Lecture Notes
Learning in Deep Artificial and Biological Neuronal Networks
HS2019

Authors:
Nik Dennler (nik.dennler@uzh.ch)
Ioan-Iutin Fodorut (ifodor@ini.uzh.ch)
Raffael Theiler (ratheile@student.ethz.ch)
Vinay Hiremath (vhirem@uzh.ch)
Farah Baracat (fbaracat@student.ethz.ch)

March 10, 2020

Disclaimer

This document is intended to help the students of the 2019 lecture "Learning in Deep Artificial and Biological Neuronal Networks" by Professor Benjamin Grewe at ETHZ in reviewing the course and preparing for the exam. It does neither guarantee completeness nor correctness and it does not claim originality. The containing information is freely gathered and often copied word-by-word from the lecture materials and other from sources, such as books, papers, blog entries and Wikipedia. The document is a collaboration between many students, some of which are noted in the authors list above.

Contents

1 Introduction

1.1 Motivation

Deep-learning, a brain-inspired weak form of artificial intelligence, allows the training of large **artificial neuronal networks** (ANNs) that, like humans, can learn real-world tasks such as recognizing objects in images. The origins of deep hierarchical learning can be traced back to early neuroscience research by Hubel and Wiesel in the 1960s, who first described the neuronal processing of visual inputs in the mammalian neocortex. Similar to their neocortical counterparts, ANNs seem to learn by interpreting and structuring the data provided by the external world. However, while on specific tasks such as playing (video) games, deep ANNs outperform humans (Minh et al, 2015, Silver et al., 2018), ANNs are still not performing on par when it comes to recognizing actions in movie data and their ability to act as generalizable problem solvers is still far behind of what the human brain seems to achieve effortlessly. Moreover, **biological neuronal networks** (BNNs) seem to learn far more effectively with fewer training examples, they achieve a much higher performance in recognizing complex patterns in time series data (e.g. recognizing actions in movies), they dynamically adapt and learn new tasks without losing performance and they achieve unmatched generalization performance to detect and integrate out-of-domain data examples (data they have not been trained with). In other words, many of the big challenges and unknowns that have emerged in the field of deep learning over the past years are already mastered exceptionally well by biological neuronal networks in our brain. On the other hand, many facets of typical ANN design might help to better understand how learning in hierarchical biological neuronal networks is organized. Recent evidence suggests that learning in biological systems is the result of the complex interplay of diverse error feedback signaling processes acting at multiple time and spatial scales, ranging from single synapses to entire networks. Untangling the parallels and differences of learning in ANNs and BNNs is the main topic of this lecture.

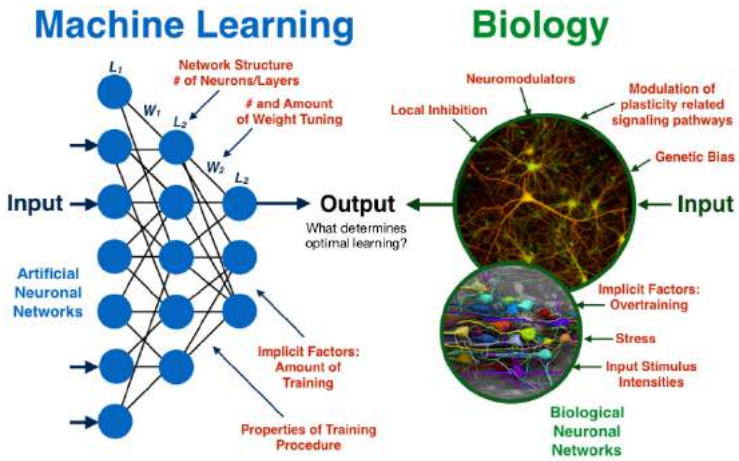


Figure 1: Comparing artificial neural networks (ANNs) with biological neural networks (BNNs).

1.2 Deep Learning

1.2.1 Applications

Driving Driverless Cars: Deep learning has become successful in solving tasks that were previously approached with rule based algortihms. The steering of a car is such an example. One can train a Convolutional Neural Network (CNN) to map raw pixels from a front-facing camera of a car directly to the steering commands[1].

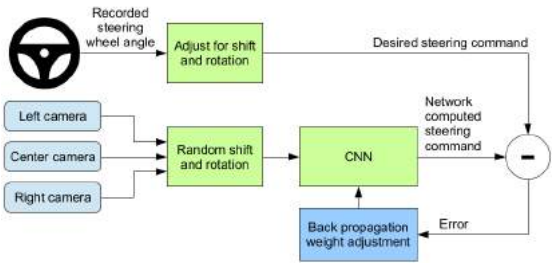


Figure 2: Training of a CNN to automatically steer a car.

Robotic Control: More complex tasks, such as the actions of robotic limbs to mimic the dynamic and agile maneuvers of animals, can be approached by deep reinforcement learning[2].

Text Generation: Natural language processing tasks, such as question answering, machine translation, reading comprehension, and summarization, are typically approached with supervised learning on task-specific data sets. If enough data provided, they can also be approached with unsupervised deep learning [3].

Speech Generation: Speech synthesis, more specifically known as text-to-speech (TTS), is a comprehensive technology that involves many disciplines such as acoustics, linguistics, digital signal processing and statistics. The main task is to convert text input into speech output. Recent advances on speech synthesis are overwhelmingly contributed by deep learning or even end-to-end techniques which have been utilized to enhance a wide range of application scenarios such as intelligent speech interaction, chatbot or conversational artificial intelligence. For speech synthesis, deep learning based techniques can leverage a large scale of <text, speech> pairs to learn effective feature representations to bridge the gap between text and speech, thus better characterizing the properties of events[4].

Image Recognition and Med. Diagnostics: Deep-learning models can be trained directly on medical data to deduce patterns and thus give precise

[1]Bojarksi, "End to End Learning for Self-Driving Cars", 2016
[2]Hwangbo, "Learning Agile and Dynamic Motor Skills for Legged Robots", 2019
[3]Radford, "Language Models are Unsupervised Multitask Learners", 2019
[4]Ning, "A Review of Deep Learning Based Speech Synthesis", 2019

diagnostics [5].

Mastering Games: Recently, several games that where though to be too complicated for machines to play proficiently, have been mastered by algorithms based on deep learning. One such example is the game Go. AlphaGo, a program developed by Google DeepMind, defeated the world champion in Go. The tree search in AlphaGo evaluated positions and selected moves using deep neural networks. These neural networks were trained by supervised learning from human expert moves, and by reinforcement learning from self-play [6].

1.2.2 Challenges

Continual learning: Being able to learn multiple tasks sequentially, rather than forgetting after learning a new task.

Robust decisions: Taking into account uncertainty and errors in inputs and feedback.

Explainable decisions: Understanding network reasoning and learning.

Security: Shared learning on confidential data. Look at the brain: Most of the data we perceive is not explicitly stored by the brain.

Composable AI systems: Combining multiple systems to solve complex tasks. Look at the brain: It is comprised of over 200 different brain areas that all learn differently.

Unsupervised learning: Learning without the need of labels by creating useful data representations. Look at the brain: It uses a combination of supervised learning, unsupervised learning and reinforcement learning[7].

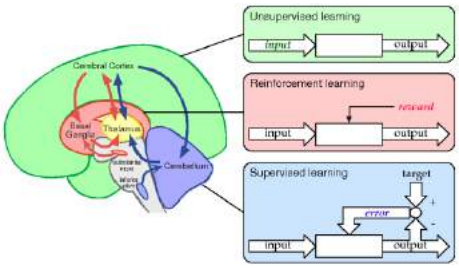


Figure 3: Supervised learning, unsupervised learning and reinforcement learning in the basal ganglia and the cerebellum.

[5] Using AI to predict breast cancer and personalize care: `http://news.mit.edu/2019/using-ai-predict-breast-cancer-and-personalize-care-0507`

[6] Silver, "Mastering the game of Go without human knowledge", 2017

[7] Doya, "Complementary roles of basal ganglia and cerebellum in learning and motor control", 2000

Fast learning: Using only a few data examples to create generalizing representations. Look at the brain: It seems to learn from extremely few examples.

1.3 The Human Brain as Universal Learning Machine

Universal Learning Machine (ULM): A simple and yet very powerful and general model for intelligent agents[8]. It is an extension of a general computer - such as Turing Machine - amplified with a universal learning algorithm. An initial untrained seed ULM can be defined by 1.) a prior over the space of models (or equivalently, programs), 2.) an initial utility function, and 3.) the universal learning machinery/algorithm. The machine is a real-time system that processes an input sensory/observation stream and produces an output motor/action stream to control the external world using a learned internal program that is the result of continuous self-optimization.

Inheritance: One criticism to typical artificial neural networks (ANNs) and supervised learning in general is, that everything is learned and nothing inherited. Contrarily, young animals (including humans) learn without seeing an enormous numbers of labeled examples. This could either lead to the belief, that animals must rely on unsupervised learning, or that most animal behavior is actually encoded in the genome. Specifically, animals are born with highly structured brain connectivity, which enables them to learn very rapidly. Because the wiring diagram is far too complex to be specified explicitly in the genome, it must be compressed through a "genomic bottleneck". The genomic bottleneck suggests a path toward ANNs capable of rapid learning[9]. There are animals, such as the simple worm *C. Elegans*, stores the entire neuronal wiring in its genome. Thus, the wiring pattern is identical for each worm.

The human brain in numbers: At 1232 grams, the human brain has about 10^{11} neurons, and more than 10^3 synapses per neuron. Specifying a connection target requires about $\log_2 10^{11} = 35$ bits/synapse. Thus it would take about $3.5 * 10^{15}$ bits ($\approx$ 400 TB) to specify all 10^{14} connections in the brain. One could therefore conclude, that the human brain is mostly learned.

Artificial General Intelligence (AGI): The intelligence of a machine that could successfully learn and perform any intellectual task similar to a human.

[8]Cool blog post: `https://www.lesswrong.com/posts/9Yc7Pp7szcjPgPsjf/the-brain-as-a-universal-learning-machine`

[9]Zador, "A critique of pure learning and what artificial neural networks can learn from animal brains", 2019

1.4 The Mammalian Neocortex

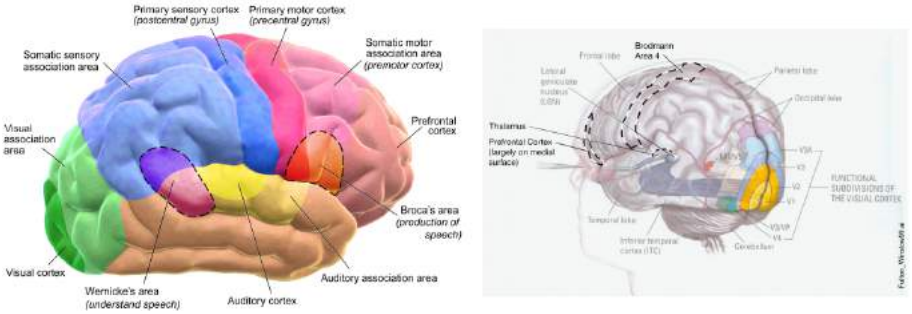


Figure 4: Illustration of the mammalian neocortex

1.4.1 Key Facts

The neocortex is ...

- the top layer of the cerebral hemispheres, 2-4 mm thick, and made up of six layers, labelled I to VI.
- part of the cerebral cortex (along with the archicortex and paleocortex, which belong to the limbic system).
- involved in higher functions such as sensory perception, generation of motor commands, spatial reasoning, conscious thought, and language.
- consists of grey matter surrounding the deeper white matter of the cerebrum.
- smaller and smoother in rats and some other small mammals, but it has deep grooves (sulci) and wrinkles (gyri) in primates and several other mammals. These folds serve to increase the area of the neocortex considerably.
- accounting for about 76% of the brain's volume of humans.

1.4.2 The 6 Layers of the Mammalian Neocortex

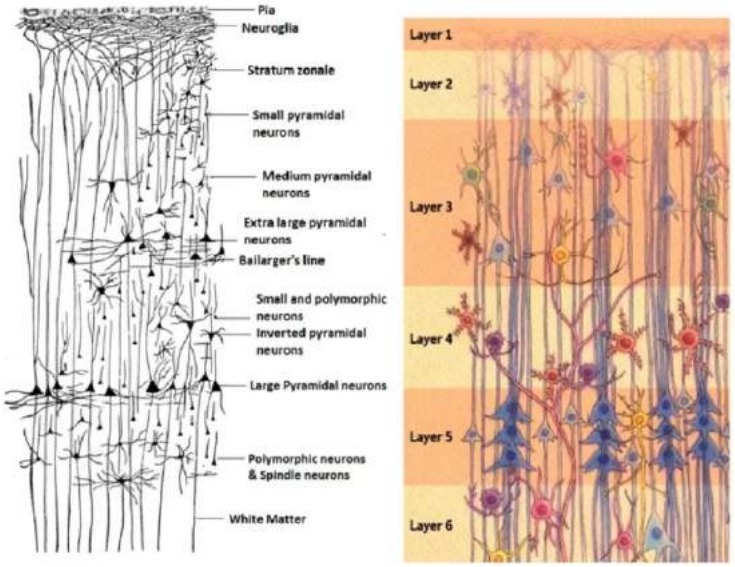


Figure 5: Caption

Layers	Components	Schematic	Afferents		Efferents
I – Molecular	Axons and Dendrites (Cell processes)		From other regions of Cortex and Brainstem	Epomedicine.com	To other regions of cortex (Intra-cortical Association functions)
II - External granular	Densely packed Stellate cells + Small pyramidal cells				
III – External pyramidal	Loosely packed Stellate cells + Medium pyramidal cells				
IV – Internal granular	Densely packed Stellate cells only			+ From Thalamus	
V – Internal pyramidal	Large pyramidal cells only (few stellate cells) – Giant Pyramidal cells of Betz			+ From Brain stem	To Brain stem & Spinal cord (Projection fibers)
VI - Multiform	Multiple sized pyramidal cells + Loosely packed stellate cells				To Thalamus

Figure 6: Caption

1.4.3 Visual Apparatus

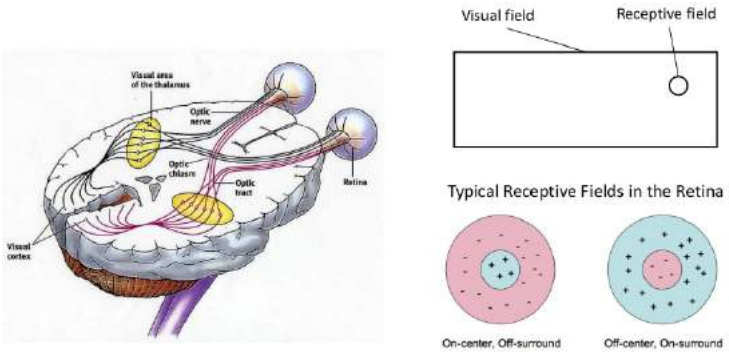


Figure 7

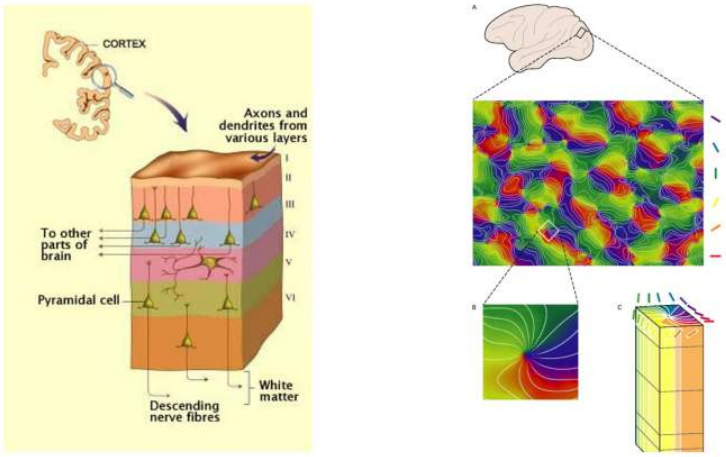


Figure 8

1.4.4 Hubel and Wiesel Experiment

The Hubel and Wiesel experiments greatly expanded the scientific knowledge of sensory processing. In one experiment, done in 1959, they inserted a microelectrode into the primary visual cortex of an anesthetized cat. They then projected patterns of light and dark on a screen in front of the cat. They found that some neurons fired rapidly when presented with lines at one angle, while others responded best to another angle. Some of these neurons responded differently to light patterns than to dark patterns. Hubel and Wiesel called these neurons "simple cells." Still other neurons, which they termed "complex cells," had identical responses to light and dark patterns. These studies showed how the visual system constructs complex representations of visual information from

simple stimulus features[10] [11], [12].

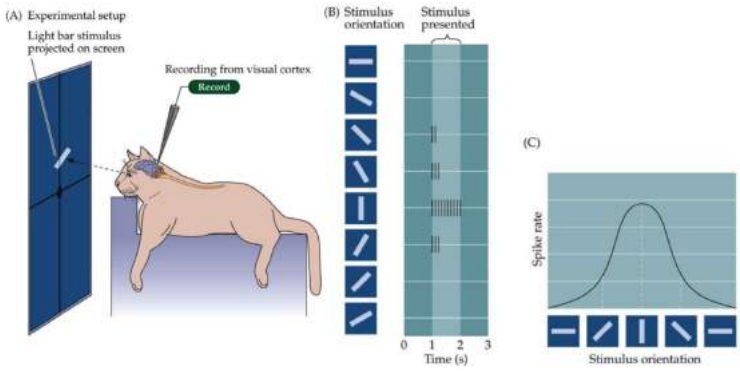


Figure 9: **A:** Hubel and Wiesels cat experiment. **B&C:** Orientation and direction selectivity in cortical neurons

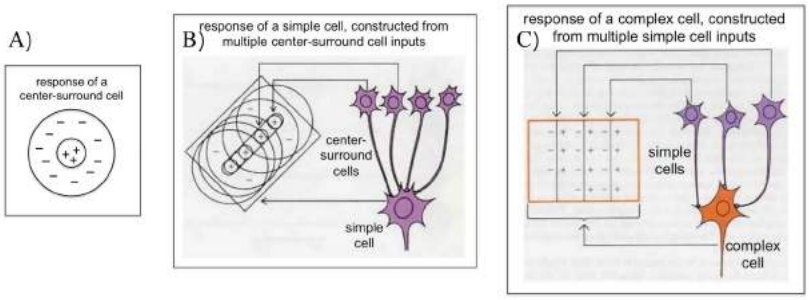


Figure 10: **A:** Depiction of center-surround cell responses. This cell is excited by light presented in a small, circular central area (plus signs) and inhibited by light in the surrounding area (minus signs). **B:** The simple cell receives input from multiple center-surround cells whose preferred areas (small circles with plus signs) are aligned in a straight line. The larger circles with minus signs again depict how light in the surrounding area inhibits the cells. **C:** A complex cell receives input from multiple simple cells that respond to lines of the same orientation (in this case, vertical) but positioned adjacently. (Plus and minus signs depict how simple cells are excited by a line in one position but inhibited by a line in an adjacent position, similar to center-surround cells. This explains why a large shape like a square won't activate the complex cell- it would trigger too much inhibition.

[10] Goldstein, 2001
[11] `https://www.brains-explained.com/how-hubel-and-wiesel-revolutionized-neuroscience/`
[12] Wiesel, "Eye, Brain, and Vision", 1988

1.5 Parallels and differences between artificial and biological networks

1.5.1 McCulloch and Pitts Neuron and the Perceptron

The McCulloch and Pitts Neuron is a highly simplified computational model of the cortical neuron. Excitatory or inhibitory inputs are aggregated; an output is produced if the aggregation reaches a certain (adjustable) threshold. The MCP takes boolean inputs only.

The Perceptron is very similar to the MCP, but it takes real valued inputs, which are weighted. Both models are linear, but by accumulating multiple units, one can achieve non-linear separation[13].

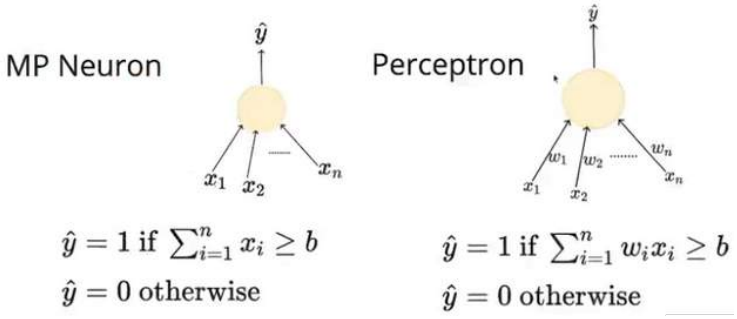


Figure 11: Comparing the MCP and the Perceptron

1.5.2 Hierarchical Models

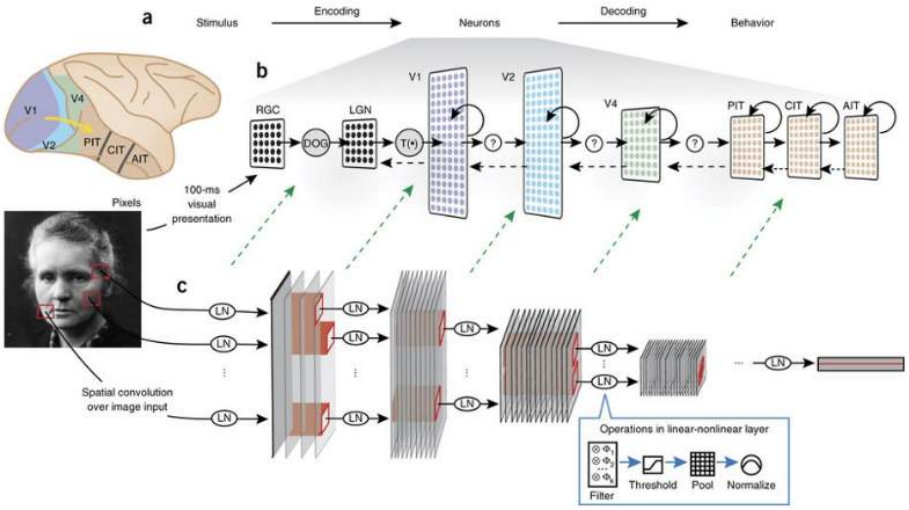


Figure 12

[13] https://medium.com/analytics-vidhya/mp-neuron-and-perceptron-model-with-sample-code-c2189edebd3f

1.5.3 Parallels

How is information processing and learning in deep networks and biological networks similar?

1.5.4 Differences

How does information processing and learning in deep networks and biological networks differ?

1.6 Experimental Methods in Neuroscience

Electrophysiology is the branch of physiology that studies the electrical properties of biological cells and tissues. It involves measurements of voltage changes or electric current or manipulations on a wide variety of scales from single ion channel proteins to whole organs like the heart. In neuroscience, it includes measurements of the electrical activity of neurons, and, in particular, action potential activity[14].

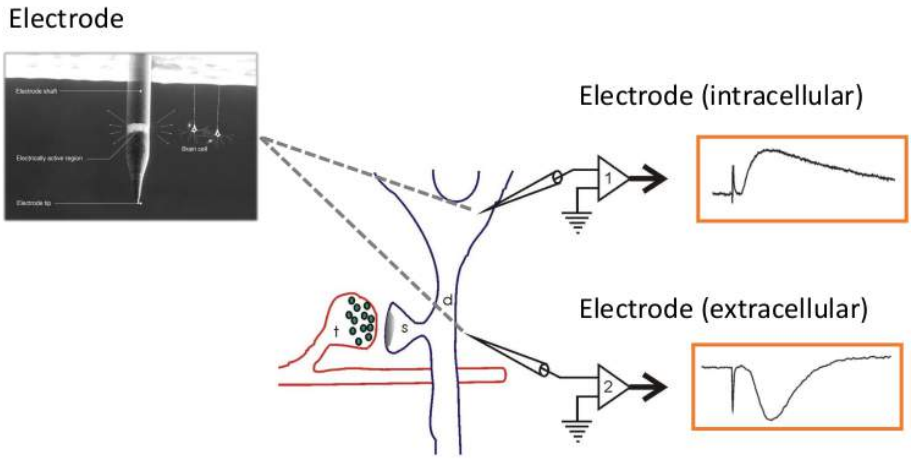


Figure 13

Calcium Imaging is a indirect measurement method which is designed to show the calcium (Ca^{2+}) status of an isolated cell, tissue or medium. Calcium imaging techniques take advantage of so-called calcium indicators, fluorescent molecules that can respond to the binding of Ca2+ ions by changing their fluorescence properties. Two main classes of calcium indicators exist: chemical indicators and genetically encoded calcium indicators (GECI). Calcium imaging can be used to optically probe intracellular calcium in living animals. This technique has allowed studies of calcium signalling in a wide variety of cell types and neuronal activity in hundreds of neurons and glial cells within neuronal circuits[15].

[14] https://en.wikipedia.org/wiki/Electrophysiology
[15] https://en.wikipedia.org/wiki/Calcium_imaging

Functional Magnetic Resonance Imaging (fMRI) indirectly measures brain activity by detecting changes associated with blood flow. This technique relies on the fact that cerebral blood flow and neuronal activation are coupled. When an area of the brain is in use, blood flow to that region also increases. The primary form of fMRI uses the blood-oxygen-level dependent (BOLD) contrast[16].

[16] https://en.wikipedia.org/wiki/Functional_magnetic_resonance_imaging

2 Training Methods for Deep ANNs

2.1 Recap: Supervised Learning in ANNs

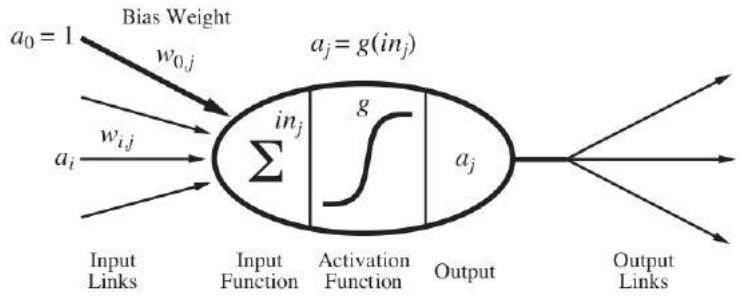


Figure 14: Simple mathematical model of a hidden neuron: From left to right: Input links constitute of weights coming from previous neurons and from a bias term. The input function, here a sum, bundles the weights. The activation function performs a nonlinear mapping of the input to a output value, which is subsequently passed on to the next neuron.

An artificial neural network (ANN) is a computational model that is vaguely inspired by the biological network of neurons constituting the brain of vertebrates. It can be used as a trainable classifier of data points. Similarly to the biological analogue, it consists of a set of computing units, or **neurons** and of directed links connecting them. The strength and sign of a link is given through its **weight**. The neurons take in a set of inputs and produce an output based on a given **input function** and a given non-linearity, the **activation function**. A schematic overview of a neuron is given in fig. 14[17]. If we bundle many neurons to a layer, and then connect multiple layers by linking the neuronal output of each layer to the neuronal input of the next layer, we get a deep neural network structure, where we differentiate between the input layer, the output layer and the in-between-laying hidden layers. Based on data on the input layer, the network will perform a **forward-pass** of the information and make a prediction (this can be a classification or regression). During **training**, the prediction is then compared with the ground-truth, or label, of the data. A **loss** is calculated based on a difference-norm between the prediction and the true label of the data point. Subsequently, weights of the network are updated. If **back-propagation** is used, which is the most common algorithm for supervised learning of ANNs, the derivative of the loss with respect to each weight is obtained, the information passed backwards and the weights adjusted accordingly.

The above described concept can be formulated mathematically. A network with L layers can be defined as

$$\boldsymbol{h}^j = \sigma^j(W^j \boldsymbol{h}^{j-1}) = \sigma^j(\boldsymbol{z}^j),\ j = 1, ..., L, \tag{1}$$

[17]Figure and caption adapted from "Russel, AI: A Modern Approach"

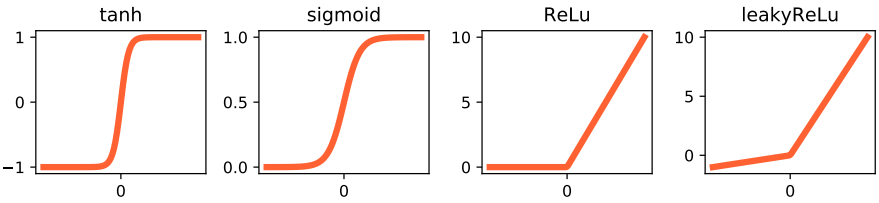


Figure 15: Common activation functions: a) $\tanh(x)$ b) $sigmoid(x)$ c) $ReLu(x)$ d) $leakyReLu(x)$

where $\boldsymbol{h}^j$ is the state of the j-th hidden layer and $\boldsymbol{z_i^j} = \sum_{k=0}^{N^j} w_{ik}^j h_i^{j-1}$ the input for the i-th neuron in the j-th hidden layer. $\boldsymbol{h}^L$ is the output layer and $\boldsymbol{h}^0 = \boldsymbol{x}$ the input layer. The forward-mapping is defined by the non-linear activation function σ^j, from which a set of commonly used examples are illustrated in fig. 15, and the weight vector W^j. Most often, the bias term of the j-th layer w_0^j is included in the weight vector, thus $W^j = (w_0^j, w_1^j, ..., w_N^j)$ defines the weight vector of a layer containing N^j neurons.
Now, let's fix the network architecture, meaning the number of neurons, the wiring scheme and the activation σ^j, and define the network parameters as all the weights w_i^j. If we define all parameters between layer j and l $(0 \leq j < l \leq L)$ as $\theta_W^{j,l} = W^k, k = j+1, ..., l$, we can write the l-th layer $\boldsymbol{h^l}$ as a function of the j-th layer $\boldsymbol{h^j}$, given the parameters $\theta_W^{j,l}$:

$$\boldsymbol{h^l} = \boldsymbol{h^l}(\boldsymbol{h^j}; \theta_W^{j,l}) \tag{2}$$

For a given data set $(\boldsymbol{x}, \boldsymbol{y}) = ((\boldsymbol{x_1}, \boldsymbol{y_1}), ..., (\boldsymbol{x_D}, \boldsymbol{y_D}))$, a global loss function $\mathcal{L}(\boldsymbol{h}^L(\boldsymbol{x}; \theta_W^{0,L}), \boldsymbol{y})$ gives a measure of the difference between the network output $\boldsymbol{h^L}(\boldsymbol{x}; \theta_W^{0,L})$ and the label y. During training, the goal is to minimize the expectation value of the global loss $\mathbb{E}_p\{\mathcal{L}(\boldsymbol{h}^L(\boldsymbol{x}; \theta_W^{0,L}), \boldsymbol{y})\}$ based on a data distribution $p(\boldsymbol{x}, \boldsymbol{y})$. Common loss functions are the **mean-squared-error** (MSE) for regression:

$$\mathcal{L}_{MSE}(\boldsymbol{h}^L(\boldsymbol{x}; \theta_W^{0,L}), \boldsymbol{y}) = \frac{1}{2D} \sum_{i=1}^{D} (\boldsymbol{h}^L(\boldsymbol{x_i}; \theta_W^{0,L}) - \boldsymbol{y_i})^2, \tag{3}$$

and the **cross-entropy loss** (CE) for classification with C classes:

$$\mathcal{L}_{CE}(\boldsymbol{h}^L(\boldsymbol{x}; \theta_W^{0,L}), \boldsymbol{y}) = -\sum_{c=1}^{C} \boldsymbol{y_c} \log \boldsymbol{h}_c^L(\boldsymbol{x_c}; \theta_W^{0,L}) \tag{4}$$

If the back-propagation algorithm is used, updating the weights is done by taking the derivatives of the loss $\mathcal{L}(\boldsymbol{h}^L(\boldsymbol{x}; \theta_W^{0,L}), \boldsymbol{y})$ with respect to all weights $\theta_W^{0,L}$:

$$w_j^l[t+1] = w_j^l[\ t\ ] - \eta \frac{\partial \mathcal{L}(\boldsymbol{h}^L(\boldsymbol{x}; \theta_W^{0,L}), \boldsymbol{y})}{\partial w_j^l} \tag{5}$$

The introduced **learning-rate** factor η does in general not need to be constant, thus it can be adaptive over time.

2.2 Backpropagation

2.2.1 Introduction

Backpropagation (BP) is a widely used algorithm in training feedforward neural networks for supervised learning[18]. It computes the gradient of the loss function with respect to the weights of the network for a single input/output example, and does so efficiently, unlike a naive direct computation of the gradient with respect to each weight individually. This efficiency makes it feasible to use gradient methods for training multilayer networks, updating weights to minimize loss; gradient descent, or variants such as stochastic gradient descent, are commonly used. The backpropagation algorithm works by computing the gradient of the loss function with respect to each weight by the chain rule, computing the gradient one layer at a time, iterating backwards from the last layer to avoid redundant calculations of intermediate terms in the chain rule; this is an example of dynamic programming[19].

2.2.2 Method

We want to calculate the weight error, which is the gradient of the loss with respect to the input of the neuron j of the l-th layer z_j^l:

$$\delta_j^l = \frac{\partial \mathcal{L}}{\partial z_j^l} \tag{6}$$

First we calculate the gradient with respect to the ultimate layer L:

$$\delta_j^L = \frac{\partial \mathcal{L}}{\partial z_j^L} = \frac{\partial \mathcal{L}}{\partial h_j^L}\frac{\partial h_j^L}{\partial z_j^L} = \frac{\partial \mathcal{L}}{\partial h_j^L}\sigma'(z_j^L) \tag{7}$$

Then, we calculate the gradient with respect to an intermediate layer l:

$$\delta_j^l = \frac{\partial \mathcal{L}}{\partial z_j^l} = \frac{\partial \mathcal{L}}{\partial z_j^{l+1}}\frac{\partial z_j^{l+1}}{\partial z_j^l} = \delta_j^{l+1}\frac{\partial z_j^{l+1}}{\partial z_j^l} = \sum_k \delta_j^{l+1} w_k^{l+1}\sigma'(z_j^l) \tag{8}$$

We note, that the gradient can also be written with a dependency to the weight (??! WHY is that? The first step does not make sense to me, the second one is mathematically not sound (flipping the derivation)):

$$\delta_j^l = \frac{\partial \mathcal{L}}{\partial z_j^l} = \frac{\partial \mathcal{L}}{\partial w_k^l}\frac{\partial w_k^l}{\partial z_j^l} = \frac{\partial \mathcal{L}}{\partial w_k^l}\frac{1}{h_k^{l-1}} \tag{9}$$

Thus, we arrive at the recursive form

$$\frac{\partial \mathcal{L}}{\partial w_k^l} = h_k^{l-1}\delta_j^l, \tag{10}$$

which gives us the weight update equation as follows:

$$w'^l{}_k = w_k^l - \eta\Delta w_k^l = w_k^l - \eta\frac{\partial \mathcal{L}}{\partial w_k^l} = w_k^l - \eta h_k^{l-1}\delta_j^l, \tag{11}$$

where η is the learning rate.

[18] Rumelhart&Hinton, "Learning representations by back-propagating errors", 1986
[19] `https://en.wikipedia.org/wiki/Backpropagation`

2.2.3 Drawbacks

- BP might not be optimal, in particular to train generative models.
- BP is not fully local and computation is not easy to parallelize.
- BP needs to store all network activations for the BP steps.
- Understanding DNN training and optimization methods might provide a different angle to understand learning in biological networks in the Brain.
- Given that the functionality of DNNs is reflected by biological networks the different variants of DNN training even provide testable hypothesis that neuroscience research can test.
- Biological plausibility issues, by Bengio[20]:
    1. The back-propagation computation is purely linear, whereas biological neurons interleave linear and non-linear operations.
    2. If the feedback paths were used to propagate credit assignment by back-propagation, they would need precise knowledge of the derivatives of the non-linearities at the operating point used in the corresponding feedforward computation.
    3. Similarly, these feedback paths would have to use exact symmetric weights (with the same connectivity, transposed) of the feedforward connections.
    4. Real neurons communicate by (possibly stochastic) binary values (spikes)
    5. The computation would have to be precisely clocked to alternate between feedforward and back-propagation phases.
    6. It is not clear where the output targets would come from.

2.3 Feedback Alignment

2.3.1 The Method

From a biological perspective, one of the biggest issues with backpropagation is that it uses the same weights for the forward and the backward pass. This is tackled with a learning method called feedback alignment[21]. Instead of using the transposed weight matrix $W^T$for the backward pass, it uses a matrix of randomly initialized and fixed weights B. On simple examples (such as MNIST), the method has been shown to work almost as well as regular back-propagation. The original illustrations and caption can be seen in fig. 16. We note the change in weight update:

Backpropagation

$$\Delta w_k^l = h_k^{l-1}\delta_{BP}{}_j^l = h_k^{l-1}\sum \delta_{BP}{}_j^{l+1} w_{jk}^{l+1}\sigma'(z_j^l) \tag{12}$$

[20]Lee&Bengio et al, "Difference Target Propagation", 2015

[21]Lillicrap, "Random synamptic feedback weights support error backpropagation for deep learning", 2016

Feedback Alignment

$$\Delta w_k^l = h_k^{l-1} \delta_{FA} {}_j^l = h_k^{l-1} \sum \delta_{FA} {}_j^{l+1} \beta_{jk}^{l+1} \sigma'(z_j^l) \tag{13}$$

where $\boldsymbol{\beta}^l$ is a random matrix with fixed weights (does not get updated) belonging to the l-th layer.

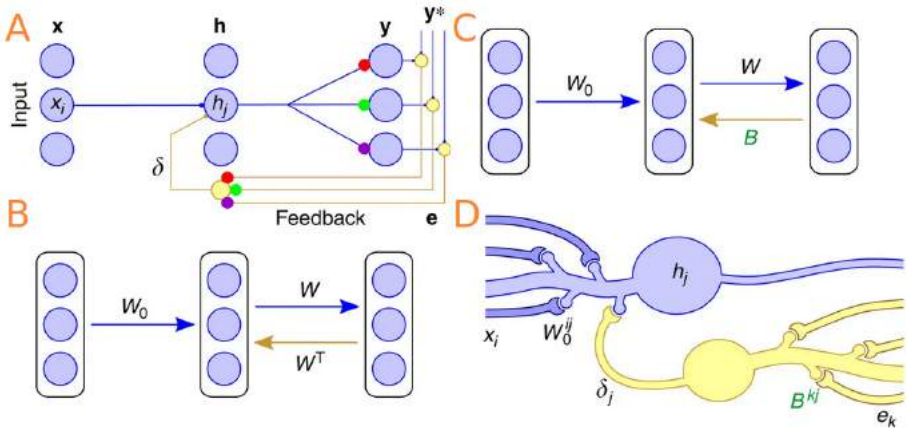


Figure 16: **(A)** The backprop learning algorithm requires that neurons know each others' synaptic weights, for example, the three coloured synapses on the feedback cell at the bottom must have weights equal to those of the corresponding coloured synapses in the forward path. **(B)** Backprop computes teaching, or modulator, vectors by multiplying the error vector $\boldsymbol{e}$ by the transpose of the forward weight matrix W, that is, $\delta_{BP} = W^T \boldsymbol{e}$. **(C)** Our feedback alignment method replaces W^T with a matrix of fixed random weights, B, so that $\delta_{FA} = B\boldsymbol{e}$. Thus, each neuron in the hidden layer receives a random projection of the error vector. **(D)** Potential synaptic circuitry underlying feedback alignment, shown for a single hidden unit (matrix superscripts denote single synapses, see main text for further explanation). This diagram is provide for illustrative purposes. There are many possible configurations that could support learning with feedback alignment, or algorithms like it, and it is this structural flexibility that we believe is important.

2.3.2 Why does FA work?

Some mathematical reasoning and the convergence proof is shown in the original paper and stated again in a follow-up paper. Intuitively; for FA the feedback weights are fixed, but if the forward weights are adapted, they will approximately align with the pseudo-inverse of the feedback weights in order to make the feedback useful. In some sense, the network learns how to learn, which is pretty dope.

2.3.3 Variations of Feedback Alignment

There are variations of feedback alignment, which show to be useful especially for deeper network architectures. The figure below shows Feedback Alignment

(FA), Direct Feedback Alignment (DFA) and Indirect Feedback alignment[22] (IFA). In fig. 18 we see the Bi-directional Feedback Alignment[23] (BFA).

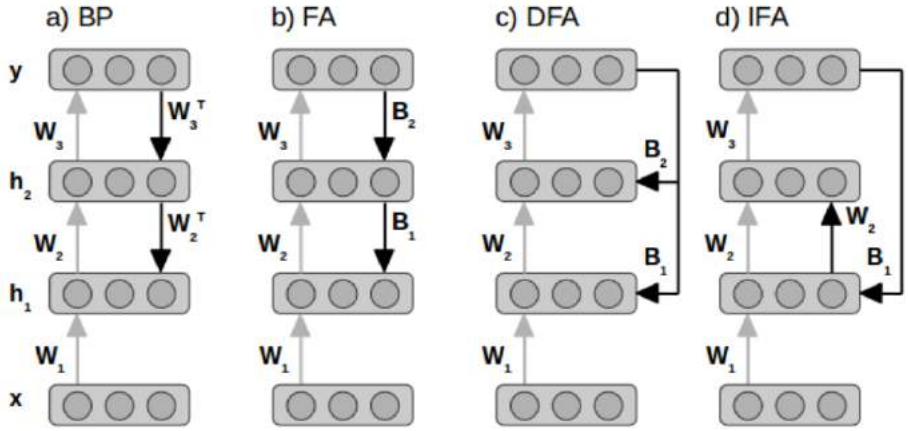


Figure 17: Overview of different error transportation configurations. Grey arrows indicate activation paths and black arrows indicate error paths. Weights that are adapted during learning are denoted as W_i , and weights that are fixed and random are denoted as B_i. **(a)** Back-propagation. **(b)** Feedback-alignment. **(c)** Direct feedback-alignment. **(d)** Indirect feedback-alignment.

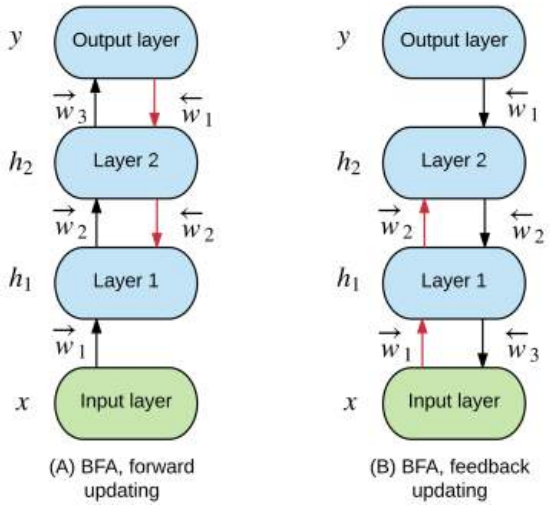


Figure 18: Bidirectional feedback alignment (BFA). Black arrows represent forward activation paths. Red arrows indicate error (gradient) propagation paths.

[22]Nøkland, "Direct Feedback Alignment Provides Learning in Deep Neural Networks", 2016
[23]Luo, "Adaptive Bidirectional Backpropagation: Towards Biologically Plausible Error Signal Transmission in Neural Networks", 2017

2.4 Target Propagation

2.4.1 The Method

The concept of Target Propagation (targetprop) goes back to LeCun 1986[24]. The intuition is simple[25]: instead of focusing solely on the "forward-direction" model ($y = f(x)$), we also try to fit the "backward-direction" model ($x = g(y)$). f and g form an auto-encoding relationship; f is the encoder, creating a latent representation and predicted outputs given inputs x, and g is the decoder, generating input representations/samples from latent/output variables. The main idea is to compute targets rather than gradients, at each layer,[26]. Like gradients, they are propagated backwards. In a way that is related but different from previously proposed proxies for back-propagation which rely on a backwards network with symmetric weights, target propagation relies on auto-encoders at each layer. Unlike back-propagation, it can be applied even when units exchange stochastic bits rather than real numbers. By its nature, target propagation can in principle handle stronger (and even discrete) non-linearities, and it deals with biological plausibility issues (1), (2), (3) and (4) described above. A sketch of the working principle is shown below.

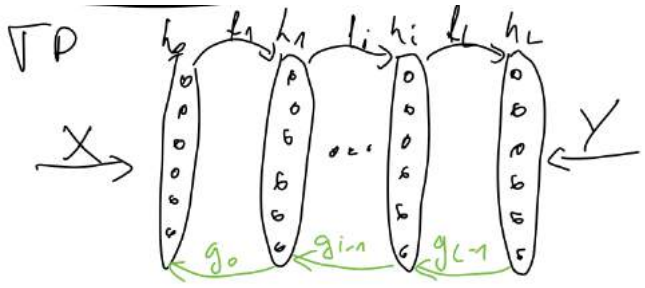


Figure 19

Forward pass

$$\boldsymbol{h}^j = f^j(\boldsymbol{h}^{j-1}) = \sigma^j(W^j \boldsymbol{h}^{j-1}) = \sigma^j(\boldsymbol{z}^j),\ j = 1, ..., L, \tag{14}$$

Backward pass

$$\hat{\boldsymbol{h}}^j = g^j(\hat{\boldsymbol{h}}^{j+1}),\ j = L-1, ..., 0, \tag{15}$$

where $\hat{\boldsymbol{h}}^j$ is the target of the j-th layer and g^j is a function that must approximate the inverse of f^{j+1}. The output target is defined as the activation of the output layer tweaked in a direction of lower loss:

$$\hat{\boldsymbol{h}}^L = \boldsymbol{h}^L - \hat{\eta}\boldsymbol{e}^L \tag{16}$$

$$\boldsymbol{e}^L = \frac{\partial \mathcal{L}}{\partial \boldsymbol{h}^L} \tag{17}$$

[24]LeCun, "Learning processes in an asymmetric threshold network", 1986

[25]`http://www.breloff.com/no-backprop-part2/`

[26]Lee&Bengio et al, "Difference Target Propagation", 2015

Learning the forward weights Based on the hidden layer targets $\hat{\boldsymbol{h}}^j$, local layer losses $\mathcal{L}^j(\boldsymbol{h}^j_{true}, \hat{\boldsymbol{h}}^j)$ are defined (for example the L2 loss). The forward weights W^j are then updated as follows:

$$\Delta W^j = -\eta_{f^j} \frac{\partial \mathcal{L}^j(\boldsymbol{h}^j_{true}, \hat{\boldsymbol{h}}^j)}{\partial W^j} \tag{18}$$

Learning the backward mapping The backward mapping g^j should approximate the inverse of the forward mapping f^{j+1}. Target propagation achieves this by "learning g^j" through an autoencoder setting of f^{j+1} (encoder) and g^j (decoder). g^j is parametrized with a non-linear activation function t^j and a backwards weight matrix $\boldsymbol{Q}^j$:

$$g^j(\hat{\boldsymbol{h}}^{j+1}) = t^j(\boldsymbol{Q}^j \hat{\boldsymbol{h}}^{j+1}) \tag{19}$$

The decoder g^j can then be trained via a reconstruction loss:

$$\mathcal{L}^j_{rec}(g^j(f^{j+1}(\boldsymbol{h}^j)), \boldsymbol{h}^j) = \|g^j(f^{j+1}(\boldsymbol{h}^j)) - \boldsymbol{h}^j\|^2 \tag{20}$$

$$\Delta \boldsymbol{Q}^j = -\eta_{g^j} \frac{\partial \mathcal{L}^j_{rec}}{\partial \boldsymbol{Q}^i} \tag{21}$$

The total loss is then composed by the local layer loss and the reconstruction loss of g^j and $(f^{j+1})^{-1}$:

$$\mathcal{L}^j = \mathcal{L}^j_{local}(\boldsymbol{h}^j_{true}, \hat{\boldsymbol{h}}^j) + \mathcal{L}^j_{rec}(g^j(f^{j+1}(\boldsymbol{h}^j)), \boldsymbol{h}^j) \tag{22}$$

2.4.2 Difference Target Propagation

So far, a vanilla Target Propagation did not seem to be successful at all. The biggest reason is, that the learned function g^j will never be a perfect inverse of f^{j+1} and thus the reconstruction loss always large. Difference Target Propagation[27] approached this problem by proposing a correction term in the propagated target $\hat{\boldsymbol{h}}^j$ that should help in getting rid of the reconstruction error:

$$\hat{\boldsymbol{h}}^j = g^j(\hat{\boldsymbol{h}}^{j+1}) + (\boldsymbol{h}^j - g^j(\boldsymbol{h}^{j+1})) \tag{23}$$

Note that if g^j is exactly the inverse, then it is the same as vanilla target propagation. An illustration to direct target propagation is shown below.

[27] Lee&Bengio et al, "Difference Target Propagation", 2015

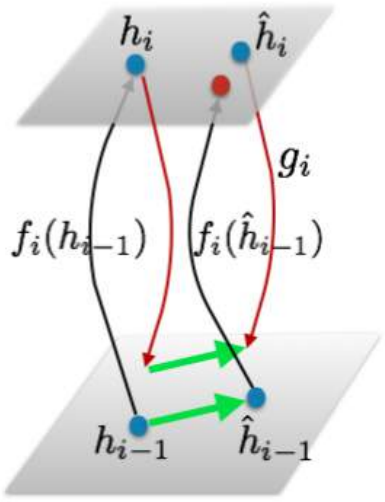


Figure 20: How to compute a target in the lower layer via difference target propagation. $f^i(\hat{h}^{i-1})$ should be closer to $\hat{h}^i$ than $f^i(h^{i-1})$

2.5 Local Learning

There are approaches that train neuronal networks using Local Layer error signals[28] [29]. This has the advantages, that activations don't occupy space in memory, and that parallelization becomes easy (each layer in own GPU, train all simultaneously). One can use a combination of Similarity Matching Loss (sim) and Cross Entropy Loss (pred).

[28] Nokland, Training Neural Networks with Local Error Signals, 2019
[29] Mostafa, "Deep Supervised Learning Using Local Errors", 2018

3 Plasticity In The Brain

3.1 Why do we need plasticity?

Why this topic is relevant:

- Biological plasticity might provide a different angle to understand the training procedures in DNNs.
- Given the effectiveness of human brain learning bio-plasticity might provide inspiration/new ideas for improved DNN training algorithms. Example: the idea of drop out (in DNNs the network randomly get rid of single neurons to prevent too much information to be stored in single neurons and weights. This was observed in neuroscience first)
- Understanding biological plasticity might help to better understand how (hierarchical) learning is organized in the brain (Learning and Memory).
- Many neural disorders such as dementia might relate to a disturbance in neuronal plasticity that cause neuronal networks to become dysfunctional. Some plasticity mechanisms /disorders lead to malfunction in the brain.
- All intelligent behavior is learned so we need plasticity to learn.
- The brain a universal learning machine, some information is genetically present at birth such as babies ability to distinguish faces but the rest is learned thus requires plasticity.

Plasticity Allows acquisition of knowledge/information and the formation of memory through experience

Memory Storage of information that can be recalled later. Learning results in memory - which has further outcome - it can change future behavior

Learning Curve Training to do tasks or reacting to stimuli, at the beginning bad and over the course of a few days vastly improves

Everything we do is *learning* and for that we need *plasticity*. An illustrative example below[30], according to the motto: "Here I get a reward, here not".

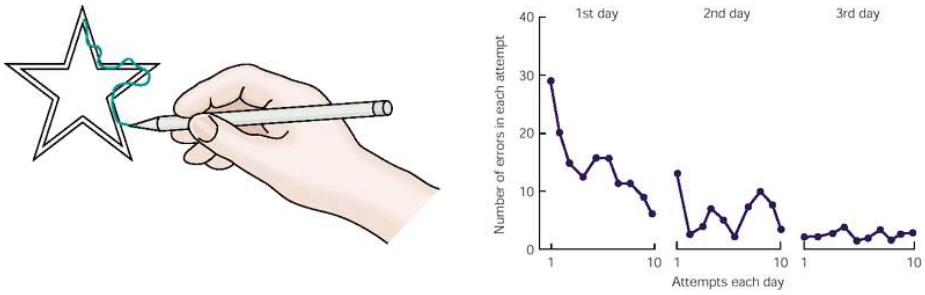


Figure 21: Motor learning task. Some human subjects have got the task to draw. Having gotten an (electrical) feedback, this subjects learned the pattern they had to draw

[30] Kandel, *Principles of neural science*, 2001

3.2 Synaptic Plasticity

3.2.1 Biological Recap

In synaptic plasticity one understands changes in reactivity of **post**- to **pre**synaptic neuron. In other words, plasticity changes of chemical synapses can strengthen or weaken transmission. This changes can last from milliseconds (short-term) to weeks or longer (long-term).

EPSP Excitatory postsynaptic potential

LTP long term potentiation; long lasting increase in synaptic strength, i.e. EPSP amplitude. Through more AMPA receptors and more released vesicles per action potential, the neuron depolarizes more after recieving an AP.

LTD long term depression; long lasting decreaes in synaptic strength; neurons do not depolarize as much.

Glutamate Neurotransmitter leading to excitability depolarization

GABA Neurotransmitter leading to inhibition hyperpolarization

Recap of Synaptic Plasticity (*see* Figure 22):

1. Action potential (AP) comes down the axon of neuron 1, this causes a change in voltage (depolarisation)
2. Ca^{2+} channels open, Ca^{2+} influx
3. Neurotransmitters (NT) get released
4. NT cross synaptic cleft and bind to the dendrite of neuron
5. bind to postsynaptic receptors (e.g. AMPA receptors), which increases the depolarisation probability.

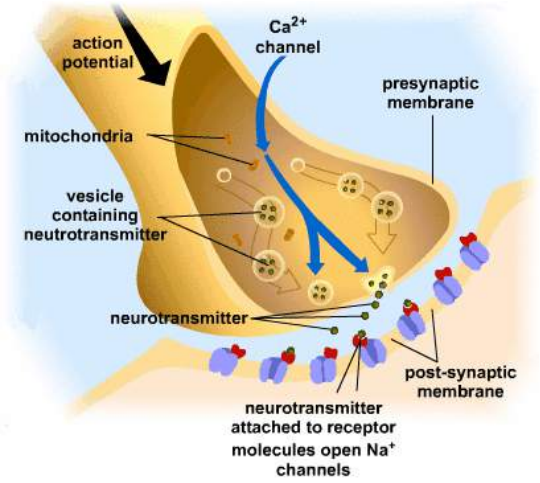


Figure 22: Pre- and post-synaptic neuron

Synaptic Plasticity Alters the interneuron Connection Strength: As the Amplitude of the EPSP increase it leads to a change in the producing and receiving neuron. This allows the neuron to scale up or down in response to the received activity. In the short term the AMPA / NMDA channels increase, synaptic density size changes, in the long term (around 30 minutes) the number of spines increase thus creating more synapses

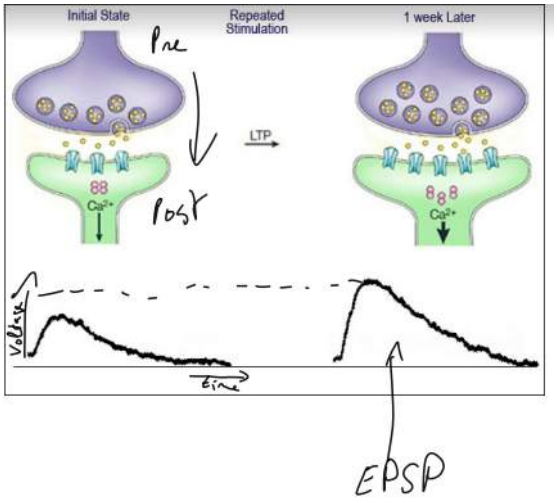


Figure 23

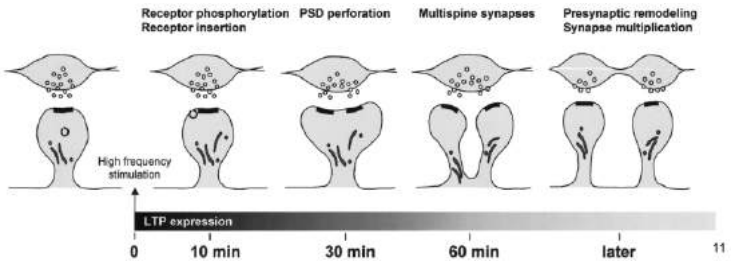


Figure 24

3.2.2 Homeostatic Plasticity

Info in this section is extracted from the paper by Turrigiano [31].

Homeostatic plasticity is necessary to regulate neural circuits; to prevent them from becoming hyper- or hypo-active. In other words, homeostatic plasticity tries to break the positive feedback loop and make the neural circuit more stable.

[31] Turrigiano. Homeostatic Synaptic Plasticity: Local and Global Mechanisms for Stabilizing Neuronal Function. 2012.

But then, the question is **why would plasticity alone destabilize the activity of neural circuits?**

- Although the synaptic-specific correlation-based mechanisms such as LTP and LTD contribute to learning and memory, they generate a powerful **destabilizing force** on network function.
- This is because when synapses undergo LTP they are better able to depolarize the postsynaptic neurons, which will increase the probability that they will undergo further LTP—leading to **unconstrained synaptic strengthening**
- As correlated activity drives strengthening of specific synapses, and the post-synaptic neuron is driven more strongly, synapses that initially were only poorly correlated with postsynaptic firing will be better able to fire the postsynaptic neuron, so they too can be- come strengthened. Hence, the specificity breaks down.

For the above mentioned reasons, we therefore need forces to prevent further excitability and constraint the neural activity. To be considered truly homeostatic, a plasticity mechanism should regulate a key parameter (such as **average neuronal firing rate**) around some set-point value (See. fig.26)

Example illustrating the importance of regulating the overall activity: [32] One way to illustrate this is to consider the problem of propagating patterned activity from the sensory periphery to higher-order neurons deep within the brain. We can draw this as a feedforward hierarchical network as in fig. fig. 25 where the activity in each layer (for example, photorecep- tors, bipolar cells, ganglion cells, lateral geniculate neurons, primary visual cortical neurons, and so on) is driven by activity in the preceding layer. We can see that in this simplified topology, the gain of transmission from one layer to the next must be close to unity for propagation to occur, otherwise the signal either dies out or specificity is lost. This is because more neurons will be recruited at successive stages, and at the highest levels all neurons will fire regardless of the pattern of firing at the input layer.

When information is propagated through a hierarchical network you want a certain amount of information expressed at each layer this prevent loss or explosion of information. This is similar to what occurs in deep networks, batch normalisation and weight decay, you want a certain activation so that there is not uniform de/activation in the layers.

[32] Turrigiano and Nelson. Homeostatic Plasticity in the developing nervous system. *Nature Reviews Neuroscience*.2004

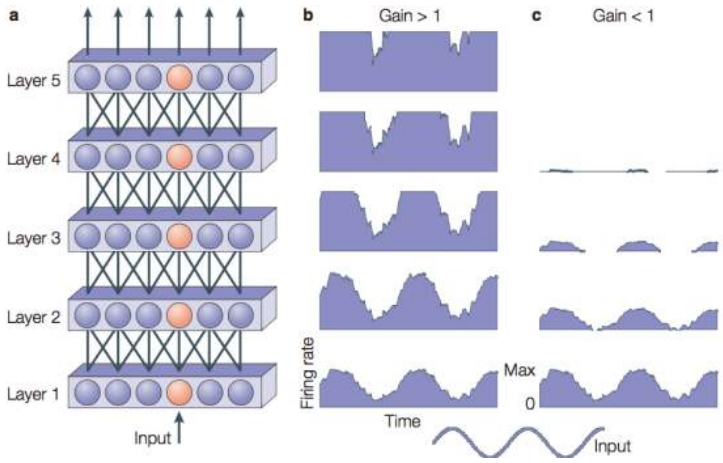


Figure 25: The problem of stability in feedforward networks. a) Schematic feedforward network consisting of five layers of neurons. b,c) Expected firing rates of the red neuron in each layer in response to a sinusoidal input. If each action potential in a preceding layer causes more than one action potential in neurons at the next layer (b), the firing rates saturate and eventually all information about the stimulus is lost. If each action potential causes less than one action potential in neurons at the next layer (c), the signals eventually fail to propagate.

Regulating the activity of a neuron around a set point: There are many phenomema that contribute to stabilization of neuronal activity. An example is pre- and postsynaptic forms of excitatory synaptic plasticity, such as **synaptic scaling**, that adjust all of a neuron's excitatory synapses up or down in the right direction to stabilize firing rate. Note that this scaling happens to all the synapses by the same "multiplicative" factor; therefore the relative synpatic stregth is preserved.

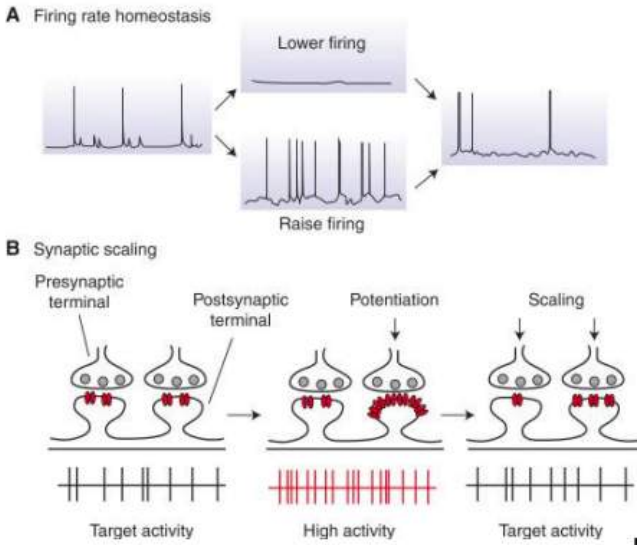


Figure 26: Homeostasis of neuronal firing through homeostatic synaptic plasticity. (A) Cartoon illustration of the phenomenon of firing rate homeostasis in dissociated neocortical networks; perturbing firing in either direction results in the homeostatic regulation of synaptic and intrinsic properties so that baseline firing rates are restored. (B) One mechanism contributing to the firing rate homeostasis illustrated in A is synaptic scaling. When activity is perturbed (illustrated here as the potentiation of some inputs through Hebbian mechanisms) this triggers synaptic scaling, which **produces a proportional reduction in strength at all synapses** of the right magnitude to return firing to baseline levels. Note that, because this mechanism scales synaptic strength up or down proportionally, the relative difference in synaptic strengths induced by Hebbian mechanisms is preserved.[Caption extracted from paper].

Regulation of firing rate in cultured networks: Initial observations using cortical cultures indicated that cortical pyramidal neurons maintain a set-point firing rate in the face of changing synaptic input. Cortical and other central neurons in culture form excitatory and inhibitory networks that develop spontaneous activity, and early studies found that blocking this activity for prolonged periods resulted in hyperactivity in these networks when activity was allowed to resume. The reciprocal manipulation — elevating network activity by reducing a fraction of inhibition — initially raises firing rates, but over many hours firing rates fall again until they approach control levels.

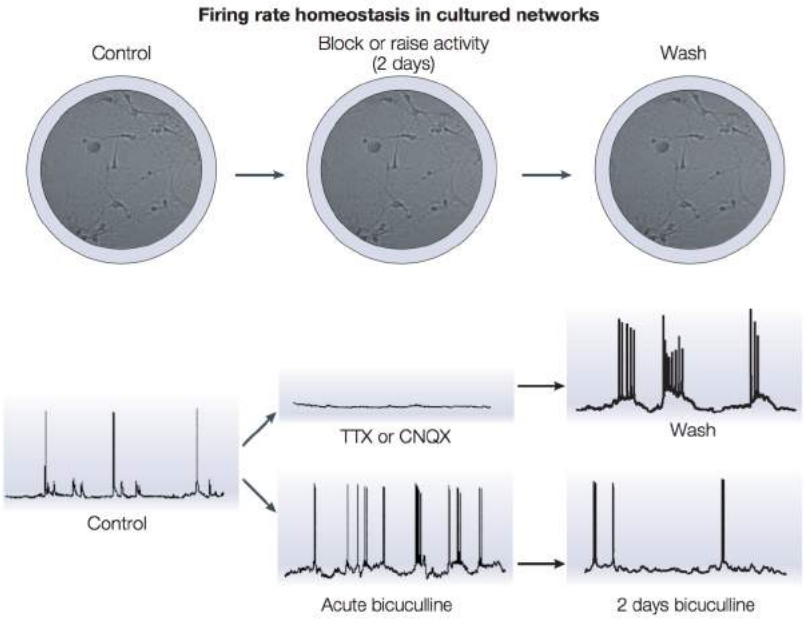


Figure 27: Evidence for firing rate homeostasis in cultured networks. Cultured cortical networks are composed of interconnected excitatory pyramidal and inhibitory interneurons, and develop spontaneous activity after a few days in vitro (control). This activity can be pharmacologically manipulated for long periods. Blockade for two days of spiking activity with tetrodotoxin (TTX), or of excitatory glutamagergic synapses with CNQX, generates a rebound phenomenon whereby the excitability of the network is increased when the drugs are removed (wash). A more direct test of the idea of firing rate homeostasis is to raise activity acutely with bicuculline (acute bicuculline), and then to follow activity over time. After two days in bicuculline, activity has returned almost to control levels (2 days bicuculline). These experiments, and others like them, indicate that homeostatic mechanisms adjust the cellular and synaptic properties of cortical networks to compensate for changes in synaptic drive.

3.2.3 Hebbian Plasticity

There are two major forms of plasticity in thenervous system: Hebbian plasticity and homeostatic plasticity. Hebian plasticity can be considered as a special form of homestatic plasticity.

Hebbian plasticity is a synaptic-basis for associative learning. In this form of plasticity synapses change their number in relation to the stimulated synapses. So, it is **synaptic-specific**. On the other hand, in homestoatic plasticity, as mentioned in the previous section, the synaptic scaling affects all the synapses (more global change in order to maintain the activity of the neural circuit around a target point).

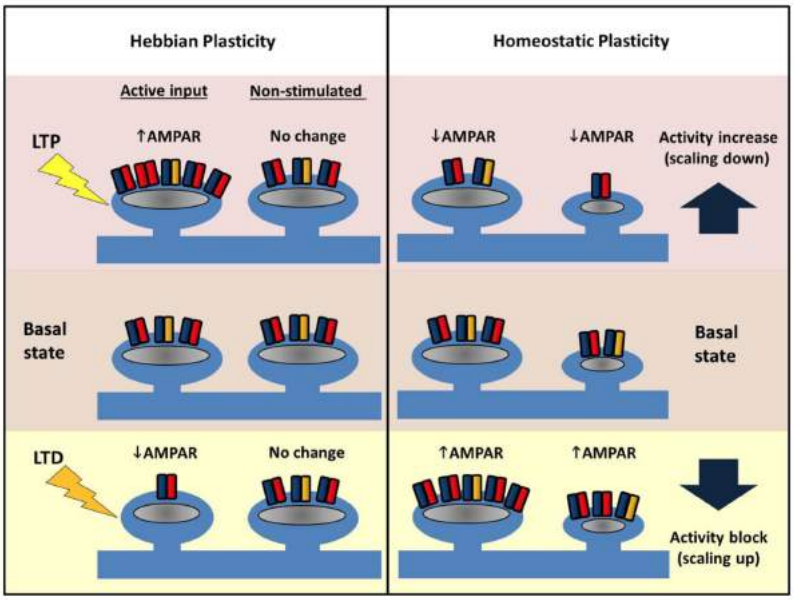


Figure 28: Comparing Hebbian and homeostatic plasticity. During Hebbian forms of plasticity synapses change their number of AMPARs in an input-specific fashion. Different patterns of activity can either cause strengthening (LTP, top left) or weakening of synapses (LTD, bottom left) via AMPAR trafficking. Potentiation or depression is limited to stimulated synapses, and neighbors are unaffected. In contrast, during homeostatic plasticity altered levels of neuronal activity drives changes in synaptic AMPAR number across the entire dendritic arbor. Blocking pre- and postsynaptic spiking with TTX causes AMPARs to accumulate at excitatory synapses (bottom right). Conversely increasing network activity (for example with a $GABA_A$R antagonist) causes a reduction in synaptic AMPAR (top right). Crucially this form of plasticity conserves the relative strength difference between synapses.

3.2.4 Some biological notions

- Synaptic scaling = Scaling up or down of the quantal amplitude of all synapses onto a postsynaptic neuron in response to long- lasting changes in neuronal activity.
- Hebbian plasticity = changes in the connection strength between two neurons as a result of correlated firing.
- AMPARS = a subtype of ligand-gated glutamate receptor; these receptors generate the majority of excitatory current at central synapses.

3.3 Hebbian Learning

Hebb's synapse: As described by Donald Hebb in *The Organization of Behavior*(1949), "When an axon of cell A is near enough to excite cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased.". Hebb postulated that this behavior of synapses in neuronal networks would permit the networks **to store memories**.

We can consider that a **Hebbian synapse is a "coincidence detector"**.

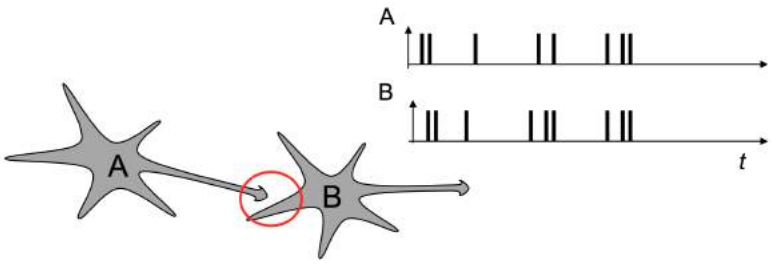


Figure 29: Hebb's synapse.

Recap

- Tuning of synapses as seen in homeostatic plasticity can be seen at specific individual synapses. When this occurs it is known as hebbian plasticity. The synapses strength increases to specifically activated neurons. Those neurons that fire together wire together associativity.
- There also can be cooperativity whereby multiple cells are required for spiking. This brings in the idea of associative learning whereby multiple features of an object linked together as it allows for better firing.

3.3.1 Spike-Timing Dependent Plasticity (STDP)

- STDP is a **temporally asymmetric** form of Hebbian learning induced by tight temporal correlations between the spikes of pre- and postsynaptic neurons[33].
- It is observed that repeated pre-synaptic spike arrival a few milliseconds before postsynaptic action potentials leads in many synapse types to Long-Term Potentiation (LTP) of the synapses. Therefore, the synaptic weight is increased.
- On the other hand, repeated spike arrival after postsynaptic spikes leads to Long-Term Depression (LTD) of the same synapse, where the synaptic weight is reduced.

[33] `http://www.scholarpedia.org/article/Spike-timing_dependent_plasticity`

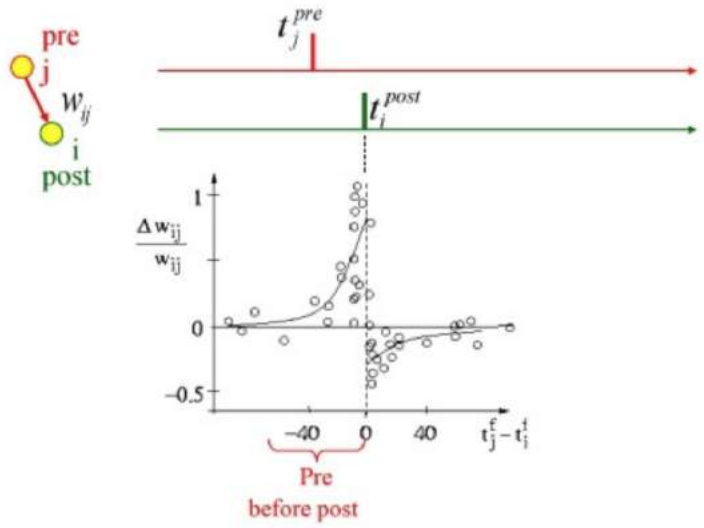


Figure 30: Spike-Timing Dependent Plasticity (schematic): The STDP function shows the change of synaptic connections as a function of the relative timing of pre- and postsynaptic spikes after 60 spike pairings. Schematically redrawn after Bi and Poo (1998) (Note that the y-axis is normalized).

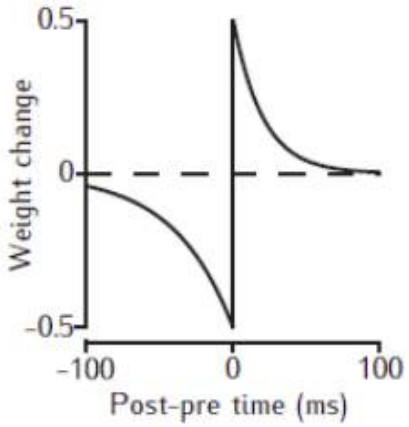


Figure 31: STPD, spike timing dependent activity, graph if the post is active then per gave input no weight change (LTD) should occur. If the post fired then activated the pre then weight increased (LTP) should occur.

$$\begin{aligned} \dot{w} &= H(\text{ pre, post }) \\ \dot{w} &= F(M,\ \text{pre, post }) \end{aligned} \tag{24}$$

Writing down hebbian learning it can be thought of as a H a function that

detects the correlation of pre and postsynaptic potentials leading to a weight change.

3.3.2 Hebb's idea and associative memory: Hebbian cell assembly

- Synaptic plasticity helps us form memories that we can recall at a later stages in our lives.
- One of the most iconic concepts of memory formation is the notion of the **Hebbian cell assembly**. Under this notion, **Hebbian learning is believed form associative memory traces or engrams**.

Hebbian cell assembly: "A Hebbian cell assembly is formed when an ensemble of neurons is repeatedly co-activated. This co-activation is thought to trigger experience-dependent "Hebbian plasticity" which causes a strengthening of the excitation connections between the co-active neurons. This so formed assembly can now be seen as an associative memory engram. The memory can be recalled by only activating a subset of the neurons within a given assembly. By dint of the strong connections, the activity can then spread to re-activate the whole assembly."[34]

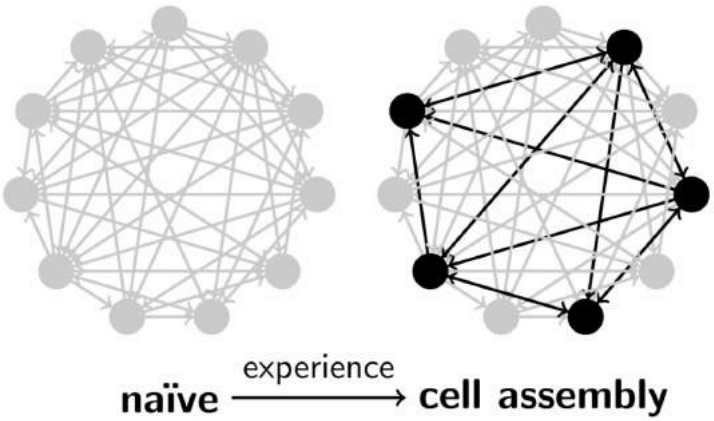


Figure 32: Formation of Hebbian cell assembly with experience

Specificity and associativity: Both are consequences of Hebbian rule.

- **Specificity** implies that only pre-synaptic neurons that are active during the depolarization of the post-synaptic neuron tend to have their synapse on the post-synaptic neuron strengthened.
- **Associativity** implies that two pre-synaptic neurons who fire at the same time, together depolarizing the same post-synaptic cell, will tend to have their synapses on the post-synaptic cell strengthened **MORE** than if either of the two cells had fired individually–the greater degree of depolarization due to the cells firing together causes the two pre-synaptic neurons to associate on the post-synaptic neuron. Further, pre-synaptic cells that

[34] https://zenkelab.org/research/synaptic-plasticity-and-homeostasis/

have a strong synapse on a post-synaptic cell can have the effect of pulling other more weakly connected cells towards the post-synaptic cell if the other cells fire at the same time as the stronger-synapsing cell.

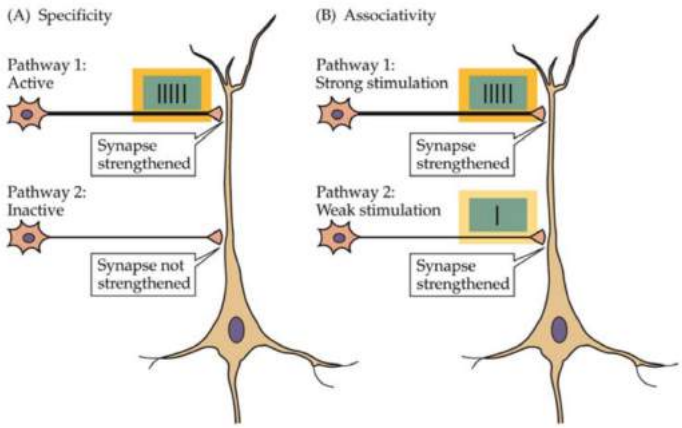


Figure 33: Two consequences of Hebbian rule: specificity and associativity

3.3.3 The Problem with Hebbian Learning

One of the main concerns of Geoffrey Hinton regarding Hebbian learning is that lack of a teaching signal. In other words, this learning rule is not error-driven; synaptic weights are updated relying on the correlation between the pre- and post synaptic activities. In his paper [35], he states that even for rather simple tasks, the "Hebbian" approach was seen as "inferior" to other error-driven methods that use the product of the pre-synaptic activity and the post-synaptic activity derivatives.

In backprop, the neurons are trained so that everyone is optimised. These errors are important for hierarchical learning to gain correct responses.

One solution for this problem is **three-factor Hebbian learning.**

3.3.4 Three-Factor Hebbian Learning

Associative (Hebbian) learning indicates association between **two factors** (two sensory inputs or an input and an output). Here the idea is to introduce another factor for modulating the plasticity. This additional factor "may represent, for example, rewards, supervised errors, summary statistics, or attentional feedback, which could be used to facilitate different types of learning by providing more global information about how well the whole network is performing or how important a current situation is."[36].

[35]The Ups and Downs of Hebb Synapses. 2003

[36]Kusmierz,Isomura and Toyoizumi. Learning with three factors: modulating Hebbian plasticity with errors. Current Opinion in Neurobiology.2017

This rule can hence be written as follows:

$$\dot{w} = F(M,\ pre,\ post) \tag{25}$$

Three-factor rules have also been called "neoHebbian" (Lisman et al., 2011; Lisman, 2017) or **"heterosynaptic" (modulatory-input dependent)**.

The M acts as the third factor, it could be the error that allows it to become error driven. Biologists are examining this could be a global reward like dopamine or local activity.

3.4 Non-Hebbian Plasticity: Heterostatic Plasticity

There are two broad categories of synaptic plasticity, generally referred to as **homosynaptic** and **heterosynaptic** plasticity.

- Homostatic plasticity is what we have been discussing with the Hebbian syapases: a synapse-specific strengthening (facilitation) or weakening (depression) based on the activity of pre- and post-synaptic neurons. In fact the three characteristics: homosynaptic plasticity, associativity and input specificity form the modern definition of the Hebbian synapse.
- Heterostatic plasticity refers to synaptic weight adaptation (facilitation or depression) based on the firing of a third modulatory interneuron. It is therefore referred to as non-hebbian plasticity.

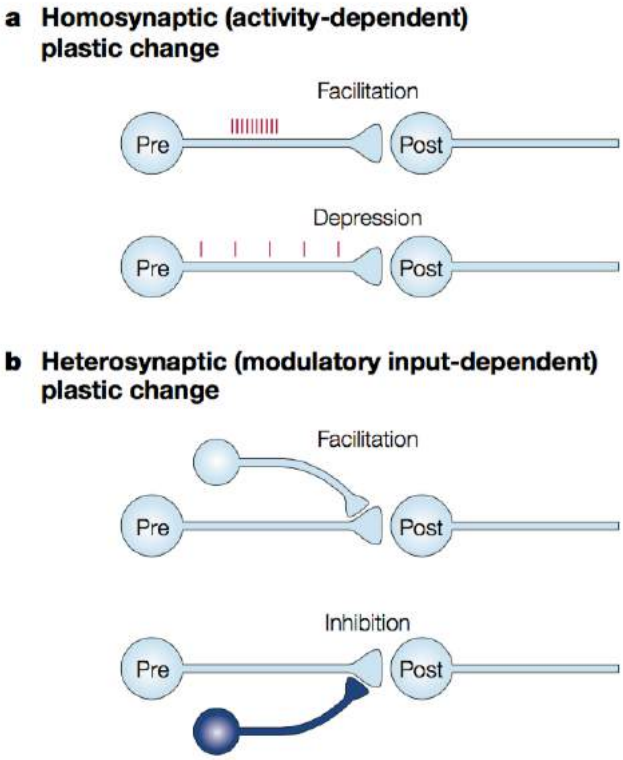


Figure 34: Homosynaptic and heterosynaptic mechanisms for long-term plasticity. a) The plastic changes that underlie long-term memory follow a homosynaptic rule, that is, the events responsible for triggering synaptic strengthening occur at the same synapse as is being strengthened. These changes can result in an increase in synaptic strength (for example, homosynaptic facilitation), or a decrease in synaptic strength (for example, homosynaptic depression). b) Synaptic strengthening between a presynaptic and a postsynaptic cell can occur as a result of the firing of a third neuron, a modulatory interneuron, whose terminals end on and regulate the strength of the specific synapse. These changes can result in an increase (heterosynaptic, modulatory facilitation) or in a decrease (heterosynaptic inhibition) in synaptic strength

Is the modulation of the input into the post from the presynaptic neuron. This then leads to the change in the post either facilitatory with excitatory inputs or inhibitory with inhibitory inputs.

Example Fly: neurons come into a brain area these act to modulate postsynaptic neurons. With greater activation the pre synatpic neurons grow irrespective of the postsynaptic activity.

Example 2: A the brain receives a reward it acts to modulate the synapse. It then changes the STDP profile it causes an increase in the synaptic strength irreseptic if the post or pre fires first. This is known as a Heterostatic Modulation.

3.4.1 Behavior time scale Plasticity

3.4.2 Time scales of synaptic plasticity (short term, LTP/LTD)

STP and STD (short term depression and potentiation) this occurs in the short term memory seconds to minutes, LTD and LTP occurs between seconds to minutes. The longer time scales the formation of new synapses as result short term processes hours to months. Then the ultra long time scales is the structural changes in brain regions. Months to lifetime.

This can be summarized as:

1. Short term facilitation/depression [seconds to hours]
2. Long term facilitation/depression
3. Generation of new synapses (synaptic pruning) [hours to months]
4. Structural changes in neuronal networks [months to lifetime]

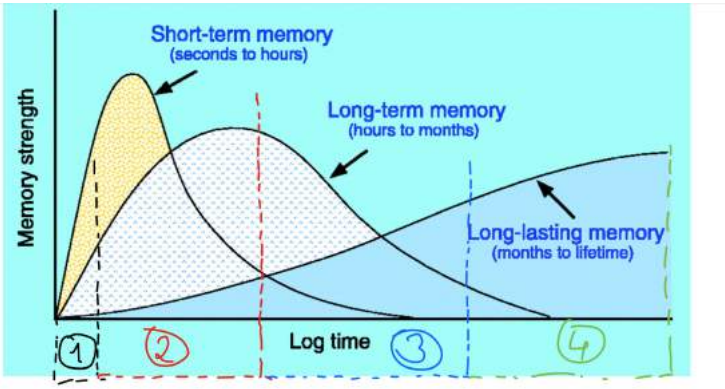


Figure 35: Timescales of Neuronal Plasticity

3.4.3 Short term plasticity (STP)

Once again, there are two types of short-term plasticity (STD): Short-Term Depression (STD) and Short-Term Facilitation (STF).

- STD is caused by depletion of neurotransmitters consumed during the synaptic signaling process at the axon terminal of a pre-synaptic neuron,
- STF is caused by influx of calcium into the axon terminal after spike generation, which increases the release probability of neurotransmitters
- STP has been found in various cortical regions and exhibits great diversity in properties (Markram et al., 1998, Dittman 2000, Wang 2006)).
- Synapses in different cortical areas can have varied forms of plasticity, being either STD-dominated, STF-dominated, or showing a mixture of both forms.

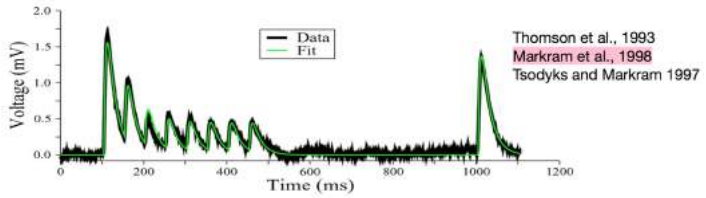


Figure 36: Short-term depression

3.5 The Hippocampus as a model system to study neural plasticity

Hippocampus (HC) is a model system of learning and memory.
Role of HC in learning and memory has been shown with rat experiments with the Morris Water Maze(MWM). MWM is large pool of opaque water where the rats are placed. The rats were trained to find and escape onto a platform which was hidden. Authors show that chronic infusion of an NMDA antagonist leads to impairment in place learning.

To summarize:

- Recent work has shown that the hippocampus contains a class of receptors for the excitatory amino acid glutamate that are activated by N-methyl-D-aspartate (NMDA) and that exhibit a peculiar dependency on membrane voltage in becoming active only on depolarization.
- Blockade of these sites with the drug aminophos-phonovaleric acid (AP5) does not affect synaptic transmission in the hippocampus, but **prevents the LTP** following brief high- frequency stimulation.

3.5.1 Most studied synapse in HC: CA3 → CA1

- The main pyramidal cell layers in HC are the CA1-4 regions (principally CA1 and CA3) and the dentate gyrus.

Schaffer Collateral/Associational Commissural Pathway: [37] This pathway is derived from axons that project from the CA3 region of then hippocampus to the CA1 region. The axons either come from neurons in the same hippocampus (ipsilateral) or from the other hippocampus (contralateral). These latter fibres are termed commissural fibres, as they cross from one hemisphere of the brain to the other. This pathway is utilised very extensively exhibits to study NMDA receptor-dependent LTP and LTD.

[37] `http://www.bristol.ac.uk/synaptic/pathways/`

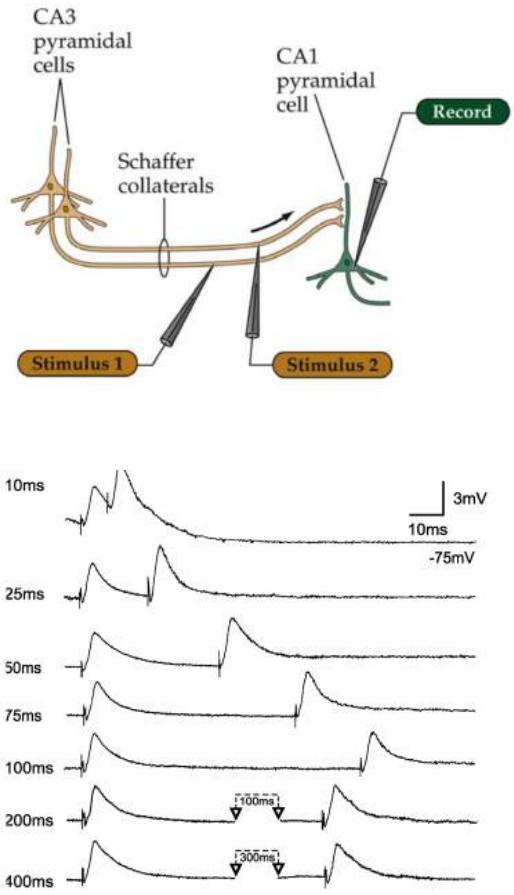


Figure 37: Heterostatic Plasticity

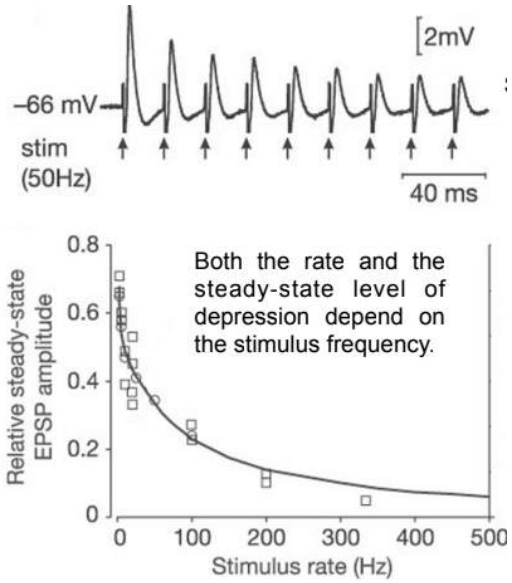


Figure 38: Heterostatic Plasticity

To test plasticity in the hippocampus the Ca3 to Ca1 pathway was modulated and the EPSP in the Ca1 was measured, this tells you the activity of the pathway. If the spiked generated overlap it leads to increased spiking strength as there is Residual Ca2+ in the cell. Short term facilitation. Short term depression at about 40ms time frame can be observed if the CA3 to CA1 pathway is stimulated at 50 hz it leads to a reduction in the EPSP which dependant on the frequency of activation.

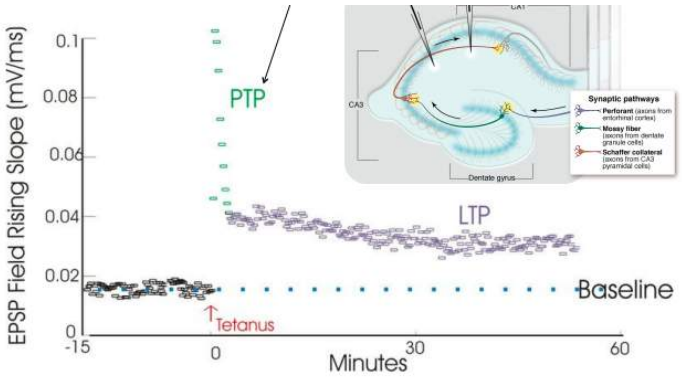


Figure 39: Heterostatic Plasticity

LTP is measure in the hippocampus. The CA3 pathway is given a fast stimulus of (range of 50 - 200) 100 hz known as tetanus. This then leads to a stronger post tetanic potentiation caused by the accumulation of Ca in the terminals as well as LTP in the long term. If the cells are stimulated at a lower time frequency 1-10 hz LTD will occur.

3.5.2 LTP and LTD induction in the Hippocampus

3.5.3 Molecular basis of synaptic plasticity

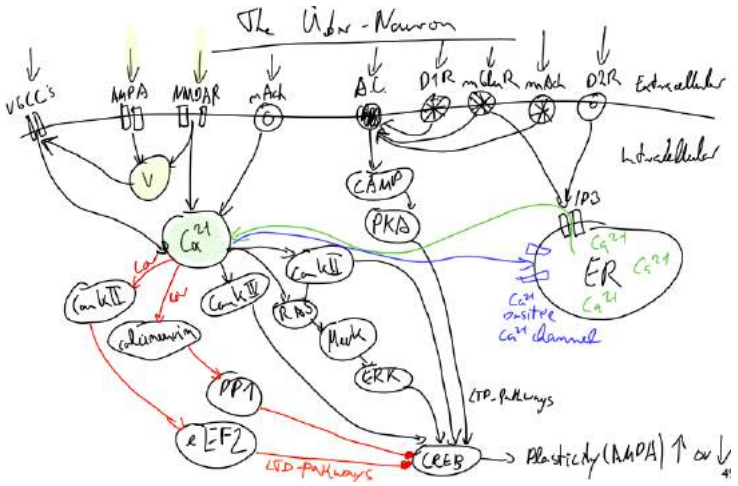


Figure 40

Intro Very simply LTP and LTD are dependent on CREB which controls the level of AMPA receptors in the cell. The level of AMPA receptors will determine how depolarised or hyperpolarised the cell becomes.

What controls LTP and LTD:

- Creb is controlled by many pathways that are dependant on Ca ions or directly by dopamine.
- Ca ion levels can increase as it enters into the cell from the external environment or released from internal stores.

How CA levels change: AMPA channel, when glutamate binds it causes depolarisation opening voltage gated Ca channels as well as NMDA channels that further depolarise the cells. Dopamine D2 when binds in leads to Ca2+ increase from the ER, this increased Ca leads.

How CA leads to CREB:

- Positive: High levels of Ca activated Camkinse 1 and 2 that leads to increased Creb and thus ampa receptors. Dopamine activated internal cell machinery that leads to increased phosphorylation (activation) of creb these both pathways are known as the LTP pathways.
- Negative: Low levels of Ca leads to Camkinse 2 and Calmodulin that reduces the phosphorylation (activation) of creb thus ampa receptors.

Summary Thus Ca is very important for LTP and LTD. Low frequency stimulation causes low Ca levels and high frequency leads to high levels.

3.6 Non-synaptic plasticity

Researchers have artificially raised the Neuronal excitability below threshold. It leads to a greater number of firings.

Researchers can modulate the axons with glutamate puffs and this will affect the action potential traveling along the axon.

Researchers can modulate dendritic excitability If the volume is smaller the epsp will summed up leading to ap, the synapse location will also modulate the excitability nearer the soma will be more excitable as there isn't a loss of charge.

3.6.1 Neuronal excitability and spike generation

3.6.2 Axonal modulation (shunting, frequency filtering)

3.6.3 Alterations of dendritic excitability

4 Learning Rules

Learning rules, as the name imply, describe methods of learning from information. Various machine learning methods that we discuss elsewhere already describe ways in which the data available to us can be used to create an objective (or cost, or loss) function which gives us something concrete to optimize so we have a model that performs well on similar data. These methods describe **global** cost functions because these expressions are in terms of high-level representations in the model, often only the final output layer representations. In a very small toy model, such as a fully-connected neural network with no hidden layers, this may provide useful information for adapting individual neuronal connections. Expanding this model to add complexities such as additional neurons hidden layers leaves us with an architecture that we can understand and can enumerate, as well as the global objective which we continue to aim for. However, we now have little understanding of how individual connections should be modified in the training process to contribute to improving the global objective defined by some global cost function which makes claims describing how output representations should change to improve the model but no inherent claims describing how the changes can be implemented.

To this end, the local learning rules we are about to discuss can alternatively be considered **local optimization principles**, as they are a small instance (typically involving only a few neurons) of our global optimization goal.

4.1 Error minimization rules

The first category of learning rules we will discuss are those that focus on optimizing with respect to some error function.

4.1.1 Perceptron learning rule

The perceptron learning rule was inspired by the model of neurons at the time, chiefly outlined in the McCulloch and Pitts Neuron. This model describes basic action potential propagation and involves multiple presynaptic neurons connected to the soma of a postsynaptic neuron. An action potential is triggered when sufficiently many presynaptic neurons (which may each contribute differently to the postsynaptic neuron based on their synaptic strengths) are activated such that the joint effects of their action potentials in the postsynaptic neuron exceeds the activation threshold, which triggers an action potential through the postsynaptic neuron.

Analogously, the perceptron learning rule involves multiple input values, which are each connected with varying weights to an output node. The value emitted by the output node depends on whether the weighted contributions of the input values exceeds a specific threshold. Learning is the process of adjusting the weights of each input as well as the output threshold to achieve the desired goal.

This can be denoted by a thresholded linear transformation. Given an input vector of values, the output falls into two cases depending on whether the linear transformation is above or below the threshold. Specifically, for an input vector

x, corresponding weights w, and a threshold b, the output can be denoted as:

$$\text{output} = \begin{cases} 0, & \text{if } w \cdot x + b \leq 0 \\ 1, & \text{if } w \cdot x + b > 0 \end{cases} \tag{26}$$

As a result, from a machine learning classification perspective, the perceptron learning rule describes a linear classifier as its decision is based on the result of a linear transformation. The standard algorithm (developed by Rosenblatt) for training according to this learning rule involves looping through every data sample, updating the weights w if and only if the current data sample x is misclassified, detailed below:

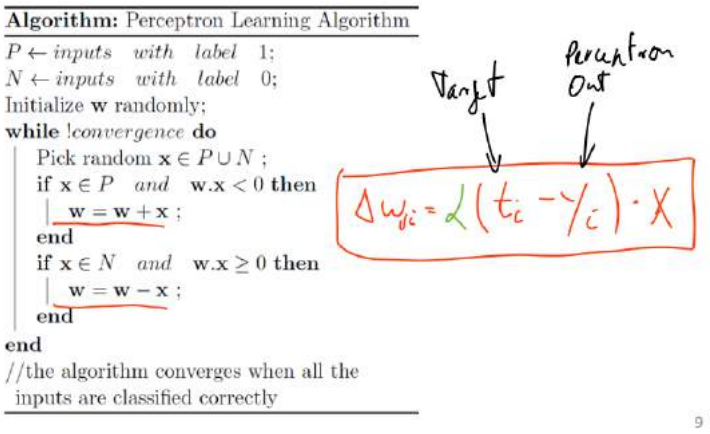


Figure 41: Algorithm for the perceptron learning rule

As shown in Figure 41, if x is a false positive, the new w is the sum of the current w and x. If x is a false negative, the new w is the difference between the current w and x. The algorithm concludes when all samples are classified correctly, i.e. w does not change after looping through all samples. If the points are not linearly separable (there exists no linear decision boundary that discriminates positive and negative samples) the algorithm will oscillate and the weights w will fail to converge.

4.1.2 ADALINE learning rule

This can be viewed as a slightly modified instance of the perceptron learning rule. In the perceptron rule, the thresholded result of the weighted sum of inputs is used for updating the weights in each iteration. In ADALINE, the weighted sum of inputs ($w \cdot x$ above) itself is used to update the weights in training.

4.1.3 DELTA learning rule

The weight updates from this equation aim to directly minimize a neuron's output error for a target value t_i and output value y_i, which can be formulated using gradient descent minimizing the squared error between these values.

This error can be formulated as:

$$E = \frac{1}{2}(t_i - y_i)^2$$

Finding the appropriate weight updates according to the gradient descent optimization method requires calculating the change in error with respect to each weight that we wish to update. This can be expressed as:

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial \frac{1}{2}(t_i - y_i)^2}{\partial w_{ji}}$$

Assuming a model structured similarly as in the perceptron and ADALINE learning rules with a single layer between inputs and the output value and using as the activation function $g(x) = \Theta(x)$ the heavyside function, this yields:

$$\nabla w_{ji} = \alpha(t_j - y_j)g'(h_j)x_i$$

where we used $h_j = x_i w_{ji}$ as inputs, $y_j = g(h_j)$ as outputs, t_j as target value, learning rate α and the derivative g' of the activation function g w.r.t..

The weight update equation for the DELTA learning rule clearly shares some similarities with that of the perceptron learning rule. Both weight update equations contain an error term calculated by the difference between the target and output values $(t_i - y_i)$ multiplied with the input x_i. However, the DELTA learning rule adds some complexities as it incorporates a learning rate α to adapt learning as well as the derivative of the activation function applied to the sum of the inputs $g'(h_j)$.

While the perceptron learning rule, particularly in light of its well-defined algorithm, defines the problem in terms of shifting hyperplanes to adapt a decision boundary, the DELTA learning rule optimizes the sum of squared error for a model with an activation function applied to a linear output. As previously mentioned in the contrast between the perceptron and ADALINE rules, the perceptron rule will either reach a stable zero-error solution (in the case of linearly separable data) or continually oscillate (otherwise). In contrast, the DELTA rule due in part to its adaptable learning rate can continually converge to a minimum error solution.

4.2 Biologically plausible rules

4.2.1 Hebbian learning

> Cells that fire together wire together.
>
> Not a direct quote by Hebb

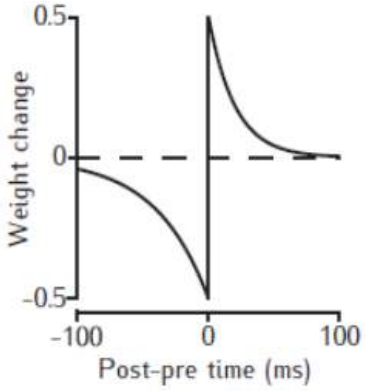


Figure 42: A schematic of theoretical pre- and postsynaptic spike timing correlation with weight updates: LTP (> 0 ms) or LTD (< 0 ms)

Spike-timing dependent plasticity is the Hebbian learning concept of weights between neuronal synapses changing over time based on the timing of their spikes. If a postsynaptic neuron fires at the same time or just after the presynaptic neuron fires, whether this is due to an action potential in the presynaptic neuron or other nearby neurons, this indicates that connections between these two neurons could be reinforced, which occurs by increasing the weights on these synaptic junctions to more efficiently propagate future action potentials. This is known as long-term potentiation. However, if a postsynaptic neuron fires just before a presynaptic neuron, the junction in the pre- to postsynaptic direction is possibly unnecessary or counterproductive so the weight of these synapses decrease over time. This is known as long-term depression.

We can start by writing a simple Hebbian weight update as the product of the input and output with some scaling factor:

$$w = \alpha \times x \times y \tag{27}$$
$$= \alpha \times x \times [w \cdot x] \tag{28}$$
$$\tag{29}$$

The more closely aligned w and x are, the larger $y = w^T x$ is, and by definition of the dot product $y = 0$ when w is orthogonal to x. This leads to the weight vector gradually pointing towards the input vector, or the cloud of input data in a dataset. We can mitigate this issue by applying a zero-mean transformation on our dataset to center the data around the origin, but this leads to a different problem of the weight vector tending to align with the direction of greatest variance.

Let's explore some other ways to express this Hebbian update rule:

$$\nabla w = \alpha \times x \times y \tag{30}$$
$$= \alpha(w \cdot x)x \tag{31}$$
$$= \alpha(x \cdot x^T) \cdot w \tag{32}$$

where the last step involves a transformation of the inner product into an outer product, which was skipped in lecture but is detailed in a dedicated lecture slide.

We note that $x \cdot x^T$ is the correlation matrix of the vector x, which we denote as C. *This is distinct from the covariance matrix.* We now arrive at:

$$\nabla w = \alpha(x \cdot x^T) \cdot w \tag{33}$$
$$= \alpha \times C \cdot w \tag{34}$$
which leads to the following update over time: (35)
$$\frac{\partial w}{\partial t} = C \cdot w \tag{36}$$

The lecture slides go into some more detail describing that applying the classical solution for this expression, $w(t) = u \times e^{\lambda \times t}$ for some vector u, as λ is positive the weight vectors will continue to increase and blow up.

4.2.2 Oja's rule

A modification of the Hebbian rule above in which a weight decay term is added. As this weight decay term is proportional to y^2, a quadratic result, it eventually limits the magnitude of the weights w to unit length while maintaining the tendency of the weights to point in the direction of maximum variance.

$$\nabla w_{oja} = \nabla w - y^2 \times w = \alpha \times x \times y - y^2 \times w \tag{37}$$
$$= \alpha \times y(x - y \times w) \tag{38}$$

4.2.3 Covariance rule

Another modification of the Hebbian rule above uses an idea similar to mean-centering of the data, but instead of transforming the data, the weights w are updated using mean-centered inputs x and outputs y.

Notation: $\langle \cdot \rangle$ denotes the mean of the variable over the input samples.

$$\nabla w_{cov} = \alpha\big[y - \langle y \rangle\big]\big[x - \langle x \rangle\big] \tag{39}$$
$$= \alpha\big[y \times x - \langle y \rangle \times x - y \times \langle x \rangle + \langle y \rangle \times \langle x \rangle\big] \tag{40}$$

$$\langle \nabla w_{cov} \rangle = \alpha\big[\langle y \times x \rangle - \langle \langle y \rangle \times x \rangle - \langle y \times \langle x \rangle \rangle + \langle \langle y \rangle \times \langle x \rangle \rangle\big] \tag{41}$$
$$= \alpha\big[\langle y \times x \rangle - \langle y \rangle \times \langle x \rangle - \langle y \rangle \times \langle x \rangle + \langle y \rangle \times \langle x \rangle\big] \tag{42}$$
$$= \alpha\big[\langle y \times x \rangle - \langle y \rangle \times \langle x \rangle\big] \tag{43}$$

The last line depicts the difference between the mean of the product of x and y, $\langle y \times x \rangle$ and the product of the means $\langle y \rangle \times \langle x \rangle$.

This rule solves a similar problem as Oja's rule, specifically the blowing up of weights over the training process. By subtracting the means when updating, weight updates can be negative as well as positive. The weights w increase when pre- and post-synaptic firing are positively correlated, and the change is proportional to the covariance of the firing rates.

4.2.4 Sanger's rule

The idea is that we use a single Hebbian neuron that points in the direction of maximum variance, as described previously, and view this as a principal component of the data. We subtract the contribution of this first principal component from the data, feeding the remaining data into a different neuron which subsequently identifies the direction of maximum variance in this data. This process can be repeated and resembles the addition of principal components in Principal Component Analysis (PCA).

4.2.5 Sejnowski's Infomax rule

The Infomax rule utilizes a nonlinear function and yields a method for implementing Independent Component Analysis (ICA).

4.2.6 Bienenstock-Cooper-Monroe rule

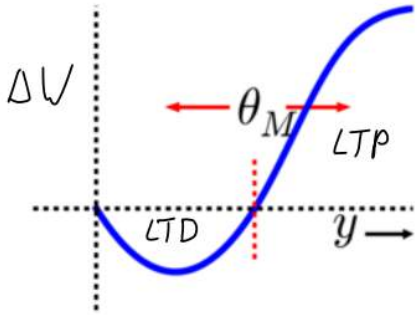


Figure 43: BCM rule

Activity is measured by y along the horizontal axis, and with no activity no change in weights takes place. Activity below a certain threshold θ_M triggers the LTD regime, and activity above this threshold triggers the LTP regime.

Measuring neuronal output across a certain window indicates that there is a biological basis for this. Above some threshold, the weight updates increased as the stimulation frequency was increased.

This was tested experimentally in the hippocampus and primary visual cortex, stimulating inputs to a neuron and measuring its spiking frequency. In this experiment, below a stimulation frequency of about 10 Hz the weight updates were negative, while above this stimulation frequency they were positive.

4.2.7 Triplet rule

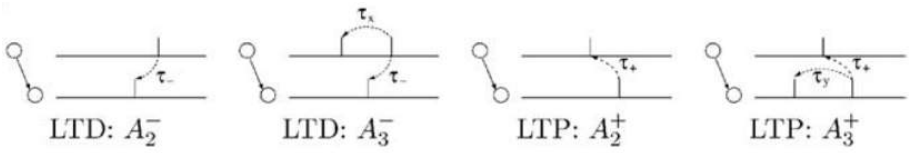


Figure 44: Examples of pre- (top) and postsynaptic (bottom) spike pairs and triplets leading to LTD or LTP

The triplet rule extends the classical Hebbian STDP idea. Instead of only looking at single presynaptic and postsynaptic spikes, multiple recent spikes within a specified time window are considered. As shown in Figure 44, LTD generally occurs when a presynaptic spike occurs just after a postsynaptic spike, even if (as in A_3^-) another presynaptic spike preceded the postsynaptic spike. Analogously, LTP tends to occur when a postsynaptic spike follows a presynaptic spike, even if (as in A_3^+) another postsynaptic spike preceded the presynaptic spike. In this fashion, the consideration of the third spikes within the time window can lead to different outcomes.

In both the triplet and BCM rules, above some threshold as the spiking frequency increases, the weight updates also increase (and are positive).

The fundamental differences between these two rules are unclear.

4.2.8 Calcium rule

As shown in Figure 45, this rule specifies that the LTP regime applies based on the amount of time above the calcium threshold while the LTD regime applies below the threshold. This rule results in the same basic STDP profile as before, and there are threshold parameters that can be changed to affect the dynamics.

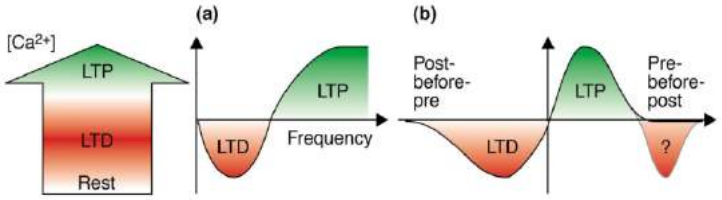


Figure 45: Calcium rule

4.2.9 Hebbian learning: Unsupervised

Error-driven learning appears to be much more necessary for deeper networks. A network was trained on the MNIST dataset (handwritten digits) using a basic Hebbian learning rule to cluster the data into separate digits and then learn a linear classifier on these digits.

4.2.10 Hebbian learning: Three-factor rules

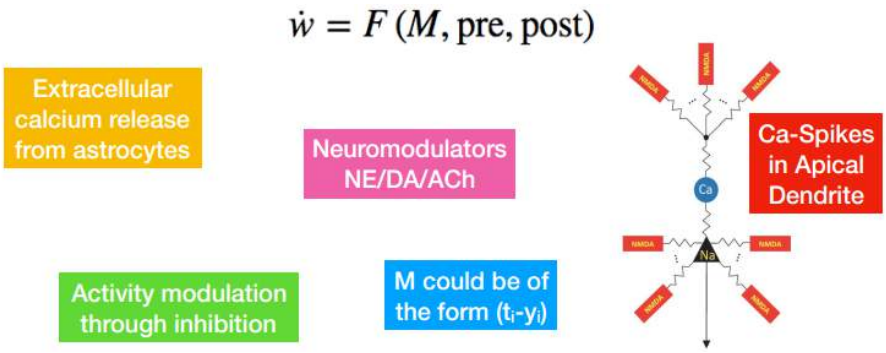


Figure 46: Possible third factors M in three-factor learning rules

Three-factor Hebbian learning rules integrate the pre- and postsynaptic firing with a third factor, M, which includes values such as the covariance-rule, TD learning, gated Hebbian learning, surprise-modulated STDP, etc.

For a biological neuron, this M factor may be viewed in a variety of ways, as shown in Figure 46.

It may be seen as a representation of error, including backpropagated error. For example, in the apical dendrites (level 5 neurons) receive feedback signals from the next hierarchical layer, and the strong calcium channels in these apical dendrites allow for error signals to trigger calcium spikes that propagate down the cell. The calcium spike is therefore a possible representation of the error from the next layer, which would model backpropagation.

Alternatively, neurons project to the next layer but some also project backwards to interneurons (and in turn, back to the apical dendritic layer), so the error signals reflect what is happening globally, in the next layer, and (through lateral inhibition, etc.) what is occurring in neighboring neurons. There is a motivation, as seen in the learning rules that are analogous to PCA, to inhibit neighboring neurons. In particular, this allows a neuron to potentially learn a useful unique representation instead of learning the same things as every other neuron.

Another possibility includes extracellular calcium release from astrocytes, as this affects the external calcium concentration but also internal concentrations in neurons, thereby indirectly affecting the plasticity of said neuron.

5 Learning As Bayesian Inference

5.1 Motivation

The main idea of this chapter is to study how to incorporate model uncertainty relying on Bayesian modelling. First, let us discuss briefly why model uncertainty is a great quantity to possess.

5.1.1 Why uncertainty matters?

On the biological side: we, humans, incorporate uncertainty in our decisions. We live in an environment where we need to act fast based on noisy, ambiguous and sparse sensory information. Therefore, decisions are usually made under substantial uncertainty. In fact a Bayesian brain hypothesis suggests that the brain follows a Bayesian probabilistic framework to make optimal decisions in presence of uncertainty [38].

For instance, our visual systems are subject to uncertainty due to noisy inputs [39]

Figure 47: Rubin's vase. Illustration of competing plausible explanations of the world. What should we see? a vase (on a black background) or two faces (on a white background)?

More formally, uncertainty is incorporated in every stage of neural computation because:

- Real sensors are noisy (e.g., low light vision)
- Real systems have finite computational resources
- Data is finite (e.g., we work with finite N in our learning problem)

Applications side: for building systems that make critical decisions, it becomes increasingly important to incorporate the model uncertainty information. For example, systems for automated diagnosis, autonomous driving or computers taking control over systems that can destabilize entire economic markets.[40]

[38] Knill DC, Pouget A (2004) The bayesian brain: the role of uncertainty in neural coding and computation. TRENDS in Neurosciences 27(12):712–719

[39] Uncertainty in visual processes predicts geometrical optical illusions

[40] Yarin Gal."Uncertainty in deep learning",PhD thesis, University of Cambridge, 2016.

5.2 Learning as an Optimization Problem

5.2.1 Recap: Learning from Data

Till now, we have talked about the standard view of learning. In this view, the recipe to learn from data is as follows:

> Consider a supervised learning problem, we are given a dataset of N training examples with their corresponding targets t $D = (x^{(n)}, t^{(n)})$ and a neural network model f_{NN}. Our goal is to find the
>
> 1. Define a global loss function, for instance a squared loss
>
> $$L(w) = \frac{1}{2} \sum_{n=1}^{N} \parallel t^{(n)} - f_{NN}(x^{(n)}, w) \parallel^2 \tag{44}$$
>
> 2. Systematically update the model parameters (i.e weights) to minimize L relying on the gradients of L. In DL, the common method is backprop.
>
> $$\Delta w = -\eta \bigtriangledown_w L(w) \tag{45}$$

Why does it work? ...

Credit assignment: "The concept of credit assignment refers to the problem of determining how much 'credit' or 'blame' a given neuron or synapse should get for a given outcome."[41] It can be formulated as an optimization problem. However, we must distinguish between pure (traditional) optimization and the optimization used for training of deep models.

5.2.2 How learning differs from pure optimization?

In ML scenarios, we try to optimize a performance metric indirectly via a loss function. In other words, initially, we care about a performance measure, P, with respect to the test set. However, since optimizing P directly is often intractable, we tend to reduce a different cost function $J(\theta)$ with the hope that this will improve P as well). In contrast, in traditional optimization problems $J(\theta)$ is a goal in itself.[42]

5.3 Neural Network Models of the brain

...

5.4 Interlude on Bayesian Modelling

Bayesian probability theory provides us with the tools to incorporate model uncertainty estimates in NN.
Following Bayesian approach, the idea is to start with an initial estimate of how the parameters are distributed and update our belief once data is observed.

[41] Richards, Lillicrap, Therien* et al. A deep learning framework for neuroscience, 2019.
[42] Chapter 8 Optimization for training Deep Models.Deep Learning

Note that: under this view the weights not single-valued (point estimates) but are random variables coming from a probability density function (pdf). In other words, weights are represented by probability distributions.

5.4.1 Model Definition

In Bayesian modelling, we need to define 2 components:

1. A **prior distribution** over the parameters space $p(w)$ which represents our prior belief on to which parameters are *likely* to have generated the data **before** we have seen the dataset.
2. A probabilistic model $p(t|x, w)$, **the likelihood distribution or function**, by which the inputs generate the output given some network parameter setting.

Once some data is observed, the weight distribution is updated to get the *posterior distribution* $p(w|D)$ of the weights given the data. Using Bayes rule, we can compute the posterior.

Bayes Rule:

$$\boxed{posterior = \frac{prior.likelihood}{evidence}} \tag{46}$$

Posterior distribution of w:

$$\boxed{p(w|D) = \frac{p(t|x, w).p(w)}{\int_\Omega p(t|x, w).p(w)dw}} \tag{47}$$

- Ω is the weight space
- The denominator in eq.47 is **the model evidence** also called **the normalizing factor**. It represents $p(t|x)$ which is computed by marginalizing the likelihood over w (integrating out over the entire parameter space). This integration is usually hard to carry out analytically for complex models. In such cases approximation is needed.

5.5 Recovering the optimization view of learning

In this section, we will try to connect between the two views to learning: the standard and probabilistic view. Both boil down to solving an optimization problem.

Assumptions

- In regression tasks, we assume a Gaussian model, i.e.

$$(t^{(n)}|x^{(n)}, w) \sim N(t^{(n)}; f_{NN}(x^{(n)}, w), \sigma^2 I) \tag{48}$$

  where $f_{NN}(x^{(n)}, w)$ and $\sigma^2 I$ are the mean and variance of the normal distribution. We can therefore write the likelihood function as the product of individual samples.

$$p(t^{(n)}|x^{(n)}, w) = \Pi_{n=1}^{N} N(t^{(n)}; f_{NN}(x^{(n)}, w), \sigma^2 I) \tag{49}$$

– Weights are independent and identically distributed (iid) and follow a normal distribution as well. Assume there are K weights (i.e dim

$$p(w) = \Pi_{k=1}^{K} N(w_k; 0, \sigma_w^2) \tag{50}$$

Interlude I: Point estimate for the posterior Using the fact that the likelihood and the priors are both normal distributions we can easily compute a single point estimate (i.e a single value that serves as a "best guess" for the weights matrix). This point estimate can be the mode of the distribution computed as

$$w^* = \arg\max_w p(\boldsymbol{w}|D) \tag{51}$$

That is the value that maximizes the posterior $p(\boldsymbol{w}|D)$, also called **maximum a posteriori estimate** of the weights. Using some analytical manipulations, we can show that[43]

$$\boldsymbol{w}^{\text{MAP}} = \arg\max_{\boldsymbol{w}} \log P(\mathcal{D}|\boldsymbol{w}) + \log P(\boldsymbol{w}) \tag{52}$$

... However, as we will see later, using the mode of the posterior doesn't help much. In fact, summarizing the posterior with a single point estimate again leads to the same problem we are trying to avoid, which is a single "best" value for the weights, i.e a single line separating the data (in the context of stochastic binary classification).

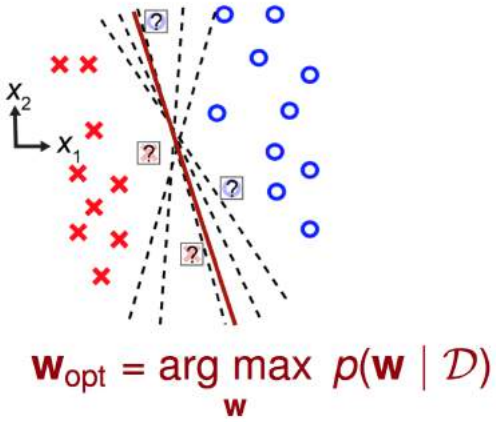


Figure 48: Maximum a posteriori estimate of the weights yields single line classifying the data

Interlude II: Bayesian Predictions In order to make predictions on new unseen data, we would have to predict $t^{(N+1)}$ for the new data $\boldsymbol{x}^{(N+1)}$.

[43] Blundell et al., *Weight Uncertainty in Neural Networks*, 2015

In the Bayesian approach, this can be done using **model averaging**. The idea here is that instead of relying on a single model to make the prediction, we average the predictions from all possible models or *model ensemble*. Therefore, the probability that the new $\boldsymbol{x}^{(N+1)}$ belongs to label $t^{(N+1)} = 1$ is given by

$$\begin{aligned} P(t^{(N+1)} = 1|\boldsymbol{x}^{(N+1)}, D) &= \mathbb{E}_{w\sim p(w|D)}[y(\boldsymbol{x}^{(N+1)}, w)] \\ &= \int_{\Omega} y(\boldsymbol{x}^{(N+1)}, w)p(\boldsymbol{w}|D)dw \end{aligned} \tag{53}$$

Once again, we need the full posterior over the weights to make the predictions. But we might ask **why not use the mode of the posterior to make the predictions?** Another way of phrasing the question: **why not use the "plug-in" solution where we collapse $p(\boldsymbol{w}|D)$ into its mode $\delta(\boldsymbol{w} - \boldsymbol{w}_{opt})$ (the one computed on *Interlude I*?**

- By construction, using the mode we ignore any uncertainty in parameter choice. We only consider the optimal case

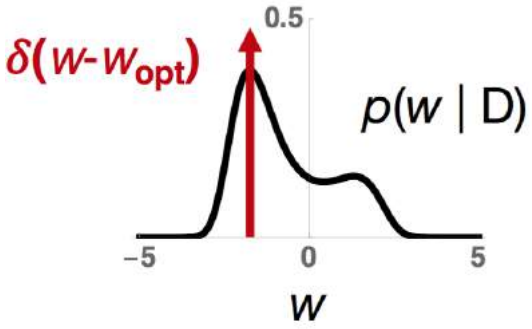


Figure 49: Posterior mode is the optimal case scenario for $p(\boldsymbol{w}|D)$

- Therefore, we are bound to make overconfident predictions
- We cannot compare different models

Now that we are convinced that we need the full posterior, we are faced with a problem.

Problem: As mentioned earlier the denominator in eq. 47 is often intractable.

$$p(\boldsymbol{w}|D) = \frac{p(t|\boldsymbol{x}, \boldsymbol{w}).p(\boldsymbol{w})}{\int_{\Omega} p(t|\boldsymbol{x}, \boldsymbol{w}).p(\boldsymbol{w})d\boldsymbol{w}}$$

Solution: Approximate $p(\boldsymbol{w}|D)$ by another, simpler distribution denoted $q(\boldsymbol{w})$ from which we can easily sample (this would become clearer later)

Discuss: Are we still "Bayesian optimal" if we follow this strategy?

...

5.5.1 Approximating the posterior $p(w|D)$

Goal: Find distribution $q(\boldsymbol{w})$ that approximates (i.e. is as similar as possible) $p(\boldsymbol{w}|D)$. For that, we need to define a measure of distance.

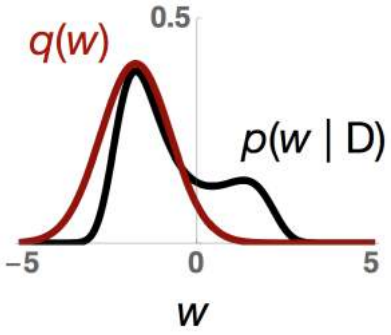


Figure 50: $q(\boldsymbol{w})$ is a surrogate distribution *similar* to $p(\boldsymbol{w}|D)$

Interim: Kullback-Leibler (KL) Divergence: The Kullback-Leibler divergence between two densities q and p is defined as

$$D_{KL}(q(\boldsymbol{w})||p(\boldsymbol{w}) = \int_{\Omega} q(\boldsymbol{w}).\log\frac{q(\boldsymbol{w})}{p(\boldsymbol{w}}d\boldsymbol{w} \tag{54}$$

Properties of KL divergence

- D_{KL} is not a true measure of distance since in general $D_{KL}(q(\boldsymbol{w})||p(\boldsymbol{w}) \neq D_{KL}(p(\boldsymbol{w})||q(\boldsymbol{w})$. It is **not symmetric** in its arguments.
- D_{KL} **does not obey** the triangle inequality
- $D_{KL}(p(\boldsymbol{w})||q(\boldsymbol{w}) \geq 0$ (**Gibb's inequality**)
- $D_{KL}(p(\boldsymbol{w})||q(\boldsymbol{w}) = 0$ **if and only if** $q(\boldsymbol{w}) = p(\boldsymbol{w})$ for almost all $\boldsymbol{w}$ (i.e. the two distributions coincide)

5.5.2 Learning as a Variational Inference (VI)

Consider a simpler weight distribution $q(\boldsymbol{w})$ parametrized by θ. The KL divergence **"from"** the true posterior **"to"** q can be written as:

$$D_{KL}(q(\boldsymbol{w};\theta)||p(\boldsymbol{w}|D) = \int_{\Omega} q(\boldsymbol{w};\theta).log\frac{q(\boldsymbol{w};\theta)}{p(\boldsymbol{w}|D}d\boldsymbol{w} \tag{55}$$

Variational learning finds the parameters θ of a distribution on the weights $q(\boldsymbol{w};\theta)$ that **minimises** the KL divergence. However note that we are minimizing over the parameter θ not the weights

$$\boxed{\theta_{opt} = \arg\min_{\theta} D_{KL}(q(\boldsymbol{w};\theta)||p(\boldsymbol{w}|D)} \tag{56}$$

Using the definition of D_{KL} in eq. 56 and considering that the posterior $p(\boldsymbol{w}|D) \propto p(\boldsymbol{w}).p(D|\boldsymbol{w})$, we can write: continue derivation from the exercise

$$\begin{aligned}
\theta_{opt} &= \arg\min_{\theta} \int q(\boldsymbol{w};\theta).log\frac{q(\boldsymbol{w};\theta)}{p(\boldsymbol{w}|D)}d\boldsymbol{w} \\
&= \arg\min_{\theta} \int q(\boldsymbol{w};\theta).\log\frac{q(\boldsymbol{w};\theta)}{p(\boldsymbol{w}).p(D|\boldsymbol{w})}d\boldsymbol{w} \\
&= \arg\min_{\theta} \int q(\boldsymbol{w};\theta).\log\frac{q(\boldsymbol{w};\theta)}{p(\boldsymbol{w})}d\boldsymbol{w} - \int q(\boldsymbol{w};\theta).\log p(D|\boldsymbol{w})d\boldsymbol{w} \\
&= \arg\min_{\theta} \mathbb{E}_{\boldsymbol{w}\sim q(\boldsymbol{w};\theta)}[-\log p(D|\boldsymbol{w})] + D_{KL}(q(\boldsymbol{w};\theta)||p(\boldsymbol{w})) \\
&= \arg\min_{\theta} F(\theta, D)
\end{aligned} \tag{57}$$

The resulting cost function is known as the **variational free energy** or **the expected lower bound** [44].
It is composed of two terms:

1. Data fitting term: the expected loss over q
2. Prior matching term: $D_{KL}(q(\boldsymbol{w};\theta)||p(\boldsymbol{w}))$. From an information theory viewpoint, this term defines how many nats needed to encode model explain

Since we choose $q(\boldsymbol{w})$ to be easy to sample from, we can therefore approximate the true mean of the distribution using the sample average.

$$\mathbb{E}_{\boldsymbol{w}\sim q}[f(\boldsymbol{w}] \approx \frac{1}{R}\sum_{r=1}^{r=R} f(\boldsymbol{w}^{(r)}); \tag{58}$$

where R is the number of samples and $\boldsymbol{w}^{(r)} \sim q(\boldsymbol{w})$

The optimization of eq.57 can be carried out using the gradient. In this case, we compute the derivative

$$\frac{\partial}{\partial\theta}\mathbb{E}_{\boldsymbol{w}\sim q}[f(\boldsymbol{w}] = \frac{\partial}{\partial\theta}\int q(\boldsymbol{w};\theta)f(\boldsymbol{w})d\boldsymbol{w} \tag{59}$$

Note that it would be easier if we swap the derivative and the expectation.

5.5.3 Bayes by Backprop (BbB)

Quick Recap:

- Bayesian inference for neural networks calculates the posterior distribution of the weights given the training data, $p(\boldsymbol{w}|D)$.
- Remember that all weights in our neural networks are represented by probability distributions over possible values, rather than having a single fixed value as is the norm (see Figure.51).

[44] Blundell, Cornebise, et al. Weight Uncertainty in Neural Networks. 2015.

- This distribution answers predictive queries about unseen data by taking expectations.
- However, the posterior is untractable and we approximate is using another parametric distribution $q(\boldsymbol{w})$
- Our goal is to find θ such that KL-divergence from $p(\boldsymbol{w}|D)$ to $q(\boldsymbol{w})$ is minimized.
- The optimization function $F(\theta, D)$ is composed of two terms: data-independent term, the prior matching and a data-fitting term, which is an expectation over q
- Now we are trying to use gradient based technique to solve this optimization problem and would love to swap the expectation and the derivative

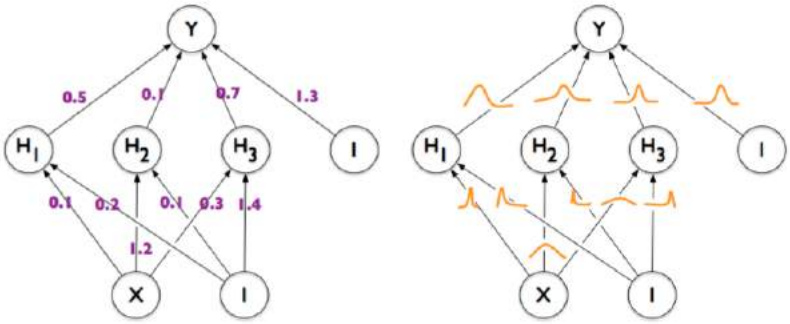


Figure 51: **Left**: each weight has a fixed value, as provided by classical backpropagation. **Right**: each weight is assigned a distribution, as provided by Bayes by Backprop.

Under certain conditions, the expectation of the derivative can be expressed as the derivative of the expectation.

Proposition: Suppose the weights are independently and modelled by a Gaussian random variables parametrized by the mean, μ_k and the variance σ_k^2, i.e.

$$\begin{aligned} w_k &\sim N(\mu_k, \sigma_k^2) \\ \theta_k &= (\mu_k, \sigma_k^2) \end{aligned} \tag{60}$$

Now let $w_k = \mu_k + \sigma_k \epsilon$; where $\epsilon \sim N(0, 1)$. Using this reparametrization trick (by introducing ϵ), $\boldsymbol{w}$ is no longer the "source of randomness"; ϵ now is. Therefore, we can propagate the derivative inside the expectation (ϵ is the random variable)[45].

[45] https://stats.stackexchange.com/questions/199605/how-does-the-reparameterization-trick-for-vaes-work-and-wh

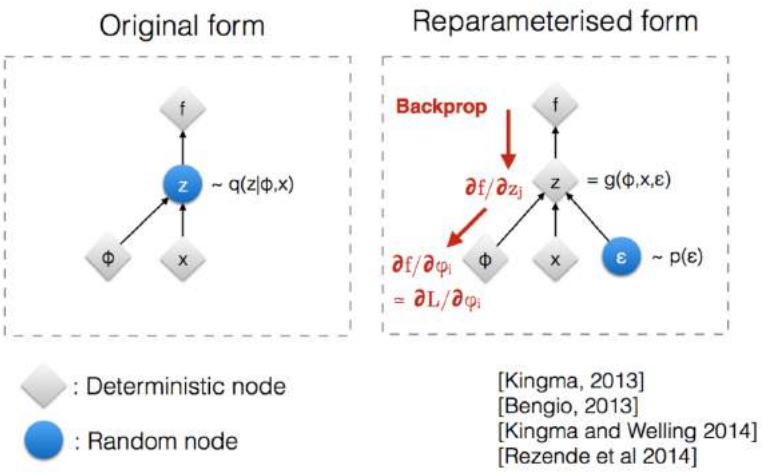


Figure 52: Reparametrization trick intuition. Extracted from..

$$\begin{aligned}\frac{\partial}{\partial\theta}\mathbb{E}_{\boldsymbol{w}\sim q}[f(\boldsymbol{w})] &= \frac{\partial}{\partial\theta}\mathbb{E}_{\epsilon}[f(\mu_k+\sigma_k\epsilon)] \\ &= \frac{\partial}{\partial\theta}\mathbb{E}_{\epsilon}[f(\boldsymbol{w}(\theta))] \\ &= \mathbb{E}_{\epsilon}[\frac{\partial}{\partial\theta}(f(\boldsymbol{w}(\theta))]\end{aligned} \tag{61}$$

- If the prior p(w) is also Gaussian, the KL divergence has as well a simple closed-form expression.see exercise
- Setting $f(\boldsymbol{w})$ in eq.61 to our loss function $-\log p(D|\boldsymbol{w})$ obtained in eq.57, we can use backprop to compute the derivatives we need (**Bayes by Backprop**).

Learning algorithm: Putting all the pieces together

1. Sample R weights $\boldsymbol{w}^{(r)}$ from the distribution $q(\boldsymbol{w})$
2. Compute sample-based loss function

$$\begin{aligned} F(\theta, D) &\approx \frac{1}{R}\sum_{r=1}^{R} \log(q(\boldsymbol{w})) - \log p(D|\boldsymbol{w}^{(r)}.\log p(\boldsymbol{w}^{(r)}) \\ &= \frac{1}{R}\sum_{r=1}^{R} \log(q(\boldsymbol{w})) - \log p(D|\boldsymbol{w}^{(r)} - \log p(\boldsymbol{w}^{(r)}) \end{aligned} \tag{62}$$

3. Compute derivatives of loss with respect to weight mean and variance, the variational posterior parameters $\theta = \left\{\mu_k, \sigma_k^2\right\}_{k=1}^{K}$ of q (exercise)
4. Update synaptic mean and variance with learning rate η

$$\begin{aligned} \mu &\leftarrow \mu - \eta\Delta\mu \\ \sigma^2 &\leftarrow \sigma^2 - \eta\Delta\sigma^2 \end{aligned} \tag{63}$$

Making Bayesian Predictions

Since it is easy to sample from q(w), we can get generate both a "point prediction" and an "error bar" using statistics from the samples, i.e;

- **Point prediction**: mean of the posterior predictive distribution

$$\bar{\boldsymbol{t}} = \mathbb{E}_{\boldsymbol{t}\sim p(\boldsymbol{t}|\boldsymbol{x})}[\boldsymbol{t}] \approx \frac{1}{R}\sum_{r=1}^{R} f_{NN}(\boldsymbol{x}, \boldsymbol{w}^{(r)}) \tag{64}$$

- **Error bar**: standard deviation of the posterior predictive distribution

$$\begin{aligned} \sigma_t^2 &= Var[\boldsymbol{t}] \approx \sigma_T^2 + \frac{1}{R}\sum_{r=1}^{R} {\boldsymbol{y}^{(r)}}^T \boldsymbol{y}^{(r)} - \bar{\boldsymbol{t}}^T\bar{\boldsymbol{t}} \\ \boldsymbol{y}^{(r)} &= f_{NN}(\boldsymbol{x}, \boldsymbol{w}^{(r)}) \end{aligned} \tag{65}$$

5.6 What do we get out of Bayes by Backprop?

1. **Prediction uncertainty** (in contract to a single curve for a vanilla neural network!)

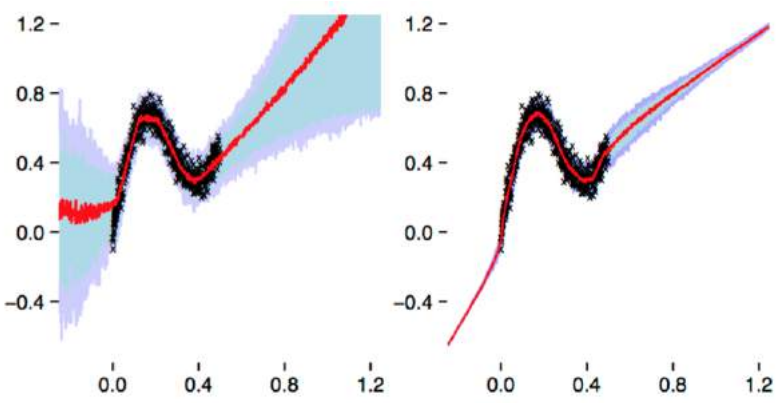


Figure 53: Regression of noisy data with interquatile ranges. Black crosses are training samples. Red lines are median predictions. Blue/purple region is interquartile range. **Left**: Bayes by Back- prop neural network, **Right**: standard neural network. Blundell et al., 2015

2. **Better generalization**: Algorithm shows good results on MNIST dataset; its learning curve is comparable to that of dropout ("although each iteration of Bayes by Backprop is more expensive than dropout – around two times slower" (Blundell et al., 2015)). One of the advantages of this approach is that a good prior can be considered as additional data.

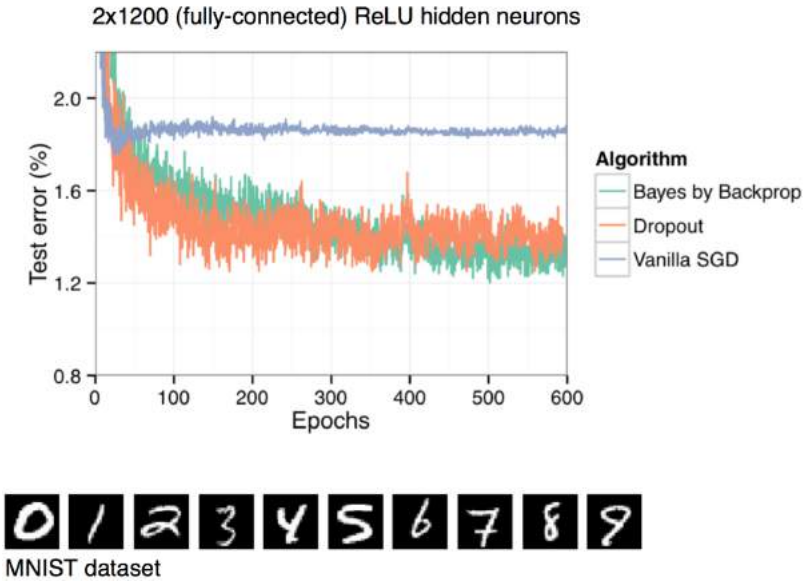


Figure 54: Test error on MNIST as training progresses.

3. **Outlier detection** which is crucial for high-risk systems

4. **Continual learning**
   Kirkpatrick et al., 2017, Nguyen et al., 2018

5.7 Drawbacks and Weaknesses of BbB

Discuss

- BbB is computationally expensive
- we assume independence (product) → may be a crucial stupid assumption
- Bayes central limit theorem

5.8 BbB and Neuroscience

Now the question is how to interpret or think about BbB from a neuroscience perspective?
One way to think about Bayes by Backprop is to see it as a model of inference with **stochastic synapses**.

BbB Algorithm	**How to think about it in the brain?**
Independent weight sampling	Noisy synaptic transmission. Every time an input is propagated, the mean weight μ is corrupted by additive Gaussian noise.
Learning changes both the mean synaptic weight μ and the level of synaptic variability σ^2	Can it be considered as "higher order" form of synaptic plasticity?
Independent synapses assumption in q(w)	Extra terms in the learning rules are local to synapses

5.8.1 Synaptic transmission in the brain

Some key facts to note [46]

- Synaptic transmission is inherently stochastic, meaning that a pre-synaptic action potential may and may not trigger neurotransmitter release.
- There is evidence that plasticity changes the properties (modifies the mean and variance of synaptic response) and number of postsynaptic receptors, as well as the presynaptic "machinery" responsible for stochastic neurotransmitter release.
- Probabilistic nature of synaptic transmission has been described as a binomial process parametrised by
  1. Number of synaptic release sites N
  2. Presynaptic release probability P_{rel}

[46] Llera1, Sacramento and Costa. Computational roles of plastic probabilistic synapses. 2019

3. Quantal amplitude q (proportional to the number of postsynaptic receptors)

This allows us to define the statistics of synaptic responses with

$$\begin{aligned} mean &= NqP_{rel} \\ Var &= Nq^2P_{rel}(1-P_{rel}) \end{aligned} \tag{66}$$

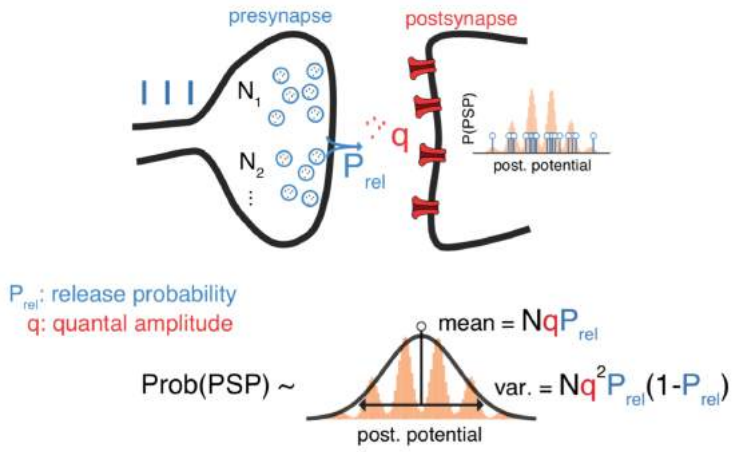


Figure 55: "When a presynaptic spike (blue vertical line on the left) occurs a presynaptic vesicle (blue circles) may release neurotransmitters (red dots) that bind to postsynaptic receptors (red) which elicits a postsynaptic potential (PSP; PSPs of different amplitudes are represented by the small vertical blue lines)" (extracted Llera,2019)

- Currently under study: what is the locus of plasticity?
  - It is currently accepted that both pre- and postsynaptic physiology can be modified during long-term potentiation (LTP) and depression (LTD)

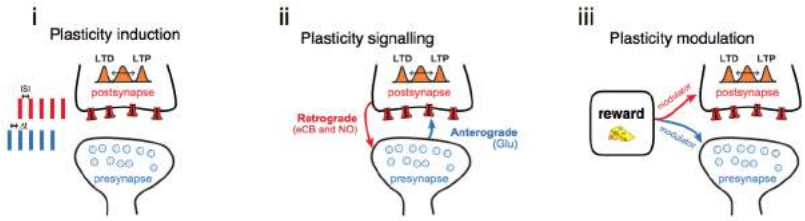


Figure 56: Long-term plasticity of probabilistic synapses. (i) Different induction protocols have been shown to trigger changes in the probability of postsynaptic responses. Schematic on the left represents pre- and postsynaptic spikes in a spike- timing-dependent plasticity protocol, which depending on the timing between pre- and postsynaptic spikes (δt) as well as the inter-spike interval (ISI) may lead to long-term potentiation (LTP) or depression (LTD). This in turn changes not only the mean synaptic response, but also its variance. (ii) Modifications to probabilistic synapses during plasticity are known to rely on specific retrograde (e.g. endocannabinoids (eCB) and nitric oxide (NO)) and anterograde signals (glutamate (Glu)). (iii) Behavioural outcomes (e.g. reward) may rely on neuromodulation (e.g. Dopamine) to regulate plasticity at probabilistic synapses. (extracted from Llera,2019)

5.8.2 Interlude on behavioral studies

Goal: This study aim is to show how the human brain makes predictions in presence of parameter uncertainty. The question the authors ask is: Do humans take advantage of Bayesian regression?

Experimental Protocol:

- Participants were asked to extrapolate a parabola from 4 noisy points. They had to adjust the location of a 5^{th} point to match the curve.
- Quadratic parameters of the parabola were generated from a prior distribution.
- After each trial, participants were shown the true parabola generating the dots as feedback. The idea here was to help them learn both the prior and the generative model.

Results:

- Out of 4 different regression models, Bayesian regression (BR) model which takes full parameter uncertainty into account) best explains human behavior
- For large noise levels BR approaches null model (prior-based regression which replaces the posterior with the prior, i.e., it does not use the likelihood.)

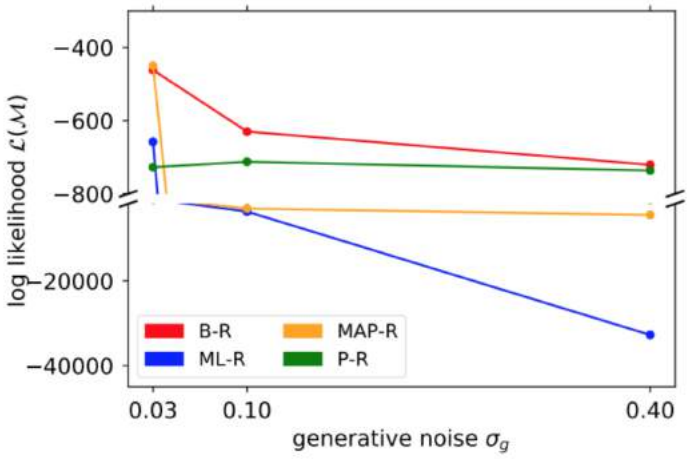


Figure 57: Log likelihoods that the models reproduce the participant responses, i.e., the y-position adjusted by the participant averaged across participants. The higher the value, the better the model performance.
B-R: Bayesian Regression, ML-R: Maximum likelihood regression, MAP-R:maximum a posteriori model, P-R: prior-based regression.

6 Prediction Errors During Perception And Learning

6.1 Motivation

There are many theories and principles about the brain such as free-energy principle, predictive coding theory and theories about error back-propagation in the brain. These different approaches attempt to explain the structure and function of the brain from a probabilistic inference perspective focusing on error propagation.

In this chapter, we, as well, consider prediction and perceptual categorization as an inference problem that is solved by the brain[47]. Under our view, in this chapter, the brain models the world as a hierarchy of dynamic systems. One key difference between this view and what we have learnt in the previous chapter on variational inference lies in the distinction between discriminative and generative models[48].

6.1.1 Discriminative Vs Generative Models

Discriminative models In Chapter 5, we were concerned with regression/-classification tasks where we were interested in obtaining the conditional distribution $p(t|\boldsymbol{x}, \boldsymbol{w})$. Models falling into this category are well-suited for supervised-learning problems. There is a teacher providing the targets which helps us learn the mapping from inputs to targets.
But what if, we don't have the targets, i.e no labelled data is at our disposal? Or what if we don't have a clear view of what we want to discriminate? Then comes the importance of generative models.

Generative models This type of modelling refers to building a **model of the data**, $p(\boldsymbol{x})$ from which we can sample. But what are our practical reasons for doing so?

- **Data efficiency**: explain the world using less data points.
- Better **understanding**: one key feature of generative models is interpretability. Generative models assume, as we will see, that the data is generated from some low-dimensional latent factors or variables to which we can assign meaning. In other words, we can associate each latent variable to a cause, something that would allow us to interpret a phenomenon.
- **Model checking** by sampling: Regression and classification models are usually complex. It is often hard to check what these models have captured from the data and what did they miss. One advantage of generative modelling is that it allows us to perform a sanity-check. Since we know the probability distribution generating the data, we can easily sample from it and compare the sampled data to the real data to see if anything is missing. This goes back to famous quote by Richard Feynman, 1988: "What I cannot create, I do not understand."

[47] Friston & Kiebal. Predictive coding under the free-energy principle. *Philosophical Transactions of The Royal Society B Biological Sciences*· June 2009.
[48] https://www.cs.toronto.edu/ duvenaud/courses/csc2541/index.html

– **Compression**: in generative modelling, we seek to explain the data (ex. some sensory input $\boldsymbol{x}$) using with a low-dimensional latent variables ($\boldsymbol{h}$). We have therefore compressed the information (from complex, high-dimensional data $\rightarrow$ low-dimensional, simpler description), which is at the core of Chaitin's 2007 "Comprehension is compression". A model is better is it explains more with less.

– Online classification: novel content?

6.1.2 Recap Bayesian inference

In the previous lecture, we formulated learning as a Bayesian inference problem. We started by defining a model using a prior on the weights (modelled as random variables following a probability distribution) and a likelihood function (which captures how likely is it that the data we see is generated under our model), then tried to update, using Bayes' rule, the weight distribution after seeing the data, i.e:

Recipe for Bayesian inference

1. **Define the model**
   - $p(\boldsymbol{w})$: could be a pre-trained model or any other prior on the weights.
   - $p(\boldsymbol{t}|\boldsymbol{x}, \boldsymbol{w})$ referred to as **forward probability**. Note that for inference we assume $\boldsymbol{w}$ fixed and we infer $\boldsymbol{t}$ from $\boldsymbol{x}$. In the machine learning view, this is a "noisy" neural network.
     $f_{NN}(\boldsymbol{x}, \boldsymbol{w}) + noise$

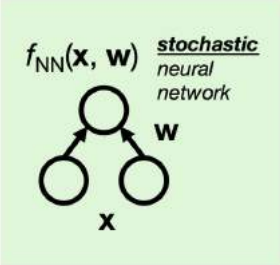


Figure 58: Computing the forward probability using a stochastic neural network.

2. **Learn/ Update model**: using Bayes' rule; where $D = \left\{\boldsymbol{x}^{(n)}, \boldsymbol{t}^{(n)}\right\}_{n=1}^{N}$

$$p(\boldsymbol{w}|D) = \frac{p(\boldsymbol{t}|\boldsymbol{x}, \boldsymbol{w}).p(\boldsymbol{w})}{\int_{\Omega} p(\boldsymbol{t}|\boldsymbol{x}, \boldsymbol{w}).p(\boldsymbol{w})d\boldsymbol{w}} \tag{67}$$

referred to as **inverse probability** because we are inverting the model.

Some comments before diving deeper

- In the previous chapter, we relied on the target $\boldsymbol{t}$. But we might ask, does the brain **always** rely on a teacher signal/ target?
- In this chapter we are building a model of the data. Such general knowledge of the world can help us know **"what to expect"**, which is important for decision-making agencies. Furthermore, a general knowledge can allow us to build our own labels on the fly; we can hence start to classify things on the fly. more details...
- In Chapter 5, we had to worry about the marginal probability $p(\boldsymbol{x})$ because we wanted apply Bayes' rule. In this chapter, **we are not fully Bayesian** and will use a frequentist approach as a shortcut. (in inference MAP)
- Why are we particularly concerned with perception? Simply because it is beleived that the brain learns a generative model for its stream of sensory data. Put differently, our brain learns a model of how sensory data are generated which can be thought of as "a particular type of probabilistic model that captures cause and effect"[49]. Perception is then equated to inverting the generic model, reasoning about causes after observing the effects.

6.2 Hierarchical Gaussian Models (HGM)

We focus on a particular class of generative models; we consider models that can be structured as a hierarchy of latent variables, $\boldsymbol{h}$ comprising of N layers:

$$\boxed{p(\boldsymbol{x}, \boldsymbol{h_1} \ldots, \boldsymbol{h_N}) = p(\boldsymbol{x}|\boldsymbol{h_1}).p(\boldsymbol{h_1}|\boldsymbol{h_2}) \ldots .p(\boldsymbol{h_{N-1}}|\boldsymbol{h_N}).p(\boldsymbol{h_N})} \tag{68}$$

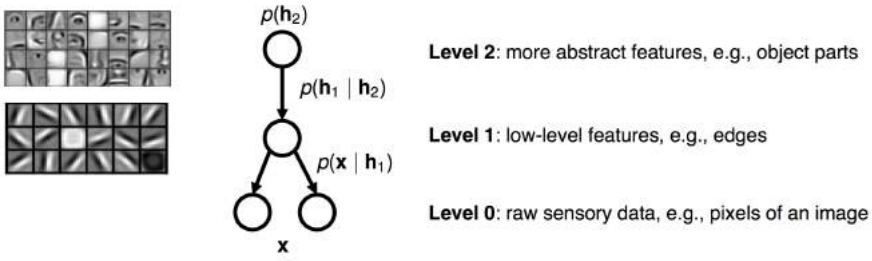


Figure 59: Example of generative model with a hierarchical structure comprised of 3 layers.

6.2.1 Perception as probabilistic inference

As explained earlier, perception can be seen as finding causes given that we have observed the effects.elaborate using an example from blog post...

[49]Nice blog post: http://boxandarrowbrain.com/2018/02/06/generative-models-in-perception/

In such context and following the same approach from Chapter 5, we perform inference on the latent variables instead of the targets $\boldsymbol{t}$ using Bayes' rule to obtain a distribution over our latent variable, given an observed sensory input $\boldsymbol{x}$.

1. **Define the model over $\boldsymbol{h}$**
   - $p(\boldsymbol{h})$
   - $p(\boldsymbol{x}|\boldsymbol{h})$ referred to as **forward probability**. Note that for inference we assume $\boldsymbol{h}$ fixed and we infer $\boldsymbol{x}$ from $\boldsymbol{h}$.
2. **Perceive** = invert the model

$$p(\boldsymbol{h}|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|\boldsymbol{h}).p(\boldsymbol{h})}{\int_H p(\boldsymbol{x}|\boldsymbol{h}).p(\boldsymbol{h})d\boldsymbol{h}} \tag{69}$$

   referred to as **inverse probability**.

6.2.2 Model Assumptions

- We assume each conditional probability distribution $p(\boldsymbol{h}_i|\boldsymbol{h}_{i+1})$ is Gaussian with mean dependent on the higher order variable $\boldsymbol{h_{i+1}}$ and covariance matrix Σ_i. Using the convention that the first layer later variable corresponds to the input, i.e. $\boldsymbol{h_0} = \boldsymbol{x}$:

$$p(\boldsymbol{h_i}|\boldsymbol{h_{i+1}}) \sim N(\boldsymbol{h_i}; \mu_i(\boldsymbol{h_{i+1}}), \Sigma_i) \tag{70}$$

- Assume a Gaussian top-level prior, **layer N** (and will briefly discuss a Laplacian prior)

$$p(\boldsymbol{h_N}) \sim N(\boldsymbol{h_N}; \mu_N, \Sigma_N) \tag{71}$$

- Note that the joint distribution $p(\boldsymbol{x}, \boldsymbol{h_1} \ldots, \boldsymbol{h_N})$ is in general not Gaussian for this model (only true for the special case of linear $\boldsymbol{\mu_i}$)
- We link two consecutive layers and introduce non-linearity, ϕ, with the following parameterization of the conditional mean:

$$\mu_i(\boldsymbol{h_{i+1}}) = \boldsymbol{w_i}.\phi(\boldsymbol{h_{i+1}}) + \boldsymbol{b_i} \tag{72}$$

  where $\boldsymbol{b_i}$ is the vector of biases
- From the previous assumptions, we can denote our model's parameters as $\boldsymbol{\Theta}$:

$$\begin{aligned} \Theta &= \{\boldsymbol{W_i}, \boldsymbol{b_i}, \Sigma_i\} \\ &= \{\mu_N, \Sigma_N\} \end{aligned} \tag{73}$$

- **Note**: we will include the bias vector as an additional weight column and set all covariance matrices Σ_i to identity matrix I:

$$I = diag(1, 1 \ldots, 1) \tag{74}$$

6.2.3 Inference under the Hierarchical Generative Model

Inference is about inversion of the internal states of the models.

Note that to perform inference we assume the model parameters Θ to be fixed.

In order to invert the internal states, we need to:

1. Find $p(\boldsymbol{h}|\boldsymbol{x};\Theta)$ using Bayes' rule in eq. 69.
2. Get a point estimate for the latent variable $\boldsymbol{h}$ using **a maximum a posteriori (MAP)** criterion. In other words, we try to find an optimal $\boldsymbol{h}$ that maximizes the posterior distribution $p(\boldsymbol{h}|\boldsymbol{x};\Theta)$ (this explains why we are not fully Bayesian).

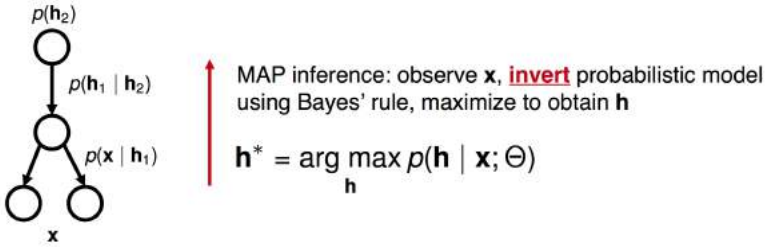


Figure 60: Inference in the hierarchical generative model

Inference as an optimization problem As we use MAP criterion to find an optimal $\boldsymbol{h}$, inference has again boiled down to solving an optimization (specifically a maximization) problem. Since the posterior probability $p(\boldsymbol{h}|\boldsymbol{x})$ is equal to the joint density $p(\boldsymbol{h},\boldsymbol{x})$ divided by $p(\boldsymbol{x})$, and we are optimizing with respect to $\boldsymbol{h}$, we can drop the denominator as follows:

$$\begin{aligned}\boldsymbol{h}^* &= \arg\max_h p(\boldsymbol{h}|\boldsymbol{x};\Theta) \\ &= \arg\max_h p(\boldsymbol{x},\boldsymbol{h})\end{aligned} \tag{75}$$

Given our model assumptions listed previously and eq.68, it would be easier to deal with the log of the density. We can then revisit our optimization problem, and turn it into minimization for use of the efficiency of gradient descent:

$$\begin{aligned}\boldsymbol{h}^* &= \arg\max_h p(\boldsymbol{h}|\boldsymbol{x};\Theta) \\ &= \arg\max_h p(\boldsymbol{x},\boldsymbol{h}) \\ &= \arg\min_h [-\log p(\boldsymbol{x},\boldsymbol{h})]\end{aligned} \tag{76}$$

The quantity we are looking to minimize is referred to as **energy** of the dynamic system.

MAP estimation using gradient descent As we have seen in the precious chapter, we can solve eq.76 for the optimal latent variable $\boldsymbol{h}^*$ using gradient descent. Using a continuous time formulation:

$$\tau \frac{d}{dt}\boldsymbol{h_i} = -\frac{\partial E(\boldsymbol{h}, \boldsymbol{x}, \Theta)}{\partial \boldsymbol{h_i}} \tag{77}$$

where t is the time step of the gradient descent. Next, let us plug the definition of our model, compute derivatives by hand (this can also be obtained with the help of an symbolic differentiation tool) and get:

insert equation for gradient descent for layer N, 0, and any middle layer i. I think the exact derivation of these equations are not important, it's just the idea of solving MAP estimation using gd that matters

$$adds \tag{78}$$

6.2.4 Quick Interim Summary

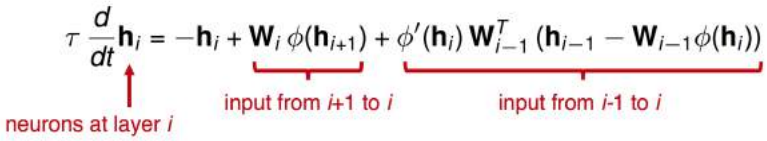
Now, let us recap what we have done so far:

- We specified how a collection of random variables depend on each other using a special type of probabilistic model, the "deep nonlinear Gaussian belief network".
- We decided to not be fully Bayesian, and do inference (i.e., determine latent variables when observing visible variables) using MAP estimation.
- We decided to use gradient descent to perform MAP estimation
- Remember that so far we have no neurons or neural networks

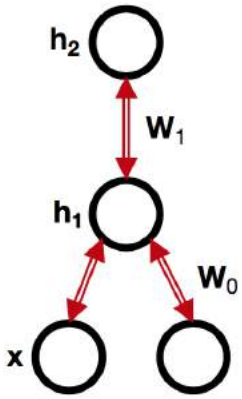
6.2.5 What did we gain?

From the gradient descent equation ended with an equation to solve to get the optimal latent variables. A close look into the different parts if this equation, eq.78, we can notice that it is composed of two parts. For any given layer i, we have

- A first part with an input coming from layer $i+1$ to layer i
- A second part with an input coming from a previous layer $i-1$ to layer i



What does it mean? This means that for inference, we rely on previous and later layers. In other words, the neural network is in fact **recurrent** rather than an **acyclic directed graphical model** (all arrows).



Solving the system eq.78 for optimality, i.e setting the derivative to zero, we get

$$\boldsymbol{h}^* = \boldsymbol{W_i}\phi(\boldsymbol{h^*_{i+1}}) + \phi'(\boldsymbol{h^*_i})\boldsymbol{W^T_{i-1}}(\boldsymbol{h^*_{i-1}} - \boldsymbol{W_{i-1}}\phi(\boldsymbol{h^*_i})) \tag{79}$$

With a closer look into this equation, we note that the first term corresponds to top-down **prediction**, while the second is a bottom-up **prediction error**.

$$\begin{aligned}\mathbf{h}^*_i &= \mathbf{W}_i\,\phi(\mathbf{h}^*_{i+1}) + \phi'(\mathbf{h}^*_i)\,\mathbf{W}^T_{i-1}\,(\mathbf{h}^*_{i-1} - \mathbf{W}_{i-1}\phi(\mathbf{h}^*_i)) \\ &= \underbrace{\mathbf{W}_i\,\phi(\mathbf{h}^*_{i+1})}_{\text{top-down "prediction"}} + \underbrace{\phi'(\mathbf{h}^*_i)\,\mathbf{W}^T_{i-1}\,\mathbf{e}^*_{i-1}}_{\text{bottom-up "prediction error"}}\end{aligned}$$

Solving the recursive equation yields the following:

$$\begin{aligned}\boldsymbol{e^*_0} &= \boldsymbol{x} - \boldsymbol{W_0}\,\phi(\boldsymbol{h^*_1}) \\ \boldsymbol{e^*_i} &= \phi(\boldsymbol{h_i})\,\boldsymbol{W^T_{i-1}}\,\boldsymbol{e^*_{i-1}}\end{aligned} \tag{80}$$

where $0 < i < N$

We can immediately see the similarity with backpropagation. However note that here:

- Errors go **forward**. They are propagated starting from the input to the top most layer N.
- Predictions and prediction errors are **summed up**. In contrast in backprop, we go **forward first**, **then** propagate the errors backward.
- We obtained a neural dynamics which yields **at the fixed point** (steady-state) both predictions and prediction errors. In other words, both predictions and predictions errors are necessary for learning!
- **We are using error propagation when computing predictions**! to clarify!

6.2.6 Unsupervised Learning

To make inference, in the previous section, we assumed the parameters Θ to be fixed. However, in fact, we would like to let our data specify them, i.e learn them from the data D. In this context, since we have no targets, this is an unsupervised learning problem.

A way of doing so in a probabilistic model is via maximum likelihood. Assume we have a set of N_d inputs $\boldsymbol{x}$, that are independent. We can then write the likelihood function as th product of individual likelihood.

$$\arg\max_{\Theta} p(D;\Theta) = \arg\max_{\Theta} \ \Pi_{d=1}^{N_D} \ p(\boldsymbol{x_d};\Theta) \tag{81}$$

Note that the individual likelihood $p(\boldsymbol{x_d};\Theta)$ can be computed by marginalizing the joint density of $\boldsymbol{x}$ and the latent variables $\boldsymbol{h}$, i.e:

$$p(\boldsymbol{x_d};\Theta) = \int_H p(\boldsymbol{x_d},\boldsymbol{h}) d\boldsymbol{h} \tag{82}$$

However, remember that the joint density is is general not Gaussian, which would make the integral evaluation hard.

Approximate algorithm to get the parameters Θ A surprisingly simple approximate algorithm exists for the HGM:

Iterate over many given x:

1. Relax the network to a fixed point:

$$\boldsymbol{h_i^*} = \boldsymbol{W_i} \ \phi(\boldsymbol{h_{i+1}^*}) + \phi'(\boldsymbol{h_i^*}) \ \boldsymbol{W_{i-1}^T} \ \boldsymbol{e_{i-1}} \tag{83}$$

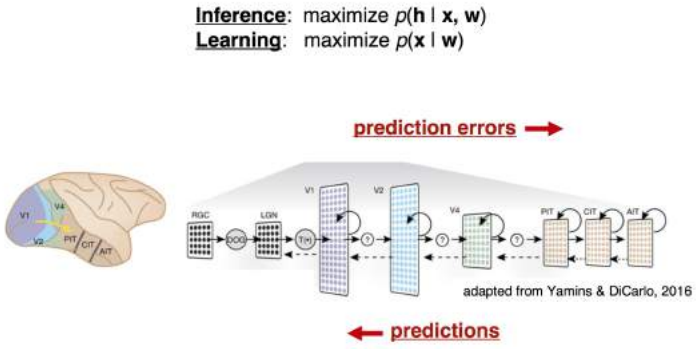
2. Update the weights using **the outer product** between errors and predictions ("Hebbian with the error")

$$\Delta \boldsymbol{W_i} = \eta \ \boldsymbol{e_i^*} \ \phi(\boldsymbol{h_{i+1}^*})^T \tag{84}$$

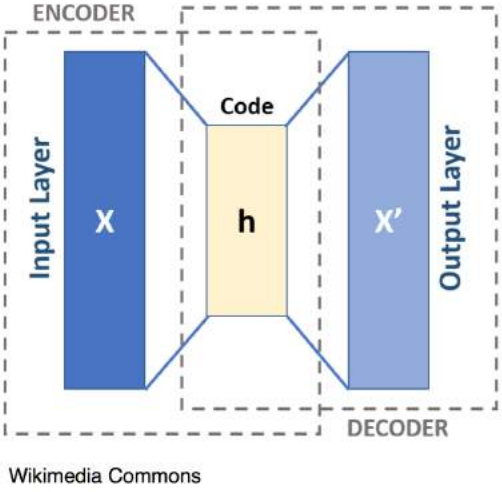
The algorithm can be derived in the framework of variational inference.This view is useful as it allows obtaining more refined versions (e.g., Rezende et al., 2014)

6.2.7 Interim summary

Stepping back again to see what we have done so far.



Discussion point: Does this way of doing unsupervised learning remind you of another neural network architecture? Yes, autoencoders. Conceptually, autoencoder is close to a probabilistic model of the data.



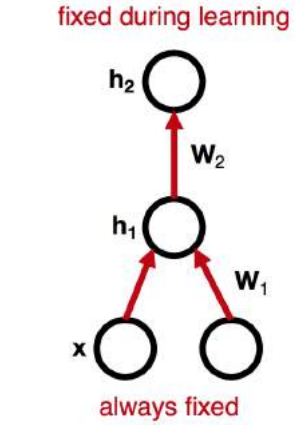
- We assumed data are generated through a **deep nonlinear Gaussian model** (a hierarchical gaussain model) $p(\boldsymbol{x}, \boldsymbol{h})$ with N layers of latent variables $\boldsymbol{h}$.
- We have seen how **perception is a probabilistic inference**, where making inference in this context is equivalent to finding $p(\boldsymbol{h}|\boldsymbol{x})$ (i.e inverting the model).

- We simplified inference by summarizing the full posterior $p(\boldsymbol{x}, \boldsymbol{h})$ by its mode **(maximum a posteriori inference**).
- We obtained a **recurrent neural network model** with a special architecture that does the job (inference of h) for us.
- Learning **(inference of W)** can be done also within the recurrent network using two quantities: **predictions** and **prediction errors**.

6.3 Dynamics for error propagation

It's nice that we have a continuous-time neural dynamics that yields both predictions and errors at steady state. But can we get something like that for our regression and classification deep learning models?

Our goal here is to use the same model to implement regression and classification. In order to recover our classical view for backpropagation, we need to modify the way we do inference and learning.



probabilistic model
of Whittington & Bogacz 2017

During inference

$$\tau \frac{d}{dt} \boldsymbol{h_N} = -\boldsymbol{h_N} \; + \; \boldsymbol{W_N} \; \phi(\boldsymbol{h_{N-1}})$$
$$\tau \frac{d}{dt} \boldsymbol{h_i} = -\boldsymbol{h_i} \; + \; \boldsymbol{W_i} \; \phi(\boldsymbol{h_{i-1}}) + \; \phi'(\boldsymbol{h_i}) \; \boldsymbol{W_{i+1}^T} \; (\boldsymbol{h_{i+1}} - \boldsymbol{W_{i+1}} \; \boldsymbol{\phi(h_i)})$$

During learning: set the top most layer of latent variables to the target outputs

$$\boldsymbol{h_N} = t \tag{85}$$

- Now errors propagate backward.

- During inference, errors always vanish, so we recover a functionally **feed-forward network** (no error terms involved).
- The model is only "truly recurrent" during learning, when we fix the output layer.
- In the limit of $\Sigma_N \to \infty$ we recover **exactly** backprop!

6.3.1 How might the brain implement this neural network?

$$\tau \frac{d}{dt}\mathbf{h}_i = -\mathbf{h}_i + \mathbf{W}_i \phi(\mathbf{h}_{i-1}) + \phi'(\mathbf{h}_i)\, \mathbf{W}_{i+1}^T (\mathbf{h}_{i+1} - \mathbf{W}_{i+1}\phi(\mathbf{h}_i))$$

seems fine, propagate "firing rate" up

are we sending "voltage" back?

How do we wire up this network? Is there a single best way to do so? There are few concers that we have to address:

- Lots of tied weights (recall the weight transport problem)
- What is ϕ? Maybe it transforms "voltage" (internal quantity) to neuron firing rate (spikes/second, transmitted to other cells)?
- But then what are the $\boldsymbol{h_i}$? Are they the firing rate, or the "voltage"?

6.3.2 Classical "predictive coding" model

7 Unsupervised And Self-Supervised Learning

7.1 Introduction to Unsupervised Learning

7.1.1 Motivation

Goeffrey Hinton (backprop[50]) suggests networks should be able to become intelligent on their own, unsupervised, without backprop.

1. **Does not need labelled data.** In practice we often assume a fixed number of labels and call them cluster centers.
2. **Is able to learn reveal the intricate structure (features) and can be used for dimensionality reduction.** This means we try to find projections in this this lower dimension space.
3. Is able to generate new data examples that are consistent with the statistics of the training data.
4. **Can predict future data examples (e.g. video frame prediction).** If we know the data representation we can try to find an inverse function out of our lower dimensional space. Which means we can sometimes reconstruct high dimensional data.
5. **Can be used to detect familiar or out-of-distribution examples.** If we freeze our clusters, and then switch to another data set, then we can calculate the new data distance to the original data cluster centers.
6. Can facilitate future supervised learning.
7. Can be used to cluster the data.
8. **Can be used to de-noise data.** We loose some information when projecting to a lower dimensional space. We optimize our method to retain the most useful information, this process might also de-noise our data.
9. Can be used to encode data in a particular way (e.g. for data compression).
10. We assume that humans have learned the world in an almost unsupervised way.

[50] Learning representations by back-propagating errors

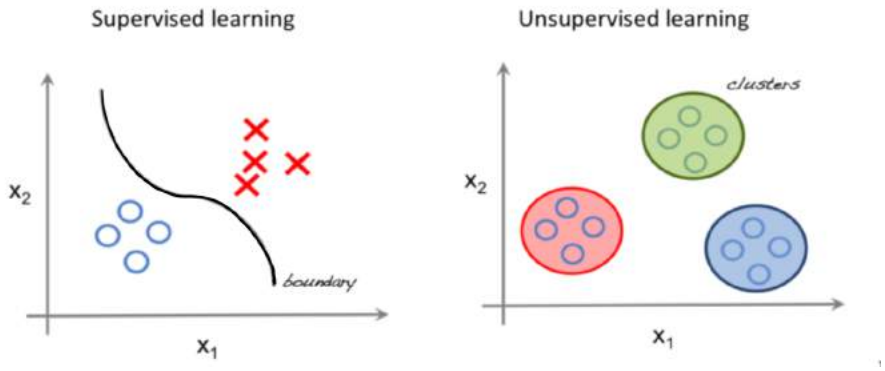


Figure 61: The Difference between a supervised regression task where the data points are separated by some function that represents a bound in between different classes and an unsupervised clustering where the algorithm highlights the intrinsic structure of the data.

7.1.2 Unsupervised Learning in the Brain

We are aware of the fact that the brain as well uses unsupervised learning. Several examples were collected for the lecture.

In 1970 an experiment was conducted on cats. The kittens were housed from birth in a completely dark room, but form the age of 2 weeks they were put in a special apparatus for an average of about 5 hours a day. The kitten stood on a clear glass platform inside a tall cylinder of which the entire surface was covered in black and white stripes (in different experiments they used horizontal and vertical stripes). Those poor cats then were virtually blind for contours perpendicular to the orientation they had experienced. They recorded single neurons from primary visual cortex and found that almost all cats had their neurons trained to be most selective in direction of the stripes presented in the experiment. Example distribution in Figure section 7.1.2. **Interpretation: The neurons are the cluster centers and they move around during learning (growing up). When presented one stimulus only, then all cluster centers group in the same optimum.**

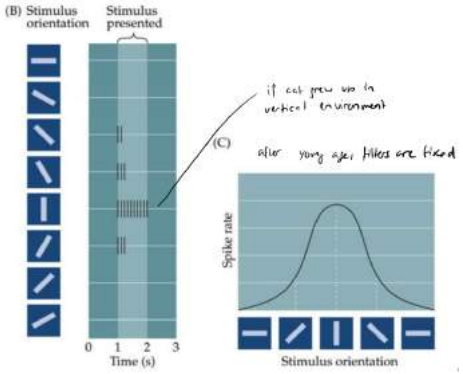


Figure 62: A spike rate curve of a single neuron with respect to the neurons orientation. Experiments show that this distribution changes in adolescent subjects and becomes more rigid with increasing age.

Another group analyzed visual cortical activity of awake ferrets during development (2007). They provide and one-sentence summary: The relation between spontaneous activity and activity evoked by natural stimuli in the primary visual cortex reveals that the cortical circuit progressively adapts its internal model to the statistical structure of the environment. [51]

[51] Spontaneous Cortical Activity Reveals Hallmarks of an Optimal Internal Model of the Environment

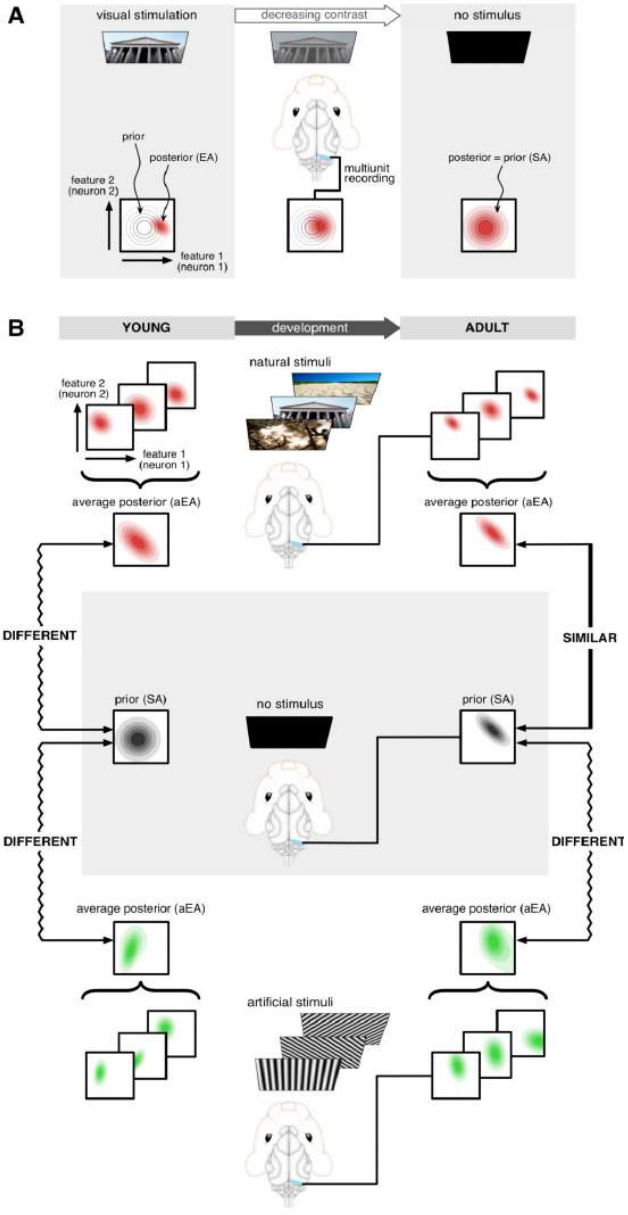


Figure 63: Notation: Evoked and spontaneous (dark) neural activity (EA and SA). Multi-neural EA (aEA) **(A)** The posterior distribution represented by EA is increasingly dominated by the prior distribution as brightness or contrast is decreased. **(B)** ferrets either receiving no stimulus (middle) or viewing natural (top) or artificial stimuli (bottom) is used to construct neural activity distributions in young and adult animals. It reveals the level of statistical adaptation of the internal model to the stimulus ensemble. The internal model of young animals (left) is expected to show little adaptation to the natural environment and thus aEA for natural (and also for artificial) scenes should be different from SA. Adult animals (right) are expected to have adapted to natural scenes and thus to exhibit a high degree of similarity between SA and natural stimuli aEA, but not between SA and artificial stimuli aEA.

We now know that these distributions adapt, but from the presented experiments it's unclear what the conditions are to trigger an adaption. Another experiment [52] shows that the relative timing of presynaptic and postsynaptic spikes plays a critical role in activity-induced synaptic orientation (single unit recording in cat V1). Induction of a significant shift required that the interval between the pair fall within +-40 ms otherwise nothing changed.

Another path to understand the learning in neural circuits leads to the recent advances in deep neural networks (DNN). Several groups tried to map layers (as in DNN layers) to cortical regions. Several mapping strategies were found. We can show that dissimilarity matrices of regions in both system look similar, especially in higher cortical regions vs deeper layers of neural networks. Interestingly, the animals we recorded from never knew any labels that were used to train the DNNs.

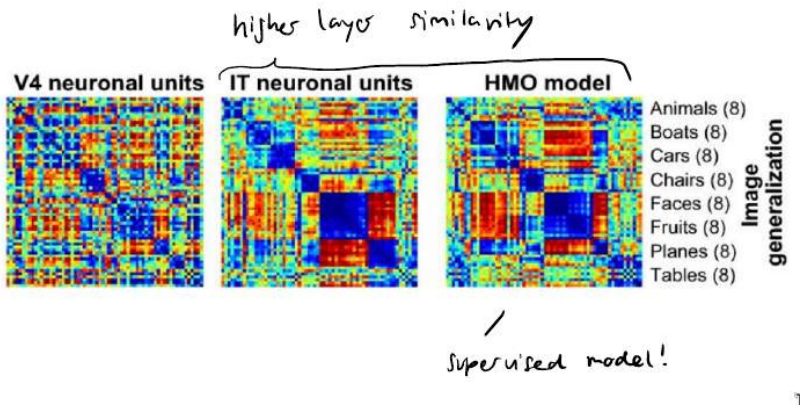


Figure 64: Confusion Matrix of V4 neural units and units in artificial networks from a comparable depth.

7.2 Sparse Coding

The sparse code is found when each sample of a given data set is encoded by the strong activation of a relatively small set of neurons. For each item to be encoded, this is a different subset of all available neurons.

Each image can be represented by a set of basis functions:

$$I(x,y) \approx \sum_i a_i \phi_i(x,y) = \hat{I} \tag{86}$$

We define an energy function which is essentially true image I minus approximation $\tilde{I}$ + regularizer S:

$$E = \sum_{x,y} [I(x,y) - \sum_i a_i \phi_i(x,y)]^2 + \lambda \sum_i S(\frac{a_i}{\tau}) \tag{87}$$

Presented in the lectures was a set of known regularizers $\mathbb{S}$:

[52] Stimulus Timing-Dependent Plasticity in Cortical Processing of Orientation

$$S \in \mathbb{S} = \{\exp^{x^2}, log(1+x^2), \|x\|\} \tag{88}$$

For each image presentation E is mimized with respect to a_i. The a_i are determined by the equilibrium solution of the differential equation (the paper does not show a proof):

$$\frac{\partial E}{\partial a_i} = \dot{a_i} = b_i - \sum_j c_{ij} a_j - \frac{\lambda}{\tau} S'(\frac{a_i}{\tau}) \tag{89}$$

$$b_i = \sum_{x,y} \phi_i(x,y) I(x,y) \tag{90}$$

$$c_{ij} = \sum_{x,y} \phi_i(x,y) \phi_j(x,y) \tag{91}$$

The ϕ_i then evolve by gradient descent on E averaged over many image presentations. The learning rule for updating ϕ is then:

$$\nabla \phi_i(x_m, y_n) = \eta \langle a_i [I(x_m, y_n) - \hat{I}(x_m, y_n)] \rangle \tag{92}$$

Comments on Parameters and Operations:

- a_i factors, should be sparse, different for each sample
- ϕ components, basis functions that are the same over all samples
- τ scaling constant
- λ scaling constant: sparseness vs information perseverance
- η learning rate
- $\langle x \rangle$ mean

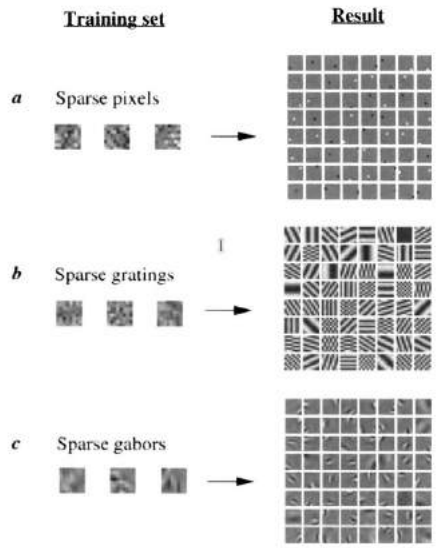


Figure 65: Training examples and the respective basis functions of sparse coding.

7.2.1 Relation to Neuroscience

The paper [53] shows that cells of sub-units in V1 have receptive fields that apply signal filtering that is very similar to sparse coding. In V2 they identified sub-units with spatial feature selectivity.

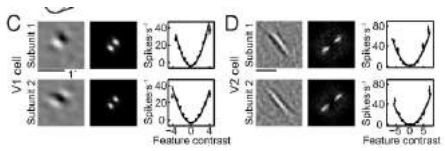


Figure 66: Sparse coding in the brain.

7.3 Infomax (ICA)

Infomax is an optimization principle for artificial neural networks and other information processing systems. It prescribes that a function that maps a set of input values I to a set of output values O should be chosen or learned so as to maximize the average Shannon mutual information between I and O. One of the applications of Infomax has been to an independent component analysis algorithm (ICA) that finds independent signals by maximizing entropy. ICA via mutual information is one way to find independent components.

The independence criterion is stronger than uncorrelatedness which is defined as:

$$\langle X_1 X_2 \rangle - \langle X_2 \rangle \langle X_1 \rangle = 0 \tag{93}$$

or

$$Cov(X_1, X_2) = \mathbf{E}[X_1 X_2] - \mathbf{E}[X_1]\mathbf{E}[X_2] = 0 \tag{94}$$

Remember: If two variables are uncorrelated, there is no linear relationship between them. However this does not mean that they are independent. The other way works: if X_1 and X_2 are independent (with finite second moments), then they are uncorrelated.

For ICA we want statistical independence

$$p(\alpha_{1:N}) = \prod_{i=1}^{N} p(\alpha_i) \tag{95}$$

To measure the degree of dependence we look at the pairwise mutual information of two random variables X, Y. Mutual information is non-negative and symmetric:

$$\begin{aligned}
\mathrm{I}(X;Y) &\equiv H(X) - H(X|Y) && (96)\\
&\equiv H(Y) - H(Y|X) && (97)\\
&\equiv H(X) + H(Y) - H(X,Y) && (98)\\
&\equiv H(X,Y) - H(X|Y) - H(Y|X) && (99)\\
I(X;Y) &= D_{\mathrm{KL}}(P_{(X,Y)} \| P_X \otimes P_Y) && (100)
\end{aligned}$$

[53] Spatial structure of neuronal receptive field in awake monkey secondary visual cortex

We use the entropy:

$$H(X) = -\sum_{i=1}^{n} \mathrm{P}(x_i) \log_b \mathrm{P}(x_i) \tag{101}$$

Idea: X, Y independent if:

$$p(X, Y) = p(X)p(Y) \tag{102}$$

For the discrete case we can then see best that last part of eq. is $log(1) = 0$:

$$\mathrm{I}(X;Y) = \sum_{y \in \mathcal{Y}} \sum_{x \in \mathcal{X}} p_{(X,Y)}(x, y) \log \left(\frac{p_{(X,Y)}(x, y)}{p_X(x)\, p_Y(y)} \right), \tag{103}$$

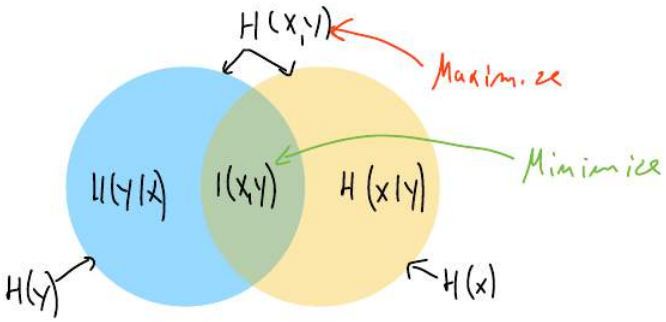


Figure 67: Venn diagram of the Infomax objective: We want to maximize the entropy and minimize the mutual information. Entropy maximization forces the network to generalize. See: principle of maximum entropy.

How do we actually implement Infomax? We can think of Infomax as a one layer linear neural network that produces

$$\mathbf{y} = \mathbf{W}\mathbf{x} \tag{104}$$

such that outputs y_i are maximally independent:

$$\forall (y_i, y_j), i \neq j, y_i, y_j \in \mathbf{y} : I(y_i, y_j) = I_{min} \tag{105}$$

but keep in mind we also maximize $I(X;Y)$. In the lecture, $\mathbf{y}$ somehow converges to a set of Gabor-like filters, but it was not specified in detail why.If somebody knows, please contact me (RT).

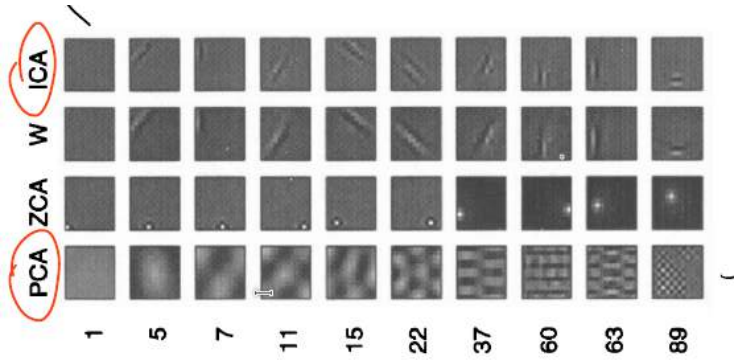


Figure 68: We can see that the PCA features are very different from what we know that the brain uses for feature representation. However the ICA representation looks like the (Gabor-like) filters from the previous chapter.

- ICA is similar to Principal Component Analysis, except that we're looking for a transformation subject to the stronger requirement of independence, rather than uncorrelatedness.
- In general, no analytic solution (like eigenvalue decomposition for PCA) exists. Thus, ICA is typically implemented using neural network models and GD.
- For the ICA NN implementation, we need an architecture and an objective function to descend/climb.
- Results in N independent (or as independent as possible) components in an N-dimensional space; these don't need need to be orthogonal (PCA is an orthogonal transformation).

7.4 Autoencoders

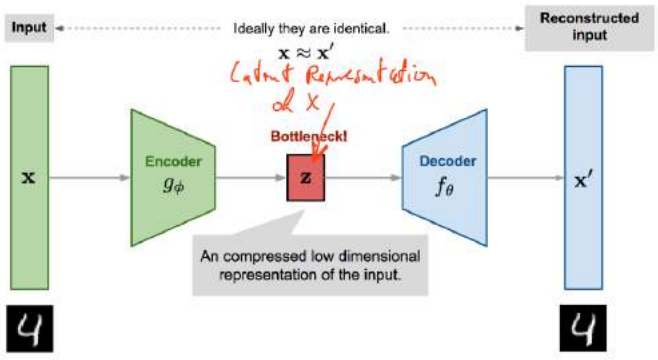


Figure 69: Autoencoder

Autoencoder loss function:

$$L_{AE} = \frac{1}{2}\|\mathbf{x} - \mathbf{x'}\|^2, \mathbf{x'} = f_\theta(g_\phi(\mathbf{x})) \tag{106}$$

The autoencoder is trained by gradient descent.
Comments on Parameters and Operations:

- ϕ parameters of encoding function (trainable)
- θ parameters of decoding function (trainable)
- g_ϕ encoding function
- f_θ decoding function

7.4.1 Semi-supervised autoencoder

The supervised AE uses the latent space to define a second decoding pathway. This path is added as another term to the loss and one calculates the gradients from two different ends. In the shared part, these gradients then merge. This is called multi-task learning (having a shared pathway for different objectives).

$$L_{SAE} = \frac{1}{t}\sum_{i=1}^{t}[L_p(\mathbf{x_i}, \mathbf{W}_{1:2}, \mathbf{y_i}) + L_v(\mathbf{x_i}, \mathbf{W}_{1:4}, \hat{\mathbf{x}}_i)] \tag{107}$$

- L_p loss of label y_i
- L_v loss of reconstruction $\hat{x_i}$
- $W_{1,2}$ weights of encoder (that produce label y)
- $W_{1,4}$ weights of encoder + decoder

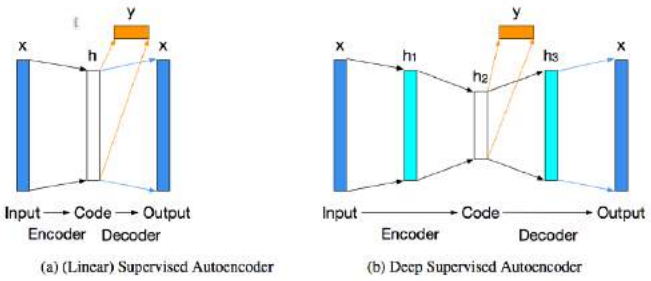


Figure 70: Semi-supervised autoencoder

7.4.2 De-noising Autoencoder

Since the autoencoder learns the identity function, we are facing the risk of "overfitting" when there are more network parameters than the number of data points. To avoid overfitting and improve the robustness, Denoising Autoencoder (Vincent et al. 2008) proposed a modification to the basic autoencoder. The input is partially corrupted by adding noises to or masking some values of the

input vector in a stochastic manner. To "repair" the partially destroyed input, the denoising autoencoder has to discover and capture relationship between dimensions of input in order to infer missing pieces. Similar to dropout.

$$\tilde{\mathbf{x}}^{(i)} \sim \mathcal{M}_{\mathcal{D}}(\tilde{\mathbf{x}}^{(i)}|\mathbf{x}^{(i)}) \tag{108}$$

$$L_{\text{DAE}}(\theta, \phi) = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{x}^{(i)} - f_\theta(g_\phi(\tilde{\mathbf{x}}^{(i)})))^2 \tag{109}$$

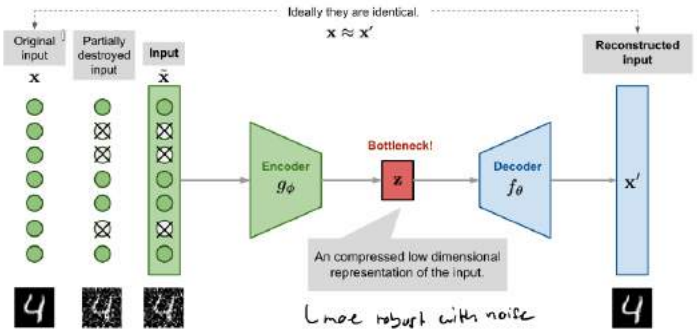


Figure 71: De-noising autoencoder

7.4.3 Sparse Autoencoder

The Sparse Autoencoder applies a sparsity constraint on the hidden unit activation to avoid overfitting and improve robustness. It forces the model to only have a small number of hidden units being activated at the same time.

$$\hat{\rho}_j^{(l)} = \frac{1}{n}\sum_{i=1}^{n}[a_j^{(l)}(\mathbf{x}^{(i)})] \approx \rho \tag{110}$$

Let's say there are s_l neurons in the l-th hidden layer and the activation function for the j-th neuron in this layer is labelled as $a_j^{(l)}(.)$, $j = 1, \ldots, s_l$. The fraction of activation of this neuron ρ^j is expected to be a small number ρ, known as sparsity parameter; a common config is $\rho = 0.05$.

$$L_{\text{SAE}}(\theta) = L(\theta) + \beta\sum_{l=1}^{L}\sum_{j=1}^{s_l} D_{\text{KL}}(\rho\|\hat{\rho}_j^{(l)}) \tag{111}$$

$$= L(\theta) + \beta\sum_{l=1}^{L}\sum_{j=1}^{s_l} \rho\log\frac{\rho}{\hat{\rho}_j^{(l)}} + (1-\rho)\log\frac{1-\rho}{1-\hat{\rho}_j^{(l)}} \tag{112}$$

Keep in mind that we specify our desired target distribution that is ρ. Common activation functions include sigmoid, tanh, relu, leaky relu, etc.

7.4.4 Contracting Autoencoder

Similar to sparse autoencoder, Contractive Autoencoder (Rifai, et al, 2011) encourages the learned representation to stay in a contractive space for better robustness.

It adds a term in the loss function to penalize the representation being too sensitive to the input, and thus improve the robustness to small perturbations around the training data points. The sensitivity is measured by the Frobenius norm of the Jacobian matrix of the encoder activations with respect to the input:

$$\|J_f(\mathbf{x})\|_F^2 = \sum_{ij} \Big(\frac{\partial h_j(\mathbf{x})}{\partial x_i}\Big)^2 \tag{113}$$

where h_j is one unit output in the compressed code $\mathbf{z} = f(x)$.

This penalty term is the sum of squares of all partial derivatives of the learned encoding with respect to input dimensions. The authors claimed that empirically this penalty was found to carve a representation that corresponds to a lower-dimensional non-linear manifold, while staying more invariant to majority directions orthogonal to the manifold.[54]

7.5 Competitive Learning

Competitive learning is a form of unsupervised learning in artificial neural networks, in which nodes compete for the right to respond to a subset of the input data. A variant of Hebbian learning, competitive learning works by increasing the specialization of each node in the network. It is well suited to finding clusters within data. Imagine that we move our neuron around that space by adjusting the weights.

- Neurons/Nodes are all the same except for their weights.
- A competitive mechanism permits neurons to compete for the right to respond to a given subset of inputs, such that only one output neuron (or only one neuron per group), is active (i.e. "on") at a time.
- The neuron that wins the competition is called a "winner-take-all" neuron and is allowed to update it's weights.
- During 'learning' individual neurons of the network learn to specialize on ensembles of similar patterns and become 'feature detectors' for different classes of input patterns.
- The competitive networks are able to recode sets of correlated inputs to a few output neurons.

7.5.1 The CL Algorithm

The competitive learning algorithm (two clusters):

[54] https://lilianweng.github.io/lil-log/2018/08/12/from-autoencoder-to-beta-vae.html#sparse-autoencoder

1. Let all inputs feed into two different nodes, so that every hidden node is connected to every input. Initialize the weights randomly between 0.0 and 1.0. Calculate the activity of each hidden node for the first input.
2. The hidden node with the highest output is the winner for the cluster to which the data point belongs.
3. The winner node updates each of its weights, thereby moving its weight vector towards the data point.
4. Repeat with the next data point.

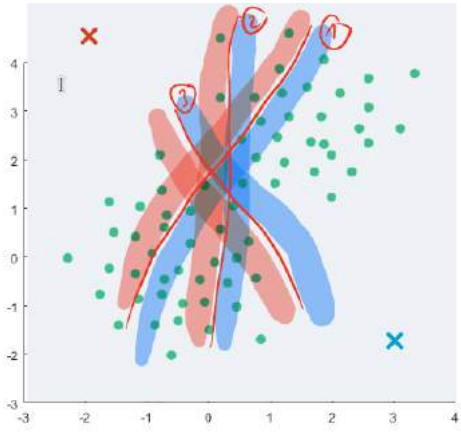


Figure 72: Illustration of how the barrier would move if the blue neuron would move upwards in direction of the cluster center and the red one downwards (3 iterations are shown). However after revisiting this example I think the blue one would occupy the lower cluster!

7.6 CL with Neural Networks

We ask for the neuron with the closest weight vector:

$$\|u - w_i\| = u^T u - 2w_i^T u + w_i^T w_i = \|u\|^2 + \|w_i\|^2 - 2y_i \tag{114}$$

$$argmin\|u - w_i\| = argmax(y_i) \tag{115}$$

We update our weights accordingly:

$$w_A(t) = \frac{1}{t}\sum_{o=1}^{t} u_t \tag{116}$$

$$\nabla w_A = \eta(u_t - w_A) \tag{117}$$

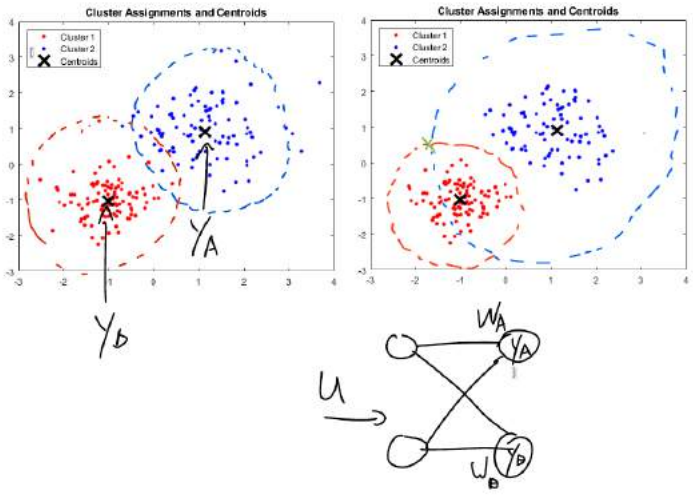


Figure 73: Competitive NN: The position of the two neurons after convergance (left). A new datapoint and the datapoints equidistance line to the cluster-center neurons (right). The network structure(bottom).

7.7 Self Organizing Maps

When a training example is fed to the network, its Euclidean distance to all weight vectors is computed. The neuron whose weight vector is most similar to the input is called the best matching unit (BMU). The weights of the BMU and neurons close to it in the SOM grid are adjusted towards the input vector.The magnitude of the change decreases with time and with the grid-distance from the BMU.

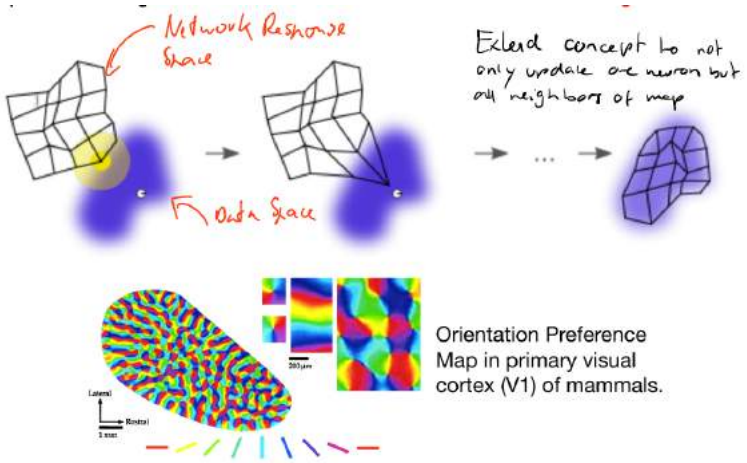


Figure 74: Self Organizing Map

7.8 Plots of Different Methods

As a summary:

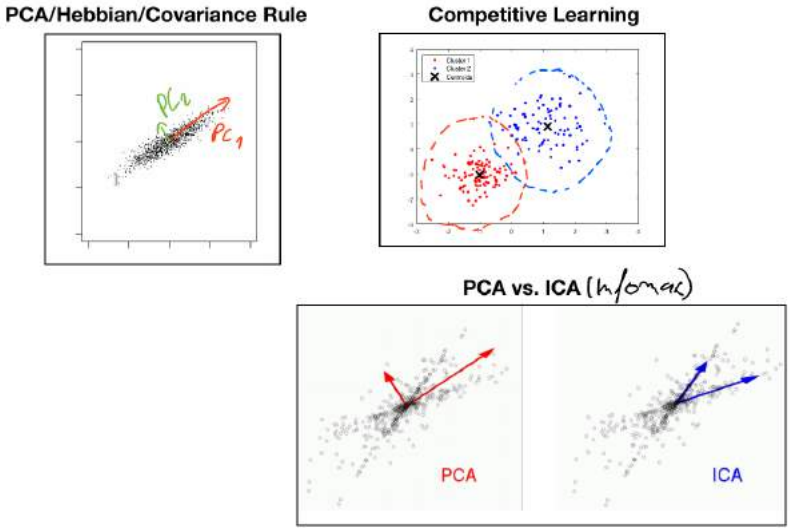


Figure 75: PCA, Competitive Learning and ICA in one picture

7.9 Probabilistic Generative Models

We know that or data has some causes v, but its hard to specify those. We do not know the underlying distribution. We can see the real world as a generative model that produced our (observable) data u. Now we want to mimic this process. We model the causes as prior $p(v, G)$ and our generative model specifies the artificial data distribution $p(u|v, G)$. We have a recognition model that maps the samples gathered in the "real" world somehow to our generative model (not clear from the slides how).

Goal: Learn a good generative model that mimics the statistics of the data generation process Approach: Given data, solve two problems:

1. Estimate the causes by computing the posterior
2. Learn all parameters G of the model and the latent space statistics.

A very basic example is a mixture of Gaussians:

$$p(u, G) = \sum_v p(u|v, G)p(v, G) \tag{118}$$

$$G = (\mu_v, \tau_v, y_v) \tag{119}$$

I assume these parameters are means, variances and mixture scaling factors.

There are several ways how one could use a neural network for these challenges. Also known as 'Maximum Likelihood Learning':

- Learn the recognition model

- Learn the generative model
- Learn/update the prior $p(z)$

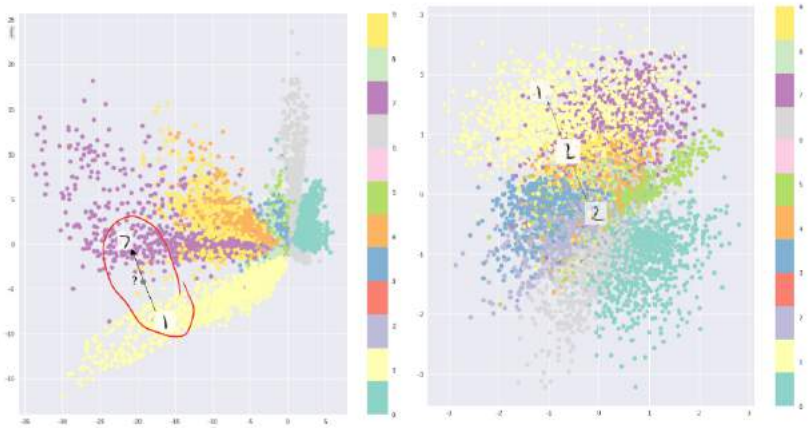


Figure 76: The difference between an traditional embedding algorithm on the left where we have huge gaps in embedding space and a generative embedding method on the right that densely covers the space. This allows us to sample.

7.10 The restricted Boltzmann Machine

A restricted Boltzmann machine (RBM) is a generative stochastic artificial neural network that can learn a probability distribution over its set of inputs. As their name implies, RBMs are a variant of Boltzmann machines, with the restriction that their neurons must form a bipartite graph. This means that in restricted Boltzmann machines there are only connections (dependencies) between hidden and visible units, and none between units of the same type (no hidden-hidden, nor visible-visible connections). Although learning is impractical in general Boltzmann machines, it can be made quite efficient for RBMs. A deep Boltzmann machine (DBM) is a type of binary pairwise Markov random field (undirected probabilistic graphical model) with multiple layers of hidden random variables.

Practical details:

- RBMS have two biases (visible and hidden).
- The hidden bias helps the RBM produce the activations on the forward pass, while
- The visible layer's biases help the RBM learn the reconstructions on the backward pass.

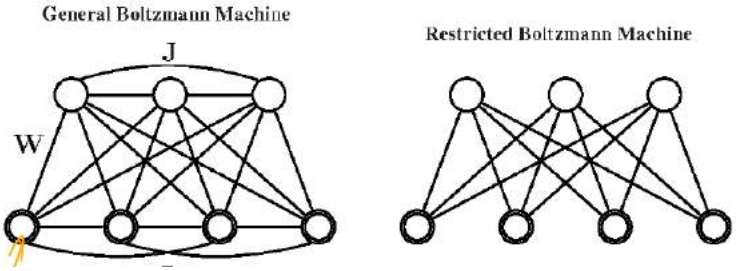


Figure 77: Difference between a general and a restricted Boltzmann machine. The weights (orange arrow) are probabilistic units with activation 0 or 1. In Boltzmann machines, information flows forward and backwards.

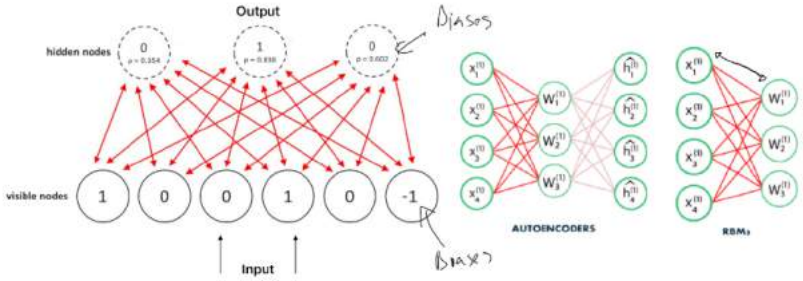


Figure 78: (right) RBMs are similar to (reverse) autoencoders but use stochastic units with particular distribution instead of deterministic distribution. The task of training is to find out how these two sets of variables are connected to each other. (left) The difference between the hidden nodes which are probabilistic and the input nodes.

7.10.1 Training the Restricted Boltzmann Machine (Contrastive Divergence)

Is not used anymore because back-propagation works so well. Attempts were made to use RBMs as dimensionality reduction algorithm. Results were okay, compared to PCA, the clusters seem more dense and separated.

Training algorithm:

1. Take a sample V_0 and compute the hidden activation vector h_0. Call $V_0 h_0^T$ (outer product) the positive gradient.
2. Sample from h_0, compute V_1, resample from V_1, compute h_1. Call $V_1 h_1^T$ the negative gradient.
3. Update weights: $\nabla w = \eta(v_0 h_0 - v_1 h_1)$

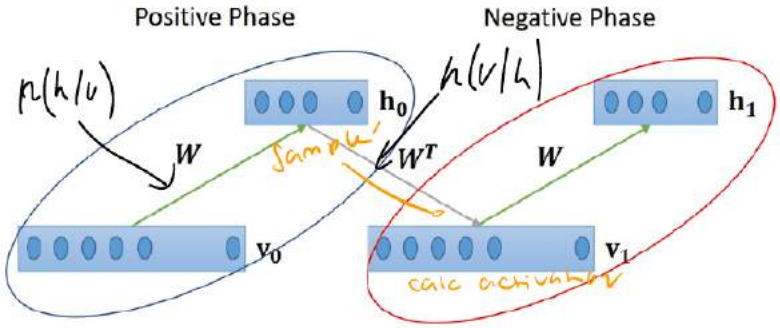


Figure 79: Visualization of the training algorithm

7.11 Helmholtz Machines

"Never really worked"

Prof. Grewe

A Helmholtz machine contains two networks, a bottom-up recognition network that takes the data as input and produces a distribution over hidden variables, and a top-down "generative" network that generates values of the hidden variables and the data itself.

Helmholtz machines were created to improve noise resilience, which is always present in natural data, and in hope that by learning economical representations of the data, the underlying structure of the generative model should reasonably approximate the hidden structure of the data set.

The two phases of the Boltzmann machine contrast the statistics of the activations of the network when input patterns are presented with the statistics of the activations of the network when it is running 'free'. This contrastive procedure involves substantial noise and is slow. The Helmholtz machine does not have this problem.

7.11.1 Wake-Sleep Algorithm

Used to train Helmholtz Machines. The wake and sleep phases are not contrastive. Rather, the recognition and generative models are forced to chase the other.

- Wake Phase: **Use the recognition weights** to perform a bottom up pass and **train the generative weights** to reconstruct the activity in each layer from the layer above.
- Sleep Phase (because the network is insensitive to input): **Use the generative weights** to generate samples from the model, propagate these from top to bottom and **then use (train) the recognition weights** to reconstruct the activities in each layer from the layer below.

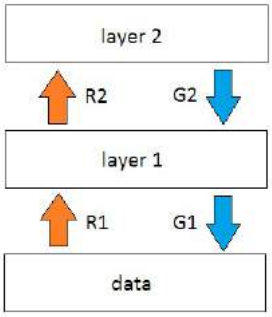


Figure 80: Layers of the neural network. R, G are weights used by the wake-sleep algorithm to modify data inside the layers.

7.12 Variational Autoencoders

The idea of Variational Autoencoder (Kingma & Welling, 2014), short for VAE, is actually less similar to all the autoencoder models above, but deeply rooted in the methods of variational bayesian and graphical model. Instead of mapping the input into a fixed vector, we want to map it into a distribution.

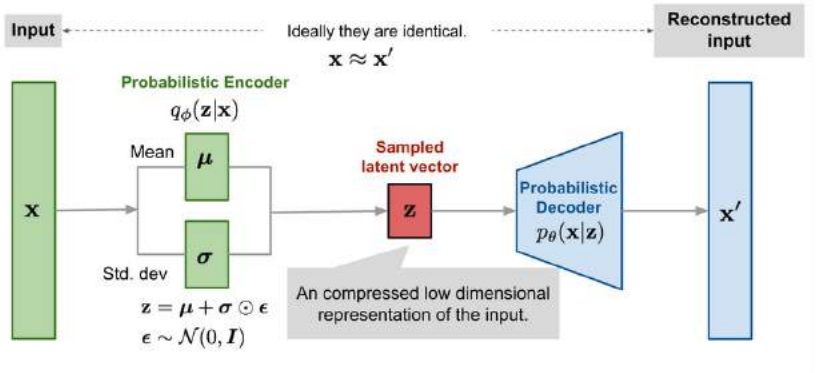


Figure 81: Illustration of variational autoencoder model with the multivariate Gaussian assumption.

7.12.1 Reparametrization Trick

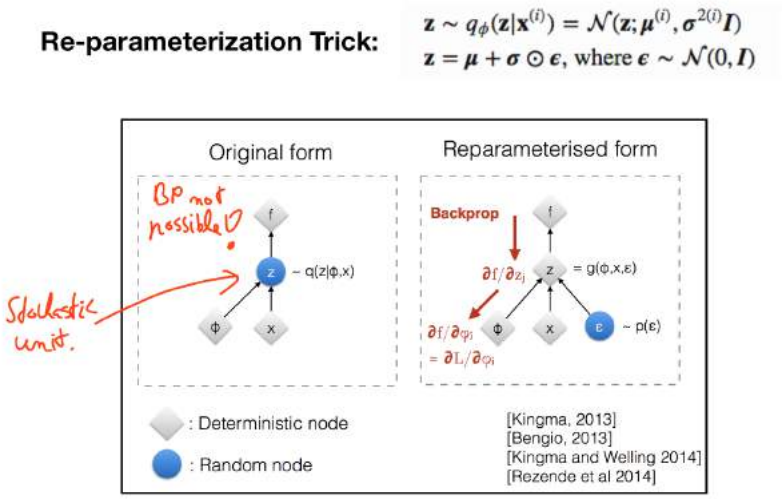


Figure 82: Reparametrization trick in detail.

7.12.2 VAE Loss

The VAE loss function is an application of the reparametrization trick. It allows to back-propagate from the decoder side to the encoder side. Without this trick, the bottleneck would be fully probabilistic without a chance to compute a gradient flow trough.

$$L_{\text{VAE}}(\theta, \phi) = -\log p_\theta(\mathbf{x}) + D_{\text{KL}}(q_\phi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z}|\mathbf{x})) \tag{120}$$

$$= -\mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}|\mathbf{z}) + D_{\text{KL}}(q_\phi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z})) \tag{121}$$

$$\theta^*, \phi^* = \arg\min_{\theta,\phi} L_{\text{VAE}} \tag{122}$$

In Variational Bayesian methods, this loss function is known as the variational lower bound, or evidence lower bound. Therefore by minimizing the loss, we are maximizing the lower bound of the probability of generating real data samples.

$$-L_{\text{VAE}} = \log p_\theta(\mathbf{x}) - D_{\text{KL}}(q_\phi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z}|\mathbf{x})) \leq \log p_\theta(\mathbf{x}) \tag{123}$$

Figure 83: We can perform vector arithmetic in the latent space which is densely covered and then use the decoder to map to a sample in data space.

7.13 Predictive Coding

Predictive coding (also known as predictive processing) is a theory of brain function in which the brain is constantly generating and updating a mental model of the environment. The model is used to generate predictions of sensory input that are compared to actual sensory input. This comparison results in prediction errors that are then used to update and revise the mental model. In short: the brain tries to predict the next input to our network.

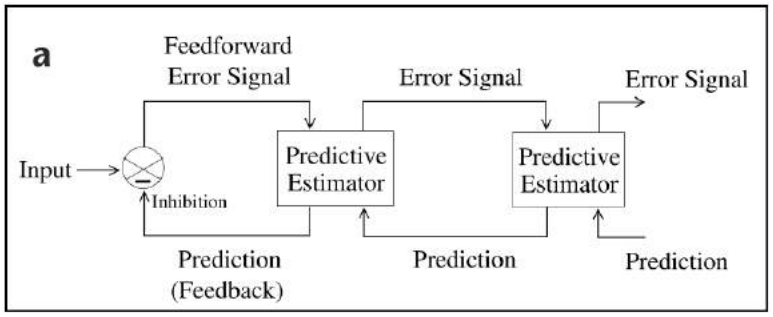


Figure 84: Inhibiting feedback prediction mechanism in the visual cortex.

7.13.1 Deep Predictive Coding: Pred-Net

The network consists of a series of repeatingstacked modules that attempt to make local predictions of the input to the module, which is thensubtracted from the actual input and passed along to the next layer.

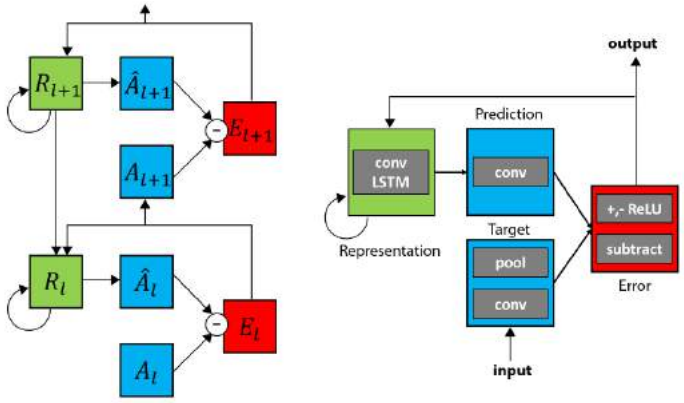


Figure 85: Figure 1: Predictive Coding Network (PredNet). Left: Illustration of information flow within two layers. Each layer consists of representation neurons (R_l), which output a layer-specific prediction ateach time step ($\hat{A}_l$), which is compared against a target (A_l) (Bengio, 2014) to produce an error term(E_l), which is then propagated laterally and vertically in the network. Right: Module operations forcase of video sequences.

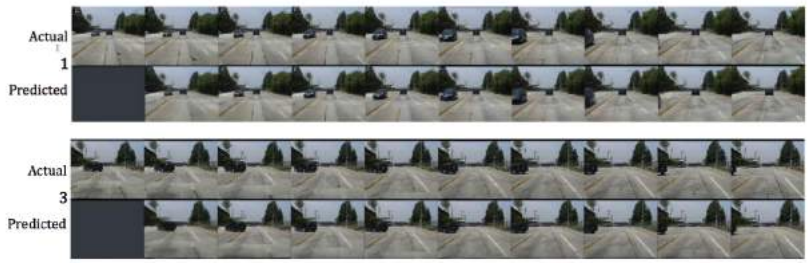


Figure 86: The PredNet in action

7.14 Pixel-RNN

The PixelRNN is a generative model for images. The network models conditional distribution of every individual pixel given previous pixels (to the left and to the top).

$$p(\mathbf{x}) = \prod_{i=1}^{n^2} p(x_i|x_1, ..., x_{i-1}) \tag{124}$$

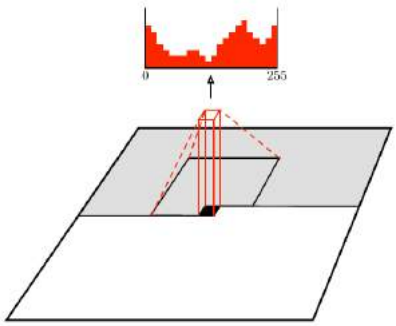


Figure 87: Distribution over color space of a single pixel in the generative process of the PixelRNN.

7.15 What you should know

You should be able to explain:

- the idea of sparse coding, including formulas.
- competitive learning in NNs.
- the different concepts of auto encoders and their cost functions.
- the Boltzmann machine.
- the Helmholtz machine and the wake-sleep algorithm.
- the basic idea behind VAEs and the re-parametrization trick.
- the idea behind predictive coding and its relation to neuroscience.
- the (dis)advantages of generative vs. non-generative UL methods.

8 Reinforcement Learning

Reinforcement learning fuses ideas from neuroscience and AI. The model describes how an agent can interact with an environment and in that environment learn to improve its actions when it comes to gathering a targeted reward.

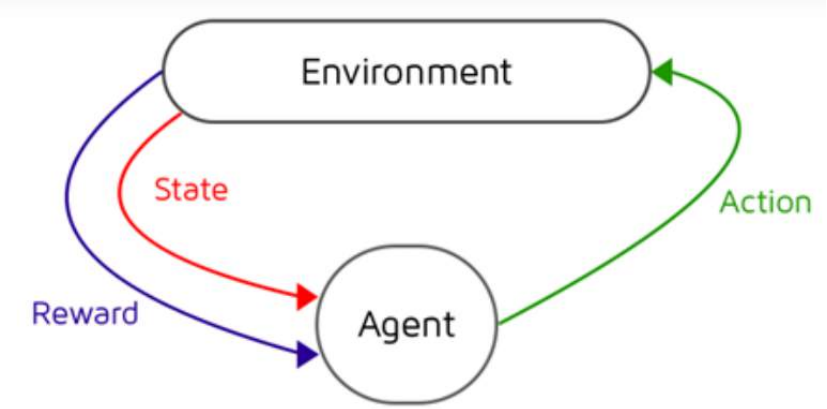


Figure 88: Connections between an agent and the environment.

- There is only a reward/supervision signal after each action.
- Feedback is often delayed and not instantaneous.
- Time needs to be taken into account.
- The agent's actions affect the subsequent data it receives.

8.1 Dopamine: Reward Prediction Error

From Schultz (89) [55]: Dopamine neurons are activated by rewarding events that are better than predicted, remain uninfluenced by events that are as good as predicted, and are depressed by events that are worse than predicted. Most dopamine nerons show phasic avtivations ... reward-predicting ... However only few phasic activations follow aversive (causing avoidance of a thing) stimuli. By signaling rewards according to a prediction error, dopamine responses have the formal characteristics of a teaching signal postulated by reinforcement learning theories.

[55] Predictive Reward Signal of Dopamine Neurons

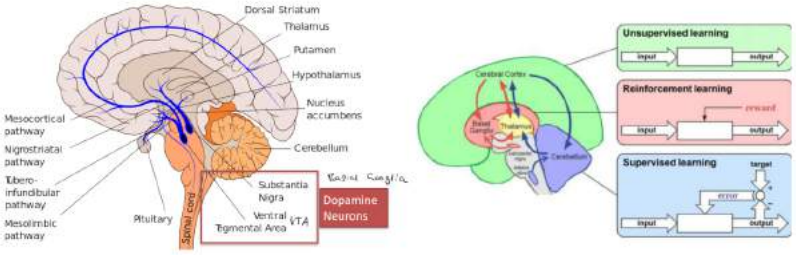


Figure 89: Dopamine neurons of Substantia Nigra and the Ventral Tegmental Area in the Basal Ganglia. (right) Mapping of different machine learning paradigmns to the brain.

If the neocortex mostly performs unsupervised learning why does the VTA strongly project to almost all cortical areas and what is the effect of DA on a cortical neuron?

- Ventral structure does not project to dorsal (where we assume RL happening).
- Gated reinforcement learning: we only want to learn relevant information
- Dopamine is connected to plasticity because it is very sensitive to new / unseen data.

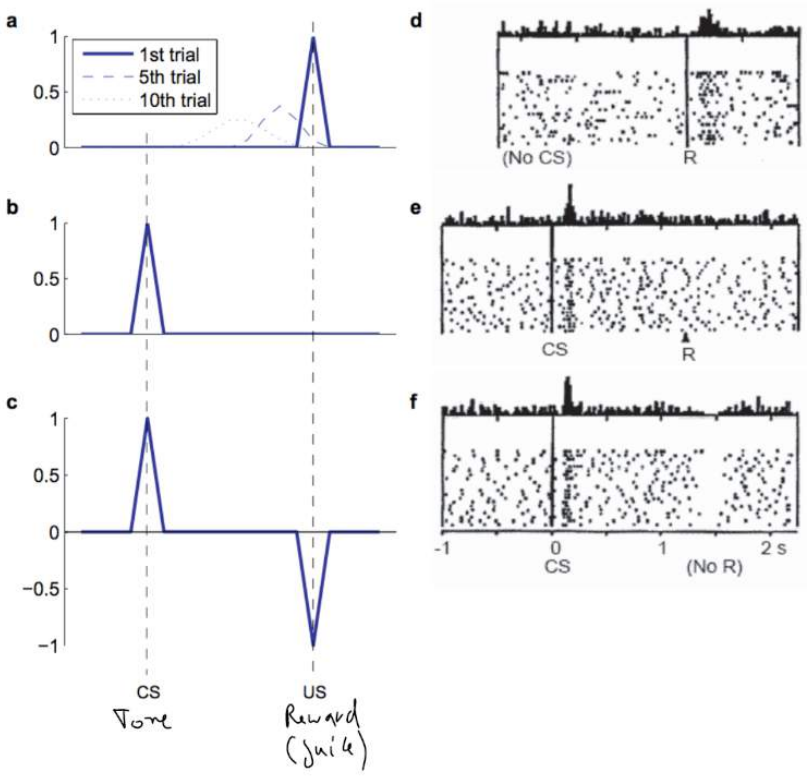


Figure 90: An animal experiment: Dopamine neurons report rewards according to an error in reward prediction. Top: drop of liquid (reward) occurs although no reward is predicted at this time. Middle: conditioned stimulus predicts a reward, and the reward occurs according to the prediction, hence no error in the prediction of reward. Bottom: conditioned stimulus predicts a reward, but the reward fails to occur because of lack of reaction by the animal.CS, conditioned stimulus; R, primary reward.

The predicted reward is further modified by other factors:

- Timing of reward: Across species, it is clear that signals related to predictionerrors are modulated by cues that predict delayed reward. Animals prefer an immediate reward over a delayed reward even when the delayed reward is more economically valuable in the long run. [56]
- Adaption to a new situation: We found that midbrain dopamine neurons rapidly adapted to the infirmation provided by reward-predicting stimuli. Responses shifted relative to the expected reward value, and the gain adjusted to the variance of reward value [57].

[56]Impact of size and delay on neural activity in the rat limbic corticostriatal system
[57]Adaptive Coding of Reward Value by Dopamine Neurons

8.2 Models of Learning Reward Prediction

Even though these models here are called predicting models, this term might mislead you. In statistics, prediction and inference (not the same) can be understood as the process of working out results from available information. In inference it's about the present, in inference about the future. However there is no learning over time involved. Here we look at update rules (designated by $\leftarrow$) which means the system changes over time - it learns. We might connect one of these learning rules to a MDP or RL task to find an optimal behavior function for our agent.

8.2.1 Rescorla Wagner Rule

Model of classical conditioning in which learning is conceptualized in terms of associations between conditioned and unconditioned stimuli. Change in value $V(s_t)$ is proportional to the difference between actual and predicted reward.

$$V(s_t) \leftarrow V(s_t) + \eta[R - V_{total}] \tag{125}$$

where:

- s_t: stimulus
- $V(s_t)$: associative strength of conditioned stimulus s_t
- R: reward
- η: learning rate
- V_{total} sum of associative strengts of all conditioned stimului (including s_t) that are presented on this trial (the n-th trial).
- $|[R - V_{total}]|$ surprise

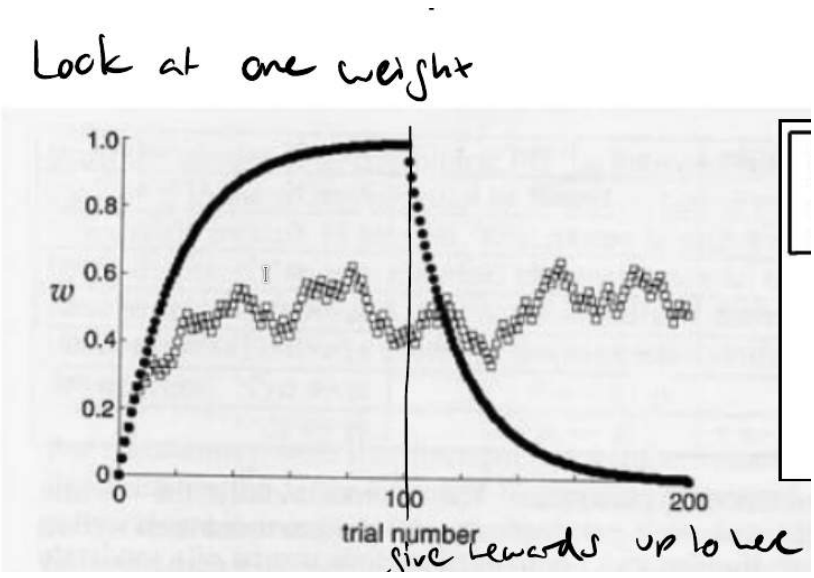


Figure 91: Typical exponential / log behavior of the iterative R-W rule.

8.2.2 Temporal Difference Rule to Q-Learning

The lecture went over these learning rules very quickly and in an unsual order. If you are a beginner, I suggest that you read the sections about MC, MRP and MDP first.

Key Idea of the temporal difference rule (TDR): Update the value of the current state based on the immediate reward and the estimated value of the next state. Interpretation: we must not look at immediate rewards only but future rewards should be taken into consideration as well on a discounted valuation. We assume that the path that our agent takes to navigate the system is given.

$$V(s_t) \leftarrow V(s_t) + \eta[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)] \tag{126}$$

where:

- $V(s_t)$: previous estimate
- r_{t+1} next reward
- $\gamma V(s_{t+1})$ discounted value on the next step

Lets now include the fundamental concept of an action to this equation. This adds one dimension to the value function and gives the agent a choice. This new function is called $Q(s, a)$.

$$Q(s, a) = \mathbb{E}_{(r',s')|s,a}[r + \gamma max_{a'}[Q(s', a')]] \tag{127}$$

By just a few trivial steps one can show that the TD rule is used to get the convex combination in the Q-Learning update rule (different from Q-Function) between old and new Q value seen in the literature:

$$Q(s, a) \leftarrow (1 - \alpha_t)Q(s, a) + \alpha_t(r + \gamma max_{a'}[Q(s', a')]) \tag{128}$$
$$Q(s, a) \leftarrow Q(s, a) - \alpha_t Q(s, a) + \alpha_t r + \gamma \alpha_t max_{a'}[Q(s', a')] \tag{129}$$
$$Q(s, a) \leftarrow Q(s, a) + \alpha_t(r + \gamma max_{a'}[Q(s', a')] - Q(s, a)) \tag{130}$$
$$Q(s, a) \leftarrow Q(s, a) + \alpha_t(r + \gamma max_{a'}[Q(s', a') - Q(s, a)]) \tag{131}$$

Given this rule, we can create and update a map over future states and actions. We can optimize w.r.t the action to get an optimal path (policy). The key idea idea is that we do not need to know any transition probabilities to learn (model), we just need an unbiased estimate from our world (sample). We can get these samples by just playing the "game". If we store the actions a and rewards r from these samples, we can directly apply Q-learning. Thus, Q-learning is considered model-free.

Keep in mind that (in the end), the optimal policy can be deducted from the optimal value function V^*:

$$V^*(x) = max_a Q^*(x, a) \tag{132}$$

8.3 Introduction to Reinforcement Learning

We saw the concept of looking at expected reward and choosing actions to maximize this reward, but the idea was not well embedded into a generalizing concept. Reinforcement learning (RL) exactly puts a name on this framework, which includes Q-Learning as well. RL is about an agent taking suitable action to maximize reward in a particular situation. It is employed by various software and machines to find the best possible behavior or path it should take in a specific situation.

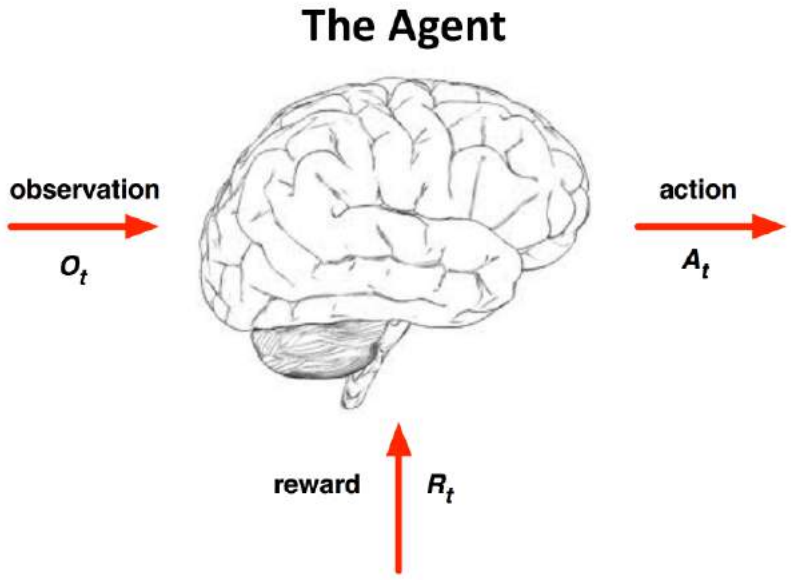


Figure 92: Influences from and to the agent in RL to the surrounding world. At each step t, the agent exectues an action A_t which the environment recieves. The agent recieves an observation O_t of the environment, for example trough a sensor and the agent is rewarded R_t by its behavior from the environment.

Reinforcement learning is based on the reward hypothesis: **All goals of an agent can be described by the maximization of expected cumulative reward.**

Example rewards:

- Fly stunt maneuvers in an RC helicopter (+ following desired trajectory, - crashing)
- Defeat the world champion at Backgammon (+ winning /- loosing)
- Manage an investment portfolio (+ more / - less money)
- Make a humanoid robot walk (+ reward for forward motion / - reward for falling over)
- Play Atari games better than humans (+/- reward for increasing/decreasing score)

The fact that reward presented to the agent is not always immediate leads to the exploration / exploitation dilemma. (Example: A financial investment may take months to mature). An agent does not intrinsically know the future implications of its actions, or (how actions interact with) dynamics of the environment.

At each point in time, the agent is in a state. because this information state is all that is necessary to fully determine the agent, it is also said that the state is markovian. This means we can throw away the histrory.

$$P(S_{t+1}|S_t) = P(S_{t+1}|S_{1:t}) \tag{133}$$

An RL agent may compute different functions on top of its state.

8.3.1 Policy

The agents behavior function callend policy maps from state s to action a. The policy may be stochastic or deterministic:

$$a = \pi(s) \tag{134}$$
$$\pi(a|s) = P(A_t = a|S_t = s) \tag{135}$$

8.3.2 Value Function

Value function is a prediction of future rewards and does so by assigning a number to every state s. It depends on a policy π to determine where the agent could go and a probability distribution $p(X'|\pi(X), X)$.

We compute this value as expectation over the joint distribution:

$$p(S_1, S_2, S_3, \dots) \tag{136}$$

The value function is defined as:

$$v_\pi = \mathbb{E}[r(s_0, \pi(s_0)) + \gamma r(s_1, \pi(s_1)) + \gamma^2 r(s_2, \pi(s_2)) + \dots] \tag{137}$$

But in the lecture, they mentioned (and it is not distinguished between the two) the conditional value function:

$$v_\pi(s_0) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r(s_t, \pi(s_t))|S_0 = s_0] \tag{138}$$

Which can be rewritten to the recursive equation (often in the literature):

$$v_\pi(s) = r(s_0, \pi(s_0)) + \gamma \sum_{s_1} p(S_1 = s_1|S_0 = s_0, \pi(s_0)) v_\pi(s_1) \tag{139}$$

8.3.3 Model

From the previous section we see that we require a probability distribution that depends on direct actions a or a policy returning actions π:

$$P_{s_t, s_{t+1}} = p(S_{t+1} = s_{t+1}|S_t = s_t, \pi(s_t)) \tag{140}$$
$$= p(S_{t+1} = s_{t+1}|S_t = s_t, A_t = a_t) \tag{141}$$

that predicts the next (immediate) reward:

$$R_{s_t} = \mathbb{E}[R_{t+1}|S_t = t, A_t = a_t] \tag{142}$$

This is called the model. Model free RL uses tricks to not compute / require this distribution. As it is often intractable. Imagine the state space being the input of a video game!

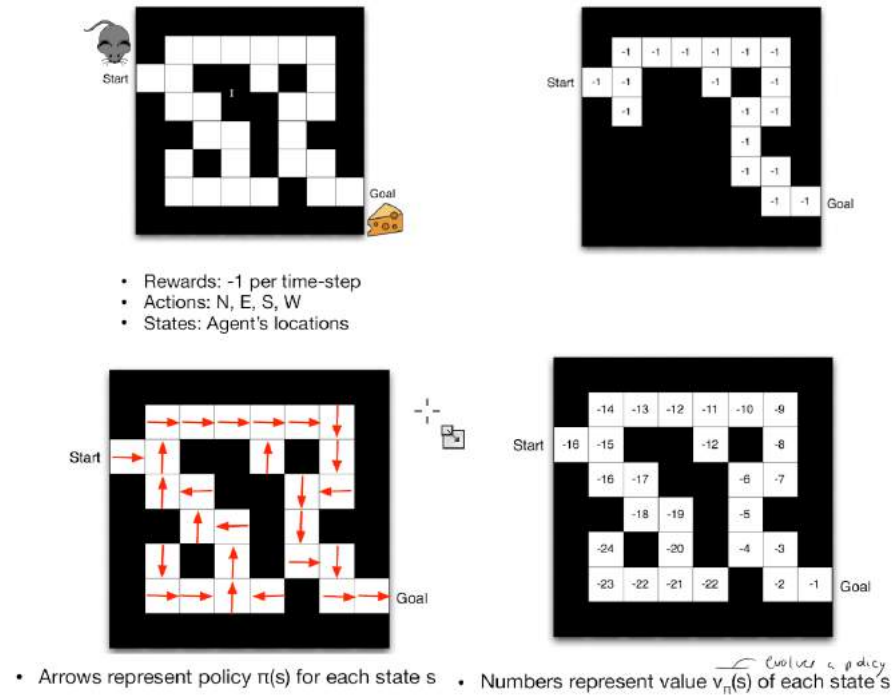


Figure 93: A small example of a mice showing all the agent related components together with some numbers. (top left) Actions, start, end and definition of other states (the maze). (top right) immediate rewards. (bottom) Policy and value function for each state.

Some details to Figure section 8.3.3:

- Agent may have an internal model of the environment.
- Dynamics: How actions change the state.
- Rewards: How much reward from each state.
- The model may be imperfect.
- Grid layout represents transition model $P^a_{s_t,s_{t+1}}$
- Numbers (top right) represent immediate reward $R^a_{s_t}$ from each state s.

We've seen different RL-subtypes that need to be distinguished. Comment: From the Bellman Theorem we know that every value function induces a policy and every policy induces a value function:

- Value Based
  - No Policy (Implicitly given by value function, see Bellman Eq.)
  - Value Function
- Policy Based
  - Policy
  - No Value Function (Even tho values can be computed off a policy)

Model Free vs model based: We can have a policy and/or a value function in each case, but we distinguish by the need for the model distribution.

8.3.4 A more sophisiticated Example: Atari

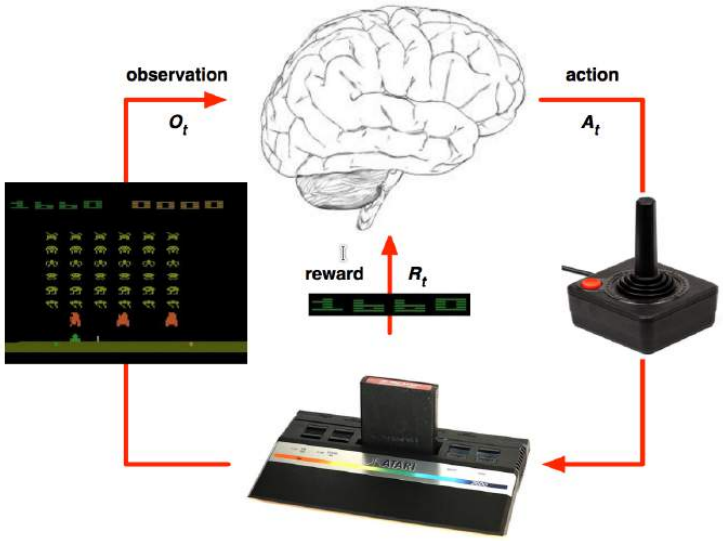


Figure 94: The Atari video gaming platform provides and ideal environment to test RL / Planning strategies. For some games, we don't know the rules and apply RL. This means we learn directly from interactive gameplay. Pick actions on a joystick and observe the pixels. For other games we know the rules. This allows to apply planning strategies where we might ask ourselves: What would the next state be? What would the score be?. We can query the future by tree search to some extent.

8.4 Planning

Even though planning appears later in the lecture it is actually the logical step before we arrive at RL. It is a more constrained view where a model of the environment is known. The agent performs computations with its model (without any external interaction). This is a simplification compared to RL where the environment is initially unknown and the agent may only discover it

by interacting with the environment. In RL, the agent improves its policy or value function.

If the agent/solver has access to the model, i.e $p(s'|s,a)$ and $r(s,a)$, and it employs it when optimizing the MDP, then we are in the planning settings (or dynamic programming , DP, setting). Otherwise, we are in the RL settings. Of course, sometimes, even though we have access to the model, still we do RL since it is hard to solve directly the MDP, and we prefer to interact with the MDP rather then solving it, i.e., we ignore the model. Remark: I think this is true for most games and therefore, the Atari example might be confusing.

In a planning scenario, we can query the future trough the emulator. We can therfore play / plan ahead to find the optimal policy by tree search. As already mentioned, this might not be possible even for simple games (check out the AlphaGo strategies).

RL on the other hand can be as well refered to as "trial-and-error" learning. However, obviously we try to guide the agent to loose the least amount of reward that is possible.

8.5 Exploration / Exploitation

Everyone is confronted with the same dilemma on a daily basis: should I keep doing what I do, or should I try something else. For example should I go to my preferred restaurant or should I try a new one, should I keep my current job or should I find a new one, etc...

In Reinforcement Learning, this type of decision is called exploitation when you keep doing what you were doing, and exploration when you try something new.

- Exploration finds more information about the environment.
- Exploitation exploits known information to maximize reward. It is usually important to explore as well as exploit

8.6 Basics

Remark: In my opinion these chapters build the foundation for RL but in the lecture they appear after RL and thus I kept that order. If you are a beginner to these topics, I highly recommend to gain some basic knowledge about probabilistic graphical models (PGM) (Bayesian networks) first. They are used from here on, but were not introduced explicitly in the lecture.

8.6.1 Markov Chain (MC)

A Bayesian network is a kind of PGM that uses a directed (acyclic) graph to represent a factorized probability distribution and associated conditional independence over a set of variables. Definition: A state S_t is Markov if and only if:

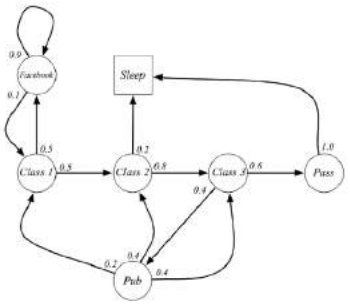
$$P(S_{t+1}|S_t) = P(S_{t+1}|S_{1:t}) \tag{143}$$

where s_t is the current state and s_{t+1} is the successor state.

The state captures all relevant information from the history. Once the state is known, the history may be thrown away. This means that the state is a sufficient statistic of the future.

The state transition matrix P defines transition probabilities from all states $m = s_t$ to all successor states $n = s_{t+1}$. :

$$P_{m,n} = \begin{pmatrix} P_{1,1} & P_{1,2} & \cdots & P_{1,n} \\ P_{2,1} & P_{2,2} & \cdots & P_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ P_{m,1} & P_{m,2} & \cdots & P_{m,n} \end{pmatrix}$$



$\mathcal{P}$ =	C1	C2	C3	Pass	Pub	FB	Sleep
C1		0.5				0.5	
C2			0.8				0.2
C3				0.6	0.4		
Pass							1.0
Pub	0.2	0.4	0.4				
FB	0.1					0.9	
Sleep							1

- C1 C2 C3 Pass Sleep
- C1 FB FB C1 C2 Sleep
- C1 C2 C3 Pub C2 C3 Pass Sleep
- C1 FB FB C1 C2 C3 Pub C1 FB FB FB C1 C2 C3 Pub C2 Sleep

Figure 95: (left) An example Markov chain showing all state transition probabilites next to its nodes. (right) The corresponding state transition matrix and results of a sampling procedure applied to this Markov chain. Be aware that this is not a PGM, the nodes are not random variables.

8.6.2 Markov Reward Process (MRP)

A Markov reward process is a stochastic process which extends either a Markov chainby adding a reward rate to each state.

Definition: A Markov Reward Process is a tuple (S, P, R, γ) where S is a finite set of states, P is a state transition probability matrix $P_{t,t+1} = P(S_{t+1}|S_t)$, $R = \mathbb{E}[R_{t+1}|S_t = s]$ is a reward function and γ is a discount in the interval $(0, 1)$

Facts:

- Mathematically convenient to discount rewards. It avoids infinite returns in cyclic Markov processes.
- Uncertainty about the future may not be fully represented.
- If the reward is financial, immediate rewards may earn more interest than delayed rewards.
- Animal/human behavior shows preference for immediate rewards.
- It is sometimes possible to use un-discounted Markov reward processes (i.e. $\gamma = 1$), e.g. if all sequences terminate.

We call G_t the return which is the total discounted reward from time step t onward.

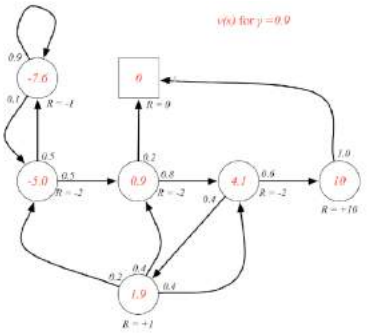
$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{144}$$

If we just look at G_t we must assume that we know the chain of events that lead to the specific rewards R, however the MRP is a stochastic process. Therefore we may compute the conditional value function given that we know where we start (s_t). We've seen this function before in the RL chapter:

$$v_{s_t} = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots | S_t = s_t] \tag{145}$$

which is just

$$v_{s_t} = \mathbb{E}[G_t | S_t = s_t] \tag{146}$$



C1 C2 C3 Pass Sleep	$v_1 = -2 - 2 * \frac{1}{2} - 2 * \frac{1}{4} + 10 * \frac{1}{8}$	=	−2.25
C1 FB FB C1 C2 Sleep	$v_1 = -2 - 1 * \frac{1}{2} - 1 * \frac{1}{4} - 2 * \frac{1}{8} - 2 * \frac{1}{16}$	=	−3.125
C1 C2 C3 Pub C2 C3 Pass Sleep	$v_1 = -2 - 2 * \frac{1}{2} - 2 * \frac{1}{4} + 1 * \frac{1}{8} - 2 * \frac{1}{16} \ldots$	=	−3.41
C1 FB FB C1 C2 C3 Pub C1 ... FB FB FB C1 C2 C3 Pub C2 Sleep	$v_1 = -2 - 1 * \frac{1}{2} - 1 * \frac{1}{4} - 2 * \frac{1}{8} - 2 * \frac{1}{16} \ldots$	=	−3.20

Figure 96: The MC of **??** extended to be a MRP. We may compute the random samples return.

8.6.3 Markov Decision Process (MDP)

Markov decision process (MDP) is a Markov reward process with decisions (actions that we can take). It is still an environment in which all states are Markov. Definition: A Markov Reward Process is a tuple (S, A, P, R, γ) where S is a finite set of states, A is a finite set of actions, P is a state transition probability matrix $P_{t,t+1} = P(S_{t+1}|S_t, A_t)$, $R = \mathbb{E}[R_{t+1}|S_t = s, A_t = a]$ is a reward function and γ is a discount in the interval $(0, 1)$

Markov decision processes formally describe an environment for reinforcement learning where the environment is fully observable. Almost all RL problems can be formalized as MDPs:

- Partially observable problems can be converted into MDPs.

- One Armed Bandits are MDPs with one state.

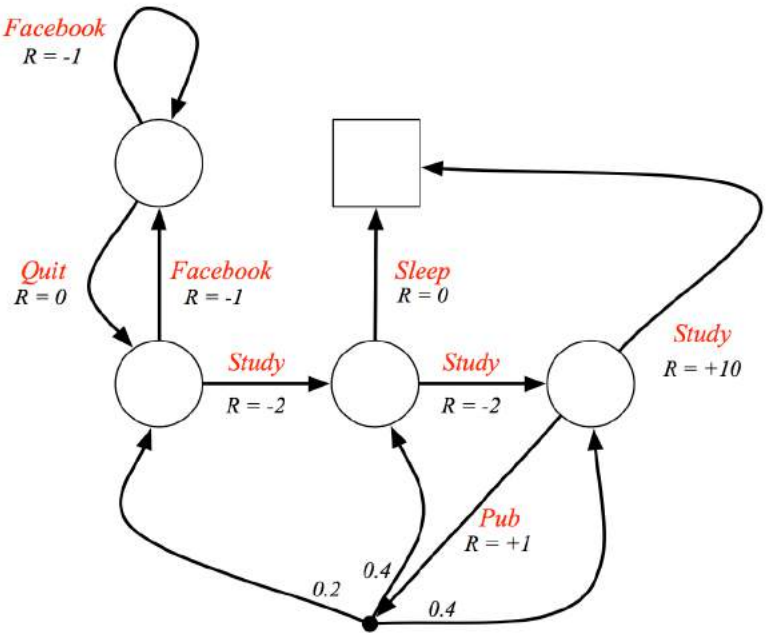


Figure 97: The MC of section 8.6.4 extended to be a MRP extended again to be an MDP. However we should be careful with the comparisons. In the MRP, our agent was guided purely by randomness and we had no choice. Now, the agent is able to directly influence its path. Note that some of the transitions are deterministic. For example if we quit facebook, we are for sure back to studying, however if were in the pup, we might be too drunk and randomness influences the outcome.

One can see that we are very close now to what we introduced in the RL section, however one key piece is missing. Given that we have choice as an agent now, how do we know the optimal behavior?

8.6.4 Bellman Equation

The Bellman equation (BE), named after Richard E. Bellman, is a necessary condition for optimality associated with the mathematical optimization method known as dynamic programming (aka. RL where the environment is known $\longrightarrow$ MDP). It allows us to "solve" the MDP problem, the problem of not knowing how to act in an environment where we are able to take action.

BE in MRP In the most simple case, we can just evaluate the Bellman expectation equation in an MRP where we have no policy to optimize.

$$
\begin{aligned}
v_{s_t} &= \mathbb{E}[G_t|S_t = s_t] && (147)\\
&= \mathbb{E}[R_{s_{t+1}} + \gamma R_{s_{t+2}} + \gamma^2 R_{s_{t+3}} + \dots |S_t = s_t] && (148)\\
&= \mathbb{E}[R_{s_{t+1}} + \gamma(R_{s_{t+2}} + \gamma R_{s_{t+3}} + \dots)|S_t = s_t] && (149)\\
&= \mathbb{E}[R_{s_{t+1}} + \gamma(G_{t+1})|S_t = s_t] && (150)\\
&= \mathbb{E}[R_{s_{t+1}} + \gamma v(S_{t+1})|S_t = s_t] && (151)\\
& && (152)
\end{aligned}
$$

This is the immediate reward R_{t+1} plus the discounted value of the successor state $\gamma v(S_{t+1})$. This is again close to the temporal difference rule. If we move the reward from the next state to the current one (by definition), we can pull R_{s_t} out of it and compute the expected value trough the sum and the equation becomes:

$$
v_{s_t} = R_{s_t} + \gamma \sum_{s_{t+1} \in S} P_{s_t, s_{t+1}} v(s_{t+1}) \tag{153}
$$

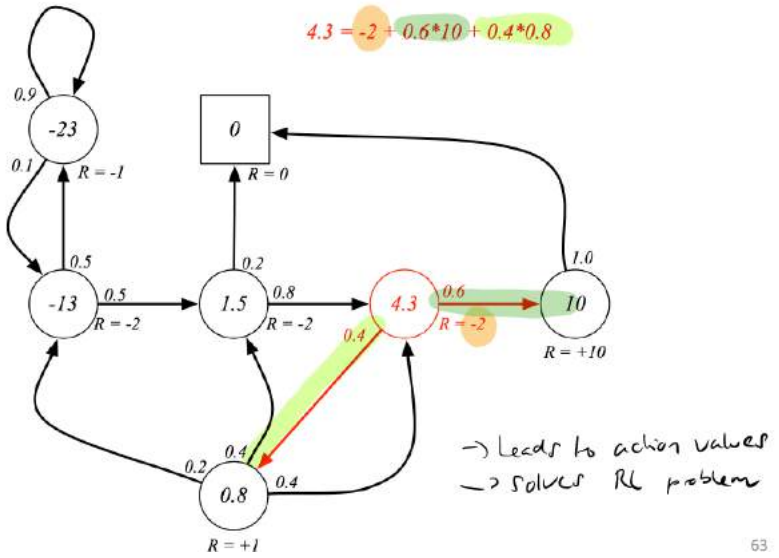


Figure 98: Example computation of the value of the red node in an MRP.

One might argue that this leads to action values by choosing the next state based on its value "score". Also this can be solved explicitly as a linear system.

BE in MDP In MDP, we must somehow include the actions. Over the entire task, the actions we take are defined to be the policy. Since we can optimize this policy, one might ask how to do this using the Bellman optimality equation.

- The state-value function $v_\pi(s)$ of an MDP is the expected return starting from state s, and then following policy π.

- The action-value function $q_\pi(s, a)$ is the expected return starting from state s, taking action a, and then following policy π.

We may rewrite both in a similar maner as we did in the MRP case.

$$v_\pi(s_t) = \mathbb{E}_\pi[G_t|S_t = s_t] \tag{154}$$
$$= \mathbb{E}_\pi[R_{s_{t+1}} + \gamma v_\pi(S_{t+1})|S_t = s_t] \tag{155}$$
$$\tag{156}$$

And if we again shift the reward ($\stackrel{!}{=}$):

$$q_\pi(s_t, a_t) = \mathbb{E}_\pi[G_t|S_t = s_t, A_t = a_t] \tag{157}$$
$$= \mathbb{E}_\pi[R_{s_{t+1}} + \gamma q_\pi(S_{t+1}, A_{t+1})|S_t = s, A_t = a] \tag{158}$$
$$\stackrel{!}{=} R^{a_t}_{s_t} + \gamma \sum_{s_{t+1} \in S} P^{a_t}_{s_t, s_{t+1}} v(s_{t+1}) \tag{159}$$
$$= R^{a_t}_{s_t} + \gamma \sum_{s_{t+1} \in S} P^{a_t}_{s_t, s_{t+1}} \sum_{a_{t+1} \in A} \pi(a_{t+1}|s_{t+1}) q_\pi(s_{t+1}, a_{t+1}) \tag{160}$$

As you can see, the notation becomes quite tedious. From now on we use $S_t = s$ and $S_{t+1} = s'$. The same applies to actions.

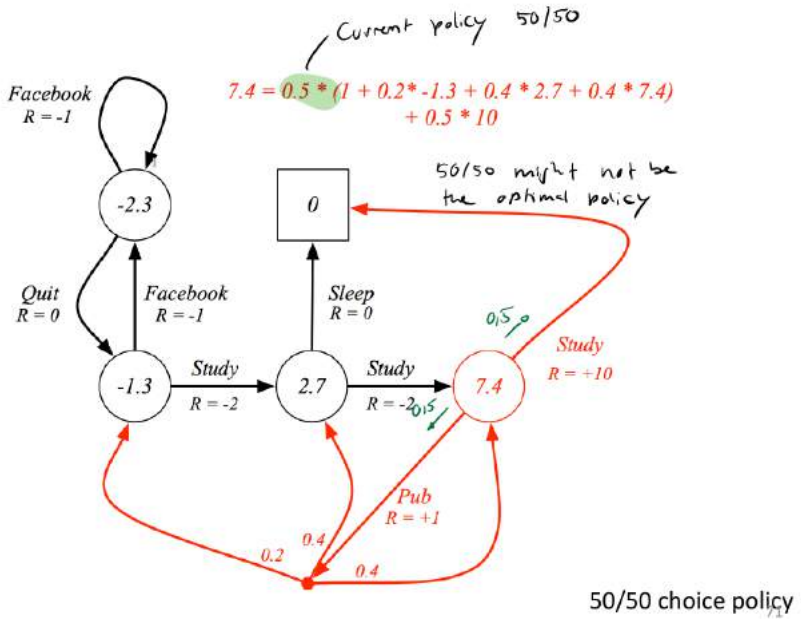


Figure 99: Example of policy based computation of the value of the red node in an MDP.

Finding the optimal action-value (Q) function
We can now define the optimal value function:

$$v^*(s) = max(v_\pi(s)) \tag{161}$$

and the optimal action-value (q) function:

$$q^*(s,a) = max(q_\pi(s,a)) \tag{162}$$

You may noticed that we depend on the policy for $v^*(s)$ and $q^*(s,a)$. An optimal policy can be found by maximizing over the optimal Q-function $q^*(s,a)$:

$$\pi^*(a|s) = \begin{cases} 1, & \text{if } a = argmax_{a \in A}(q^*(s,a)) \\ 0, & \text{otherwise} \end{cases}$$

There is always a deterministic optimal policy for any MDP. If we know $q^*(s,a)$, we immediately have the optimal policy.

In the last step, this theorem allows us to assemble the Bellman optimality update equations:

We now use the optimal value function ($v^*(s) = max(v_\pi(s))$) to get the optimal Q-function:

$$q^*_\pi(s,a) = R^a_t + \gamma \sum_{s' \in S} P^a_{s,s'} v^*(s') \tag{163}$$

Note that the agent has to average, because we can't choose. This is decided by the environment. Finally we may write down the Bellman optimality equations for v^* and q^*:

$$v^*(s) \leftarrow max_a[R^a_s + \gamma \sum_{s' \in S} P^a_{s,s'} v^*(s')] \tag{164}$$

$$q^*(s,a) \leftarrow R^a_s + \gamma \sum_{s' \in S} P^a_{s,s'} max_{a'}[q^*(s',a')] \tag{165}$$

Bellman figured out that our policy being optimal means that we can be greedy w.r.t these optimality equations. If we update over and over again, we will converge to a fixed point which is the optimal policy.

8.7 Deep Reinforcement Learning

In the previous sections, distributions were always treated as discrete tables. This is not possible for large state/action spaces. Therefore, functional approximations to these functions must be found. We've already seen that deep neural networks (DNN) are function approximators in their nature. One can parametrize a policy or Q-function and use DNN to estimate its parameters directly from state space.

For optimization purposes, we define a loss function. This loss follows from moving Q inside of expectation.

$$L = \frac{1}{2}[r + max_{a'} Q(s',a') - Q(s,a)]^2 \tag{166}$$

We find the optimal action-value function Q^* by parametrizing it with θ (make it DNN compatible).

$$Q^*(s,a) = max_\pi \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots | s_t = s, a_t = a, \pi] \tag{167}$$

and then minimize the loss w.r.t θ. Comment: Often, we approximate $\mathbb{E}$ by sampling.

$$L_i(\theta_i) = \mathbb{E}_{s,a,r,s' \ U(D)}[r + \gamma max_{a'} Q(s', a'|\theta_i^-) - Q(s, a|\theta_i)] \tag{168}$$

This is again the temporal difference rule.

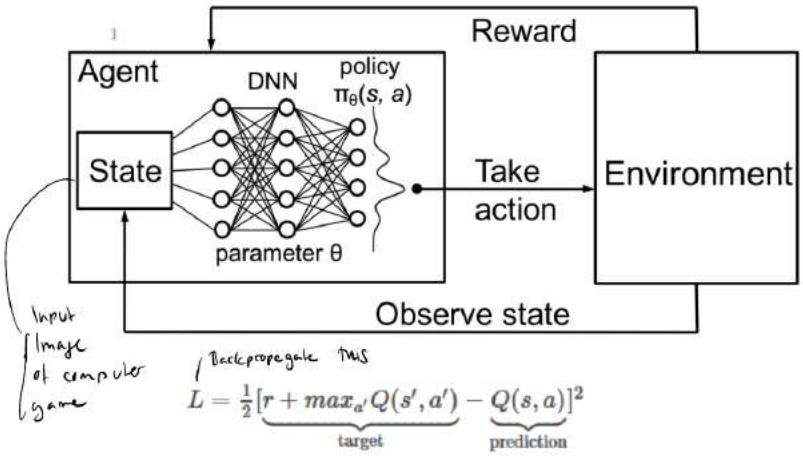


Figure 100: Main building blocks of a deep RL system.

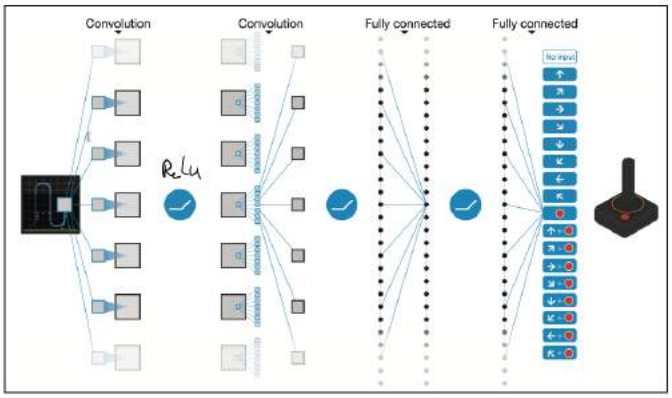


Figure 101: A deep RL system trained directly from input (Atari game video output) to actions of the controlling joystick.

9 Why Spikes

Preface- This chapter (as most can tell by the title) deals with the advantages and disadvantages of using spikes (action potentials) in order to convey information. Traditional hardware computing uses binary 'steps': 0 to 1, GND to VND (strictly speaking the rising edge and falling edges of a step) to transfer information and this chapter will demonstrate why biological spikes are superior.

9.1 What is a Neuronal Spike?

By now most readers should be aware of how an action potential is generated and propagates through a neuron,but here is a short recap.

action potential.jpg action potential.jpg

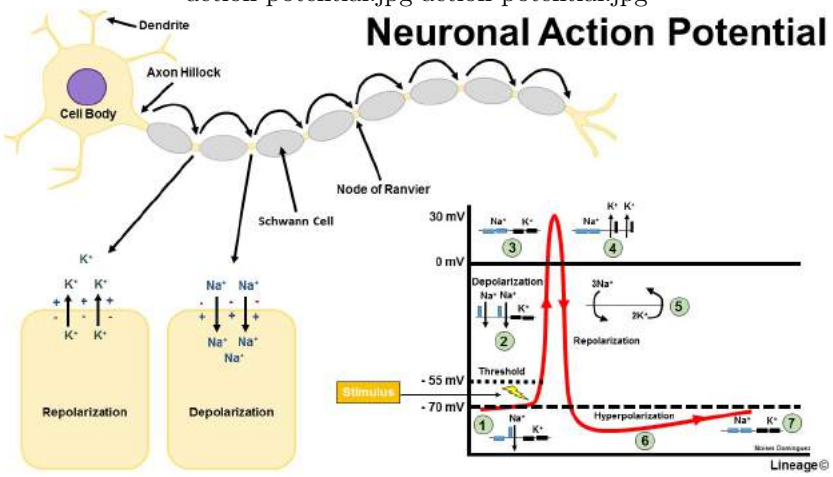


Figure 102: Quick Action Potential Diagram.

- Neuron is at resting potential (-70mV) which is determined by the balance between extracellular and intracellula K+, Na+. and Cl- concentrations (see later section for Hodgkin Huxley model)
- Neuron receives inputs at its dendrites which raises the membrane potential. If the membrane potential is below the threshold potential (55mV), no AP is generated. If the resulting membrane potential is above the threshold, cell depolarization occurs.
- Voltage gated Na+ channels open causing Na+ influx into the cell which raises the membrane voltage which forces neighboring Na+ channels to also open. This continues up until a peak membrane potential of 50mV.
- The Na+ channels begin to deactivate and voltage gated K+ channels begin to open causing K+ ion efflux. This rapidly lowers the membrane potential to below the baseline/resting potential levels (-80 -90 mV) which is called hyper-polarization.

  During this stage, the refractory period, the neuron is unable to fire again while the original ion balance/concentration is reestablished via Na+/K+ ATPases.

9.2 Why does a neuron spike?

9.2.1 (Dis)advantages of digital (spike) vs. non-digital communication

There are several disadvantages to sending digital signals along a channel. One of them is quantization loss:

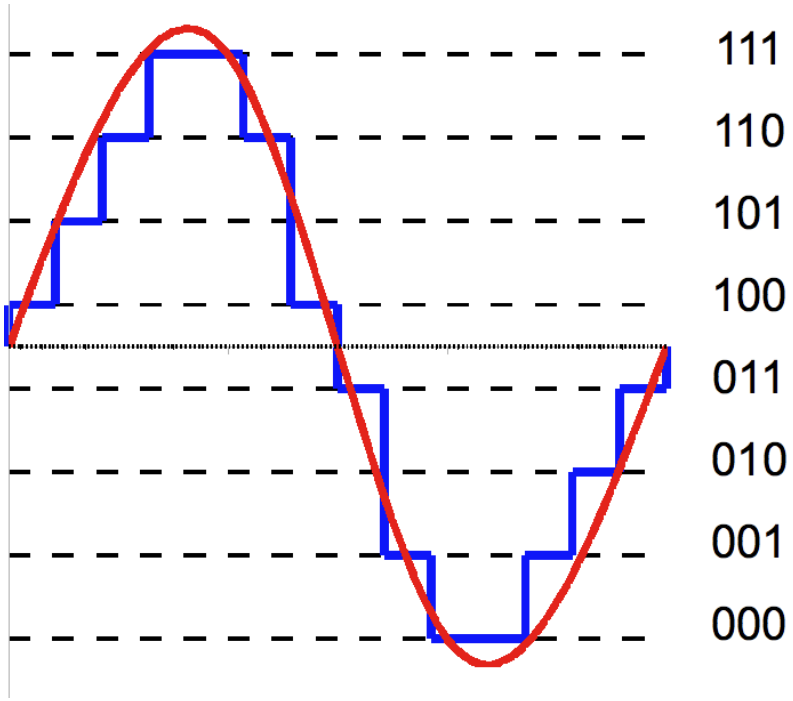


Figure 103: Quantization error

As clearly seen- the output waveform (blue) is not perfectly in line with the original analogue input waveform (red). The calculation to determine quanta is straightforward.

As an aside- nonlinear quantization techniques and schemas exist. One can take advantage of alternative quantization techniques (below) to digitize an analogue signal that has a different entropy, or more information concentrated at lower values at the cost of reducing the resolution in higher values.

The most striking argument for Analogue activity is giving by Shannon's Information Capacity. A calculation done for Crab neurons demonstrates that an analogue channel transfers 2,500 6,000 bits/s while a digital channel would have 50 220 bits/s: an order of magnitude inferior to an analogue channel

9.2.2 Analogue communication in the Retina, C. Elegans, Locust

Not every single neuron of every single creature is digital and spiking. There are several examples of species (C. Elegans, Cockroaches, Locusts) and even neuron families in mammals (Photo receptors, Horizontal cells, Olfactory Granule Cells) where neurons don't utilize spiking.

An entire textbook (Neurones without Impulses- their significance for vertebrate and invertebrate nervous systems- Alan Roberts) exists to cover these example, but the lecture focuses on 1 case: C. Elegans.

C. Elegans has 302 neurons. None of them spike in the traditional AP generating manner.

However recently, a group found spiking-"like" activity in an AWA (olfactory sensory) neuron type.

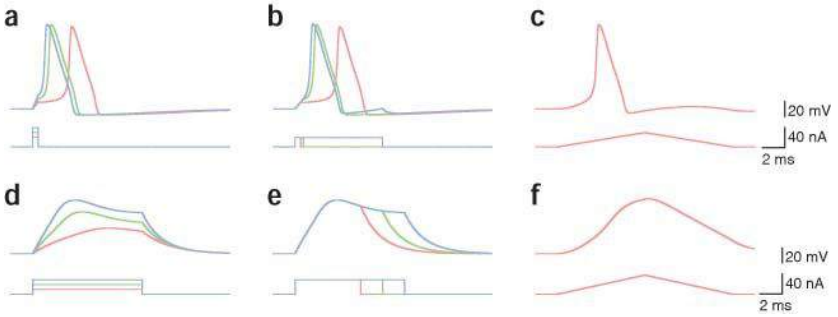


Figure 104: Spiking LIKE activity in C elegans

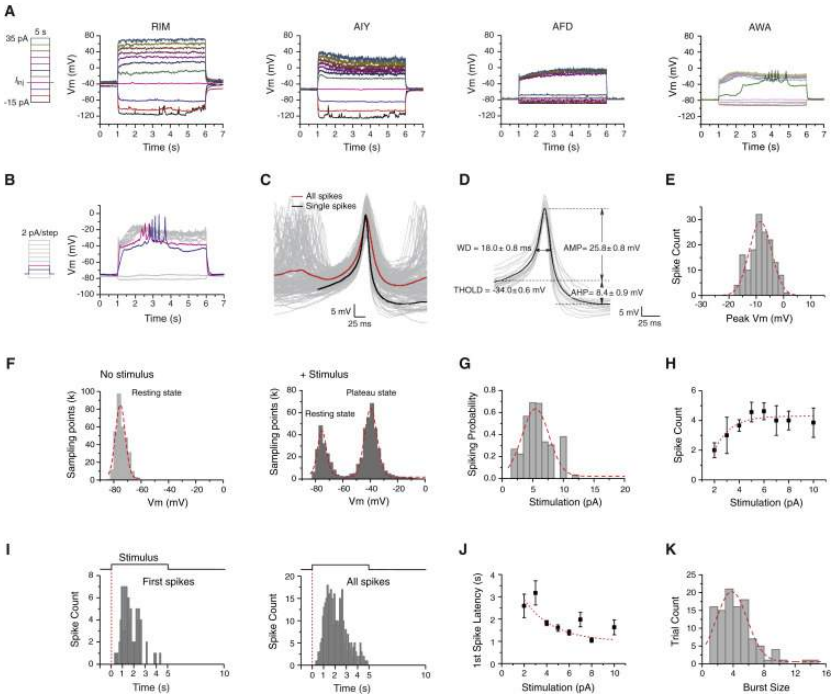


Figure 105: Firing types in C elegans

After a series of back-and-forth arguments, this evidence proved to be inconclusive and the scientific consensus remains set on the fact that C Elegans neurons do not generate action potentials, but the resulting "graded" potentials are quite interesting.

9.2.3 The different types of Action Potentials

9.3 Why Spike- from Biology?

It has been demonstrated from a signal processing standpoint that digital spikes are inferior to analogue communications. It has also been demonstrated that there are insects that work perfectly well without spikes. So why use spikes?

Several reasons:

- Spikes Synchronize Internal Processes -A paramecium (single celled organism) uses spikes to forcefully/instantly orient its motile cilia (motors) as part of avoidance behavior
- Spikes send local information within a Cell -The ameoba uses Mechanosensetive Calcium channels to generate a spike which causes local contraction/-compression to generate movement away from an object.
- Calcium spikes regulates homeostatic processes -Ca2+ intracellular signaling cascade/pathways are the foundations of molecular biology. Seeing Ca2+ influx follows an action potential, it would be wasteful to have to come up with a different mechanism to mediate Ca2+ levels. Spiking does this as a byproduct.
- Cells send information across long distances -Any analogue wavelength would decay over time, and be vulnurable to noise fluctuations. Neurons have tricks to reduce this (myelination, increasing axon diameter in sea squids), but ultimately spikes guarantee an intact and reconstructable signal being delivered.
- Energy Efficiency -Self Explanatory. A human uses 100 Watts, the brain takes up 20W. Compare that to the power supply unit of a standard desktop (300W) or a high end rig (800W)

9.4 How to measure spiking activity in a biological neuron?

How would one measure a voltage level change of anything? Use a voltmeter!

The only caveat to this is probe design. Simply placing the tip of the probe in contact with the membrane is usually insufficient (especially at the 1 2 μmeter level), thus there are several techniques to properly "clamp" the membrane.

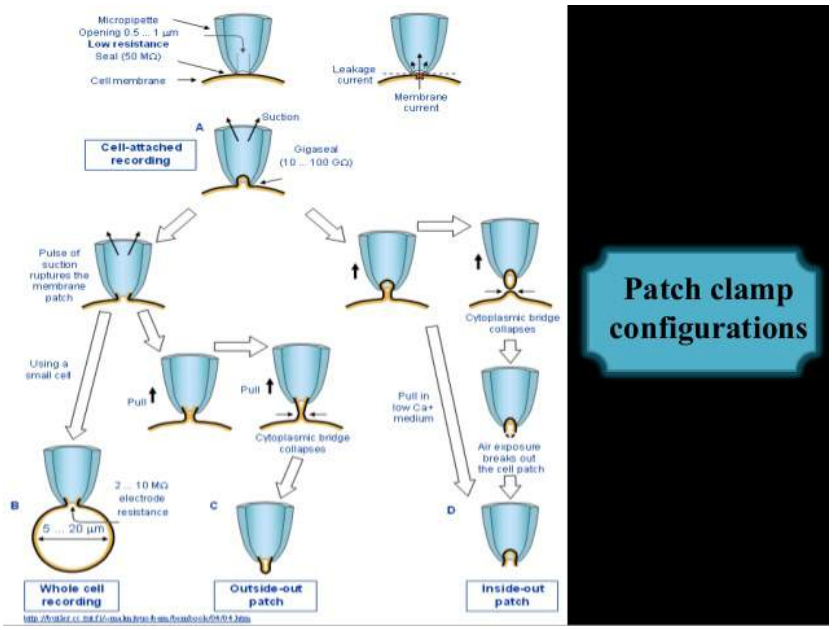


Figure 106: Patch Clamp configurations

Electrode design is actually quite a hardware challenge for electrical engineers and improvements are being made every year.

Alternatively Neuronal Spiking can be recorded via fluoresence techniques (Voltage Indicators)- Ace1Q-mNeon & Ace2N-mNeon are examples.

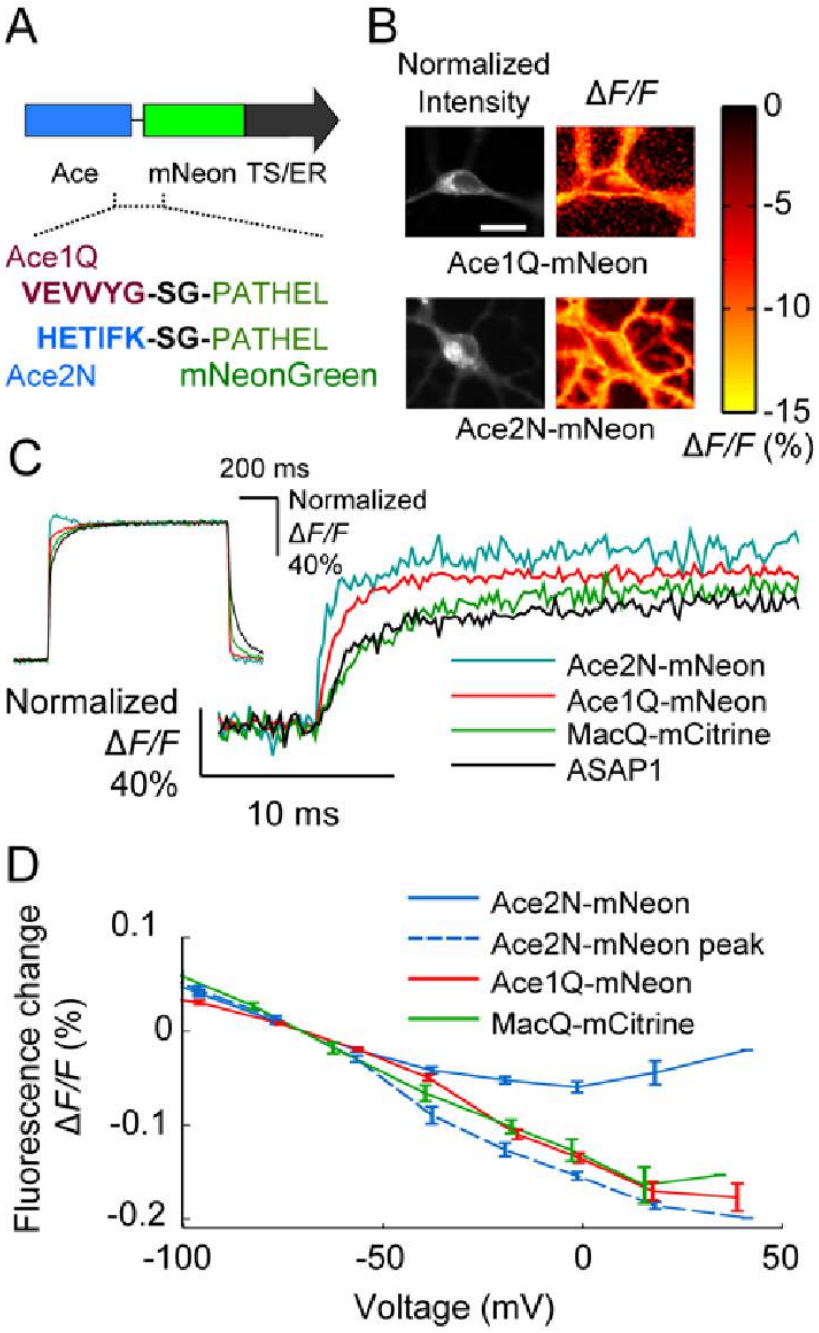


Figure 107: Voltage Indicators

GEVIs undergo a conformations change in response to a voltage change which changes their fluorescence levels. Several Examples are given

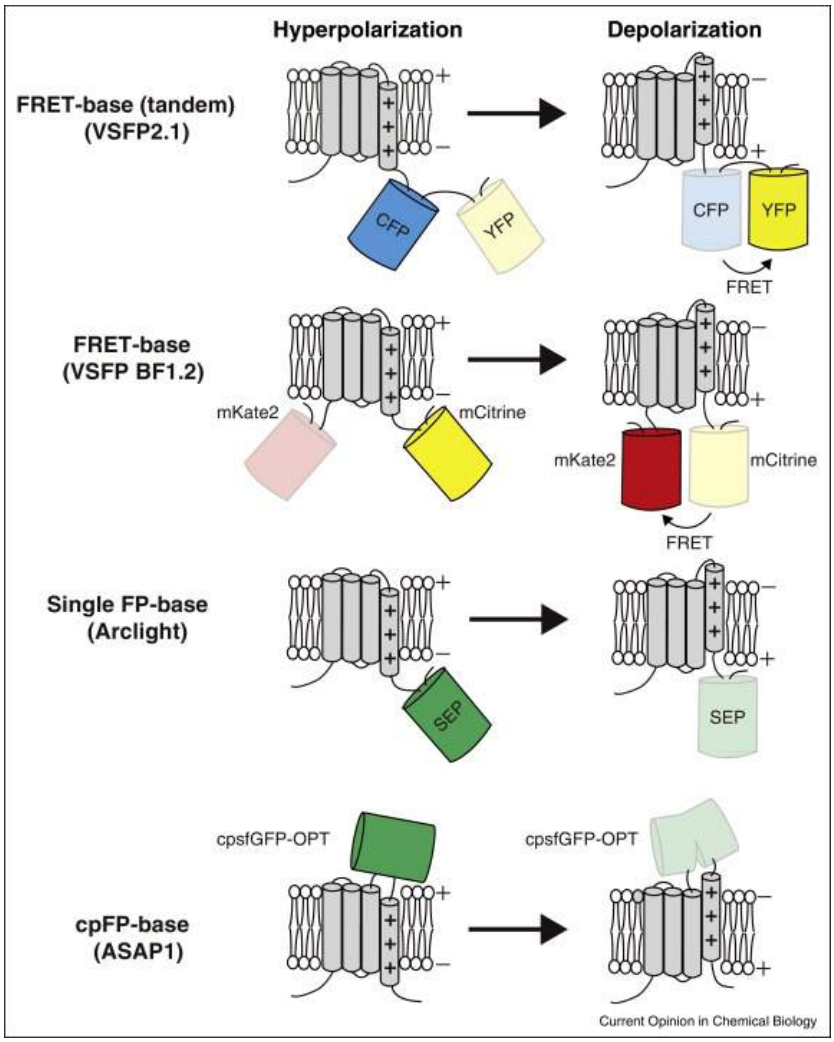


Figure 108: More Voltage Indicators

Finally there are Calcium Indicators that do exactly the same thing but undergo a conformational shift in response to a change in intracellular calcium concentration.

9.5 Temporal coding schemes with spikes

Now that we've established that spikes are the method of information transportation in the brain, it is necessary to find a way to decode them.

There are several ways in which spiking data can encode information.

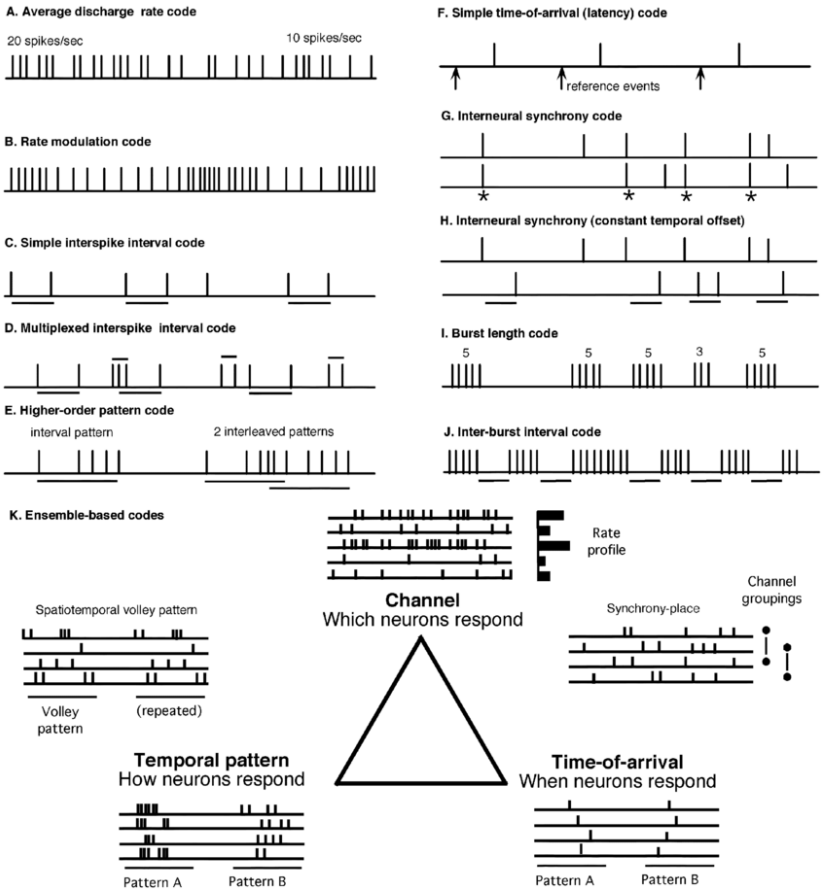


Figure 109: Types of Spike Coding

These can be divided into 2 primary categories: Rate coding (information = firing rate) and Temporal coding (information = time scheme)

The lecture gives several examples

- Spike RATE coding in the Cat VI encodes direction selectivity
- Phase Coding in Place Cells in the Hippocampus to determine position
- Temporal (Latency) Coding to determine Interaural time difference

9.5.1 Deep Learning with "Time to first Spike"

An Artifical Spiking Neural Network has been trained to utilize the time to first spike scheme.

The (MNIST) 2D image information was encoded in the following manner

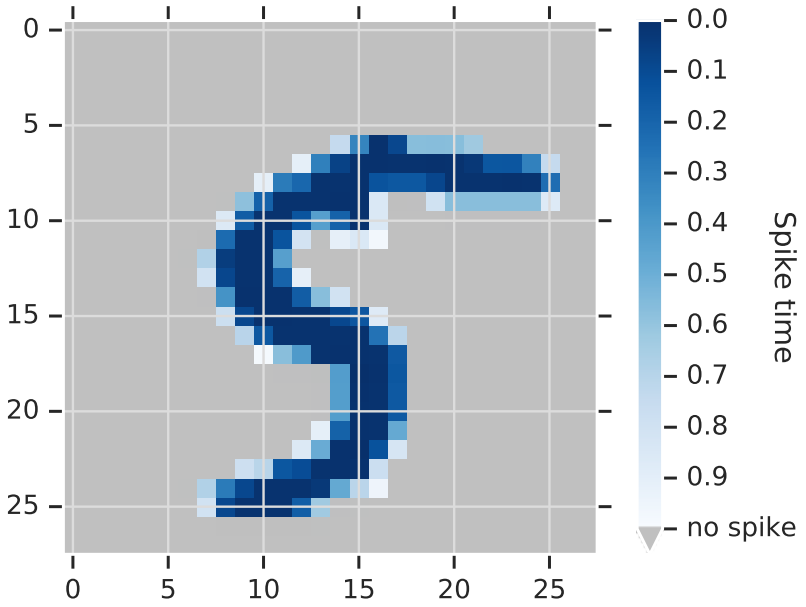


Figure 110: TTFS Encoding of MNIST image

This paper modelled spikes via the Alpha Function

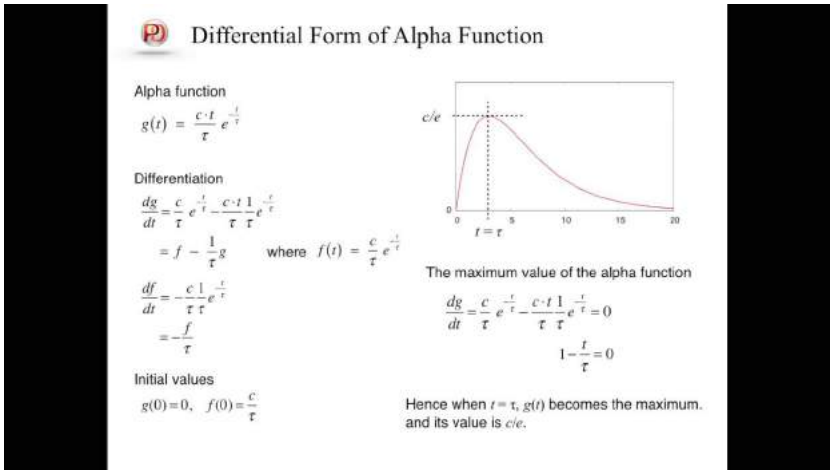


Figure 111: Alpha Function

And it can be trained to solve Boolean tasks (AND, OR, XOR), as well as MNIST classification

9.5.2 Hodgkin-Huxley Model

Mathematically modelling neuronal firing can be done in different ways. The classic model is the Hodgkin-Huxley Model.

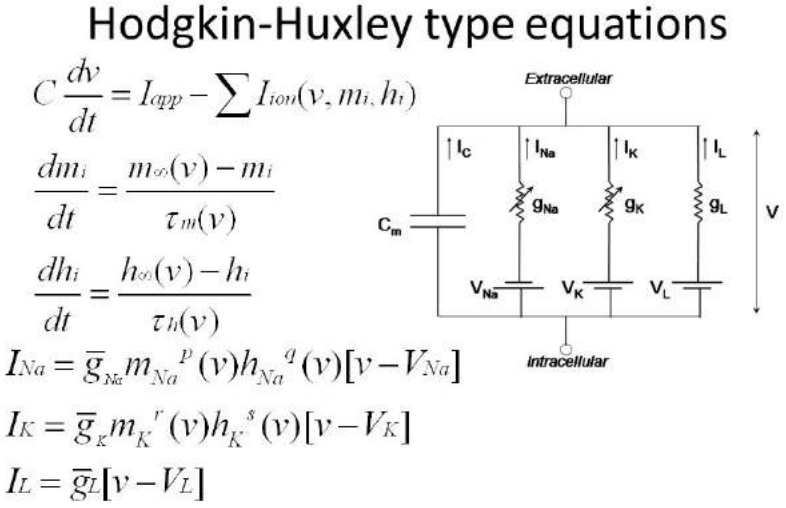


Figure 112: Hodgkin-Huxley

However multiple other models (Fitzhugh-Nagumo, Leaky Integrate & Fire, Galves–Löcherbach, HTM models) exist and can be derived.

Keep in mind that these attempt to biologically model neurons, and not artificial neuron networks (ADALINE, MLP, McCulloch-Pitts Neurons, etc.)

10 Deep Learning With Spikes

10.1 Motivation

Neurobiology mostly uses spiking neural networks. Neurons output spikes, which are binary events and localized in time.

So how do hidden units learn?

Bottom-up approach:

- Start with a random network model
- Include data driven plasticity model
- Observe function → Limited success in learning useful hidden layer representations

One outcome would be Spike-time dependent plasticity (STDP), which is a way the weights can be adjusted. So far not very useful in building networks; the weights tend to blow up. Over the years, people went over this concept and tried to improve.

Top-down approach:

- Start with function in mind
- Derive suitable plasticity rules
- Build functional network models

Deep learning is an example of a top-down framework. Two questions remain:

The algorithmic question: How to compute the gradient?

The conceptual question: Which functions are learned?

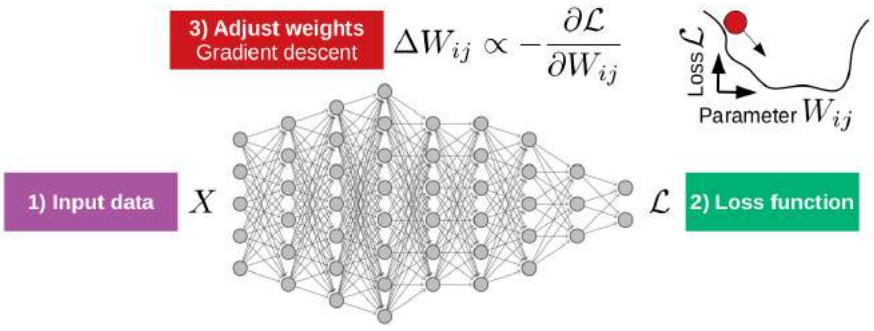


Figure 113: Deep learning as a top-down framework

10.2 Recap: Spiking Neuron Models

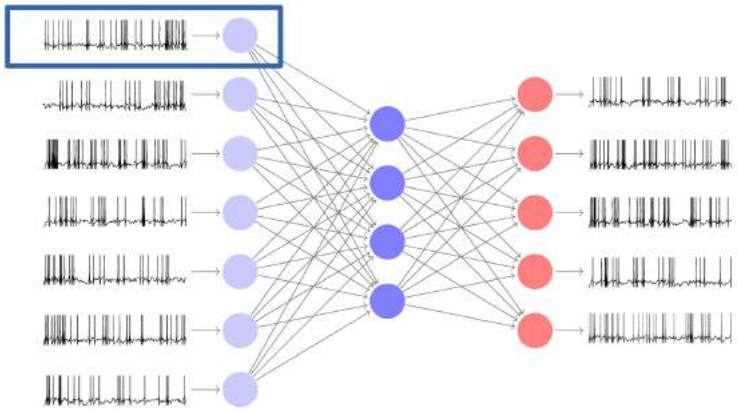


Figure 114: Illustration of a Spiking Neural Network

- Spiking networks consist of spiking neurons
- Network modeling largely relies on simplified neuron models

10.2.1 Biophysics of neuronal signal transmission

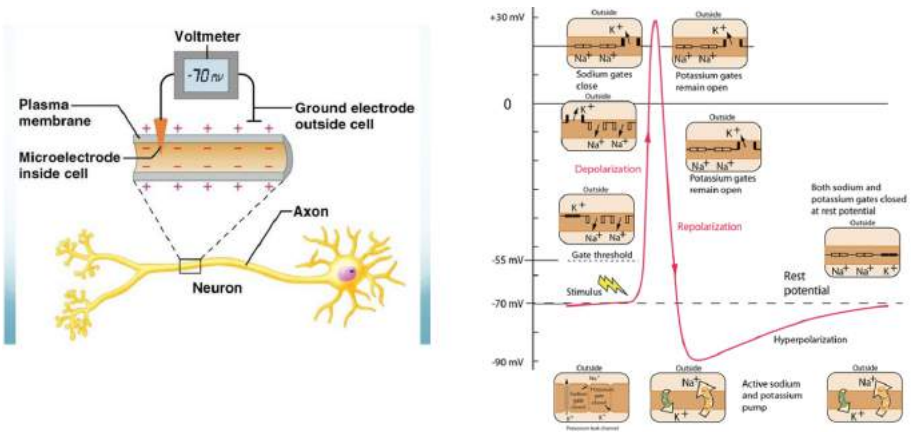


Figure 115

10.2.2 From biophysical to reduced neuron models

In order to build a phenomenological model of neuronal dynamics[58], we describe the critical voltage for spike initiation by a formal threshold θ. If the voltage $U_i(t)$ (that contains the summed effect of all inputs) reaches θ from below, we say that neuron i fires a **spike**. The moment of threshold crossing defines the firing time $t_i^{(f)}$. The model makes use of the fact that neuronal action

[58]Incredibly useful and free book by Wulfram Gernster et. al, where most of the content of this section is copy-pasted from: `https://neuronaldynamics.epfl.ch/online/index.html`

potentials of a given neuron always have roughly the same form. If the shape of an action potential is always the same, then the shape cannot be used to transmit information: rather information is contained in the presence or absence of a spike. Therefore action potentials are reduced to 'events' that happen at a precise moment in time.

Leaky integrate-and-fire neuron: Neuron models where action potentials are described as events are called 'Integrate-and-Fire' models. No attempt is made to describe the shape of an action potential. Integrate-and-fire models have two separate components that are both necessary to define their dynamics:

1. An equation that describes the evolution of the membrane potential $U_i(t)$
2. A mechanism to generate spikes.

The variable U_i describes the momentary value of the membrane potential of neuron i. In the absence of any input, the potential is at its resting value U_{rest}. If an experimentalist injects a current $I(t)$ into the neuron, or if the neuron receives synaptic input from other neurons, the potential $U_i(t)$ will be deflected from its resting value. The basic electrical circuit representing a leaky integrate-and-fire model consists of a capacitor C in parallel with a resistor R driven by a current $I(t)$, as shown in the figure below. The differential equation for describing the leaky-integration of the voltage is given by

$$\tau \frac{\mathrm{d}}{\mathrm{d}t} U(t) = -\left(U(t) - U_{\mathrm{rest}}\right) + R_{\mathrm{in}} I(t), \tag{169}$$

where $\tau = RC$ is the time constant of the circuit.

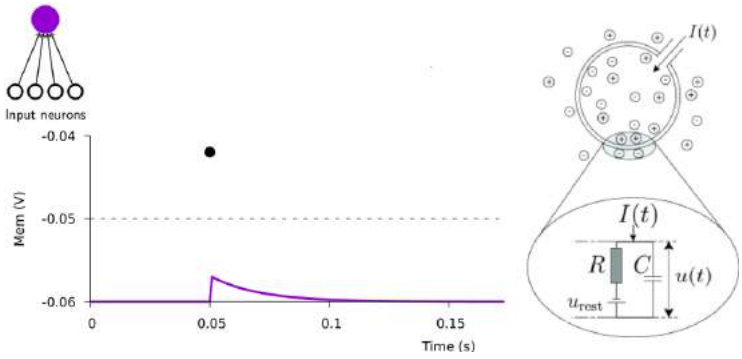


Figure 116: Passive Membrane

Now the second part of the leaky integrate-and-fire neuron is the firing and re-setting of the voltage after the neuron-specific threshold has been reached. At the firing time $t^f : U(t^f) = \theta$, the neuron fires (with a not-here-to-be-defined spike-form), the firing time is noted and immediately after the voltage reset to a new value $U_{rest} < \theta$:

$$\lim_{\delta \to 0; \delta > 0} U(t^{(f)} + \delta) = U_{rest} \tag{170}$$

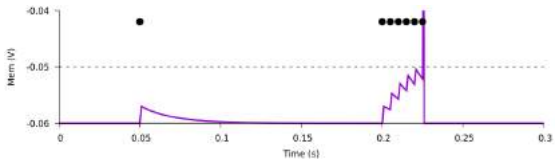


Figure 117

Exponential postsynaptic currents: We also want to model the synapse. Activation of a presynaptic neuron results in a release of neurotransmitters into the synaptic cleft. The transmitter molecules diffuse to the other side of the cleft and activate receptors that are located in the postsynaptic membrane. In both cases, the activation of the receptor results in the opening of certain ion channels and, thus, in an excitatory or inhibitory postsynaptic transmembrane current (EPSC or IPSC). The main mechanism for carrier transport underlying this current is diffusion of the ions passing from the extracellular space into the cell. Instead of developing a mathematical model of the transmitter concentration in the synaptic cleft, we keep things simple and describe transmitter-activated ion channels as an explicitly time-dependent conductivity. This conductivity change most often modelled as an exponentially decaying function, to represent the effect of closing ion channels. The following differential equation describes the evolution of the postsynaptic current $I(t)$:

$$\frac{\mathrm{d}}{\mathrm{d}t}I(t) = -\frac{I(t)}{\tau_{\mathrm{syn}}} + S(t) \tag{171}$$

and is derived starting from the decaying exponential we want to model:

$$I(t) = I_0 e^{\frac{-t}{\tau}} \tag{172}$$

$$ln(I(t)) = ln(I_0) + ln(e^{\frac{-t}{\tau}}) \tag{173}$$

$$= ln(I_0) - \frac{t}{\tau}|\frac{dI(t)}{dt} \tag{174}$$

$$\frac{1}{I(t)}\frac{I(t)}{dt} = -\frac{1}{\tau} \tag{175}$$

$$\frac{dI(t)}{dt} = -\frac{I(t)}{\tau} \tag{176}$$

Considering the changes arising due to the discrete APs, the term $S(t)$ added in Equation 171 determines an instantaneous increase in the postsynaptic current proportional to the synaptic weight.

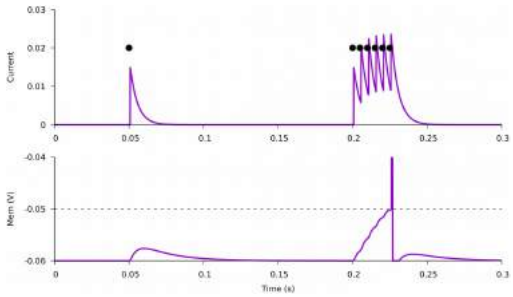


Figure 118

The spike response model (SRM): ?? Also here So far, we have described neuronal dynamics in terms of systems of differential equations. There is another approach called the 'filter picture'. In this picture, the parameters of the model are replaced by (parametric) functions of time, generically called 'filters'. The neuron model is therefore interpreted in terms of a membrane filter as well as a function describing the shape of the spike and, potentially, also a function for the time course of the threshold. Together, these three functions establish the Spike Response Model (SRM). More info here: https://neuronaldynamics.epfl.ch/online/Ch6.S4.html

Mathematically speaking, we integrate over the differential equation, then replace the integration times multiplications with convolutions of filter kernels over the spikes:

$$\mathrm{U}_i(t) = \sum_j w_{ij} \left(\epsilon * S_j\right)(t) + \left(\eta * S_i\right)(t) + U_{\mathrm{rest}} \tag{177}$$

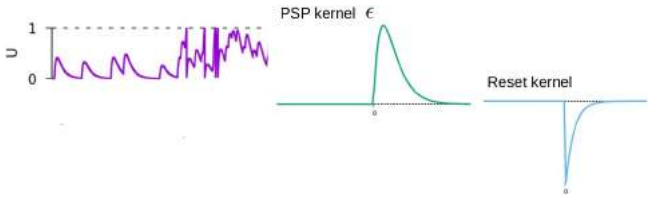


Figure 119

Towards functional neural network models: We want this

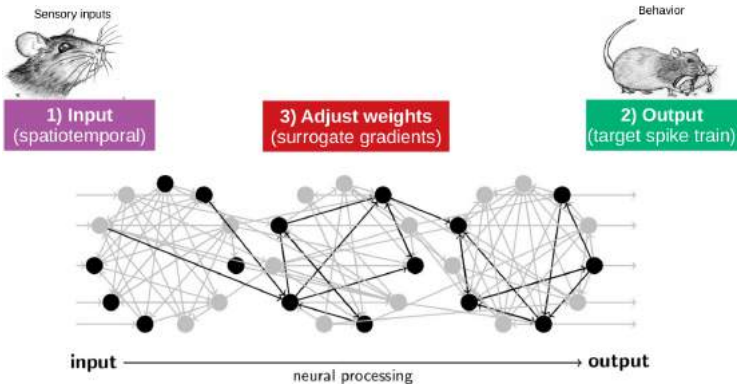


Figure 120

10.3 Dealing with the Vanishing Gradient Problem

10.3.1 Defining the problem

Can we do supervised learning in spiking multi-layer networks with a local online learning rule? We want to compare output spikes with target spikes. Let's try:

Van Rossum distance between output and target spike trains: There is different ways of representing a spike train. If the spikes are seen to be discrete units, the spike train $S(t)$ is given simply by

$$S(t) = \sum_{i}^{M} \delta(t - t_i) \tag{178}$$

Replacing the delta function associated with each spike with an exponential function, that is, add an exponential tail to all spikes, leads to another definition of a spike train:

$$S(t) = \sum_{i}^{M} H(t - t_i) \exp -(t - t_i)/t_c, \tag{179}$$

where $H(t)$ is the heavyside function. The loss between the target spikes distance $\hat{S}$ and the output spikes distance S can be defined as the Van Rossum distance:

$$\mathcal{L} = \frac{1}{2} \int_{-\infty}^{t} [\lambda * (\hat{S}(s) - S(s))]^2 ds \tag{180}$$

The problem with our spike model and such a loss becomes evident when we try to differentiate the loss with respect to the single weights:

$$-\frac{\partial \mathcal{L}}{\partial w_k} = \int_{-\infty}^{t} \lambda * (\hat{S}(s) - S(s)) \lambda * \frac{\partial S(t)}{\partial w_k} ds \tag{181}$$

The partial derivation $\frac{\partial S(t)}{\partial w_k}$ This derivative is problematic because for most neuron models, it is zero except at spike times at which it is not defined. Thus it forces the gradient to vanish.

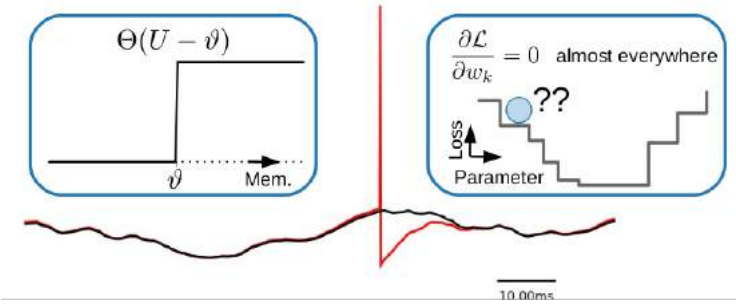


Figure 121

10.3.2 A history of struggle

Noise injection : [59] Write smth here.

Differentiate firing times : [60] (skipped in lecture)

Make spikes differentiable : [61]

Force hidden units "on target" : [62]

e.g. firing-rate approaches : [63]

[59] Pfister, Toyoizumi, Barber & Gerstner (2006), Gardner, Sporea Grüning (2015)
[60] Bohte, Kok, Poutre (2002), Gütig & Sompolinski (2006), Gütig (2016), Mostafa (2018)
[61] Huh & Sejnowski (2018)
[62] Gilra & Gerstner (2017), Nicola Clopath (2017)
[63] Hunsberger & Eliasmith (2015), Lee et al. (2016), ...

10.3.3 Surrogate gradients & SuperSpike

Idea[64]: Replace the non-differentiable heaviside function with the differentiable sigmoid function σ, but only in the backward-pass. In the forward pass, leave it as a heaviside function. The equivalent in machine learning would be "Straight-through estimators"[65]. This procedure leads to the replacements:

$$\frac{\partial S_i}{\partial w_{ij}} \quad \rightarrow \quad \sigma'\left(U_i\right) \frac{\partial U_i}{\partial w_{ij}} \tag{182}$$

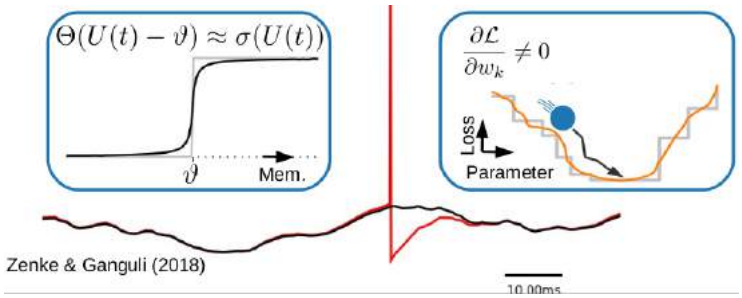


Figure 122

If now the membrane potential $U_i(t)$ is written in the integral form as a spike response model (SRM_0)

$$\mathrm{U}_i(t) = \sum_j w_{ij} \left(\epsilon * S_j\right)(t) + \left(\eta * S_i\right)(t) + U_{\mathrm{rest}}, \tag{183}$$

where ϵ is the causal membrane kernel (corresponding to the postsynaptic potential) and η captures spike dynamics and reset. With some steps that are briefly explained in the paper, one gets for the gradient descent learning rule for a single neuron the following expression:

$$\frac{\partial w_{ij}}{\partial t} = r \int_{-\infty}^{t} \lambda * (\underbrace{\left(\epsilon * S_j\right)(s)}_{\text{Pre}} \underbrace{\sigma'\left(U_i(s)\right)}_{\text{Post}}) \underbrace{e_i(s)}_{\text{Error signal}} \; ds \tag{184}$$

Here, r is the learning rate, $e_i(s) = \lambda * (\hat{S}_i - S_i)$ the error signal and λ the eligibility trace ("Ca transient"). This learning rule is called **SuperSpike**. We can divide this rule into three factors:

Pre Presynaptic activity

Post Postsynaptic activity

Error signal Specific feedback

The pre- and postsynaptic activity are combined in a multiplicative manner, which can be seen as the Hebbian term (which is "STDP"-like). σ' is the voltage nonlinearity, thus the learning rule is voltage based.

[64] Zenke & Ganguli, "SuperSpike: Supervised Learning in Multilayer Spiking Neural Networks", 2018

[65] Begio et al., "Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation", 2013

10.3.4 Hidden layers

What about training the hidden layers? The learning rule for hidden weights is

$$\frac{\partial w_{ij}}{\partial t} \equiv \sum_k e_k(t)\epsilon * \left[w_{ki}\epsilon * \left(\epsilon * S_j(t)\sigma'\left(U_i\right)\right)\sigma'\left(U_k\right)\right]. \tag{185}$$

Biologically seen this is problematic, because

1. It requires symmetric weights
2. There are downstream activities

One way to overcome those issues is by applying feedback-alignment, which is well documented in section 2. Note that all quantities computed online. Temporal credit assignment through dynamics at the synaptic level (eligibility trace).

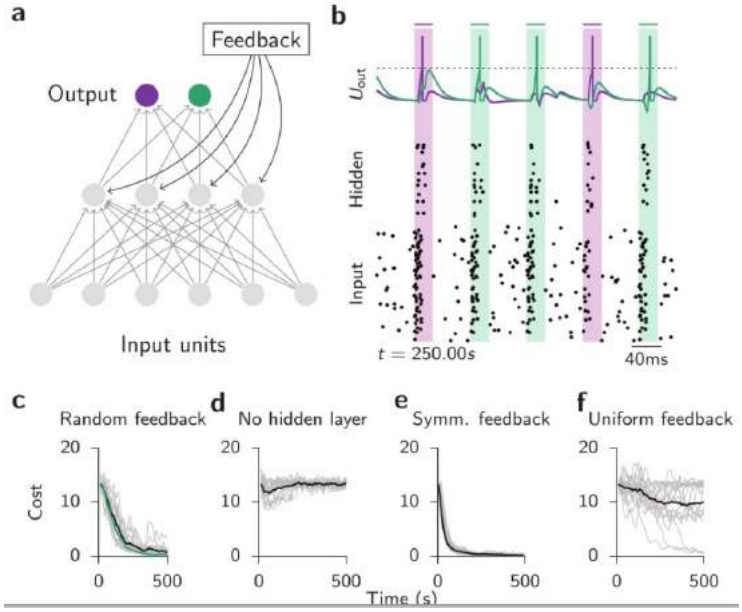


Figure 123: Network trained to solve a non-linearly separable classification problem with noisy input neurons. **(a)** Sketch of network layout with two output units and four hidden units. **(b)** Snapshot of network activity at the end of training with random feedback. Four input patterns from two non-linearly separable classes are presented in random order (shaded areas). In between stimulus periods, input neurons spike randomly with 4 Hz background firing rate. **(c)** Learning curves of 20 trials with different random initialisations (gray) for a network with random feedback connections that solves the task. The average of all trials is given by the black line. The average of 20 simulation trials with an additional regularization term (see section 3) is shown in green. **(d)** Same as panel c but for a network without hidden units that cannot solve the task. **(e)** Same as panel c but for symmetric feedback. **(f)** Same as panel c but for uniform ("all ones") feedback connections.

10.3.5 Sequence-to-sequence learning

Because zero error was achieved with simple tasks using different types of feedback signal, the learning rule can be put to work in harsher conditions, under more challenging tasks: making it associate an spatio-temporal target output pattern to a repeating frozen Poisson noise input. In this case, a larger, 3-layer net was used (100 in, variable number hidden, 100 out), but an output pattern matching the target was achieved with only 32 hidden neurons. The random feedback performs worse than a network that was trained without a hidden layer, but with symmetrical weights.

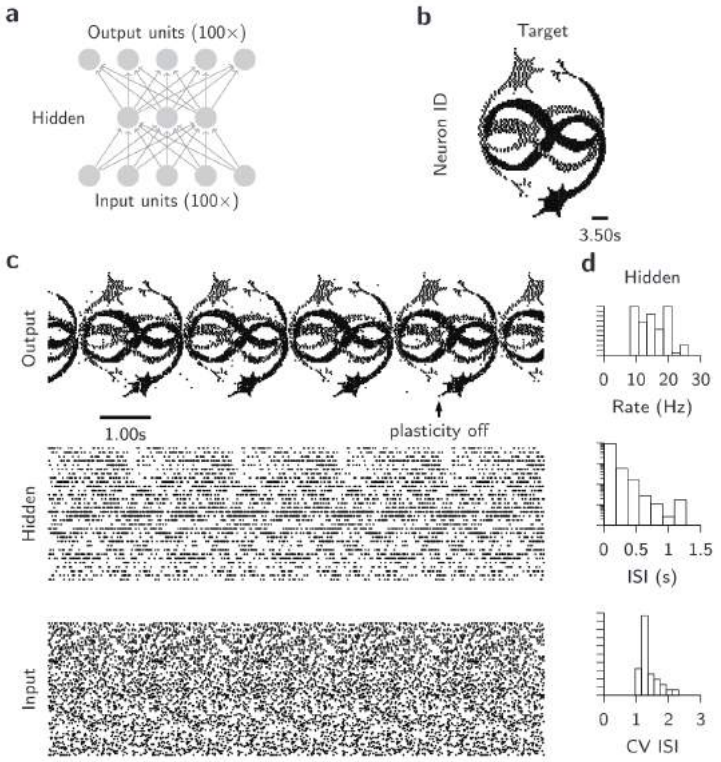


Figure 124: Network trained to solve a nonlinearly separable classification problem with noisy input neurons. (a) Sketch of network layout with two output units and four hidden units. (b) Snapshot of network activity at the end of training with random feedback. Four input patterns from two nonlinearly separable classes are presented in random order (shaded areas). In between stimulus periods, input neurons spike randomly with 4 Hz background firing rate. (c) Learning curves of 20 trials with different random initializations (gray) for a network with random feedback connections that solves the task. The average of all trials is given by the black line. The average of 20 simulation trials with an additional regularization term (see section 3) is shown in green. (d) Same as panel c but fora network without hidden units that cannot solve the task. (e) Same as panel c but for symmetric feedback. (f) Same as panel c but for uniform ("all ones") feedback connections.

The rule can also be used as an auto-encoder network, since it can be provided the same pattern as both input and output. It is able to reconstruct the output pattern with high fidelity while having a number of hidden units smaller than the number of units in the input and output layers.

10.4 Spiking nets and temporal coding

The RNN term is used in its widest sense, that of networks with states evolving in time based on well-defined dynamic recurrent equations. An important fact to note is that, while recurrent synaptic connections between neurons in a network give rise to recurrent dynamics, they are not absolutely necessary, as dynamical recurrent can arise without them. This is the case of neurons or synapses which have states which evolve according to internal dynamics: the current state depends on the previous state and the next state depends on the current state, thus state-full units are inherently recurrent.

This idea can be very well applied to SNNs, with computations necessary to update a cell state that can be unrolled in time as seen in Figure 125.

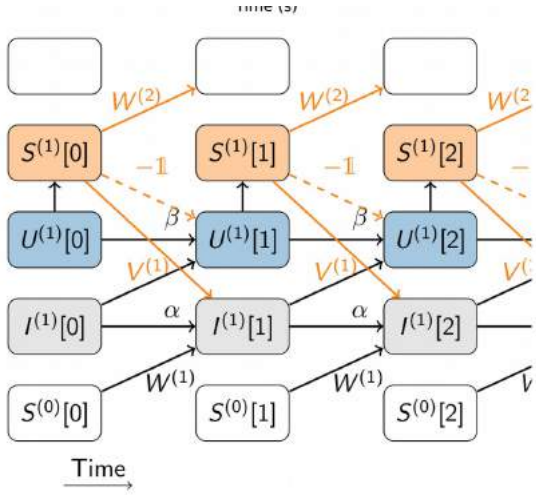


Figure 125: Illustration of the computational graph of a SNN in discrete time. Time steps flow from left to right. Input spikes S(0) are fed into the network from the bottom and propagate upwards to higher layers. The synaptic currents I are decayed by α in each time step and fed into the membrane potentials U. The U are similarly decaying over time as characterized by β. Spike trains S are generated by applying a threshold non-linearity to the membrane potentials U in each time step. Spikes causally affect the network state (orange connections). First, each spike causes the membrane potential of the neuron that emits the spike to be reset. Second, each spike may be communicated to the same neuronal population via recurrent connections V(1). Finally, it may also be communicated via W(2) to another downstream network layer or, alternatively, a readout layer on which a cost function is defined.

The Back-propagation rule can be applied to RNNs. In this case the recurrence is "unrolled" meaning that an auxiliary network is created by making

copies of the network for each time step. The unrolled network is simply a deep network with shared feed-forward weights W(l) and recurrent weights V(l), on which the standard BP applies:

$$\Delta W_{ij}^{(l)} \propto \frac{\partial \mathcal{L}[x]}{\partial W_{ij}^{(l)}} = \sum_{m=0}^{t} \delta_i^{(l)}[m] y_j^{(l-1)}[m] \tag{186}$$

$$\Delta V_{ij}^{(l)} \propto \frac{\partial \mathcal{L}[x]}{\partial V_{ij}^{(l)}} = \sum_{m=1}^{t} \delta_i^{(l)}[m-1] y_j^{(l)}[m-1] \tag{187}$$

$$\delta_i^{(l)}[n] = \sigma'\left(a_i^{(l)}[n]\right)\left(\sum_k \delta_k^{(l+1)}[n] W_{ij}^{(T,l)} + (\sum_k \delta_k^{(l)}[n+1] V_{ij}^{(T,l)}\right) \tag{188}$$

Applying BP to an unrolled network is referred to as back-propagation through time (BPTT).

11 Neuromorphic Systems 1

11.1 History and motivation

11.1.1 History

Neuromorphic Engineering (NE) Neuromorphic engineering is a relatively young field that attempts to build physical realizations of biologically realistic models of neural systems using electronic circuits implemented in very large scale integration technology. The concept was coined by Carver Mead in the late 1980s. Carver Mead together with John Hopfield and Richard Feynmann started the Physics of Computation course at Caltech, which later transformed to the fully-flagged Phd/MSc program called "Computation and Neural Systems". Notable alumni are INI professors like Tobi Delbruck, Shi-Chi Liu and Giacomo Indiveri. Carver Mead also coined Moore's Law: the number of transistors in a dense integrated circuit doubles about every two years.

11.1.2 Physical comparisons: channel vs. transistor

NE is possible because the physics of voltage activated membrane channels and transistors are closely related. When MOSFETs are operated in the weak-inversion regime (sub-threshold), the main mechanism of carrier transport is that of diffusion, as it is for ions flowing through proteic channels across neuron membranes:

- K^+ conductance of cell membrane increases exponentially with the membrane voltage, until it approaches saturation at the maximal opening of the channel.
- Drain-to-Source Current (I_{DS}) flowing through a MOS transistors increases exponentially with the Gate-to-Source Voltage (V_{GS}) below a specific "threshold" voltage and quadratically above this value.

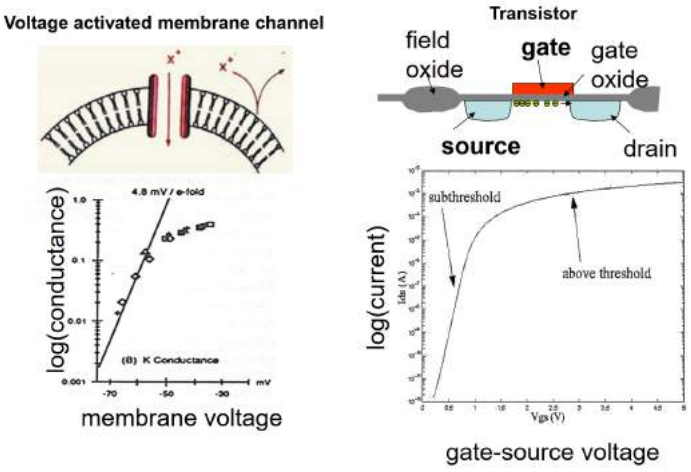


Figure 126: Comparing a voltage activated membrane channel to the MOS transistor.

11.1.3 Physical comparisons: brain vs. computer

Even though conventional computation powered by silicon has seen immense gains in the past decades, as predicted by Moore's law, natural computation is still at least a million times more power efficient.

Some notable differences between brain-natural and artificial computation are presented below:

Artificial Computation	Brain
Fast clocked system (GHz)	Self-timed, data-driven (0.1 Hz to tens kHz)
Bit-perfect deterministic logical state	Synapses are stochastic
Memory distant to computation	Memory local to computation
High-bit resolution	Low-bit resolution with neurons as quantizers

11.1.4 Physical comparisons: cell vs. ALU

When looking at individual neuronal cells structurally we can define a set of computational steps executed by each main components:

- **Channels** execute a form of multiplication as observed in Figure 126.
- **Synapses** host populations of channels which open at the arrival of a PSP. The effect of each channel in the population is summed up, making the synapse behave like a Multiply-Accumulate unit.
- **Cell membrane** hosts complementary channels which can both increase the membrane voltage or decrease it.
- **Dendrites** connect to pre-synaptic neurons through both excitatory and inhibitory synapses, leading to analog additions and subtractions being executed at the level of single dendrites.
- **Dendritic trees** perform another level of analog computation, integrating the potentials arriving from each dendrite into the cell body.
- **Axons** transmit digital spike events (APs) to distant cells.

11.1.5 Neuromorphic VLSI circuits

Detailed biophysical models of cortical circuits are derived from neuroscience experiments. Neural networks models are designed, with realistic spiking neurons and dynamic synapses, then mapped into analog circuits and integrated in large numbers on VLSI chips. [cite: computational intelligence book]

One example of analog integrated circuit with the functional characteristics of real nerve cells can be seen in Figure 127. Because the physics underlying the conductivity of silicon devices and biological membranes is similar, the 'silicon neuron' is able to emulate efficiently the ion currents that cause nerve impulses and control the dynamics of their discharge. It operates in real-time and consumes little power, and many 'neurons' can be fabricated on a single silicon chip. [cite: a silicon neuron] This neuron implements the Hodgkin-Huxley model which expresses the membrane conductance effects of Na^+ and K^+ ion currents and of the passive leak at rest and during an action potential. The

variable conductance is implemented using the *follower-integrator* circuit. The follower-integrator is composed of a transconductance amplifier configured in negative feedback mode with its output node connected to a capacitor. In weak-inversion the circuit behaves as a first-order low-pass filter with a tunable conductance. [cite neuromorphic silicon neurons].

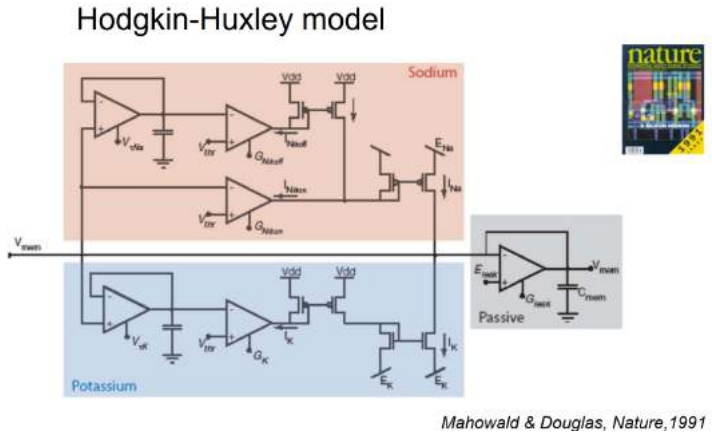


Figure 127: A silicon neuron circuit.

Processors harnessing both digital and analog techniques have been built in the past decade and they are presented in more detail in Lecture 12.

11.2 Modelling vision and audition

11.2.1 Dynamic Vision Sensor: silicon retina

The human retina contains around 10^8 analog receptors, 10^6 ganglion cells with spiking outputs and 10^9 dynamic range modulating cells while consuming less than 3 mW of power. The retinal cone cells perform a special function of adjusting the operating point and the gain according to background illumination. This is driven by an active cell adaptation mechanism which is manifested as a shift in the light intensity-response curve, providing high sensitivity even in very low-light conditions. This property of the retina has inspired inventors in building state-of-the-art imaging sensors like the Dynamic Vision Sensor, whose main component, the Pixel, is depicted in Figure 131.

The continuous-time front end photoreceptor (inspired from the adaptive photoreceptor) is the correspondent of the cone cell. The photoreceptor circuit comprises a photodiode whose photocurrent is sourced by a saturated NMOS transistor. This well-known transimpedance configuration converts the photocurrent logarithmically into a voltage and also holds the photodiode clamped at a virtual ground.

The bipolar cell is built as a precision self-timed switched-capacitor differentiator. Local capacitor ratio matching gives the differencing circuit a precisely defined gain for changes in log intensity, thus reducing the effective imprecision of the comparators that detect positive and negative changes in log intensity. The DC mismatch is removed by balancing the output of the differencing circuit to a reset level after the generation of an event.

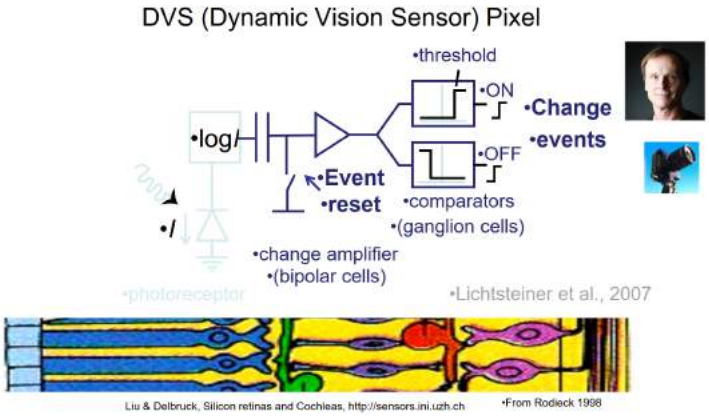


Figure 128: The Pixel.

The last stage is the comparison of the output of the inverting amplifier against global thresholds that are offset from the reset voltage to detect increasing and decreasing changes. If the input of a comparator overcomes its threshold, an ON or OFF event is generated. This part can be regarded as a ganglion cell and is build using cheap two-transistor comparators.

Each pixel independently and in continuous time quantifies local relative intensity changes to generate spike events. These events appear at the output of the sensor as an asynchronous stream of digital pixel addresses. These address-events signify scene reflectance change and have sub-millisecond timing precision. This communication protocol is called Address-Event-Representation (AER). The output data rate depends on the dynamic content of the scene and is typically orders of magnitude lower than those of conventional frame-based imagers.

The Dynamic and Active Pixel Vision Sensor (DAVIS) combines active pixel technology with the DVS temporal contrast pixel. The two streams of frames and events are output concurrently. That way, the DAVIS produces conventional frames, which are the basis of all existing machine vision, and the event stream that allows quick responses with sparse data and high dynamic range. [http://sensors.ini.uzh.ch/sensors-21.html]

Some demonstrative applications of the DVS/DAVIS are optical-flow, ultra-low power (2ms reaction at 4% CPU usage) object tracking (RoboGoalie), autonomous driving and drone flight, high-speed robotics.

An example algorithm used for object tracking with the DVS uses spatio-temporal coherence:

1. For each event find nearest cluster
2. If the event is withing the cluster, move the cluster, else send new cluster
3. Periodically manage cluster lifetime

11.2.2 Dynamic Audio Sensor: silicon cochlea

The Dynamic Audio Sensor is an asynchronous event-based silicon cochlea. The board takes stereo audio inputs and the custom chip asynchronously outputs a

stream of address-events representing activity in different frequency ranges. As such it is a silicon model of the cochlea, the auditory inner ear. The system has also been called AER-EAR.

It uses cascaded second-order sections (SOS) to model the physical oscillation of the basilar membrane. These drive half-wave rectifier circuits, which model inner hair cells. These in turn drive multiple pulse frequency modulator circuits, which model ganglion cells with different spike thresholds. The resonance of individual sections can be adjusted with local digital-to-analog converters.

This chip includes a variety of features including a matched binaural pair of cochleas, on-chip digitally controlled biases, on-chip microphone preamplifiers, and open-sourced host software APIs and algorithms. A bus-powered USB board enables easy interfacing to standard PCs for control and processing (Fig. 1), whilst a parallel AER port allows direct connection to other dedicated spiking neuromorphic hardware.

Each cochlea consists of a 64-stage cascaded filter bank stage. The cascaded architecture is preferred over a coupled bandpass architecture so we can achieve better matching and sharp high frequency roll-off. Each filter stage consists of a second-order-section (SOS) filter which is biased by a Complementary Lateral Bipolar Transistor (CLBT) ladder to improve matching [11]. A differential readout of each SOS output drives its own halfwave rectifier (HWR) circuit, and the HWR output drives 4 pulse-frequency modulators (PFMs). The PFM circuits implement an integrate-and-fire model with a threshold (VT). The four PFMs have individual global thresholds (VT1 to VT4), allowing volume encoding by selective activation of PFMs. Compared with regularly-sampled audio systems, the PFM outputs are transmitted asynchronously, reducing latency to the analog delay along the filter bank and increasing temporal resolution to microseconds.

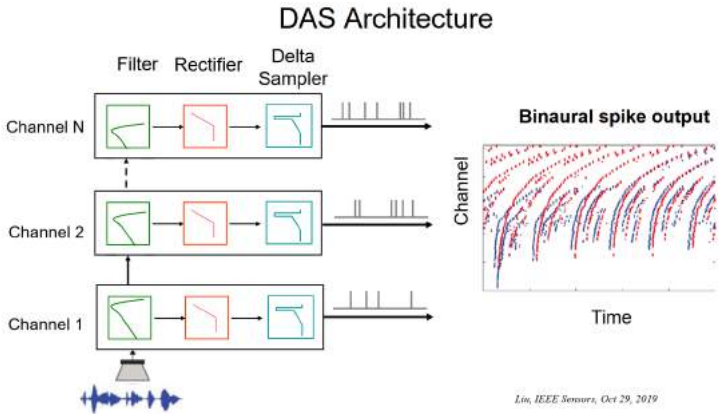


Figure 129: The DAS architecture.

11.2.3 CAVIAR

CAVIAR (Convolution AER VIsion Architecture for Real-time), was the largest multichip multilayer AER real-time, frame-free vision system (Figure 13.13) built at the time, with a combined total of 45K neurons and 5M synapses

(Serrano-Gotarredonaet al. 2009). This system has four custom mixed-signal AER chips, five custom digital AER interface components and it performs up to 12G synaptic operations per second. It is capable of achieving millisecond object recognition and tracking latencies, illustrating the computational efficiency of AER systems. The CAVIAR vision system is composed of the following custom chips: the DVS temporal contrast retina chip, a set of programmable kernel convolution chips, a 2D winner-take-all (WTA) object chip, and a delay line chip and learning chip.

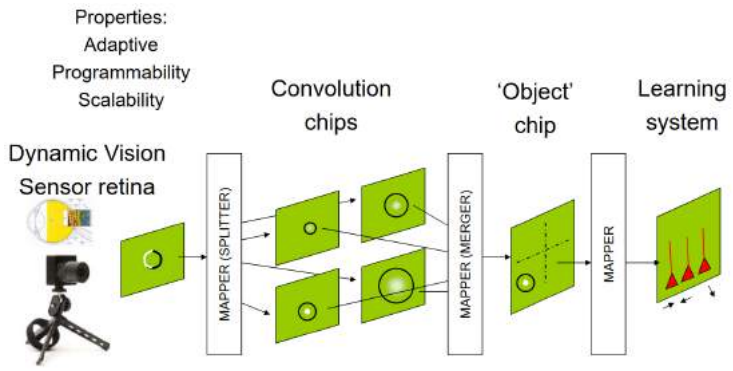


Figure 130: CAVIAR architecture.

11.3 Sensors driving event-driven Deep Learning networks

11.3.1 Background of deep networks and biological aspirations

Yann LeCun et al. (1989) used back-propagation to learn the convolution kernel coefficients directly from images of hand-written numbers. Learning was thus fully automatic, performed better than manual coefficient design, and was suited to a broader range of image recognition problems and image types. This was the origin of LeNet, a very well known 5-layer CNN model.

The DAVE project consists of a small off-road robot that uses an end-to-end learning system to avoid obstacles solely from visual input. The DAVE robot has two cameras with analog video transmitter. The video is transmitted to a remote computer that collects the data, runs the automatic driving system, and controls the robot through radio control. A convolutional network takes the left and right camera images (YUV components) and is trained to directly predict the steering angle of a human driver. The CNN ran at 10 FPS on 1.5GHz Celeron, was trained on about 90k frames. This led to DARPA LAGR program.

At the beginning of CNNs, everybody used threshold or sigmoid units, but cortical Firing frequency - Current (F-I) curves have a different look. To model this curve, biologists introduced Rectified Linear Units (ReLU) to ML, more than a decade before mainstream AI started using it. ReLU allows unbounded values to pass through while providing an essential non-linearity. It helps the back=propagation algorithm avoid the vanishing gradients problem and produces sparse feature maps.

The max-pooling operation is, in neuroscience terms, akin to a neuron's firing rate being equal to the firing rate of its highest firing input, like in he well-known Winner-Takes-All networks. Because neurons pool from many neurons, it's hard to devise a neuron that could straightforwardly do this. But the pooling operation was inspired by the discovery of complex cells and originally started as an averaging operation, something trivially achievable by neurons. Max-pooling, however, has been found to be more successful in terms of object recognition performance and fitting biological data and is now widely used.

11.3.2 Spiking sensors + deep network systems

When interfacing neuromorphic sensors like the DAVIS or DAS with deep networks, a data-driven approach is used. In data-driven pipelines, processing is triggered only by input events so processing times, dissipation power and latencies can be reduced. Some examples of projects developed using this approach and neuromorphic sensors are the Predator and the RoShamBo.

The predator/prey robot consists of a CNN that is trained and run on data from a Dynamic and Active Pixel Sensor (DAVIS) mounted on a Summit XL robot (the predator), which follows another one (the prey). The CNN is driven by both conventional image frames and dynamic vision sensor "frames" that consist of a constant number of DAVIS ON and OFF events. The network is thus "data driven" at a sample rate proportional to the scene activity, so the effective sample rate varies from 15 Hz to 240 Hz depending on the robot speeds.

RoShamBo is system which uses the DVS sensor to drive a 5-layer CNN into playing the game of rock-paper-scissors against a human. CNN input images are generated at a variable, data-driven rate between 1-200 Hz by accumulating asynchronous DVS address-events into 64x64 pixel 2D histograms of a constant total number of events. The DVS + NullHop is so fast that the symbol can be recognized as the person is opening it's hand. Thus the system shows the symbol to beat the human before they complete their throw.

11.3.3 Data-driven deep networks: exploiting sparsity

NullHop is a CNN accelerator that exploits the spatial sparsity of neuron activations to accelerate the computation and reduce memory requirements. The flexible architecture allows high utilization of available computing resources across kernel sizes ranging from 1x1 to 7x7. NullHop can process up to 128 input and 128 output feature maps per layer in a single pass.

The accelerator implements one convolutional stage followed by a ReLU transformation and then a max-pooling stage. The ReLU and max-pooling stages can be disabled. To implement the full forward pass in a CNN, the accelerator evaluates convolutional stages one after another in a sequential manner. The input feature maps and the kernel values for the current convolutional layer are stored in two independent SRAM blocks.

The input decoding block is able to directly skip over zero pixels in the compressed input feature maps without wasting any MAC operation. NullHop uses a novel sparse matrix compression algorithm allowing it to encode feature maps with 1.x bits/zero pixel, achieving superior memory usage.

DeltaNet is a Recurrent Neural Network (RNN) architecture in which each neuron transmits its value only when the change in its activation exceeds a

threshold. RNNs require a large weight memory storage that is expensive to allocate to on-chip static random access memory (SRAM). The execution of RNNs as delta networks is attractive because their states must be stored and fetched at every time-step. By caching neuron activations, computations can be skipped where inputs change by a small amount from the previous update. The purpose of a delta network is to transform a dense matrix-vector multiplication (for example, a weight matrix and a state vector) into a sparse matrix-vector multiplication followed by a full addition. This transformation leads to savings on both operations (actual multiplications) and more importantly memory accesses (weight fetches).

A project which harnesses the power of DeltaNets is the hardware system for continuous speech recognition. It consists of a single-layer RNN with 256 gated recurrent unit (GRU) neurons and is driven by input features generated either from the output of a filter bank running on the ARM core of the FPGA in a PmodMic3 microphone setup or from the asynchronous outputs of a spiking silicon cochlea circuit. The microphone setup achieves 7.1 ms minimum latency and 177 frames-per-second (FPS) maximum throughput while the cochlea setup achieves 2.9 ms minimum latency and 345 FPS maximum throughput. The low latency and 70 mW power consumption of the DeltaRNN makes it suitable as an IoT computing platform.

11.3.4 Conversion of deep ANN to deep SNN

[almost all from rueckauer et al.] Inference in very large deep networks like GoogLeNet and VGG-16, requires substantial computational and energy costs, thus limiting their use in mobile and embedded applications. Recent work have shown that the event-based mode of operation in SNNs is particularly attractive for reducing the latency and computational load of deep neural networks.

The most straight-forward way to combine the advantages of these models is to take the parameters of a pre-trained ANN and to map them to an equivalent-accuracy SNN. The basic principle of converting ANNs into SNNs is that firing rates r_i^l of spiking neurons correlates with the its original graded activation a_i^l of analog neurons.

The spiking neuron integrates inputs $z_i^l(t)$ until the membrane potential $V_i^l(t)$ exceeds a threshold $V_{thr}\epsilon R^+$ and a spike is generated. Once the spike is generated, the membrane potential is reset. Two types of reset are possible: reset the membrane potential back to a baseline, typically zero and reset by subtraction, or "linear reset mode", subtracts the threshold V_{thr} from the membrane potential at the time when it exceeds the threshold:

$$r_i^l(t) = \begin{cases} \left(V_i^l(t-1) + z_i^l l(t)\right)\left(1 - \Theta_{t,i}^l\right) & \text{reset to zero} \\ V_i^l(t-1) + z_i^l l(t) - V_{thr}\Theta_{t,i}^l) & \text{reset by subtraction} \end{cases} \tag{189}$$

The input to first-layer neurons is then related to the ANN activations via $z_i^l = V_{thr} \cdot a_i^l$. In order to relate these ANN activations to the SNN spike rates, an average is done on the membrane Equation 189 over the simulation time. The resulting rates are:

$$r_i^l(t) = \begin{cases} a_i^l r_{max} \cdot \frac{V_{thr}}{V_{thr}+\epsilon_i^l} - \frac{V_i^l(t)}{t\cdot(V_{thr}+\epsilon_i^l)} & \text{reset to zero} \\ a_i^l r_{max} - \frac{V_i^l(t)}{t\cdot V_{thr}} & \text{reset by subtraction} \end{cases} \tag{190}$$

Some important operators of standard ANNs need a special treatment in the conversion process.

In a spiking network, a **bias** can simply be implemented with a constant input current of equal sign as the bias. Alternatively, one could present the bias with an external spike input of constant rate proportional to the ANN bias.

The **data-based weight normalization** mechanism is based on the linearity of the ReLU unit used for ANNs. It can simply be extended to biases by linearly rescaling all weights and biases such that the ANN activation a is smaller than 1 for all training examples. Denoting the maximum ReLU activation in layer l as $\lambda^l = max[a^l]$, then weights W^l and biases b^l are normalized to $W^l \rightarrow W^l \frac{\lambda^{l-1}}{\lambda^l}$ and $b^l \rightarrow b^l/\lambda^l$.

The analog **input activations** are interpreted as constant currents. The input to the neurons in the first hidden layer is obtained by multiplying the corresponding kernels with the analog input image x:

$$z_i^1 = V_{thr} \left(\sum_{j=1}^{M^0} W_{ij}^1 x_j + b_i^1 \right) \tag{191}$$

This results in one constant charge value z_i^1 per neuron i, which is added to the membrane potential at every time step. The spiking output then begins with the first hidden layer.

Softmax is commonly used on the outputs of a deep ANN, because it results in normalized and strictly positive class likelihoods. In this case output spikes are triggered by an external Poisson generator with fixed firing rate. To determine if a neuron should spike, we compute the softmax on the membrane potentials of all neurons, and use the resulting values in range of [0, 1] as rate parameters in a Poisson process for each neuron.

The max-pooling operation mechanism is based on output units contain gating functions that only let spikes from the maximally firing neuron pass, while discarding spikes from other neurons. The gating function is controlled by computing estimates of the pre-synaptic firing rates, e.g., by computing an online or exponentially weighted average of these rates.

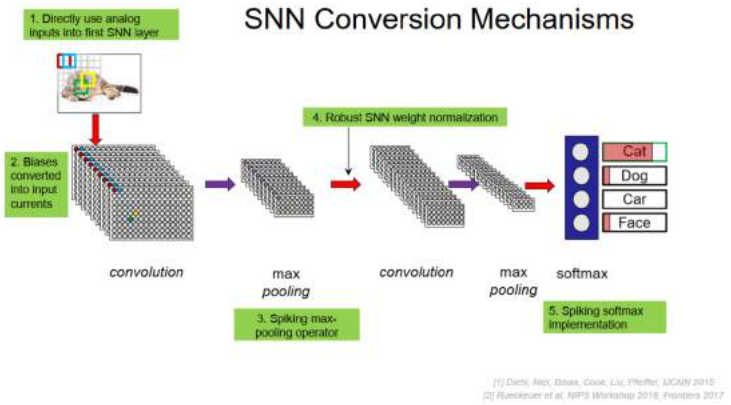


Figure 131: Converting different modules of ANNs into SNNs.

Alternatively, the converted SNN can be exported for use in spiking simulators like pyNN or Brian2.

12 Neuromorphic Systems 2

12.1 Neuromorphic approaches

Neuromorphic Engineering has its roots in the in the early 70' *subterranean group*, formed by Carver Mead, Max Delbruck and Paul Mueller. This group studied membrane channel biophysics showing the all-or-nothing property of single channels and that membrane conductance was a function of the fractional population of channels in the open state. The group also developed methods based on statistics of current flow that enabled them to measure single channel current and the shape of the energy barrier.

After the group ended its work, Carver turned his whole lab to the pursuit of neuromorphic electronic engineering while working on the first silicon retina chip with Misha Mahovald, an undergraduate biology student. They published it in 1991. Also in 1991, Misha Mahovald and Rodney Douglas proposed a conductance-based silicon neuron and showed that it had properties remarkably similar to those of real cortical neurons.

Neuromorphic computing is today present in many digital and mixed-signal devices, encompassing a wide-range of approaches. Some of examples are:

1. ARM-based simulated spiking neural networks: SpiNNaKer - a custom digital computing architecture that is fully programmable, provides fixed-point arithmetic in pseudo real-time and is optimized for spiking computational neuroscience simulations

2. Asynchronous simulation of neurons and synapses: Loihi - an asynchronous multi-core chip for simulating spiking neural networks and on-chip learning. It has a low-power consumption and is optimised for prototyping and scalability.

3. Large-scale asynchronous spiking neural network: TrueNorth - an asynchronous multi-core, large-scale spiking neural network fabric. It is reconfigurable, has binary (1 bit) synapses and computes everything in pseudo real-time with 1 ms frames. It is optimized for low-power and low-latency processing tasks.

4. Wafer-scale analog neural accelerators: BrainScaleS - a full custom large-scale analog-digital neural processing system. It has limited precision due to it's analog components but works in accelerated time (x1000) and is designed for faithful reproduction of neuroscience simulations.

5. Real-time neuromorphic large-scale system: BrainDrop - a full custom analog-digital neural engineering framework computing engine. It works in analog subthreshold so it is noisy and has limited precision. It is a real-time system with ultra-low power-consumption that is scalable but has no on-chip learning.

6. Real-time on-line learning neuromorphic chips: DYNAPs - a full custom analog-digital models of cortical circuits. Uses analog subthreshold and thus is noisy and has limited precision but provides real-time and ultra-low power computation. It is scalable and onlie reconfigurable. It is desinged for implementing principles of neural computatoin in real-time behaving agents.

Thus all the efforts put into this field can be divided in three large directions:

1. Neuromorphic Engineering - represents the fundamental research done in this field on emulating neural function using subthreshold analog circuits and asynchronous digital chips.

2. Neuromorphic Computing - is application driven and can be seen as a conservative approach including simulating neural networks and building custom VLSI circuits.

3. Neuromorphic Devices - is oriented towards new emerging memory technologies like memristive devices, the use of in-memory computing and high-density arrays.

12.2 Neuromorphic electronic circuits

12.2.1 The approach

The neuromorphic engineering approach is centered on learning to build artificial neural processing systems that can interact intelligently with the physical world. It combines multiple disciplines such as neuroscience physics, computer science, electrical engineering etc. It exploits device physics to directly emulate biophysical properties of neural systems. Biophysical processes happen in time so in this approach time is represented implicitly in the physical processes the produce emulating computation, without explicit time references in these circuits. This approach aims to implement robust computation in autonomous egents leading to cognitive behavior.

Why analog subthreshold? See the first part of the previous lecture.

Neuromorphic computation has represented a radical paradigm shift by challenging well-established assumptions poised by conventional computing architectures (e.g. von Neumann).

The first essential property of neuromorphic computing is this ability to exploit physical space more efficiently through fine grain parallelism, not needing time-multiplexing of multiple tasks onto common resources. Memory and computation are co-localized reducing data transfer time and storage problems. Neurons generally don't activate all at the same time (excepting in case of conditions like Epilepsy), thus they can be though of as sparsely activating in space and time, as opposed to conventional computers which can use most of their computing resources at a point in time. Neuromorphic circuits can be driven only at the arrival of an input signal, thus they can inherently exploit the data-driven computational paradigm.

The second property is that time is represented implicitly allowing for real-time interaction with the environment. By modulating circuit time constants, one can adjust responses with input signal dynamics. This property allows neuromorphic circuits to be synchronized with real-world events by default while providing low-power and low-bandwidth processing of sensory signals.

12.2.2 Analog circuits

Current-mode CMOS circuits operated in the subthreshold, or weak-inversion, regime can be used to implement log-domain filters. Current-mode log-domain

CMOS filters have favorable properties, such as wide dynamic range at low supply voltage, compactness, linearity and low power consumption. These properties are becoming increasingly important for biomedical applications that require extremely low-power dissipation and neuromorphic circuits that attempt to reproduce the biophysics of biological neurons and synapses. [cite giacomo ..]

An example of classical log-domain integrator is presented in Figure 132. This circuit's linear transfer function can be easily derived by applying the translinear principle on the V_{gs} loop highlighted by the arrows: given the exponential relationship between the subthreshold currents of the p-FETs and their V_{gs} voltages we can write: $I_{th} \cdot I_1 = I_2 \cdot I_{out}$. In subthreshold, the output n-FET M_{out} produces a current that changes exponentially with its gate voltage V_c. Differentiating I_out with respect to V_c and combining the result with the capacitor equation $C\frac{d}{dt}V_C = I_2 - I_\tau$ we obtain:

$$\tau \frac{d}{dt} I_{out} + I_{out} = \frac{I_{th}}{I_\tau} I_{in} \tag{192}$$

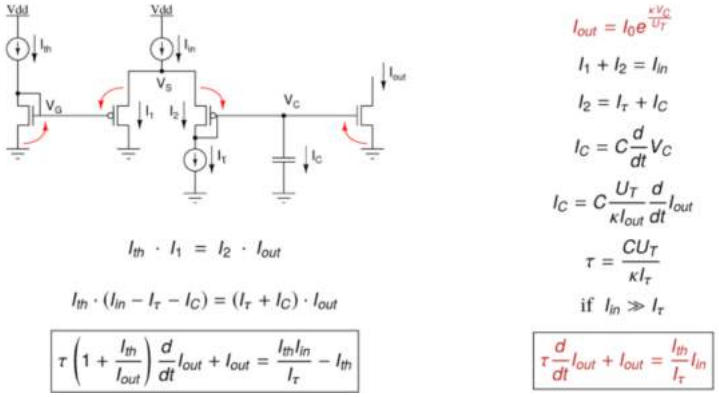


Figure 132: Classical log-domain integrator.

The DPI is a CMOS current-mode circuit that operates in the subthreshold regime integrating voltage pulses. However, rather than using a single pFET to generate the appropriate I_w current, via the translinear principle (Gilbert, 1975), it uses a differential pair in negative feedback configuration. This allows the circuit to achieve LPF functionality with tunable dynamic conductances: Input voltage pulses are integrated to produce an output current that has maximum amplitude set by V_w, V_t, and V_{thr}. [silicon neuron circuits] It has additional advantages of providing a compact layout, better matching properties and lower power consumption. The **differential-pair integrator** is used to model synaptic dynamics.

It comprises only 3 n-FETs, 2 p-FETs and 1 capacitor. The two current sources are implemented using two subthreshold MOSFETs: one n-FET for the I_{in} current and one p-FET for the I_t current. Following a similar derivation to the one used in the classical log-domain integrator, the characteristic equation is obtained as observed in Figure 133

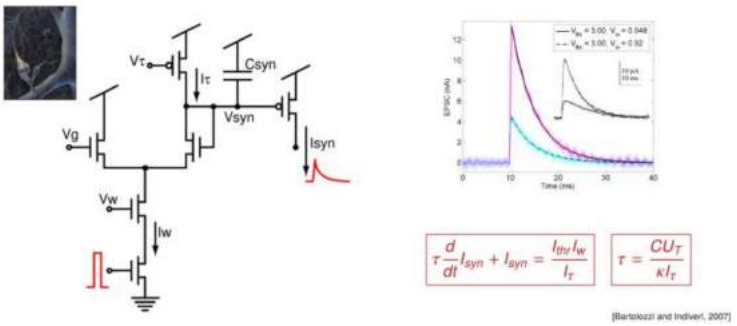


Figure 133: Differential Pair Integrator circuit.

Additional circuits can be attached to the DPI synapse to extend the model with extra features typical of biological synapses and implement various types of plasticity. For example, by adding two extra transistors, we can implement voltage-gated channels that model NMDA synapse behavior. Similarly, by using two more transistors, we can extend the synaptic model to be conductance based (Kandel, Schwartz, Jessell, 2000). Furthermore, the DPI circuit is compatible with previously proposed circuits for implementing synaptic plasticity, on both short timescales with models of short-term depression (STD) (Rasche Hahnloser, 2001; Boegerhausen, Suter, Liu, 2003) and on longer timescales with spike-based learning mechanisms, such as spike timing-dependent plasticity (STDP).

The DPI neuron circuit is a variant of the generalized IF neuron and is depicted in Figure 134. The input DPI low-pass filter (yellow, ML1 - ML3) models the neuron's leak conductance. A spike event generation amplifier (red, MA1 - MA6) implements current-based positive feedback (modeling both sodium activation and inactivation conductances) and produces address-events at extremely low-power. The reset block (blue, MR1 - MR6) resets the neuron and keeps it in a reset state for a refractory period, set by the V_{ref} bias voltage. An additional DPI filter integrates the spikes and produces a slow after hyper-polarizing current I_g responsible for spike-frequency adaptation (green, MG1 - MG6).

By applying a current-mode analysis to both the input and the spike-frequency adaptation DPI circuits, it is possible to derive a simplified analytical solution:

$$\tau \frac{d}{dt} I_{mem} + I_{mem} \approx \frac{I_{th} I_{in}}{I_\tau} - I_g + f(I_{mem}) \tag{193}$$

$$\tau_{ahp} \frac{d}{dt} I_g + I_g = \frac{I_{thr} I_{ahp}}{I_{\tau_{ahp}}} \tag{194}$$

The state of the art version of this neuron circuit consumes one order of magnitude less power than the circuit depicted in Figure 134 and two orders of magnitude less power than the digital implementation of the I&F neuron.

Given the exponential nature of the generalized IF neuron's non-linear term $f(I_{mem})$, the DPI-neuron implements an adaptive exponential IF model (Brette and Gerstner, 2005). This IF model has been shown to be able to reproduce a wide range of spiking behaviors, and explain a wide set of experimental measurements from pyramidal neurons. [silicon neuron circuits]

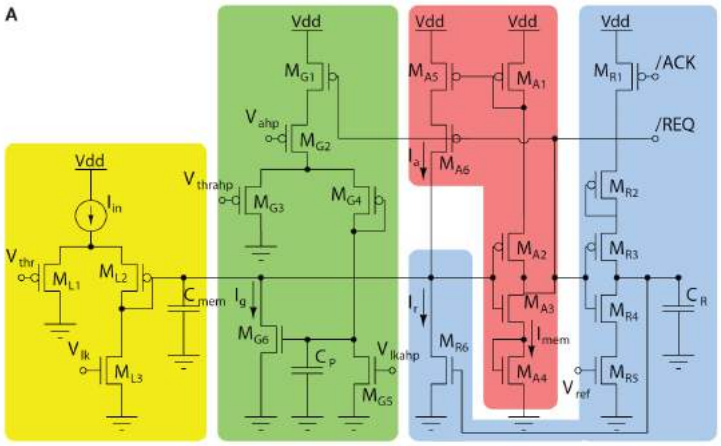


Figure 134: Differential Pair Integrator Neuron.

12.2.3 Neuromorphic processors

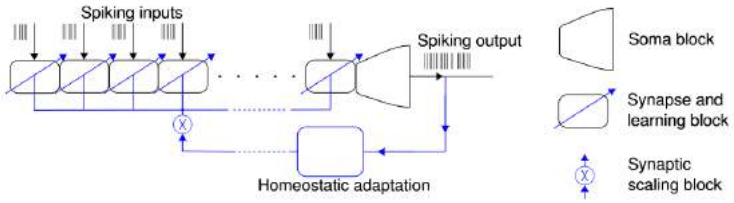


Figure 135

Typical spiking neural network chips have the elements described in Figure 135. Multiple instances of these elements can be integrated onto single chips and connected among each other either with on-chip hard-wired connections or via off-chip reconfigurable connectivity infrastructures. The most relevant characteristics of processors build based on analog circuits working in subthreshold are:

- Slow temporal, non-linear dynamics
- Massively parallel operation
- Inhomogeneous, imprecise and noisy
- Adaptation and learning is done at multiple time-scales
- Fault tolerant and mismatch insensitive by design
- Fast asynchronous digital routing circuits
- Re-programmable network topology and connectivity

12.3 Neuromorphic processing chips

Neuromorphic agents

When building processing chips, one can be inspired from life-forms that have brain and use them in every-day tasks, like mammals, insects etc. The physical properties of computational elements in brains can also be used as a reference from the computational resource usage standpoint, for example a bee's brain has a weight of 1 mg, a volume of 1 mm^3 and manages to squeeze almost 1 million neurons in it. The energy per operation is approximated to $10^{-15} J/spike$. Functional principles used by these life-forms can be used in the development of neuromorphic agents. Some of these are:

1. Exploiting physical space in the most efficient way possible (e.g. squeezing cortical collumns in the neocortex)
2. Let time represent itself
3. Use both analog and digital computing elements
4. Exploit non-linearities and temporal dynamics
5. Leverage noise, variability and stochasticity
6. Minimize wiring (maximize local connectivity)
7. Optimize for processing complex spatio-temporal (dynamic and noisy) signals
8. Re-use computational principles for sensory processing, motor control and cognitive computing

DYNAP-SEL

The DYNAP-SEL (Dynamic Neuromorphic Aynch Processor with Self Learning) is built based on an analog and digital co-design. It has on-chip inference and learning but also provides a reconfigurable architecture. The SRAM and TCAM memory cells are distributed and it uses capacitors for state dynamics. It is considered to be ideal for integration with binary, non-volatile resistive memory devices, for integration with dynamic, volatile/non-volatile memristive devices and can be integrated in 3D VLSI technology. It has a batter Fan In-/Out and can do 12x more synaptic operations per second per watt than True North. It also uses 5x less energy per synaptic event and more than 2x less energy per spike.

Neuromorphic vs. conventional processors

Pros	Cons
Low-latency	Limited resolution
Ultra low-power	High variability, noisy

Programming a neuromorphic chip involves:

Good	Bad
Real-time sensory-motor processing	High accuracy pattern recognition
Sensory fusion and on-line classification	High precision number crunching
Low-latency decision making	Batch processing of data sets

1. defining the neural computational primitives
2. composing multiple primitives for context dependent processing
3. configuring the network structure and parameters
4. training the neural network with different learning methods

12.3.1 Configuring

Robot navigation

In this work, a mixed-signal analog-digital neuromorphic processor ROLLS was interfaced with a neuromorphic dynamic vision sensor (DVS) mounted on a robotic vehicle. An autonomous neuromorphic agent that is able to perform neurally inspired obstacle-avoidance and target acquisition was developed. The implemented neuronal architectures for obstacle avoidance, presented in Figure 136: Violet OL and OR circles represent obstacle detecting neuronal populations. Orange DL and DR circles are motor driving populations. sp is the speed-setting population, and exp is the constantly firing population that sets the default speed. Thin arrows show excitatory non-plastic connections realized on the ROLLS chip, whereas colors and numbers show the weights.

On-line detection of iEEG High-Frequency Oscillations (HFO)

An HFO is a series of spontaneous EEF events in the frequency range between 80 and 500 Hz consisting of at least four oscillations that clearly stand out from the baseline. HFO in human iEEG are used to identify epileptogenic brain tissue during epilepsy surgery. However, current methods typically analyse the raw data offline using complex time-consuming algorithms so NCS group at INI developed a compact neuromorphic sensory processing system-on-chip that can monitor the iEEG signals and detect high frequency oscillations in real-time using spiking neural networks.

The integrated device has an analog front-end that can extract predefined spectral features and encode them as address-events, and a neuromorphic processor core that implements a network of integrate and fire neurons with dynamic synapses. The front-end amplifies the bio-signals, filters them in specific frequency bands and converts filter outputs to asynchronous events. The DYNAP block is configured to implement a spiking neural network which receives these events, processes them via its dynamic synapses and neurons, and detects HFO in two different frequency bands.

The Low Noise Amplifier is the most critical block employed in the front-end, which ensures linear amplification and systematic noise suppression. It includes an Operational Transconductance Amplifier (OTA) in capacitive feedback configuration with MOS Bipolar structure as resistive elements.

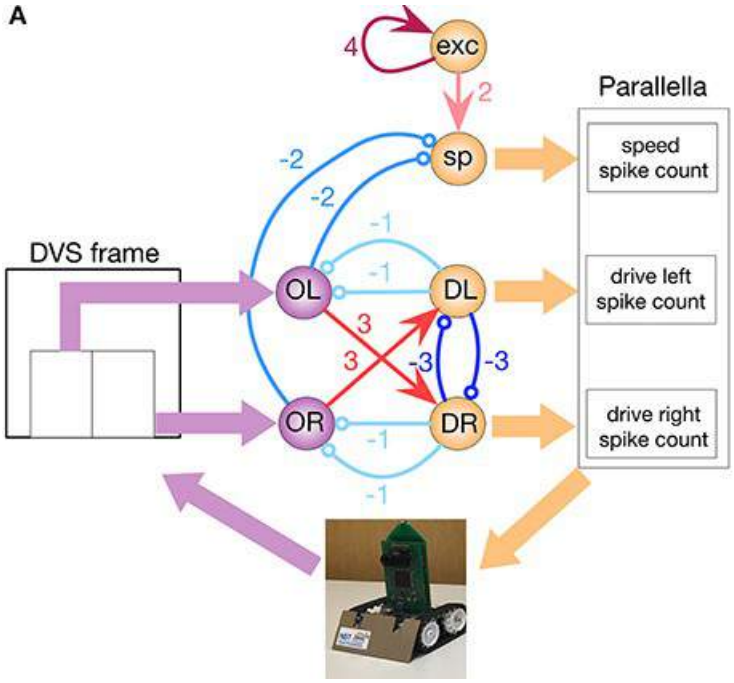


Figure 136: Robotic navigation using neuromorphic processors.

Another important component of the analog front-end is the asynchronous ADC, which enables the analog front-end to interface with the event-based processing core and translates the input data to a sequence of spikes. Here, each spike corresponds to a polarized, adjustable amount of change in the analog input signal.

The estimated power consumption of the front-end is $6.2W$/channel and the area-on-chip for a single channel is 0.15 square millimetres. The SNN classifier provides 90.5% sensitivity and 67.7% specificity for detecting high frequency oscillations. This is the first feasibility study towards identifying relevant features in intracranial human data in real-time on-chip using event-base processors. [cite from lecture]

12.3.2 On-line classification of EMG signals

An accurate description of muscular activity plays an important role in the clinical diagnosis and rehabilitation research. The electromyography (EMG) is the most used technique to make accurate descriptions of muscular activity. The EMG is associated with the electrical changes generated by the activity of the motor neurons. Typically, to decode the muscular activation during different movements, a large number of individual motor neurons are monitored simultaneously, producing large amounts of data to be transferred and processed by the computing devices. In this paper, an alternative approach is followed, that can be deployed locally on the sensor side. It proposed a neuromorphic implementation of a spiking neural network (SNN) to extract spatio-temporal information of EMG signals locally and classify hand gestures with very low-

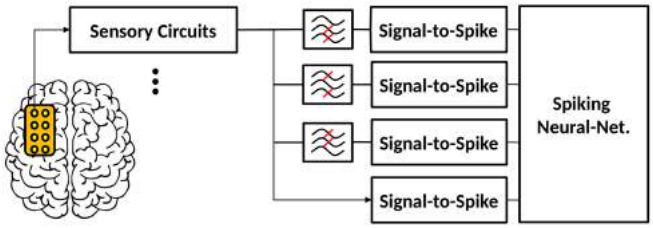


Figure 137: On-line detection of iEEG HFO.

power consumption. A mixed-signal analog/digital neuromorphic processor was used

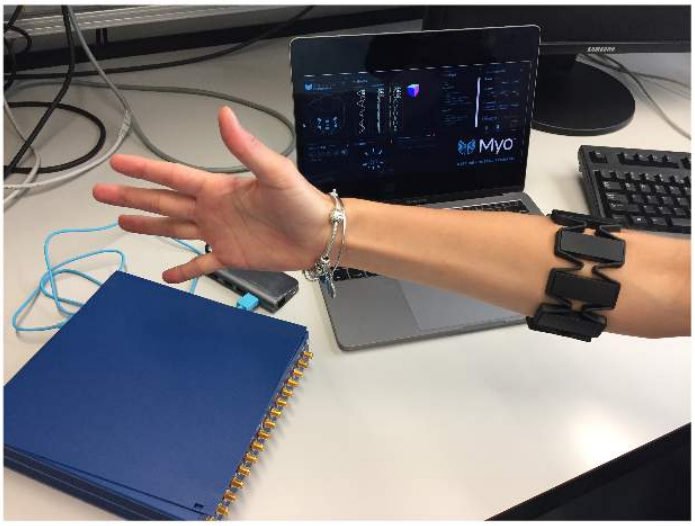


Figure 138: MYO device used obtain EMG signals

EMG signals related to two movements of open and closed hand gestures were recorded, then converted into asynchronous Address-Event Representation (AER) signals. These signals were provided as input to a recurrent spiking neural network implemented on an ultra-low power neuromorphic chip and the chip's response was analyzed. The recurrent network was configured as a Liquid State Machine (LSM) as a means to classify the spatio-temporal data. The Separation Property (SP) of the liquid states was evaluated for the two movements. Results show that the activity of the silicon neurons can be encoded in state variables for which the average state distance is larger between two different gestures than it is between the same ones measured across different trials.

The LSM type of RNNs proposes the ingenious idea of a two layer network comprised of spiking neurons with dynamic synapses in which the first layer is set up with randomly connected recurrent weights to project the input signal to a higher dimensional space. The complex dynamics of the synapses and neurons,

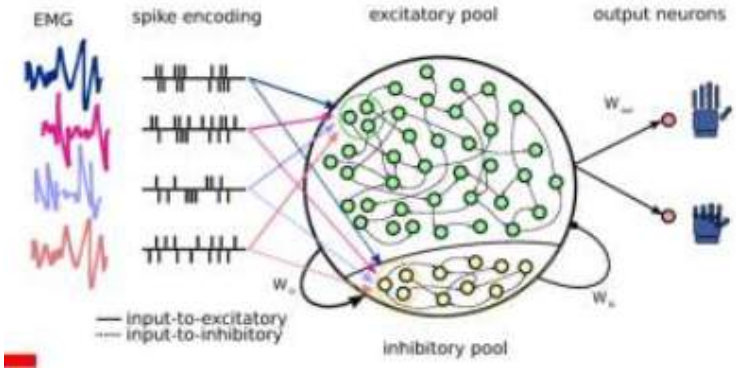


Figure 139: Processing pipeline of input EMG data, encoded in AER, passing through the reservoir and ending into the actuators.

and the high dimensional space provide a pool of basis functions on which the projected data can be separated because of the Kernel properties of the LSM. As a result, the second layer can simply be trained on the basis of a gradient decent learning rule which is powered by the already separated data from the first random layer. The synapse and neural dynamics are directly and faithfully emulated by the analog circuits of neuromorphic chips and the high variability of the LSM, imposed in software by the randomness of the first layer weights is easily achieved on the neuromorphic chips when using sub-threshold analog circuits to implement the neuron and synapse circuits. [cite from lecture]

12.3.3 Learning

There exist many pre-synaptic (input) driven learning algorithms that are hardware friendly, but all have in common 3 main aspects:

- Bi-stable synapses
- Redudancy
- Variability

Ensemble learning

Intrinsic variability and diverse activation patterns are often identified as fundamental aspects of neural computation for information maximization and transmission. The strategy of combining large numbers of variable and imprecise computing elements to carry out robust computation is also followed by a wide set of traditional machine learning approaches. These approaches work on the principle of combining the output of multiple inaccurate computational modules that have slightly different properties, to optimize classification performances and achieve or even beat the performances of single accurate and complex learning systems. A set of similar theoretical studies showed that the coexistence of multiple different time-scales of synaptic plasticity (e.g., present due to mismatch in the time-constants of the DPI synapse circuits) can dramatically improve the memory performance of ANN. [chica et al]

Hopfield/attractor networks

Mechanisms operating at the network level can also allow neural processing systems to form short-term memories, consolidate long-term ones, and carry out nonlinear processing functions such as selective amplification (e.g.,to implement attention and decision making). An example of such a network-level mechanism is provided by "attractor networks." These are networks of neurons that are recurrently connected via excitatory synapses, and that can settle into stable patterns of firing even after the external stimulus is removed. Different stimuli can elicit different stable patterns, which consist of specific subsets of neurons firing at high rates. Each of the high-firing rate attractor states can represent a different memory. To make an analogy with conventional logic structures, a small attractor network with two stable states would be equivalent to a flip-flop gate in CMOS.

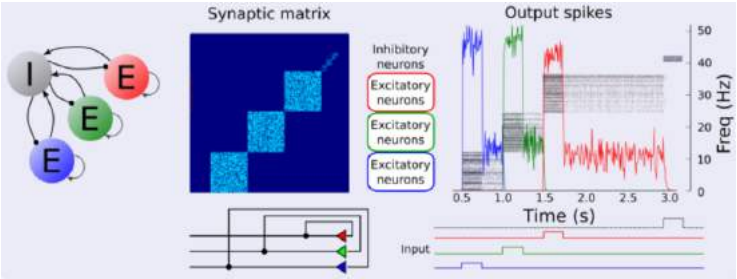


Figure 140: Hopfield - attractor - winner-takes-all networks

A particularly interesting class of attractor networks is the one of soft winner-take-all (sWTA) neural networks. In these networks, groups of neurons both co-operate and compete with each other. Cooperation takes place between groups of neurons spatially close to each other, while competition is typically achieved through global recurrent patterns of inhibitory connections. When stimulated by external inputs, the neurons excite their neighbors and the ones with highest response suppress all other neurons to win the competition. Thanks to these competition and cooperation mechanisms, the outputs of individual neurons depend on the activity of the whole network and not just on their individual inputs. [indiveri,liu 205]

Spike-based back-propagation

Discussed in Zenke's lecture.

Long eligibility traces

To achieve long biological realistic time-scales that do not interfere with signal transmission and learning, it is necessary to develop circuits with time scales that range from seconds to hours. In order to optimize the circuit's area to allow the dense integration of thousands of synapses and neurons on a single chip, the capacitance of the homeostatic control circuits must be small, therefore long time scales can only be achieved by extremely small currents. An example of such a circuit is the ultra-low leakage cell shown in Figure 141. This circuit

increases or decreases its output voltage V_{thr} by controlling the direction of a very small current across the channel of the LLC p-FET to slowly charge or discharge the capacitor C_F.

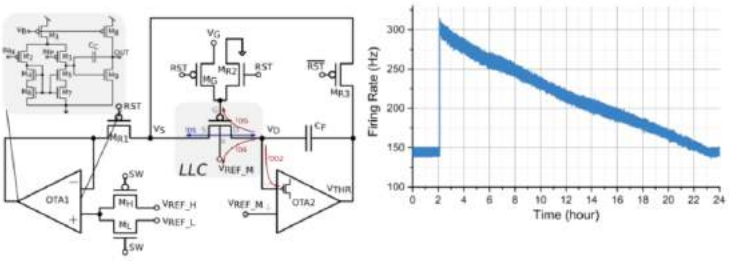


Figure 141: Low Leakage Cell and its long eligibility trace

Three factor learning circuits

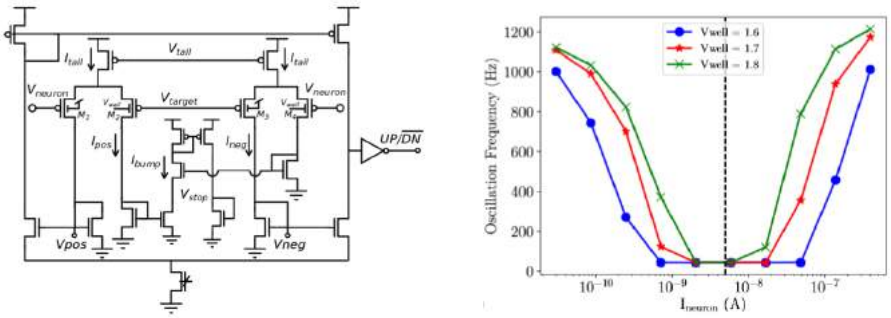
Event-driven neuromorphic hardware with on-line learning capabilities enables the low-power local processing of signals on the edge sensors. Implementing such hardware requires having an always-on online learning operation in order to continuously adapt to the changes in the environment. Therefore, as the data is continuously streaming, there cannot be a separation between the training and the testing phase. Such constraint thus asks for a continuous time learning strategy which includes a mechanism to stop changing the weights when the system has reached an optimal operating point, so that it does not over-fit the input data and it generalizes to unseen patterns of the learned class. Circuits with spike-based local gradient-descent learning rule are proposed, that comprise also this additional "stop-learning" feature and that have a wide range of configurability options over the learning parameters.

In this paper, the authors propose spike-based circuits implementing a local gradient-descent based learning rule (delta rule) which is empowered by the additional stop-learning feature. This is done through a modified version of the Bump circuit.

The learning circuits implement Delta-rule which is a learning algorithm minimizing the Least Mean Square (LMS) error of a single-layer neural network whose cost function is defined as the difference between a target desired output signal T and the network output signal y, for a given set of input patterns signals x, weighted by the synaptic weight parameters w. Specifically, this learning rule, given the learning rate α, sets the corresponding weight change between the i^{th} input and the j^{th} output neuron to be:

$$\delta w = \alpha(T_j - y_j)x_i \tag{195}$$

This algorithm was previously implemented using the Bump circuit by comparing the rate of the neuron spikes to a target value. The rates are calculated through low-pass filtering the spikes by using a Diff-Pair Integrator (DPI) circuit which receives spike inputs, low pass filters them, and generates output currents proportional to the rate of the spikes. The Bump circuit provides us with the analog value and the direction of the difference between the neuron's



(a) (b)

Figure 142: VWBump circuit and the stop-learning region controlled by the back-gate voltage V_w

and target's spike rate, along with a flag for the similarity between the two signals which is ideal for implementing Delta update rule.

The width of the transfer characteristics of the Bump current is a dead region where the target and the neuron are "similar enough". This dead region directly implements the "stop-learning" region where the error of the network is close to negligible and hence no update in the synaptic weights is necessary. The VW Bump circuit has the advantage of having an electrical control over the width of the bump current characteristic function.

12.4 Computational primitives

12.4.1 Cortical microcircuit

The cortical microcircuit explains the intracellular responses to pulse stimulation in terms of the interactions between three basic populations of neurons, and reveals the following features of cortical processing that are important to computational theories of neocortex. First, inhibition and excitation are not separable events. Activation of the cortex inevitably sets in motion a sequence of excitation and inhibition in every neuron. Second, the thalamic input does not provide the major excitation arriving at any neuron. Instead the intracortical excitatory connections provide most of the excitation. Third, the time evolution of excitation and inhibition is far longer than the synaptic delays of the circuits involved. This means that cortical processing cannot rely on precise timing between individual synaptic inputs.

Three populations of neurons interact with one another: one population is inhibitory (GABA cells, solid synapses), and two are excitatory (open synapses), representing superficial (P2 + 3) and deep (P5 + 6) layer pyramidal neurons. The layer 4 spiny stellate cells are incorporated with the superficial group of pyramids. Each population receives excitatory input from the thalamus, which is weaker (dashed line) to deep pyramids. The inhibitory inputs activate both $GABA_A$ and $GABA_B$ receptors on pyramidal cells. The thick line connecting GABA tO P5+6 indicates that the inhibitory input to the deep pyramidal population is relatively greater than that to the superficial population. However, the increased inhibition is due to enhanced $GABA_A$ drive only. The $GABA_B$

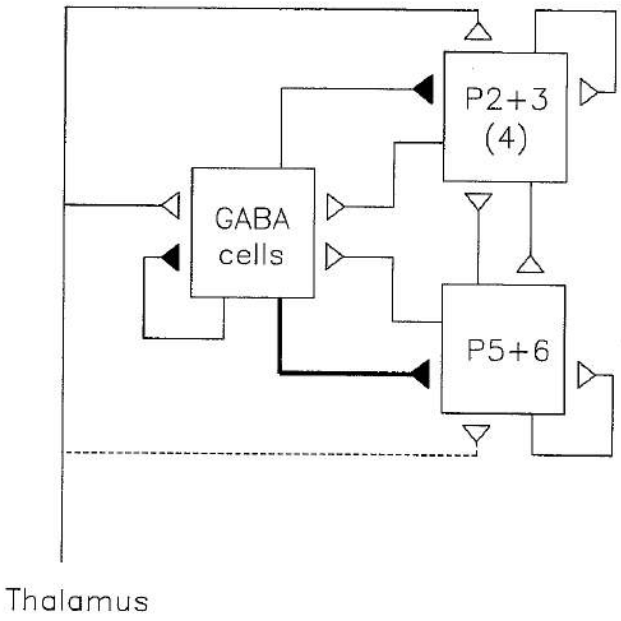


Figure 143: Model of cerebral cortex that successfully predicts the intracellular responses of cortical neurons to stimulation of thalamic afferents

inputs to p5+6 is similar to that applied to P2 + 3.

12.4.2 WTA

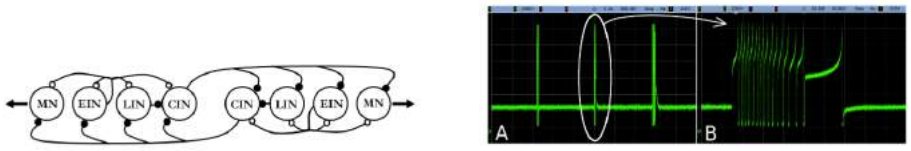
Discussed in section 12.3.3.

12.4.3 Intrinsic oscillators and Central Pattern Generators

In animals, a basic block of locomotor control is the Central Pattern Generator (CPG), a neural network capable of generating coordinated pattern of rhythmic activity, driven by very simple input signals, which are sufficient to modulate locomotion patterns.

In particular the animal that has been mostly studied in this context is the lamprey. In the lamprey the presence of mutually inhibitory connections organizes neurons in antagonist segments or units. The spinal cord system consists of about 100 units (segments), each containing an oscillatory neural network. In each of these segments the left-right alternated spiking activity has a frequency ranging from about 0.1 Hz to 8-10 Hz. Each side of a segment can generate bursting activity, independent of the other side.

The setup used to implement the neural model comprises a mixed signal analog/digital VLSI device interfaced, for prototyping purposes, to a commercially available FPGA board. To provide the model's input signal, we first created an oscillatory network on the chip. This network is composed of two pools of 4 silicon neurons that are sparsely and recurrently connected. Connectivity within each pool has ratio $c_{rec} = 0.5$ (indicates the probability that a neuron



(a) (b)

Figure 144: A. Spinal cord CPG unit, comprising two sides with multiple types of neurons. The neurons are coupled by both excitatory and inhibitory connections. Filled circles represent inhibitory synapses while open circles the excitatory. B. Silicon neuron measurements

in the pool connects to any other neuron in the same pool). The two pools inhibit each other via inhibitory connections with a connectivity ratio $c = 0.5$. This interplay of recurrent excitation, spike-frequency adaptation and external inhibition creates a stable pattern of oscillation, when neurons are driven by a constant injection current.

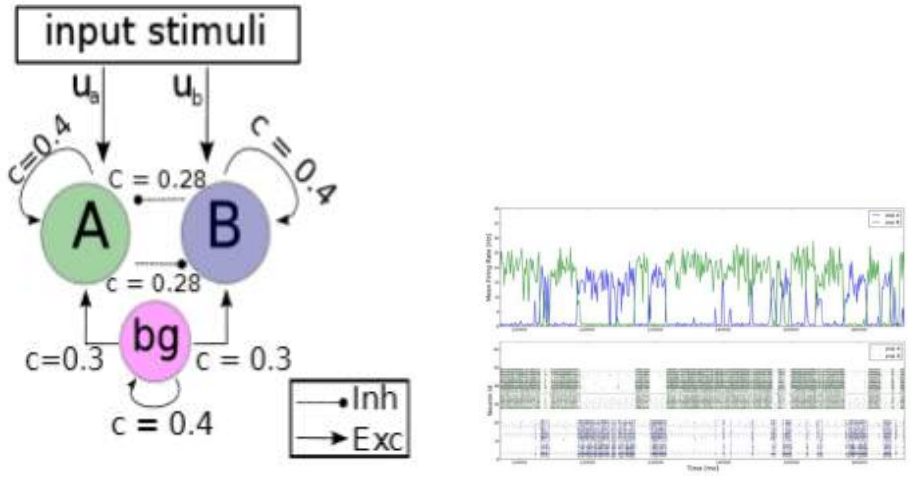
The neuron produces periodic bursting, lasting approximately 60 ms. with an inter-burst interval of about 1.5 s. The spiking frequency during the burst is about 35 Hz.

12.4.4 Perceptual bistability

The ability of embodying information in the dynamics of a recurrent neural network, which can persist also in the absence of external stimulation and transition between meta-stable states, represents a fundamental processing capability of neural systems. In biologically-inspired neural network models, it has often been assumed that an attractor in phase space represents an internal or an external source of information. From a biological perspective, recurrent spiking neural network models have expressed the dynamics of bistability in their firing rates. The architecture of the network is illustrated in Figure 145 is organized in three populations of neurons A, B and background (bg). The number of neurons in population A and B is $Na = Nb = 22$, while the background population counts $Nbg = 12$ neurons. Each population is recurrently connected, with sparse connectivity. Populations A and B inhibit each other via direct inhibitory connections.

Long-lasting spiking activity in the cortex, that lasts much longer than the typical time scales of synapses and membrane potentials, is the neural correlate of working memory and perceptual decision making. In theoretical neuroscience models this activity is assumed to encode sensory inputs which relate to different alternative possible decisions or percepts. This activity is accumulated in different pools of neurons that, due to their recurrent excitatory connections, are capable of sustaining persistent activity, even after the input stimulus that triggered it is removed.

The observed behaviour of the neuromorphic attractor network, when all neurons are stimulated by a constant injection of current, is an alternation of perceptual dominance among the two different activity states with very long time constants, orders of seconds. Figure 145 shows five seconds of recordings of such behaviour. It shows the mean firing rate activity over time of neurons



(a) (b)

Figure 145: A. Network structure B. Neuromorphic chip measurements

grouped in populations. Continuous green line depicts neurons in population A, and the dashed blue line represents neurons in population B. An irregular alternation of high activity is evident, and only one of the two populations of neurons can be found in the up state (80 Hz) of activity: this is a confirmation that the network is operating in a winner-take-all regime.

12.4.5 Neural state machines

A finite-state machine (FSM) is a mathematical model of computation used to design both computer programs and sequential logic circuits. It is conceived as an abstract machine that can be one of a finite number of states. State-dependent computation is one of the main signatures of cognition. Recently, it has been shown how it can be used as a computational primitive in spiking neural networks for constructing complex cognitive behaviors in neuromorphic agents. In spiking neural networks, Neural State Machines (NSMs) provide a generic computational model for implementing state-dependent computations. Similar to finite-state automata, NSMs are defined by four sets of Boolean variables, which include a set of states $S = s1, s2, s3...$, a set of excitatory input signals $E = e1, e2, e3...$, a set of inhibitory input signals $I = i1, i2, i3...$, and a set of transition signals $T = t1, t2, t3....$ Each transition of T links one source state of S to at least one target state of S and is also linked to an additional set of input signals. The co-occurrence of input signals and source state signals are used to switch transitions on and off.

Multiple NSMs can interact with each other. They have been used as a modular building block in Spiking Neural Networks (SNNs) to construct complex cognitive computations in neuromorphic agents, such as solving Constraint Satisfaction Problems (CSPs). Synthesis of a target FSM in neuromorphic VLSI neural networks is presented in Figure 147.

Subfigure (A) contains the state diagram of the high-level behavioral model.

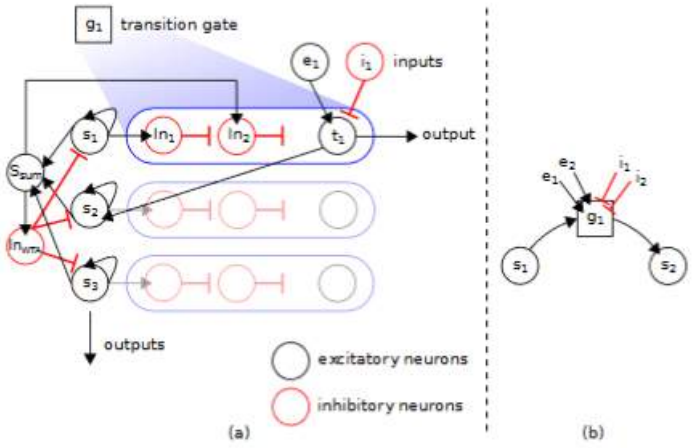


Figure 146: NSM structure its schematic representation

Circles represent states and arrows indicate the transitions between them, conditional on input symbol X. In this example state machine, the active state flips between S_1 and S_2 in response to X and outputs either the response A or the response B, depending on the previous state.

Subfigure (B) represents the computational substrate composed of three sWTA networks: two "state-holding" networks (vertical and horizontal rectangles) and a third transition network (central square). The shaded circles in each sWTA represent populations of spiking neurons that are in competition through a population of inhibitory neurons (not displayed). The state-holding sWTA networks are coupled population-wise (-labeled arrow, red with red, blue with blue, etc.) to implement working memory. Solid arrows indicate stereotypic couplings, and the dashed arrows indicate couplings that are specific to the FSM (in this case the one shown in A). The gain and threshold in the transition sWTA are configured such that each population becomes active only if both of its inputs are presented together. The sWTA competition ensures that only a single population in the network is active at any time. An additional output sWTA network is connected to the transition network to represent the output symbols. To program a different state machine, only the dashed arrows need to be modified.

Subfigure (C) depicts the multineuron chips used in the neuromorphic setup, whicj feature a network of low-power IF neurons with dynamic synapses. The chips are configured to provide the hardware neural substrate that supports the computational architecture consisting of sWTA shown in B. Each population of an sWTA network is represented in hardware by a small population of recurrently coupled spiking neurons ($N = 16$), which compete against other populations via an inhibitory population. [neftci 2013]

12.4.6 WTA-based constraint-satisfaction problems

Mostafa et al. proposes a recurrent neural network modeled as a continuous-time dynamical system, that can solve constraint satisfaction problems. Discrete variables are represented by coupled Winner-Take-All (WTA) networks, and

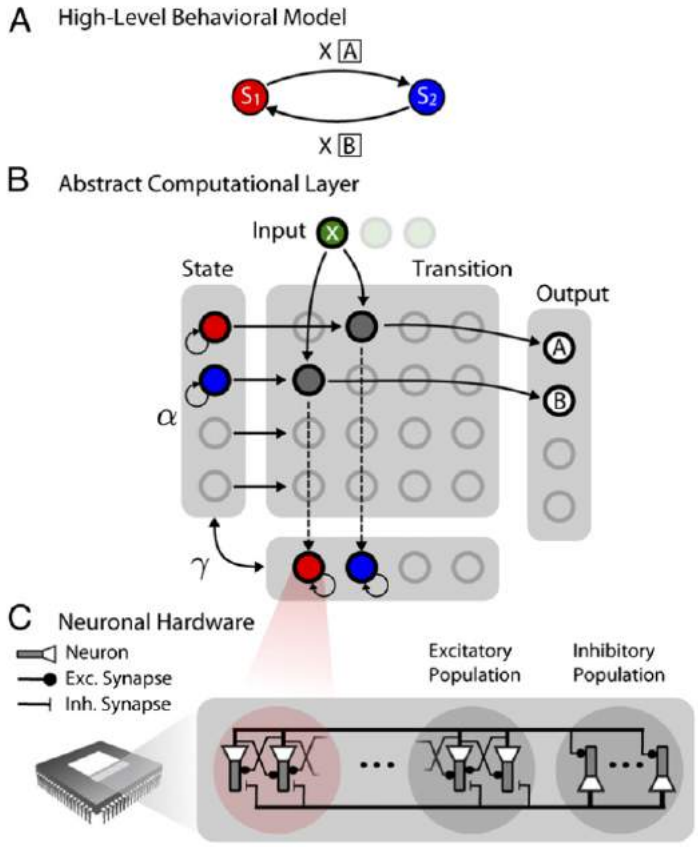


Figure 147: FSM synthesis

their values are encoded in localized patterns of oscillations that are learned by the recurrent weights in these networks.

The basic building block of the proposed network is the WTA circuit in which multiple excitatory populations are competing through a common inhibitory population. When the excitatory populations of the WTA network receive inputs of different amplitudes, their activity will increase and be amplified due to the recurrent excitatory connections. This will in turn activate the inhibitory population which will suppress activity in the excitatory populations until an equilibrium is reached. Typically, the excitatory population that receives the strongest external input is the only one that remains active (the network has selected a winner). By properly tuning the connection strengths, it is possible to configure the network so that it settles into a stable state of activity (or an attractor) that persists after input removal.

Constraints over the variables are encoded in the network connectivity. Although there are no sources of noise, the network can escape from local optima in its search for solutions that satisfy all constraints by modifying the effective network connectivity through oscillations. If there is no solution that satisfies all constraints, the network state changes in a seemingly random manner and its

trajectory approximates a sampling procedure that selects a variable assignment with a probability that increases with the fraction of constraints satisfied by this assignment. External evidence, or input to the network, can force variables to specific values. When new inputs are applied, the network re-evaluates the entire set of variables in its search for states that satisfy the maximum number of constraints, while being consistent with the external input.

The results demonstrate that the proposed network architecture can perform a deterministic search for the optimal solution to problems with non-convex cost functions. The network is inspired by canonical microcircuit models of the cortex and suggests possible dynamical mechanisms to solve constraint satisfaction problems that can be present in biological networks, or implemented in neuromorphic electronic circuits.

12.5 Conclusion

The main research objective is to implement computational neuroscience models in low-power, mixed signal, hybrid memristive/CMOS technology to build efficient on-line learning and processing systems.

An important guiding principle is exploiting the physics of semiconductors to implement efficient neuromorphic cognitive agents that interact intelligently with the environment.

13 Learning In Recurrent Neuronal Networks (RNNs)

13.1 Motivation: Why is it so important to understand RNN learning?

1. The mammalian cortex is highly recurrent - it will help us to better understand the brain.
2. The core reason that recurrent nets are more exciting is that they allow us to operate over sequences of vectors: Sequences in the input, the output, or in the most general case both. Thus better understanding RNNs might help us to develop new, powerful RNN algorithms that are able to learn long and complex sequences.
3. In programming terms be interpreted as running a fixed program with certain inputs and some internal variables. Viewed this way, RNNs essentially describe programs. In fact, it is known that RNNs are Turing-Complete in the sense that they can be used to simulate arbitrary programs (with proper weights).
4. RNNs perform exceptionally well in language-modeling, the task of predicting the probability of the next word in a sequence.

13.2 RNNs in machine learning

13.2.1 What is a Recurrent Neural Network? (RECAP)

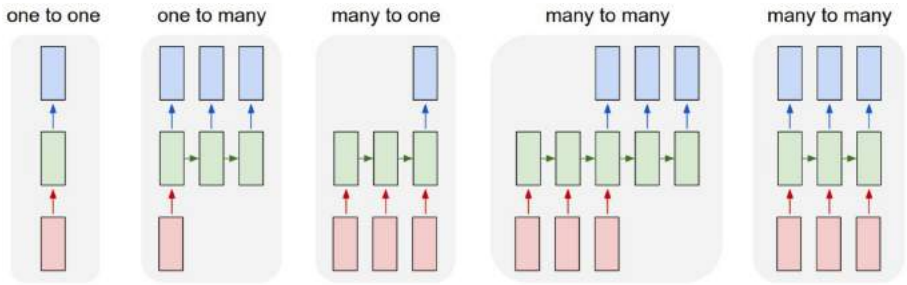


Figure 148: Each rectangle is a vector and arrows represent functions (e.g. matrix multiply). Input vectors are in red, output vectors are in blue and green vectors hold the RNN's state (more on this soon). From left to right: **(1)** Vanilla mode of processing without RNN, from fixed-sized input to fixed-sized output (e.g. image classification). **(2)** Sequence output (e.g. image captioning takes an image and outputs a sentence of words). **(3)** Sequence input (e.g. sentiment analysis where a given sentence is classified as expressing positive or negative sentiment). **(4)** Sequence input and sequence output (e.g. Machine Translation: an RNN reads a sentence in English and then outputs a sentence in French). **(5)** Synced sequence input and output (e.g. video classification where we wish to label each frame of the video). Notice that in every case are no pre-specified constraints on the lengths sequences because the recurrent transformation (green) is fixed and can be applied as many times as we like.

Recurrent Neural Networks (RNNs) add an interesting twist to basic neural networks. A vanilla neural network takes in a fixed size vector as input which limits its usage in situations that involve a 'series' type input with no predetermined size. Recurrent nets allow us to operate over sequences of vectors: Sequences in the input, the output, or in the most general case both (A sequence means, that the elements can have dependency on each other and that the order matters!). A few examples that may make this more concrete[66], are shown in fig. 148. The size of the input or output sequence is flexible, i.e. does not change the architecture of the model. Each network state gets an indices for the sequence. Since the sequence is often related with time progression, the index is choosen to be t. The main difference in architecture compared to conventional ANNs is, that recurrent loops are allowed, i.e. inputs from previous layer states of the network. Looking at a one-to-one neural network with one hidden layer, we can write the output state $y(t)$ and the hidden layer state $h(t)$ as follows:

$$
\begin{aligned}
y(t) &= g_y(W_y h(t) + b_y) && (1)\\
h(t) &= g_h(W_x x(t) + W_h h(t-1) + b_h) && (2)\\
&= g_h((W_{hh} W_{xh}) \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}) \\
&= g_h(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}),
\end{aligned}
$$

where we include the bias in the W matrix. If we want to display the network over all sequences graphically, i.e. the computational graph, we can unroll it as displayed in fig. 151.

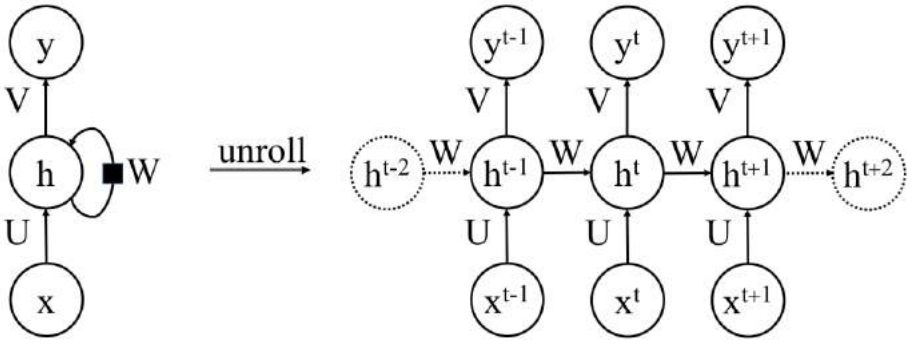


Figure 149: Computational graph of a many-to-many network: Unrolling the network over all sequences.

This gives us another perspective: For any fixed sequence length s, the unrolled recurrent network corresponds to a feedforward network with s hidden layers. The two main differences to a feedforward network is, that the inputs are processed and outputs produced in sequence, and that the same parameters are used for all layers / all time steps, i.e. the same functions U, V, W applied

[66]Cool blog by Karpathy (Teslas AI director): `http://karpathy.github.io/2015/05/21/rnn-effectiveness/`

over all time steps[67](Not to be confused with all epochs!!!).

13.2.2 Back-Propagation Through Time (BPTT)

The unfolding shown in fig. 151 is the first step of a particular network training algorithm, which is called Back-Propagation Through Time (BPTT). The second step is applying our known backpropagation algortihm to the unrolled network to calculate all weight updates.

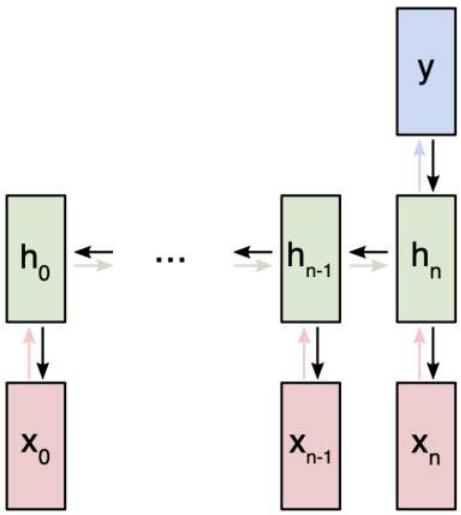


Figure 150: Illustrating the backpropagation-through-time algorithm on a many-to-one RNN.

There are several drawbacks to BPTT:

Costly parameter update: Especially for long sequences, the parameter update for a shallow layer is the same as updating a parameter in an extremely deep feedforward network. One way to fix this is using a truncated BPTT algortithm. It processes the sequence one timestep at a time, and every $k1$ timesteps, it runs BPTT for $k2$ timesteps, so a parameter update can be cheap if $k2$ is small. Consequently, its hidden states have been exposed to many timesteps and so may contain useful information about the far past, which would be opportunistically exploited[68].

Exploding gradients: The gradients coming from the deeper layers have to go through continuous matrix multiplications because of the the chain rule, and as they approach the earlier layers. If they have large values (> 1) they get larger and eventually blow up and crash the model (NaN-values!). This can be solved by gradient clipping; which places a predefined threshold on the gradients to prevent it from getting too large. Note that this only changes the length, and not the direction, of the gradients.

[67]Here another helpful entry: `https://towardsdatascience.com/recurrent-neural-networks-d4642c9bc7ce`

[68]Sutskever, "Training Recurrent Neural Networks", 2013

Vanishing gradients: A similar problem arises if the gradients have small values (< 1). They will shrink exponentially until they vanish and make it impossible for the model to learn. This issue cannot be solved as simple; hence it requires to use shorter sequences or to make fundamental change in the RNN architecture.

13.2.3 Long-Short-Term-Memory (LSTM) Networks

Long Short-Term Memory (LSTM) is a feature of a RNN that tackles the problems arising from long sequences / deep networks by a clever memory management[69] [70]. A common LSTM unit is composed of a cell, an input gate **i**, an output gate **o**, a forget gate **f** and a gate gate **g**. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell. A comparison between a normal RNN cell and a LSTM cell is given in the figure below.

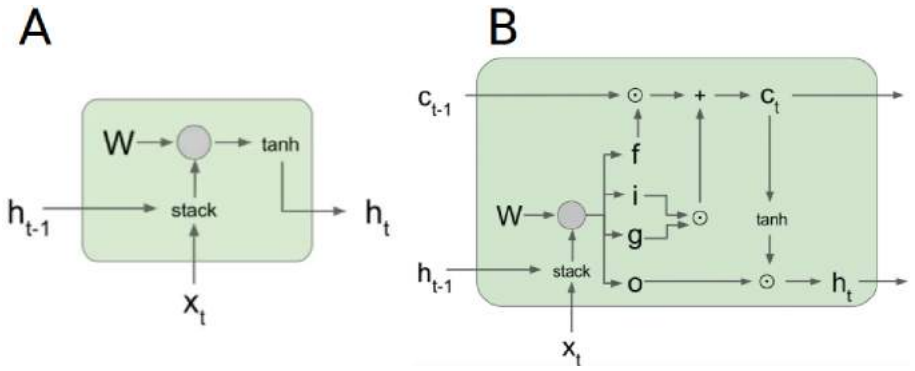


Figure 151: A comparison between a normal RNN cell **A** and a LSTM cell **B**.

The gate vector can be written as

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}, \tag{3}$$

where $W = \begin{pmatrix} W_i & W_f & W_o & W_g \end{pmatrix}$. The cell state is defined as the following:

$$\begin{aligned} c_t =& f \odot c_{t-1} + i \odot g \\ =& \sigma(W_{hf} h_{t-1} + W_{xf} x_t) \odot c_{t-1} \\ &+ \sigma(W_{hi} h_{t-1} + W_{xi} x_t) \odot \tanh(W_{hg} h_{t-1} + W_{xg} x_t) \end{aligned} \tag{4}$$

The hidden state is then given by

$$\begin{aligned} h_t &= o \odot \tanh(c_t) \\ &= \sigma(W_{hho} h_{t-1} + W_{xho} x_t) \odot \tanh(c_t) \end{aligned} \tag{5}$$

[69] Hochreiter&Schmidhuber, "Long Short-Term Memory", 1997

[70] Great blog post explaining LSTM: `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`

The practicality of having this particular cell structure is evident if we look at multiple cells at once, i.e. the processing over multiple sequences, as it is done in fig. 157. Training works again with Back-propagation-Through-Time. The gradient can now be passed without being interrupted, i.e. the problems of costly weight updates, vanishing and exploding gradients should not occur anymore.

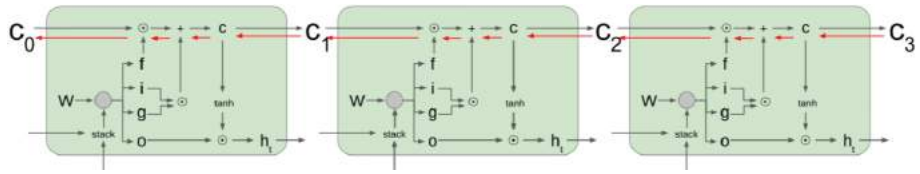


Figure 152: Illustration of LSTM over many sequences. Red arrow denotes the gradient, which can flow uninterruptedly.

13.2.4 Gated Recurrent Units (GRUs)

A Gated Recurrent Unit (GRU) is another way of using gated information in recurrent neural networks. It has fewer parameters than LSTM, as it lacks an output gate. GRU's performance on certain tasks of polyphonic music modeling and speech signal modeling was found to be similar to that of LSTM. GRUs have been shown to exhibit even better performance on certain smaller datasets. An illustration of a GRU is shown below.

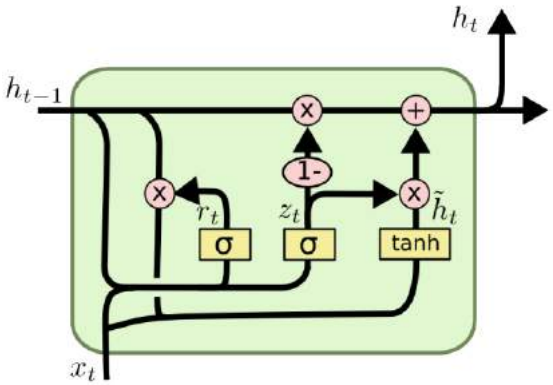


Figure 153: Gated Recurrent Unit (GRU)

The equations that define the update gate vector z_t, the reset gate vector r_t and the output vector h_t are given by

$$
\begin{aligned}
z_t &= \sigma\left(W_z \cdot [h_{t-1}, x_t]\right), \\
r_t &= \sigma\left(W_r \cdot [h_{t-1}, x_t]\right), \\
\tilde{h}_t &= \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right), \\
h_t &= (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t.
\end{aligned}
$$

13.3 RNNs in the Brain

There are claims[71] that networks of the AlexNet[72] type successfully predict properties of neurons in visual cortex. Thus one natural question arises: how similar is an ultra-deep residual network to the primate cortex?[73] A notable difference is the depth. While a residual network has as many as 1202 layers, biological systems seem to have two orders of magnitude less, if we make the customary assumption that a layer in the NN architecture corresponds to a cortical area. In fact, there are about half a dozen areas in the ventral stream of visual cortex from the retina to the Inferior Temporal cortex. Notice that it takes in the order of 10ms for neural activity to propagate from one area to another one (remember that spiking activity of cortical neurons is usually well below 100 Hz). The evolutionary advantage of having fewer layers is apparent: it supports rapid (100msec from image onset to meaningful information in IT neural population) visual recognition, which is a key ability of human and non-human primates. It is intriguingly possible to account for this discrepancy by taking into account **recurrent connections within each visual area**. Areas in visual cortex comprise six different layers with lateral and feedback connections, which are believed to mediate some attentional effects and even learning (such as backpropagation). "Unrolling" in time the recurrent computations carried out by the visual cortex provides an equivalent "ultra-deep" feedforward network, which might represent a more appropriate comparison with the state-of-the-art computer vision models.

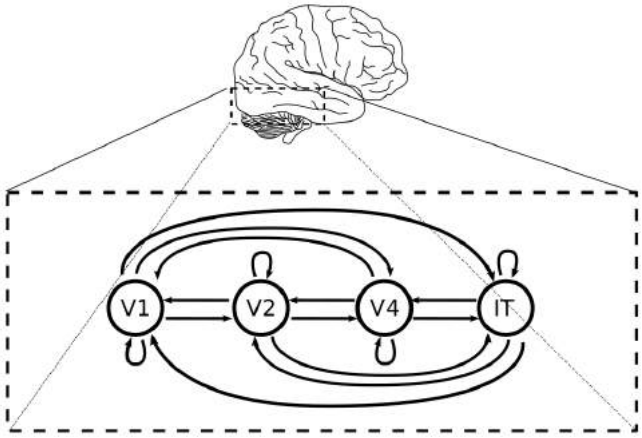


Figure 154: Multi-state (Fully) Recurrent Neural Network

[71] Yamins&Dicarlo, "Using goal-driven deep learning models to understand sensory cortex", 2016.

[72] Krizhevsky&Sutskever&Hinton, "Imagenet classification with deep convolutional neural networks", 2012

[73] This paragraph is copied from Liao&Poggio, "Bridging the Gaps Between Residual Learning, Recurrent Neural Networks and Visual Cortex", 2016

13.3.1 Recurrent Projections in the Cat Brain

By mapping the circuit of cat primary visual cortex (V1), it is evident, that there are recurrent projections involved[74].

13.3.2 Evidence for RNNs in the Rodent Brain

Similary, it has been shown in rodents, that the communication between columns is organized by multiple highly specific horizontal projection patterns[75].

Population coding is a method to represent stimuli by using the joint activities of a number of neurons. In population coding, each neuron has a distribution of responses over some set of inputs, and the responses of many neurons may be combined to determine some value about the inputs.

...

13.3.3 Evidence for RNNs in the Primate Brain

The brain has both a feed-forward structure and recurrent pathways: Information can get sent back from one area to a previous one or echo around the same area multiple times[76]. Studies suggest this extra processing helps the brain interpret challenging visual information, such as objects that are occluded or viewed from unusual angles. A recent study[77] found images that are difficult for a feed-forward model to classify but easy for humans and monkeys to interpret, although they take slightly longer to classify these challenging images than normal ones. This delay suggests that some recurrent processing is involved. The researchers then looked at how neural activity in the monkey's brain evolves as these images are processed. A benefit of convolutional neural networks is that the response of different layers in the model can be used to predict the response of neurons in different brain areas. The researchers found that the feed-forward model predicts the activity of neurons fairly well at early stages (up to 0.1 second into the response) but struggles at later time points. When a convolutional neural network is not performing well, researchers in computer vision tend to add more layers to it, making it 'deeper.' The authors tested whether such deeper networks could better predict neural responses to their challenging images, under the assumption that a network with more layers, which compute over space, resemble recurrent pathways, which compute over time. These deeper networks were indeed better than the shallower model at predicting neural activity at later time points. Finally, the authors added recurrent connections to the structure of their original model and found that responses at later time points in the model better matched later time points in the data. Specifically, when recurrent connections were added to this 'shallower' network, it predicted neural activity as well as the deeper model did. Overall, this work strongly suggests that

[74] Binzegger&Douglas&Martin, "A Quantitative Map of the Circuit of Cat Primary Visual Cortex", 2004

[75] Narayanan, "Beyond Columnar Organization: Cell Type- and Target Layer-Specific Principles of Horizontal Axon Projection Patterns in Rat Vibrissal Cortex.", 2015

[76] Citing from the webpage that represents the two exercise papers of the lecture: `https://www.simonsfoundation.org/2019/05/23/recurrent-connections-improve-neural-network-models-of-vision/`

[77] Paper from exercise class: Kar&DiCarlo, "Evidence that recurrent circuits are critical to the ventral stream's execution of core object recognition behavior", 2019

recurrent processing is an important contributor to computation in the visual system.

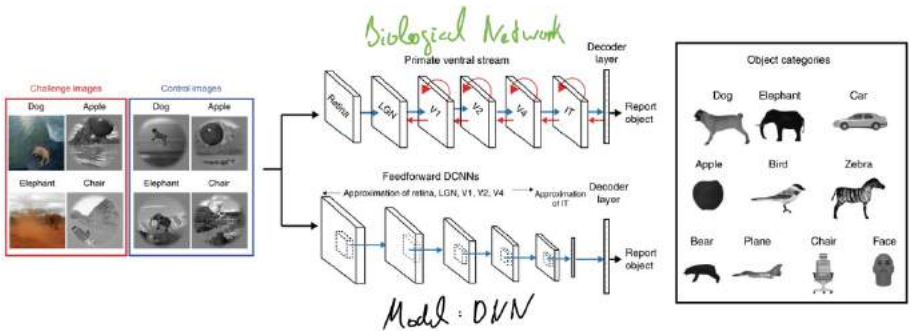


Figure 155: Both primates (humans and macaques) and feedforward DCNNs were tasked to identify which object is present in each test image (1,320 images). Top: the stages in the primate ventral visual pathway (retina, lateral geniculate nucleus (LGN), areas V1, V2, V4, and the IT cortex), which is implicated in core object recognition. We can conceptualize each stage as rapidly transforming the representation of the image and ultimately yielding the primates' behavior (that is, producing a behavioral report of which object was present). The blue arrows indicate the known anatomical feedforward projections from one area to the other. The red arrows indicate the known lateral and top-down recurrent connections. Bottom: a schematic of a similar pathway commonly present in DCNNs. These networks contain a series of convolutional and pooling layers with nonlinear transforms at each stage, followed by fully connected layers (which approximate macaque IT neural responses) that ultimately gives rise to the models' 'behavior'. Note that the DCNNs only have feedforward (blue) connections.

13.4 RNNs in Theoretical Neuroscience

13.4.1 Hopfield Network

A Hopfield network is a form of recurrent artificial neural network popularized by John Hopfield in 1982[78], but described earlier by Little in 1974. Hopfield nets serve as content-addressable ("associative") memory systems with binary threshold nodes. They are guaranteed to converge to a local minimum, but will sometimes converge to a false pattern (wrong local minimum) rather than the stored pattern (expected local minimum)[79].

[78] Hopfield, "Neural networks and physical systems with emergent collective computational abilities", 1982

[79] `https://pathmind.com/wiki/hopfieldnetworks`

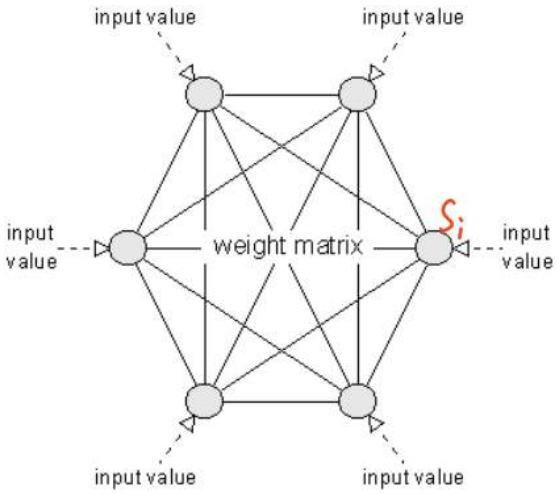


Figure 156: Hopfield Network with six units

A Hopfield network has various units, which have a binary state (1/0). The units update asynchronously or synchronously with the following rule:

$$\begin{aligned} S_i = +1 \quad &\text{if} \quad x_i = \textstyle\sum_{j \neq i} w_{ij} \cdot S_j > \theta_i \\ S_i = -1 \quad &\text{if} \quad x_i = \textstyle\sum_{j \neq i} w_{ij} \cdot S_j < \theta_i \end{aligned}$$

Here, S_i is the i-th unit of the Hopfield network and θ_i is the threshold. One can define an energy term as

$$E = -\frac{1}{2} \sum_{i,j} w_{ij} S_i S_j + \sum_i \theta_i S_i \tag{196}$$

With each update step, the energy either stays constant or decreases.

13.4.2 Reservoir Computing

Reservoir computing is a framework for computation that may be viewed as an extension of neural networks. Typically an input signal is fed into a fixed (random) dynamical system called a reservoir (for example a RNN). Hereby, the dynamics of the reservoir map the input to a higher dimension. Then a simple readout mechanism is trained to read the state of the reservoir and map it to the desired output. The main benefit is that training is performed only at the readout stage and the reservoir is fixed[80]. A really cool thought is, that basically every (abstract or physical) dynamic system can be used as the reservoir[81], including a water tank[82], an electronic circuit or parts of the brain itself!

Echo State Network: Recurrent neural network with a random and sparsely connected (1%) hidden layer / reservoir, works in discrete time, different activity update modes.

[80] `https://en.wikipedia.org/wiki/Reservoir_computing`
[81] Tanaka, "Recent Advances in Physical Reservoir Computing: A Review", 2019
[82] Fernando&Sojakka, "Pattern Recognition in a Bucket", 2003

Liquid State Machine: Biologically plausible, spiking RNN reservoir, randomly connected, continuous in time, asynchronous integration, uses linear discriminate units.

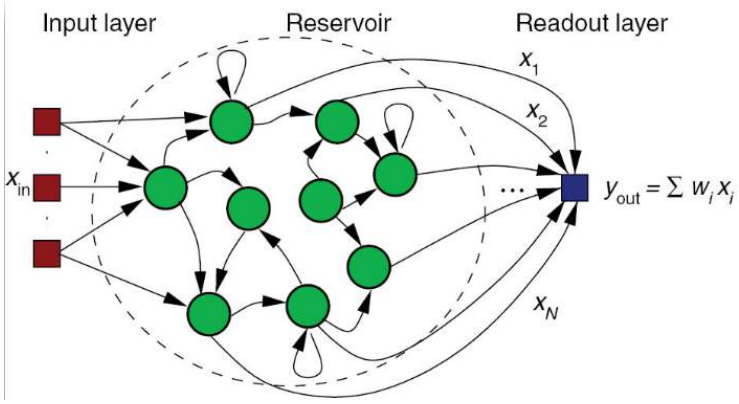


Figure 157: Reservoir computing

13.4.3 Learning Algorithms

Learning via Backpropagation Through Time (BPTT): See above. ...

Learning via Real Time Recurrent Learning (RTRL): A general approach to training an arbitrary recurrent network by adjusting weights along the error gradient. This algorithm usually requires very low learning rates because of the inherent correlations between successive node outputs.

...

First-Order, Reduced and Controlled Error (FORCE) Learning: ...

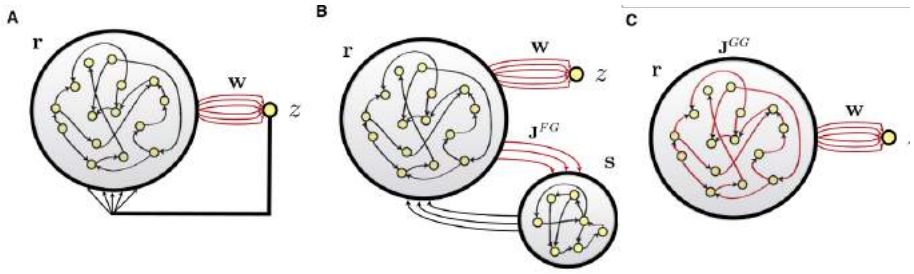


Figure 158: First-Order, Reduced and Controlled Error (FORCE) Learning. In all three cases, a recurrent generator network with firing rates **r** drives a linear readout unit with output z through weights **w** (red) that are modified during training. Only connections shown in red are subject to modification. **(A)** Feedback to the generator network (large network circle) is provided by the readout unit. **(B)** Feedback to the generator network is provided by a separate feedback network (smaller network circle). Neurons of the feedback network are recurrently connected and receive input from the generator network through synapses of strength J^{FG} (red), which are modified during training. **(C)** A network with no external feedback. Instead, feedback is generated within the network and modified by applying FORCE learning to the synapses with strengths J^{GG} internal to the network (red).

Self Organizing Recurrent Networks (SORN) It combines three distinct forms of local plasticity to learn spatio-temporal patterns in its input while maintaining its dynamics in a healthy regime suitable for learning[83]. The SORN learns to encode information in the form of trajectories through its high-dimensional state space reminiscent of recent biological findings on cortical coding. All three forms of plasticity are shown to be essential for the network's success:

1. STDP Rule
2. Weight Normalization Rule
3. Distributed Processing Rule

[83]Lazar, "A Self-Organizing Recurrent Neural Network", 2009

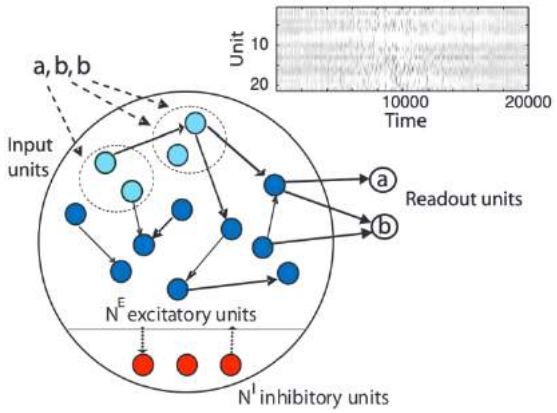


Figure 159: Self Organizing Recurrent Networks (SORN)

14 Continual-, Meta- And Transfer-Learning

14.1 Motivation

In section 1.2.2 we listed the standing challenges in deep learning research, from which we now want to discuss continual learning (the capability to learn multiple tasks sequentially) and meta-learning, which can allow to learn fast and from few-data only. So why do we need continual and meta-learning?

- For many applications we don't have large training data-sets (medical imaging, robotics, recommendations, real-world agent training).
- Life-long learning systems should quickly adapt to new tasks, but not forget previous ones (can't learn every task/classifier from scratch).
- Sometimes our training data has a long-tail, meaning that there are only a few data points for a large range of categories.
- Concept learning enables humans to extrapolate from learned tasks to a similar task.

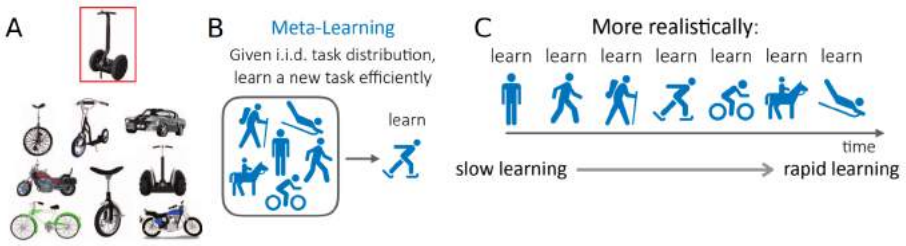


Figure 160: The principle of learning the learn.

14.2 Meta-Learning with ANNs

Meta-learning, also known as "learning to learn", intends to design models that can learn new skills or adapt to new environments rapidly with a few training examples. There are three common approaches[84]:

1. learn an efficient distance metric (metric-based)
2. use (recurrent) network with external or internal memory (model-based)
3. optimize the model parameters explicitly for fast learning (optimization-based)

A good meta-learning model should be trained over a variety of learning tasks and optimized for the best performance on a distribution of tasks, including potentially unseen tasks. Each task is associated with a dataset $\mathcal{D}$, containing both feature vectors and true labels. The optimal model parameters are:

$$\theta^* = \arg\min_\theta \mathbb{E}_{\mathcal{D}\sim p(\mathcal{D})} \left[\mathcal{L}_\theta(\mathcal{D})\right] \tag{197}$$

[84] Awesome blog post by Lilian at OpenAI, where apparently half of the lecture was taken from: `https://lilianweng.github.io/lil-log/2018/11/30/meta-learning.html`

It looks very similar to a normal learning task, but one *dataset* is considered as one *data sample*. The concept of *Few-shot classification* is an instantiation of meta-learning in the field of supervised learning. The dataset $\mathcal{D}$ is often split into two parts, a support set $\mathcal{S}$ for learning and a prediction set $\mathcal{B}$ for training or testing, $\mathcal{D} = \langle S, B \rangle$.

Another popular view of meta-learning decomposes the model update into two stages:

1. A classifier f_θ is the "learner" model, trained for operating a given task
2. In the meantime, a optimizer g_ϕ (the "meta-learner"), learns how to update the learner model's parameters via the support set $S, \theta^{'} = g_\phi(\theta, S)$.

In the final optimization step, one needs to update both θ and ϕ to maximize:

$$\mathbb{E}_{L \subset \mathcal{L}} \left[\mathbb{E}_{S^L} \subset \mathcal{D}, B^L \subset \mathcal{D} \left[\sum_{(\mathbf{x},y) \in B^L} P_{g_\phi(\theta, S^L)}(y|\mathbf{x}) \right] \right] \tag{198}$$

14.2.1 Metric-based ML

The core idea in metric-based meta-learning is similar to nearest neighbors algorithms and kernel density estimation. The predicted probability over a set of known labels y is a weighted sum of labels of support set samples. The weight is generated by a kernel function k_θ, measuring the similarity between two data samples.

$$P_\theta(y|\mathbf{x}, S) = \sum_{(x_i, y_i) \in S} k_\theta(\mathbf{x}, \mathbf{x}_i)\, y_i \tag{199}$$

To learn a good kernel is crucial to the success of a metric-based meta-learning model. Metric learning is well aligned with this intention, as it aims to learn a metric or distance function over objects. The notion of a good metric is problem-dependent. It should represent the relationship between inputs in the task space and facilitate problem solving. A few models are introduced, that learn embedding vectors of input data explicitly and use them to design proper kernel functions:

Prototypical Networks [85] use an embedding function f_θ to encode each input into a M-dimensional feature vector. A prototype feature vector is defined for every class $c \in \mathcal{C}$, as the mean vector of the embedded support data samples in this class.

$$\mathbf{v}c = \frac{1}{|S_c|} \sum (\mathbf{x}i, y_i) \in S_c f\theta(\mathbf{x}_i) \tag{200}$$

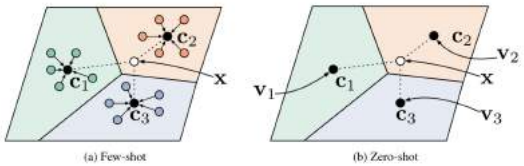


Figure 161: Prototypical networks in the few-shot and zero-shot scenarios.

[85] Snell&Swersky& Zemel, "Prototypical Networks for Few-shot Learning", 2017

The distribution over classes for a given test input $\mathbf{x}$ is a *softmax* over the inverse of distances between the test data embedding and prototype vectors.

$$P(y = c|\mathbf{x}) = \text{softmax}(-d_\varphi(f_\theta(\mathbf{x}), \mathbf{v}c)) = \frac{\exp(-d\varphi(f_\theta(\mathbf{x}), \mathbf{v}c))}{\sum c' \in \mathcal{C} \exp(-d_\varphi(f_\theta(\mathbf{x}), \mathbf{v}_{c'}))} \tag{201}$$

where d_φ can be any distance function as long as φ is differentiable. In the paper, they used the squared euclidean distance. The loss function is the negative log-likelihood:

$$\mathcal{L}(\theta) = -\log P_\theta(y = c|\mathbf{x}) \tag{202}$$

Siamese Neural Networks are composed of two twin networks and their outputs are jointly trained on top with a function to learn the relationship between pairs of input data samples. The twin networks are identical, sharing the same weights and network parameters. In other words, both refer to the same embedding network that learns an efficient embedding to reveal relationship between pairs of data points. Convolutional Siamese neural networks have been applied to one-shot image classification[86].

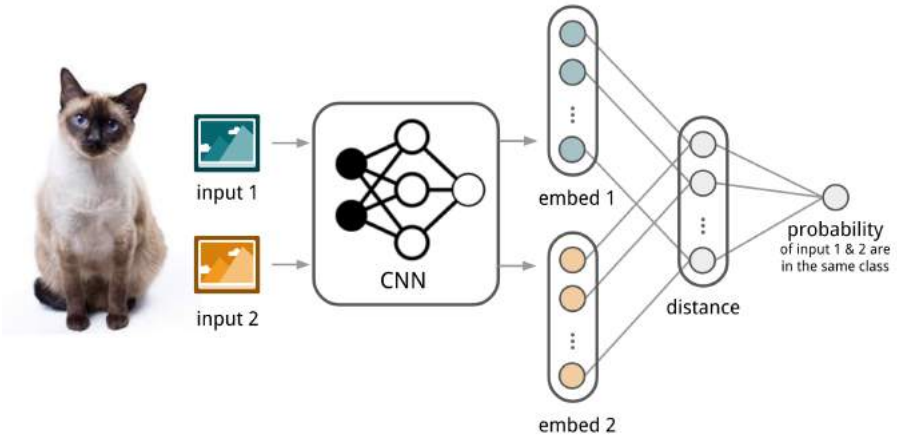


Figure 162: The architecture of convolutional siamese neural network for few-show image classification.

Training: The siamese network is trained for a verification task for telling whether two input images are in the same class. It outputs the probability of two images belonging to the same class.

1. First, convolutional siamese network learns to encode two images into feature vectors via a embedding function f_θ which contains a couple of convolutional layers.

2. The L1-distance between two embeddings is $|f_\theta(\mathbf{x}i) - f\theta(\mathbf{x}_j)|$.

[86]Koch&Zemel&Salakhutdinov, "Siamese Neural Networks for One-shot Image Recognition", 2015

3. The distance is converted to a probability p by a linear feedforward layer and sigmoid. It is the probability of whether two images are drawn from the same class.

4. Intuitively the loss is cross entropy because the label is binary.

$$p(\mathbf{x}_i, \mathbf{x}j) = \sigma(\mathbf{W}|f\theta(\mathbf{x}i) - f\theta(\mathbf{x}j)|) \tag{203}$$

$$\mathcal{L}(B) = \sum_{(\mathbf{x}_i, \mathbf{x}j, y_i, y_j) \in B} \infty_{y_i = y_j} \log p(\mathbf{x}_i, \mathbf{x}j) + (1 - \infty_{y_i = y_j}) \log(1 - p(\mathbf{x}_i, \mathbf{x}_j)) \tag{204}$$

Testing: The siamese network processes all the image pairs between a test image and every image in the support set. The final prediction is the class of the support image with the highest probability. Given a support set S and a test image $\mathbf{x}$, the final predicted class is:

$$\hat{c}_S(\mathbf{x}) = c(\arg\max \mathbf{x}_i \in SP(\mathbf{x}, \mathbf{x}_i)) \tag{205}$$

where $c(\mathbf{x})$ is the class label of an image $\mathbf{x}$ and $\hat{c}(.)$ is the predicted label.

Matching Networks [87] aim at learning a classifier c_S for any given (small) support set $S = \{x_i, y_i\}_{i=1}^k$ (k-shot classification). This classifier defines a probability distribution over output labels y given a test example $\mathbf{x}$. Similar to other metric-based models, the classifier output is defined as a sum of labels of support samples weighted by attention kernel $a(\mathbf{x}, \mathbf{x}_i)$ - which should be proportional to the similarity between $\mathbf{x}$ and $\mathbf{x}_i$.

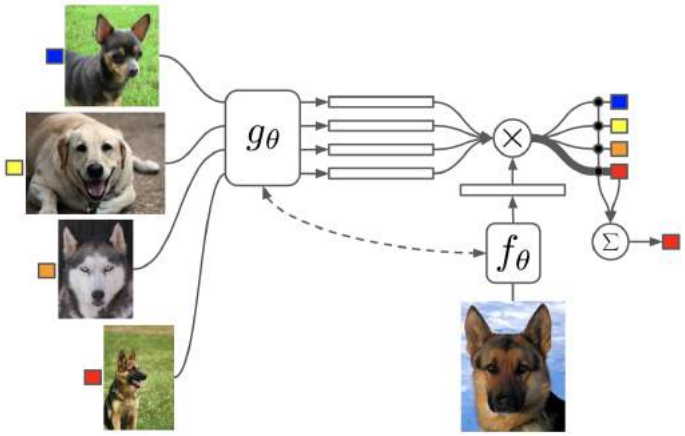


Figure 163: The architecture of Matching Networks

$$c_S(\mathbf{x}) = P(y|\mathbf{x}, S) = \sum_{i=1}^{k} a(\mathbf{x}, \mathbf{x}_i) y_i, \text{ where } S = (\mathbf{x}i, y_i)i = 1^k \tag{206}$$

[87] Vinyals et al., "Matching Networks for One Shot Learning", 2016

The attention kernel depends on two embedding functions, f and g, for encoding the test sample and the support set samples respectively. The attention weight between two data points is the cosine similarity, cosine(.), between their embedding vectors, normalized by *softmax*:

$$a(\mathbf{x}, \mathbf{x}_i) = \frac{\exp(\text{cosine}(f(\mathbf{x}), g(\mathbf{x}i))}{\sum j = 1^k \exp(\text{cosine}(f(\mathbf{x}), g(\mathbf{x}_j))} \tag{207}$$

The embedding has to be choosen carefully. In a simple version, an embedding function is a neural network with a single data sample as input. Taking a single data point as input might not be enough to efficiently gauge the entire feature space. Therefore, the Matching Network model further proposed to enhance the embedding functions by taking as input the whole support set S in addition to the original input, so that the learned embedding can be adjusted based on the relationship with other support samples.

Relation Network [88] is similar to siamese network but with a few differences:

- The relationship is not captured by a simple L1 distance in the feature space, but predicted by a CNN classifier g_ϕ. The relation score between a pair of inputs, $\mathbf{x}i$ and $\mathbf{x}j$, is $rij = g\phi([\mathbf{x}_i, \mathbf{x}_j])$ where $[.,.]$ is concatenation.
- The objective function is MSE loss instead of cross-entropy, because conceptually RN focuses more on predicting relation scores which is more like regression, rather than binary classification, $\mathcal{L}(B) = \sum_{(\mathbf{x}i,\mathbf{x}j,y_i,y_j)\in B}(rij - \mathbf{1}y_i = y_j)^2$.

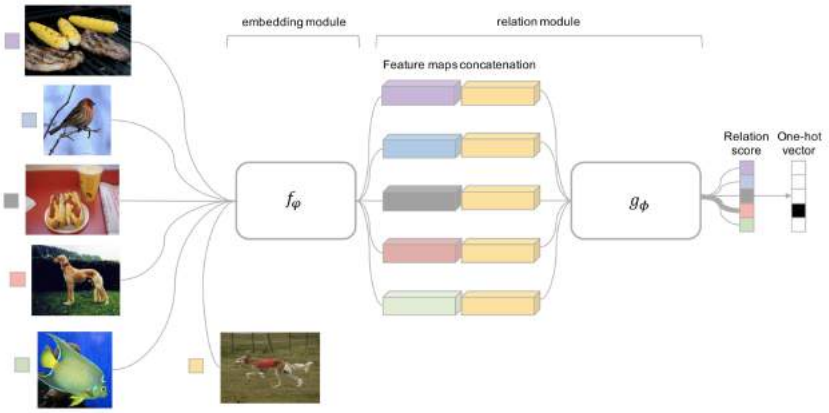


Figure 164: Relation Network architecture for a 5-way 1-shot problem with one query example.

14.2.2 Model-based ML

Model-based meta-learning models make no assumption on the form of $P_\theta(y|x)$. Rather it depends on a model designed specifically for fast learning — a model

[88] Sung et al., "Learning to Compare: Relation Network for Few-Shot Learning" 2018

that updates its parameters rapidly with a few training steps. This rapid parameter update can be achieved by its internal architecture or controlled by another meta-learner model.

Hypernetworks [89] are networks that generate the weights of a target model based on task identity. Continual learning (CL) is less difficult for this class of models thanks to a simple key feature: instead of recalling the input-output relations of all previously seen data, task-conditioned hypernetworks only require rehearsing task-specific weight realizations, which can be maintained in memory using a simple regularizer. Besides achieving state-of- the-art performance on standard CL benchmarks, additional experiments on long task sequences reveal that task-conditioned hypernetworks display a very large capacity to retain previous memories.

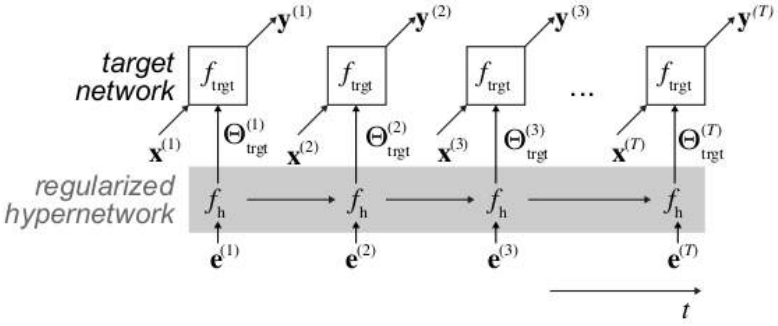


Figure 165: **Task-conditioned hypernetworks for continual learning.** Commonly, the parameters of a neural network are directly adjusted from data to solve a task. Here, a weight generator termed hypernetwork is learned instead. Hypernetworks map embedding vectors to weights, which parameterize a target neural network. In a continual learning scenario, a set of task-specific embeddings is learned via backpropagation. Embedding vectors provide task-dependent context and bias the hypernetwork to particular solutions.

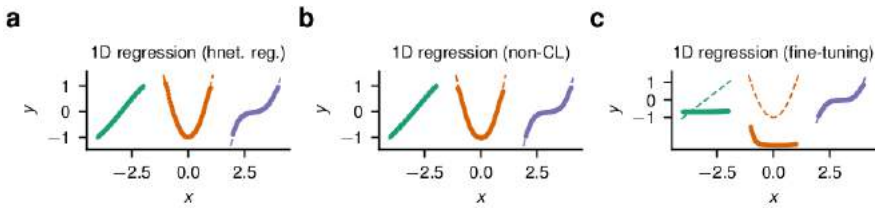


Figure 166: **1D nonlinear regression.** **(a)** Task-conditioned hypernetworks with output regularization can easily model a sequence of polynomials of increasing degree, while learning in a continual fashion. **(b)** The solution found by a target network which is trained directly on all tasks simultaneously is similar. **(c)** Fine-tuning, i.e., learning sequentially, leads to forgetting of past tasks. Dashed lines depict ground truth, markers show model predictions.

[89] Oswald&Henning&Sacramento et al., "Continual Learning with Hypernetworks", 2019

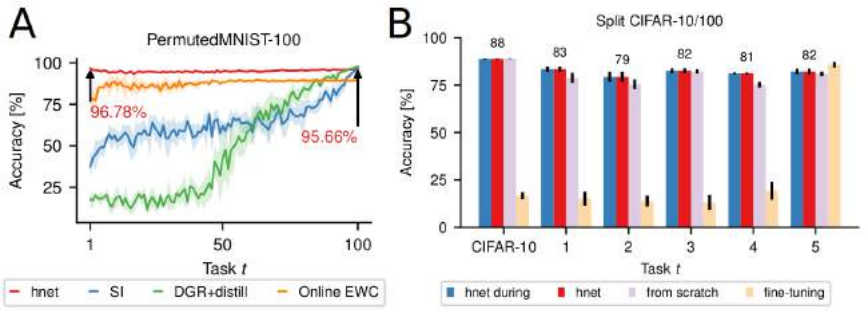


Figure 167: **Experimental results.** **(A)** Experiments on the permuted MNIST benchmark. Final test set classification accuracy on the t-th task after learning one hundred permutations (PermutedMNIST-100). Task-conditioned hypernetworks (hnet, in red) achieve very large memory lifetimes on the permuted MNIST benchmark. Synaptic intelligence (SI, in blue; Zenke et al., 2017), online EWC (in orange; Schwarz et al., 2018) and deep generative replay (DGR+distill, in green; Shin et al., 2017) methods are shown for comparison. **(B)** Split CIFAR-10/100 continual learning benchmark. Test set accuracies (mean ± STD, n = 5) on the entire CIFAR-10 dataset and subsequent CIFAR-100 splits. The hypernetwork-protected ResNet-32 displays virtually no forgetting; final averaged performance (hnet, in red) matches the immediate one (hnet-during, in blue). Furthermore, information is transferred across tasks, as performance is higher than when training each task from scratch (purple).

14.2.3 Optimization-Based ML

Deep learning models learn through backpropagation of gradients. However, the gradient-based optimization is neither designed to cope with a small number of training samples, nor to converge within a small number of optimization steps. Is there a way to adjust the optimization algorithm so that the model can be good at learning with a few examples? This is what optimization-based approach meta-learning algorithms intend for. Look at **LSTM Meta-Learner**, **Model-Agnostic Meta-Learning (MAML)** and **Reptile** for further information (not covered in class).

Model-Agnostic Meta-Learning (MAML) is a fairly general optimization algorithm, compatible with any model that learns through gradient descent. Let's say our model is f_θ with parameters θ. Given a task τ_i and its associated dataset $(\mathcal{D}^{(i)}\text{train}, \mathcal{D}^{(i)}\text{test})$, we can update the model parameters by one or more gradient descent steps (the following example only contains one step):

$$\theta' i = \theta - \alpha \nabla \theta \mathcal{L}^{(0)} \tau_i(f\theta)$$

where $\mathcal{L}^{(0)}$ is the loss computed using the mini data batch with id (0).

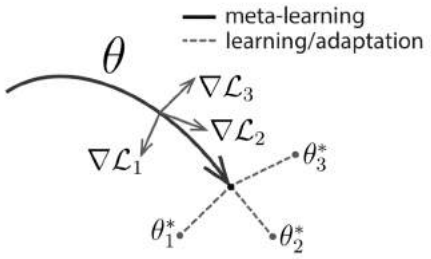


Figure 168: Diagram of MAML.

The above formula only optimizes for one task. To achieve a good generalization across a variety of tasks, we would like to find the optimal θ^* so that the task-specific fine-tuning is more efficient. Now, we sample a new data batch with id (1) for updating the meta-objective. The loss, denoted as $\mathcal{L}^{(1)}$, depends on the mini batch (1). The superscripts in $\mathcal{L}^{(0)}$ and $\mathcal{L}^{(1)}$ only indicate different data batches, and they refer to the same loss objective for the same task.

$$\theta^* = \arg\min_{\theta} \sum_{\tau_i \sim p(\tau)} \mathcal{L}\tau_i{}^{(1)}(f\theta' i) = \arg\min \theta \sum_{\tau_i \sim p(\tau)} \mathcal{L}\tau_i{}^{(1)}(f\theta - \alpha\nabla_\theta \mathcal{L}\tau_i{}^{(0)}(f\theta)) \tag{208}$$

$$\theta \leftarrow \theta - \beta\nabla_\theta \sum_{\tau_i \sim p(\tau)} \mathcal{L}\tau_i{}^{(1)}(f\theta - \alpha\nabla_\theta \mathcal{L}\tau_i{}^{(0)}(f\theta)); \text{ updating rule} \tag{209}$$

LSTM Meta-Learner [90] is proposed to learn the exact optimization algorithm used to train another learner neural network classifier in the few-shot regime. The parametrization of the model allows it to learn appropriate parameter updates specifically for the scenario where a set amount of updates will be made, while also learning a general initialization of the learner (classifier) network that allows for quick convergence of training.

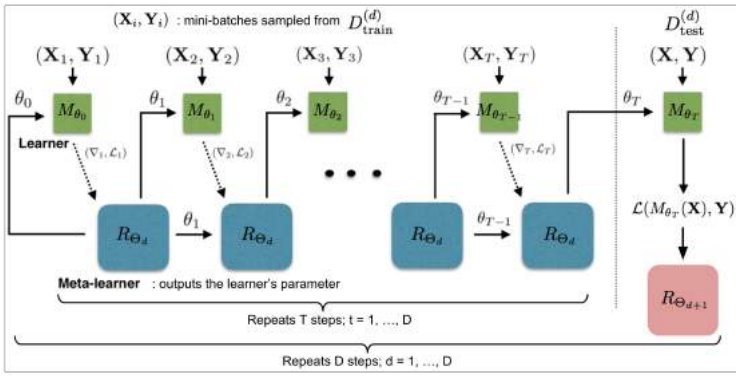


Figure 169: How the learner M_θ and the meta-learner R_Θ are trained. Computational graph for the forward pass of the meta-learner. The dashed line divides examples from the training set D_{train} and test set D_{test}.

[90] Ravi&Larochelle, "Optimization as a Model for Few-Shot Learning", 2017

14.3 Meta-Learning in the Brain

...

14.3.1 Meta-Learning in the Prefrontal Cortex

Over the past 20 years, neuroscience research on reward-based learning has converged on a canonical model, under which the neurotransmitter dopamine "stamps in" associations between situations, actions and rewards by modulating the strength of synaptic connections between neurons. However, a growing number of recent findings have placed this standard model under strain. A recent study introduces a new theory, where the dopamine system trains another part of the brain, the prefrontal cortex, to operate as its own free-standing learning system. This new perspective accommodates the findings that motivated the standard model, but also deals with a wider range of observations.

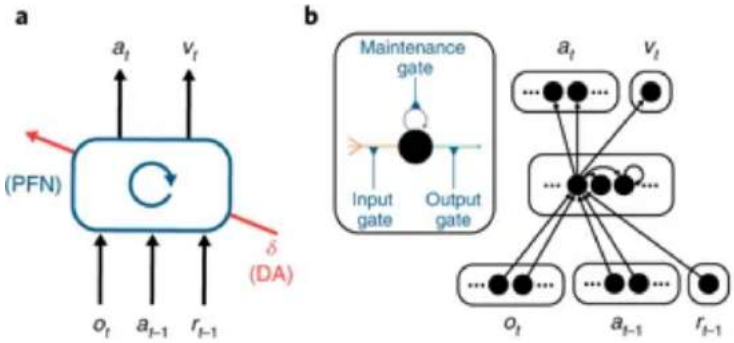


Figure 170: **Meta-RL architecture learns across episodes to learn efficiently within an episode. a** Agent architecture. The prefrontal network (PFN), including sectors of the basal ganglia and the thalamus that connect directly with PFC, is modeled as a recurrent neural network, with synaptic weights adjusted through an RL algorithm driven by dopamine (DA); o is perceptual input, a is action, r is reward, v is state value, t is time-step and σ is RPE. The central box denotes a single, fully connected set of LSTM units. **b** A more detailed schematic of the neural network implementation used in the simulations.

14.3.2 Meta-Learning via Neuromodulation

Neuromodulators play an important role in meta-learning in the brain. Some of the key modulators are listed below: Neuromodulatory systems can be seen to mediate the global signals that regulate the distributed learning mechanisms in the brain[91]. Based on the review of experimental data and theoretical models, some key modulators are described below:

Dopamine is proposed to act as a "global learning" signal, critical to prediction of rewards and action reinforcement.

[91] Doya, "Metalearning and neuromodulation", 2002

Serotorin is proposed to control the balance between short and long term reward prediction, essentially by variably "discounting" expected future reward sums that may require too much expenditure to achieve.

Norepinephrine is proposed to facilitate "wide exploration" by stochastic action selection (control exploration vs. exploitation).

Acetylcholine is proposed to facilitate the balance between memory storage and memory renewal, finding an optimal balance between stability and effectiveness of learning algorithms for the specific environmental task.

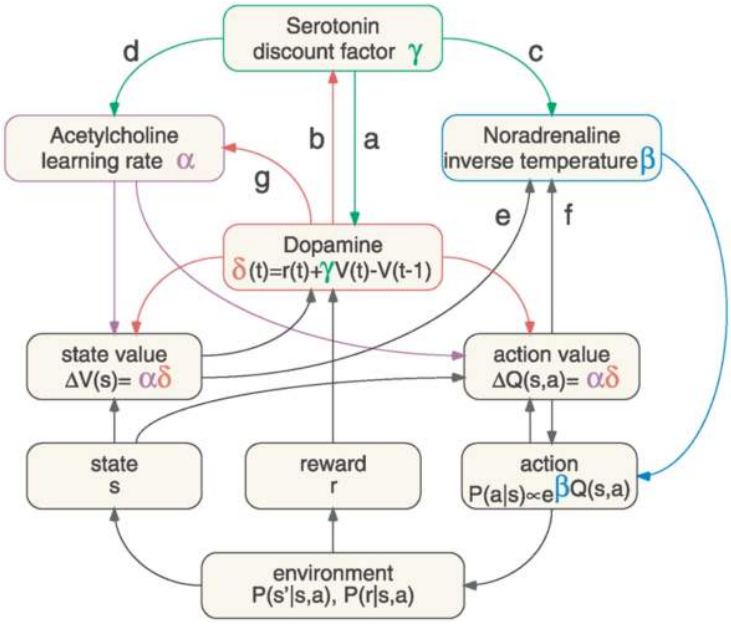


Figure 171: Computational diagram: Metalearning via neuromodulation.

A Questions

Mock exam

1. Why Spikes (Lecture 9): Several techniques have been developed to better understand neuronal activity. Please answer the following questions on methods used to investigate action potentials.

   1.1. When recording the membrane potential of a neuron, which changes can be observed during an action potential?

   1.2. Explain in a few sentences how the Patch Clamp technique works and what it can be used for.

   1.3. Some techniques for imaging neuronal activity involve functional fluorescent indicators. Which signals can be recorded with these techniques?

   1.4. Give one reason why an individual neuron might benefit from spiking.

2. Plasticity in the Brain (Lecture 3): Homeostatic Plasticity involves mechanisms which allow neurons to adapt to altered environments whilst maintaining a desired functionality. Answer the following questions about Homeostatic Plasticity.

   2.1. A potent toxin prevents postsynaptic ion channels from opening. This leads to a rapid decay of postsynaptic potentials (EPSPs). However, after the toxin is removed, the amplitude of postsynaptic potentials is significantly increased in comparison to pre-perturbation. How could this behavior be explained?

   2.2. What does the term "scaling" refer to in Homeostatic Plasticity?

   2.3. Homeostatic Plasticity is thought to keep local properties of neurons within a certain (physiological, functional, . . .) range. Discuss in a few sentences how this principle could be used in artificial neural networks.

   2.4. What is Heterosynaptic (Heterostatic) Plasticity and which synaptic mechanisms could be involved (give one example)?

3. Learning as Bayesian Inference (Lecture 5): Using Bayesian Inference techniques to train neural networks brings some advantages as well as some new challenges. Answer the following questions about Bayesian inference.

   3.1. What is $p(w|D)$? Can it be related to a set of weights obtained with error backpropagation?

   3.2. How is $p(D)$, the marginal likelihood, problematic? Explain the approach to solve the problem.

   3.3. What are relevant properties of the Kullback-Leibler divergence?

   3.4. How do $p(w)$ and $q(w)$ compare to each other when $DKL(q(w)||p(w)) = 0$? Learning in deep artificial and biological neuronal networks

4. Unsupervised and Self-Supervised Learning (Lecture 7): Autoencoders are neural networks that learn efficient data encodings in an unsupervised manner. Please answer the following questions on Autoencoders.

4.1. In which case can we say that Autoencoders yield a compressed low dimensional representation of the input?

4.2. What does the term "latent representation" mean in the context of Autoencoders?

4.3. Describe how you would use Autoencoders to perform de-noising on a set of inputs.

4.4. Describe the two phases in the wake-sleep algorithm.

5. Deep learning with Spikes (Lecture 10): SuperSpike was recently proposed as an online rule for learning in spiking neural networks. Answer the following questions on SuperSpike.

5.1. Why can SuperSpike be considered a three-factor rule?

5.2. The rule includes two convolutions (mathematical operation). What is the algorithmic justification and what is a possible biological interpretation?

5.3. How is the loss of the neural network defined when using this rule?

5.4. How does this rule solve the spike non-linearity problem?

6. Learning Rules (Lecture 4): The DELTA Rule and the Perceptron Learning Rule are both similar but also very different in some properties. Answer the following questions about the DELTA- and the Perceptron Learning rules.

6.1. What do the DELTA Rule and the Perceptron Learning Rule have in common?

6.2. Explain in a few sentences, how the DELTA rule works.

6.3. What is the main difference between the two rules with respect to how they have been derived / developed?

6.4. Identify the following equation and explain the terms of the formula $\Delta Wij = (tiyi)g0(hi)Xj$

7. Reinforcement Learning (Lecture 8): Error driven learning is key in reinforcement learning and several learning rules exist. Answer the following questions on different learning rules in RL.

7.1. The Rescorla-Wagner Rule is formalized as: $V(st) \longleftarrow V(st) + \eta[Rsum(V(St))]$. What does the term $Rsum(V(St))$ represent?

7.2. Name the following rule and correct the mistakes: $V(st) \longleftarrow V(st) + [rt + \gamma V(St)V(St)]$

7.3. Instead of a state value, Q-learning introduces an action value Q. Which two variables does Q depend on and what do they represent?

7.4. When new values are predicted, values of subsequent steps are discounted. What is the rationale behind this?

Jeopardy

1. **What is ADALINE?** See the appropriate section in the Learning Rules chapter.
2. **What is the KL divergence?** An asymmetric measure of difference between two functions, often probability distributions, that does not obey the triangle inequality. $D_{KL}(P||Q) \geq 0$ is also known as the Gibbs' inequality.
3. **What is the forward and backward methods in PyTorch?** Pytorch.autograd
4. **What is fluorescence?** This signal can be detected when dopamine binds to dLight1.1
5. **What is uncorrelated?** This is what $x1$ and $x2$ are, given $< x1, x2 > - < x1 >< x2 >$
6. **What is the Hippocampus?** Brain area for memory.
7. **What is the NMDA receptor?** Responding to glutamate, this receptor opens for sodium and calcium ions.
8. **What is symmetry?** This property is the reason why kl is not a real distance.
9. **What is the prior matching term?** Besides data fitting, this term is also minimized in variational inference.
10. **What is convolution?** In the superspike rule, this operation is used to generate eligibility traces.
11. **What is two photon microscopy?** This method uses an infrared laser to image e.g. Ca2 activities.
12. **What is independent component analysis (ICA)?** This unsupervised method enforces independence of its components.
13. **What is VAE?** In this probabilistic autoencoder the latent representation of an input is sampled from a set of means and standard deviations
14. **What is a Helmholtz Machine?** This network is trained via the Wake/sleep algorithm.
15. **What are sodium and potassium ions?** The ions are responsible for generating an action potential.
16. **What is the spike non-linearity?** This property of SNNs makes it difficult to train them.
17. **What are amplitude an duration (resp what is shape)?** For consectutive action potentials these properties tend to stay constant.
18. **What is a calcium ion?** CamK2 is dependent on this tiny second messenger. It is used to justify

19. **What is the reparametrization trick?** deltaWij = xiyj is commonly use to formalize the simplest form
20. **What is the hebbian learning rule?**
21. **What is the temporal difference rule?**
22. **What is the triplet rule?** In this rule a presynaptic, event followed by ps ev
23. **What is facilitation/post tetanic potentiation?** In short-term plasticity this gain may precede depletion.

More questions

1. **What are the differences between the weights of a deep network and the connections between biological neurons?** In deep learning the weights are binary, if a neuron is activity it causes a change in the next layer. In biology if the action potential is modulated (axonal by myelinated and other factors) it will determine if the post neuron is activated, the dendritic epsp can be modulated before it reaches the axon hillock by summation and inhibition signals, the cell can in a refractory period or in excitability this means the effective connective between neurons can be modulated, also the inhibitory nts can modulate the epsp.
2. **Is synaptic plasticity the only mechanism that neurons can use to alter their effective connectivity?**
3. **Is weight tuning by BP the only mechanism in DNNs that affect the weights?**
4. **What is the difference between BCM rule and triplet rule?**
5. If the neocortex mostly performs unsupervised learning why does the Ventral Degmental Area (VTA) strongly project to almost all cortical areas?
6. What is the effect of dopamine (DA) on a cortical neuron?

More Questions

1. Lecture 1: Introduction
    1.1. How heavy is the brain, how many neurons and how many synapses?
    1.2. What is the neocortex?
    1.3. Describe Hubel&Wiesel experiment and its outcomes
    1.4. Difference between MP neuron and Perceptron?
    1.5. What are parallels and differences of information processing and learning in ANNs and BNNs?
    1.6. What is duration of a spike?
    1.7. How fast do APs propagate along axon?
    1.8. What is electrophysiology?

1.9. What is Calcium Imaging?

1.10. What is fMRI?

2. Lecture 2: Training Methods for Deep ANNs

2.1. What is backpropagation? (Qualitatively and with formulas)

2.2. What are some issues of BP regarding performance?

2.3. What are some issues of BP regarding biological plausibility?

2.4. Explain feedback alignment

2.5. Explain target propagation

2.6. What problem does Direct Target Propagation solve and how?

2.7. How could biological neurons differ between FF and BP error signals?

2.8. How do biological neurons change their incoming weights (synaptic connections)?

2.9. Do biological neurons follow a purely local update rule?

2.10. What is the depth (hierarchical layers) of biological neuronal networks of the human/primate visual system?

3. Lecture 3: Plasticity

3.1. Describe a spike generation, starting at the action potential of the presynaptic neuron.

3.2. What are short and long term effects of an increased EPSP?

3.3. Describe associative learning.

3.4. What is homeostatic plasticity?

3.5. What is spike-timing dependent plasticity?

3.6. What is the problem with Hebbian Learning? [Hint: Hinton]

3.7. What is also called heterosynaptic plasticity?

3.8. What is the hippocampus?

3.9. What ion is mostly involved in controlling LTP and LTP?

3.10. Give some examples for non-synaptic plasticity

3.11. What are the differences between the weights of a deep network and the connections between biological neurons? In deep learning the weights are binary, if a neuron is activity it causes a change in the next layer. In biology if the action potential is modulated (axonal by myelinated and other factors) it will determine if the post neuron is activated, the dendritic epsp can be modulated before it reaches the axon hillock by summation and inhibition signals, the cell can in a refractory period or in excitability this means the effective connective between neurons can be modulated, also the inhibitory nts can modulate the epsp.

3.12. Is synaptic plasticity the only mechanism that neurons can use to alter their effective connectivity?

4. Lecture 4: Learning Rules

4.1. What is the perceptron rule?

4.2. What is ADALINE?

4.3. What is the DELTA rule?

4.4. What is Hebbian rule, Oja rule and Covariance rule?

4.5. Describe what Sangers rule does

4.6. What is the BMC rule?

4.7. Is weight tuning by BP the only mechanism in DNNs that affect the weights?

4.8. What is the difference between BCM rule and triplet rule?

5. Lecture 5: Unsupervised/Selfsupervised Learning

   5.1.

6. Lecture 6: Unsupervised/Selfsupervised Learning

   6.1.

7. Lecture 7: Unsupervised/Selfsupervised Learning

   7.1. Tell me about sparse coding

   7.2. Tell me about competitive learning in NN

   7.3. Tell me about different concepts of autoencoders and their cost functions

   7.4. What is a Boltzmann machine?

   7.5. Tell me about the Helmholtz machine and the wake-sleep algorithm

   7.6. What is the re-parametrization trick?

   7.7. What is the idea behind predictive coding and what's the neuroscientific background?

   7.8. What's the advantages and disadvantages of generative and non-generative unsupervised learning methods?

   7.9. Tell me about the Variational Autoencoder

8. Lecture 8: Reinforcement Learning

   8.1. What is a dopamine neuron? Explain

   8.2. What is the Rescola-Wagner rule?

   8.3. What is the Temporal difference rule?

   8.4. What is Q-learning?

   8.5. State the reward hypothesis.

   8.6. What is a policy, what a value function?

   8.7. What is Bellman's theorem?

   8.8. How do we know if a policy is optimal?

   8.9. What is the loss function in Deep Reinforcement Learning?

   w

9. Lecture 10: Unsupervised/Selfsupervised Learning

    9.1.

10. Lecture 11: Neuromorphic Systems 1

    10.1. What neuron property does Neuromorphic engineering try to emulate and how?

11. Lecture 12: Neuromorphic Systems 2

    11.1.

12. Lecture 13: Recurrent Neural Networks

    12.1. What is back-propagation through time? What are the issues?

    12.2. What is one way to solve the exploding, but not the vanishing gradient problem?

    12.3. What are LSTM networks?

    12.4. Consider the typical depth of a ResNet and compare it with the visual cortex. If there are discrepancies, how could one explain that they both perform fairly well on image processing tasks?

13. Lecture 14: Continual-, Meta- and Transfer Learning

    13.1.