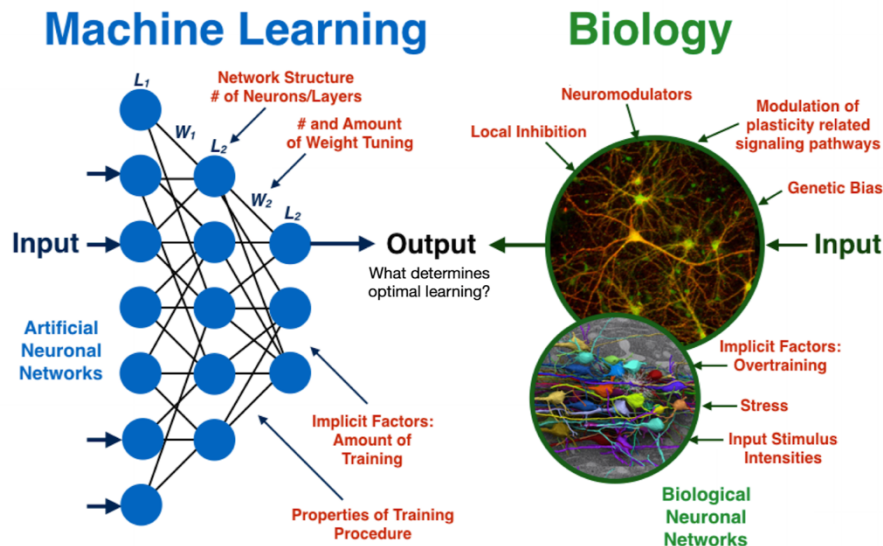


# LDABNN – Benjamin Grewe – 2022/2023

## Introduction – Lecture 01 – Benjamin Grewe

Motivation: Advancing Deep Learning and Neuroscience



### Course Overview:

- 21.09 – Introduction. Brain – Deep Net Analogies. (Grewe)
- 28.09 – Plasticity in the Brain. (Grewe)
- 05.10 – Training Methods for Deep Artificial Nets. (Grewe)
- 12.10 – Learning Rules. (Grewe)
- 19.10 – Deep Reinforcement Learning and the Dopamine System. (Grewe)
- 26.10 – Neural Solutions for Unsupervised Learning. (Grewe)
- 02.11 – Continual learning in artificial and biological systems. (Sorbaro)
- 09.11 - Why spikes? Introduction to SNNs. (Grewe)
- 16.11 – Deep Learning with Spikes. (Zenke)
- 23.11 – Recurrent Neuronal Networks. (Aceituno)
- 01.12 – Predictive Coding. (Aceituno)
- 07.12 – Neuromorphic Systems. (Indiveri)
- 15.12 – Comparing Representations of Biological and Artificial Networks. (Grewe)
- 22.12 – Self Organization. (Malsburg)

### What to Expect? What will you be able to do after the Lecture?

- Able to read and interpret systems neuroscience papers and to draw inspiration from natural network systems.
- Able to train biologically plausible neuronal networks (FA and spiking).
- Overview about biological learning rules and principles of learning.
- Know the relation of dopamine and RL.
- Understand the basic deep learning concepts of meta-learning, unsupervised learning and recurrent neuronal networks.

### Examples of Deep Learning Applications

- DL Driving Driverless Cars.
- DL for Robotic Control ("Learning Agile and Dynamic Motor Skills for Legged Robots").

- DL for Speech Generation.
- DL for Text Generation (GPT-3).
- Image Recognition and Medical Diagnostics (“Using AI to predict breast cancer and personalize care”).
- DL for Playing Games (“Innateness, AlphaZero and Artificial Intelligence”)

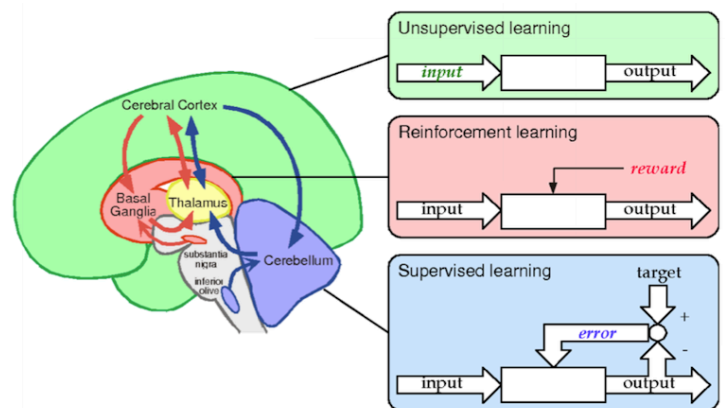
### DL Challenges – Solved by the Human Brain?

Papers: “A Berkeley View of Systems Challenges for AI” & “Complementary roles of basal ganglia and cerebellum in learning and motor control”.

- **Continual Learning.** Being able to learn multiple tasks sequentially.
- **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.
- **Composable AI systems.** Combining multiple systems to solve complex tasks.

(The human brain is comprised of over 200 different brain areas that all learn differently).

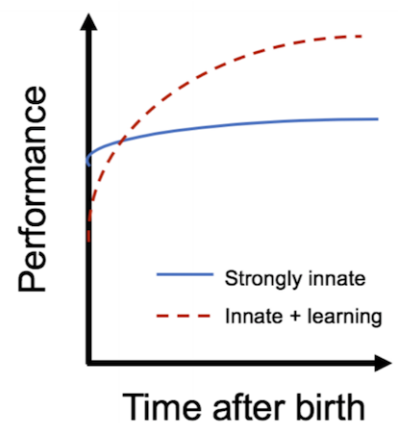
- **Unsupervised learning of useful data representations.**
- **Fast learning and generalization from a few data examples.** (There are only  $10^7$  seconds in a year, so a child would need to ask one question very second of his life to receive a comparable volume of labeled data. In fact, there is a mismatch between the available pool of labeled data and how quickly children learn. Thus, children cannot only rely on supervised algorithms to learn to categorize objects.



### The Human Brain as Universal Learning Machine

Is the brain a Universal Learning Machine? A species using the mixed strategy may thrive if that strategy achieves a higher asymptotic level of performance.

Papers: “Deep Learning: A Critical Appraisal” & “A Critique of Pure Learning: What Artificial Neural Networks can Learn from Animal Brains” & “A Path Towards Autonomous Machine Intelligence”.



### The C. Elegans genome stores neuronal wiring!

The simple worm C. Elegans, for example, has 302 neurons and about 7000 synapses and in each individual of an inbred strain, the wiring pattern is exactly the same. (Chen et al., 2006).

### The Human Brain is mostly Learned.

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  $3.5 \times 10^{15}$  bits (approx. 400 TB) to specify all  $10^{14}$  connections in the brain.

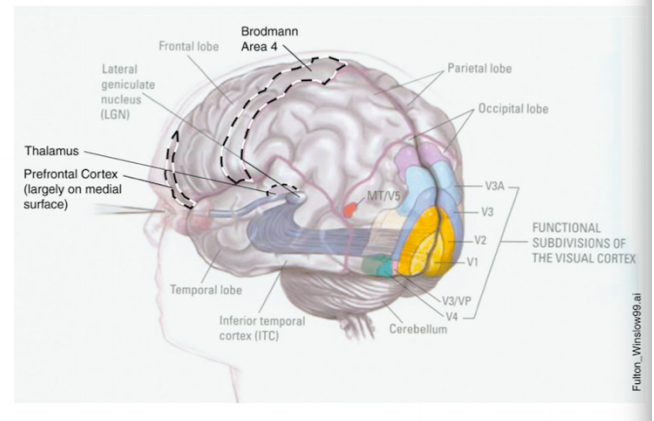
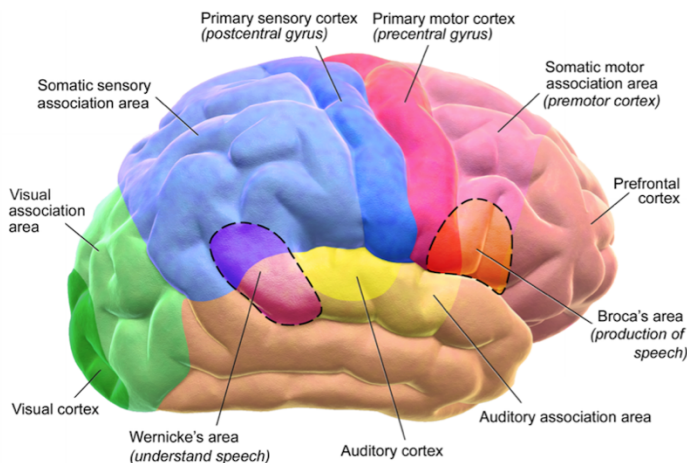
### How did Human Intelligence Emerge?

The brain capacity has been constantly increasing during our evolution.

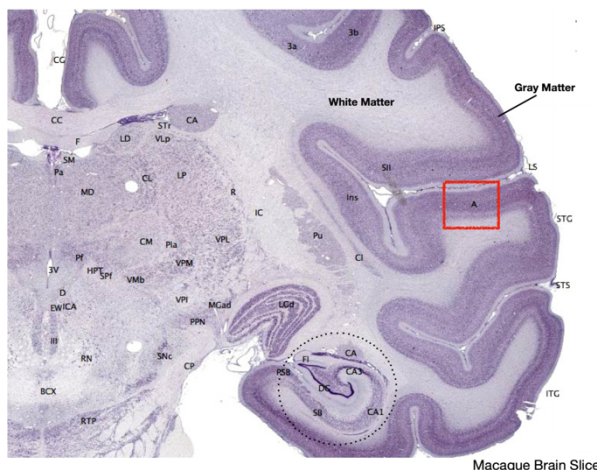
Paper: “The Evolution of Intelligence in Mammalian Carnivores”.



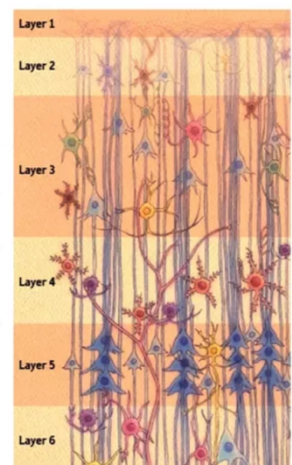
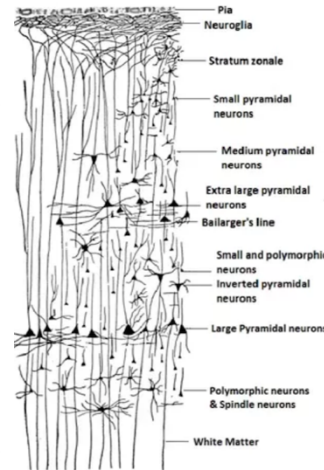
## The Mammalian Neocortex – The Soul of Human Intelligence



- It is the top layer of the cerebral hemispheres, 2-4 mm thick, and made up of six layers, labelled I to VI.
- The neocortex is part of the cerebral cortex (along with the archicortex and paleocortex, which belong to the limbic system).
- It is involved in higher functions such as sensory perception, generation of motor commands, spatial reasoning, conscious thought, and language.
- The neocortex consists of grey matter surrounding the deeper white matter of the cerebrum.
- While the neocortex is smaller and smoother in rats and some other small mammals, 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.
- In humans the neocortex accounts for about 76% of the brain's volume.



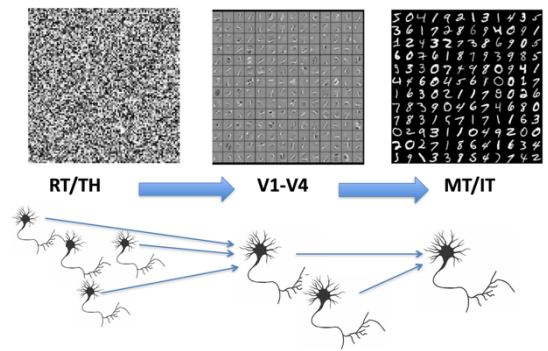
Macaque Brain Slice



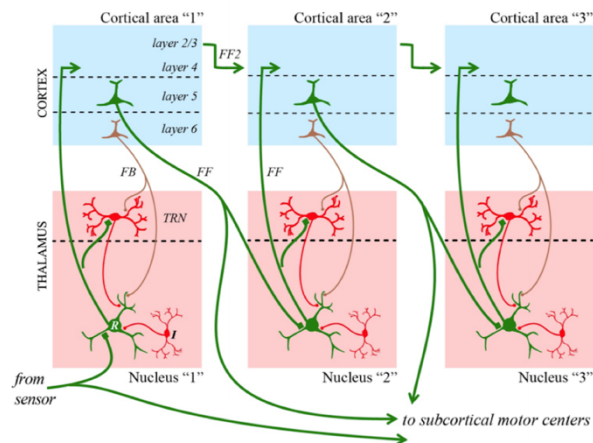
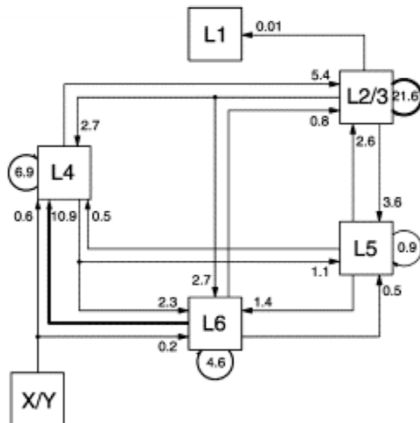
| Layers                   | Components                                                                      | Schematic | Afferents                                  | Efferents                                                         |
|--------------------------|---------------------------------------------------------------------------------|-----------|--------------------------------------------|-------------------------------------------------------------------|
| I – Molecular            | Axons and Dendrites (Cell processes)                                            |           | From other regions of Cortex and Brainstem | 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                                                       |

Modern AI was inspired by the Neocortex.

- Hierarchical Information Processing in the Neocortex



- The Canonical Cortical Circuit  
Paper: "Neuronal Circuits of the Neocortex".



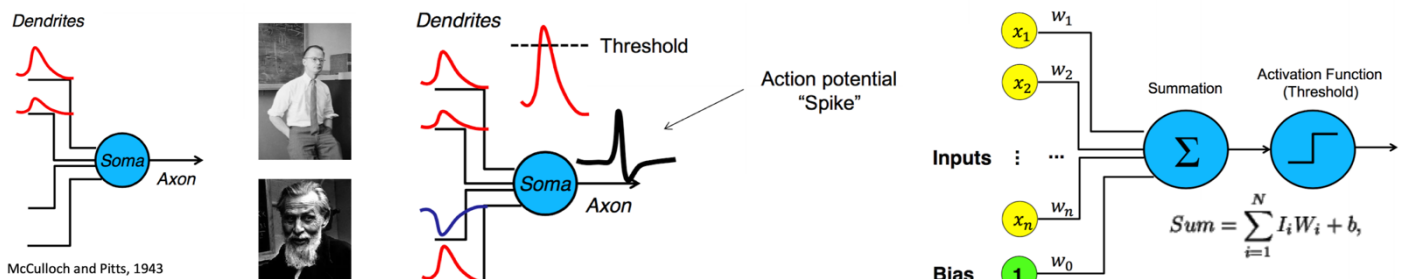
Hubel & Wiesel Experiments in the late 50s

- Orientation and Direction Selective Neurons in Cortex.

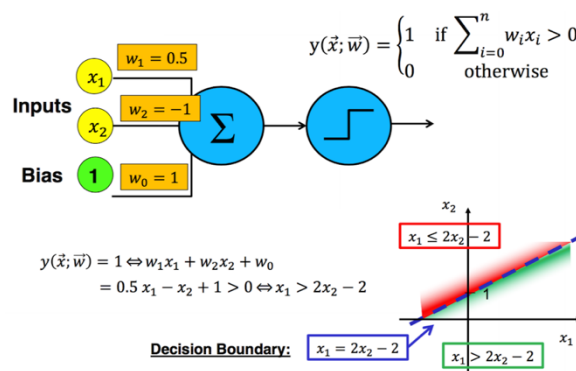
The McCulloch and Pitts Neuron (MCP) was inspired by Cortical Neurons.

The Perceptron was developed based on the MCP Neuron by Frank Rosenblatt in 1957.

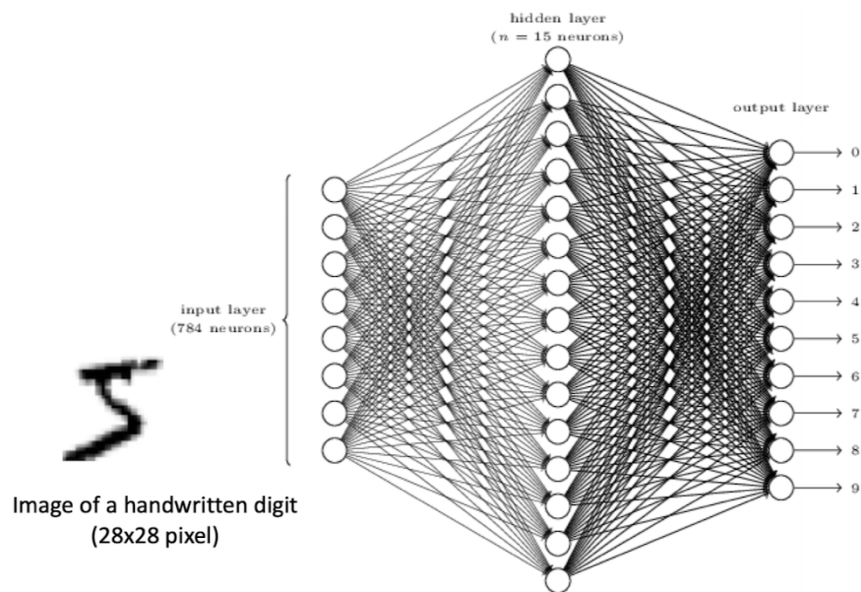
MCP-Neuron integrates only binary values. While the Perceptron integrates non-boolean values where every value is associated with a weight.



The Perceptron as Feature Detector.

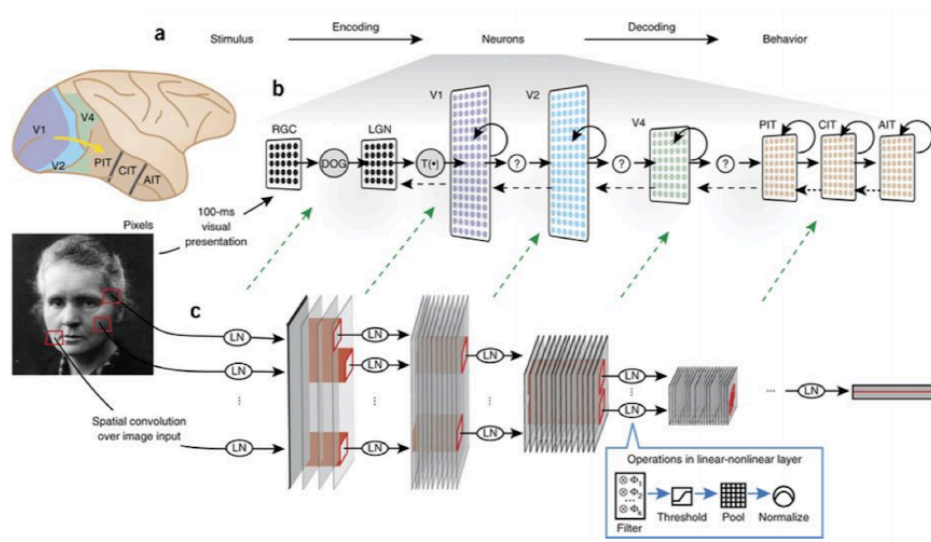


## Solving Complex Classification Tasks with MLPs



## Parallels between Artificial and Biological Networks

Papers: “Using goal-driven deep learning models to understand sensory cortex” & “Performance-optimized hierarchical models predict neural responses in higher visual cortex” & “Deep Supervised, but Not Unsupervised, Models May Explain IT Cortical Representation”.



## Plasticity in the Brain – Lecture 02 – Benjamin Grewe

### Why is this Topic important?

- 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.
- Understanding biological plasticity might help to better understand how (hierarchical) learning is organized in the brain (Learning&Memory).
- Many neural disorders such as dementia relate to a disturbance in neuronal plasticity that cause neuronal networks to become dysfunctional.



## Content of the Lecture

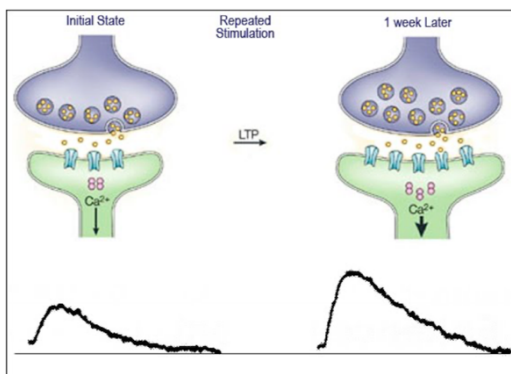
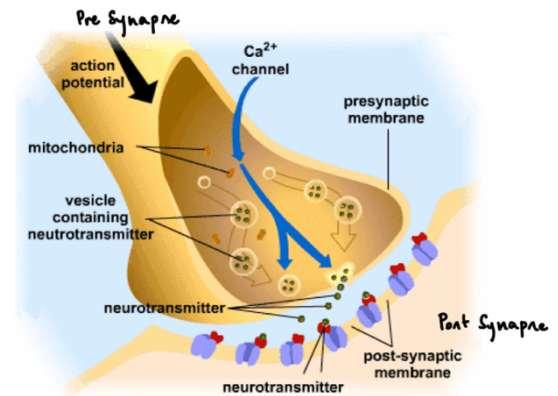
- Why Do We Need Plasticity?
- Synaptic Plasticity
  - Time Scales of Synaptic Plasticity (Short Term, LTP/LTD).
  - Homeostatic Plasticity.
  - Hebb's Idea and STDP.
  - Heterosynaptic Plasticity.
- The Hippocampus as a Model System to Study Neural Plasticity
  - LTP and LTD Induction in the Hippocampus.
  - Molecular Basis of Synaptic Plasticity.
- Non-Synaptic Plasticity
  - Neuronal Excitability and Spike Generation.
  - Axonal Modulation (Shunting, Frequency Filtering).
  - Alterations of Dendritic Excitability.

## Defining Learning, Memory and Plasticity

- **Plasticity**, it allows the acquisition of knowledge/information and the formation of a memory through experience.
- **Memory**, it is a storage of information that can be recalled at a later stage in time.
- **Note**, learning results in memory – which has a further outcome – it can change future behavior.

## Synaptic Plasticity – A Short Recap of Synaptic Function

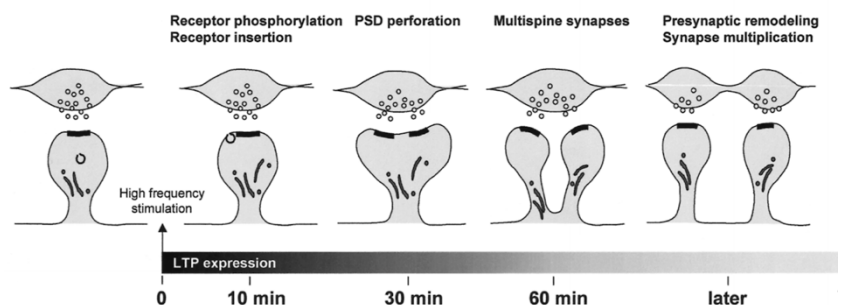
In the presence of a presynaptic action potential, Calcium channels open allowing an increase of calcium, such that glutamate in vesicles fuses with the synapses and crosses them. Then AMPA are activated and neurotransmitters attach to the receptors. EPSP happens.



Amplitude increases with the number of receiving AMPA channels, hence with LTP the amplitude of EPSP increases due to an increase of neurotransmitters released and received.

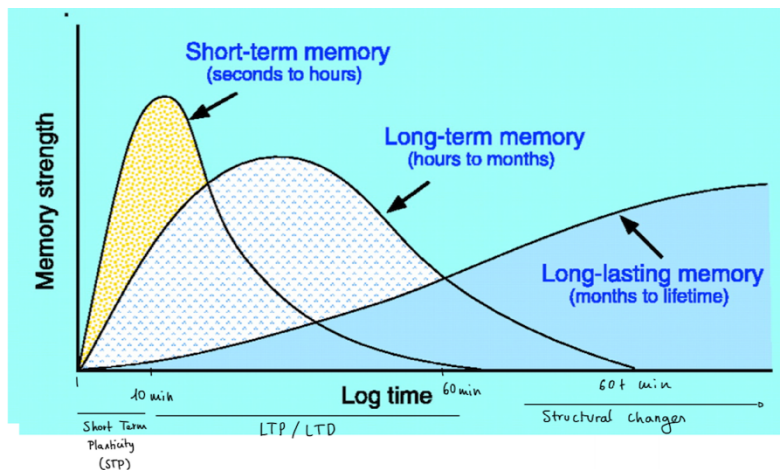
## Synaptic Plasticity Alters the Inter-Neuron Connection Strength

1. Synaptic Density Size Changes (short term).
2. AMPA/NMDA ratio changes (short term).
3. Number of spines changes (long term).



**Note that NMDA stays constant!**

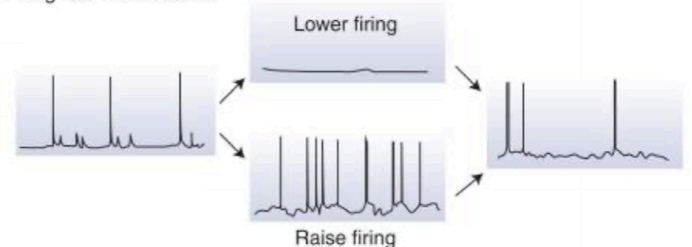
## Timescales of Neuronal Plasticity



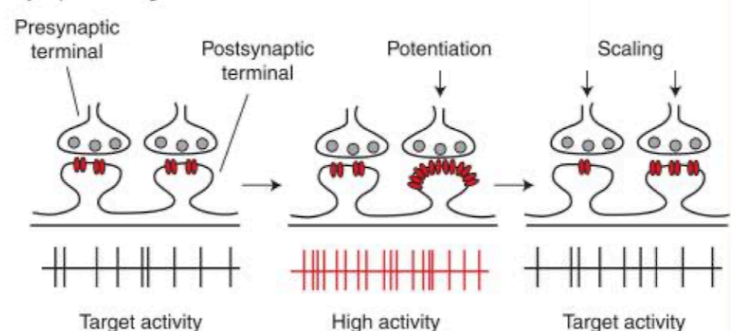
### Necessity of Homeostatic Plasticity

Homeostatic plasticity is a mechanism that ensures that the activity of neurons among levels remains constant. It is the process by which the brain adjusts the strength of its synapses to maintain a consistent level of activity. This process helps to balance the overall activity of the brain and maintain a stable internal environment. For example, LTP may occur in response to a particularly strong or meaningful stimulus, resulting in an increase in synapse strength. This increase in strength may be necessary for the formation of a new memory. However, if the increased strength of the synapses were to persist indefinitely, it could lead to an imbalance in activity in the brain. Homeostatic plasticity can help to restore balance by adjusting the strength of other synapses in response to the LTP-induced increase. In this way, LTP and homeostatic plasticity can work together to support the formation of long-term memories while also maintaining the overall stability of the brain.

#### A Firing rate homeostasis



#### B Synaptic scaling

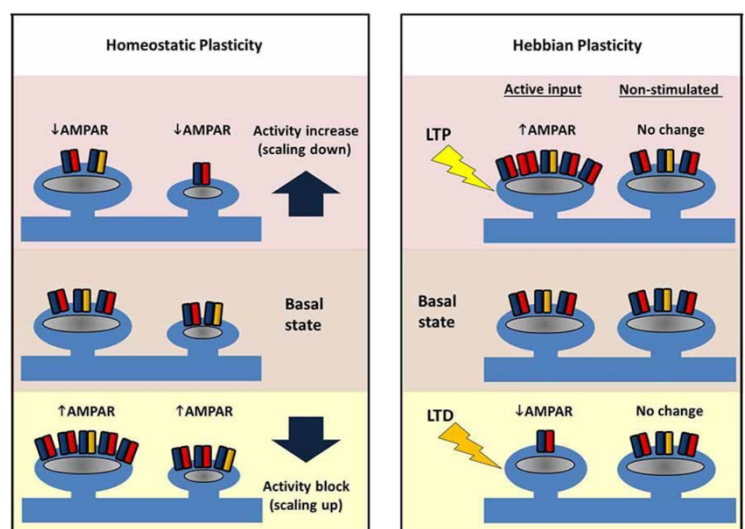


Papers: “*Homeostatic Plasticity in the Developing Nervous System*” & “*Homeostatic Synaptic Plasticity: Local and Global Mechanism for Stabilizing Neuronal Function*”.

### Homeostatic & Hebbian Plasticity

From *The Organization of Behavior* by Donald Hebb, 1949.

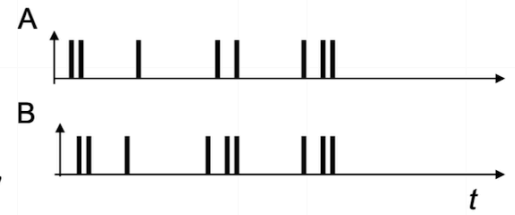
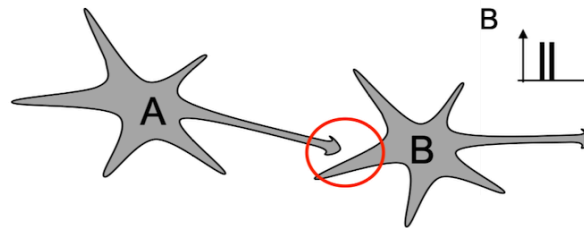
“When an axon of cell A is near enough to excite cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A’s efficiency, as one of the cells firing B, is increased”.





Hebb postulated that this behavior of synapses in neuronal networks would permit the networks to store memories. A Hebbian Synapse is a “coincidence detector”.

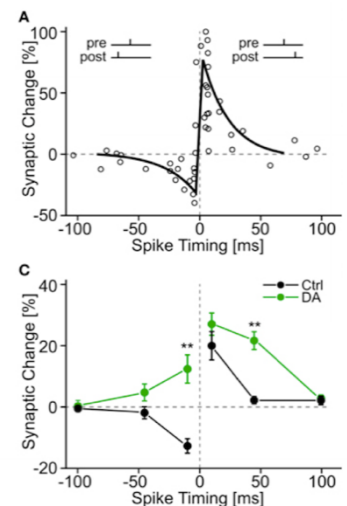
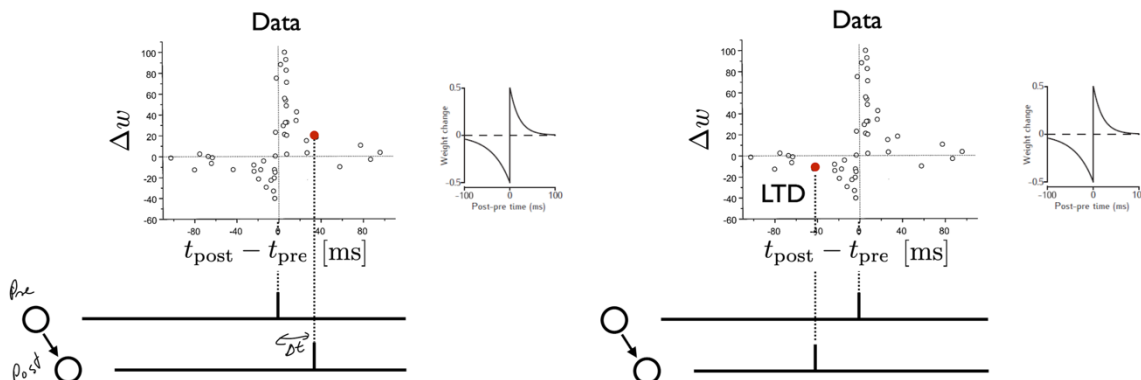
The first real demonstration of this paradigm can be found in STDP.



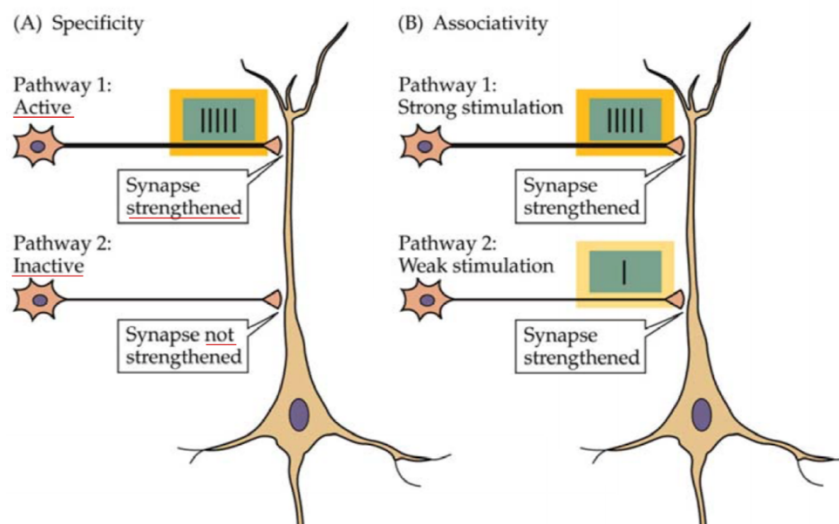
### Examples of Hebbian Learning – Spike Timing Dependent Plasticity (STDP)

STDP represents a form of neural plasticity, it refers to the process by which the strength of a synapse is modified based on the timing of action potentials in the neurons. According to the STDP rule, if an action potential in one neuron (the presynaptic neuron) occurs shortly before an action potential in a second neuron (the postsynaptic neuron), the synapse between the two neurons becomes stronger. On the other hand, if the action potential in the presynaptic neuron occurs after the action potential in the postsynaptic neuron, the synapse becomes weaker.

Papers: “Synaptic Modifications in Cultured Hippocampal Neurons: Dependence on Spike Timing, Synaptic Strength, and Postsynaptic Cell Type” & “Gain in Sensitivity and Loss in Temporal Contrast of STDP by Dopaminergic Modulation at Hippocampal Synapses”.



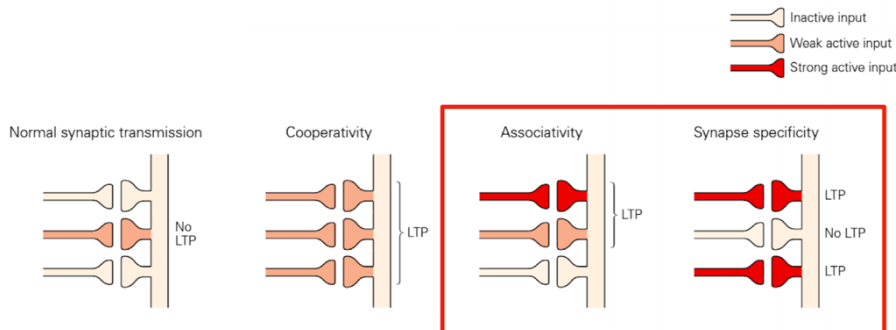
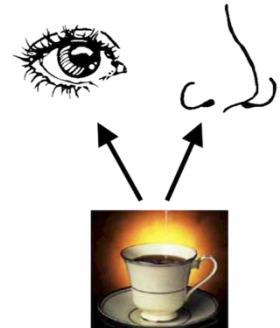
### Hebb's Idea How Neurons Can Learn Associations



## Hebbian LTD and LTP are Input Specific

1. Cooperativity (induction threshold).
2. Input/Synapse Specificity.
3. Enables Associative Learning.

$$\dot{w} = H(\text{pre}, \text{post})$$



The weight update is a function  $H$  that evaluates time pre and post.

Papers: “*Neural Ensemble Dynamics Underlying a Long-Term Associative Memory*” & “*The Ups and Downs of Hebb Synapses*” & “*Neuromodulated Spike-Timing-Dependent Plasticity, and Theory of Three-Factor Learning Rules*”.

## What is Geoffrey Hinton’s Problem with Hebbian Learning?

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

### One Solution: Three Factor Hebbian Learning Rules

The three-factor Hebbian learning rule adds two additional factors to the original Hebbian learning rule:

- **Coincidence:** the strength of a synapse is only modified if the activity of the two neurons is correlated in time.
- **Repetition:** the more often the activity of the two neurons is correlated, the greater the effect on the strength of the synapse.
- **Intensity:** the stronger the activity of the two neurons, the greater the effect on the strength of the synapse.

According to the three-factor Hebbian learning rule, the strength of a synapse is increased when the activity of the two neurons is correlated in time, is repeated, and is strong. Conversely, the strength of a synapse is decreased when the activity of the two neurons is not correlated in time, is not repeated, or is weak.

## Non-Hebbian Plasticity – Heterosynaptic Plasticity

Heterosynaptic Plasticity refers to the process by which the strength of one synapse is modified in response to activity at a different synapse.

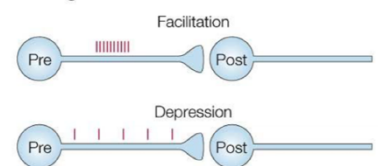
Papers: “*Is Heterosynaptic Modulation Essential for Stabilizing Hebbian Plasticity and Memory*” & “*Heterosynaptic Plasticity Underlies Aversive Olfactory Learning in Drosophila*”.

## Homosynaptic vs Heterosynaptic Plasticity

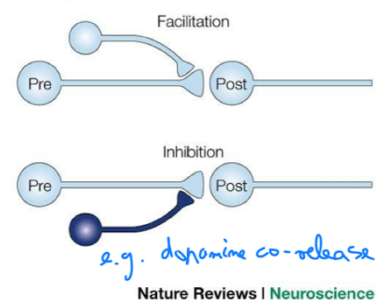
There are two broad categories of synaptic plasticity, generally referred to as **homosynaptic** and **heterosynaptic** plasticity.

- **Homosynaptic** plasticity is what we have been discussing with the Hebbian synapses: a synapse-specific strengthening (facilitation) or weakening

### a Homosynaptic (activity-dependent) plastic change



### b Heterosynaptic (modulatory input-dependent) plastic change



Nature Reviews | Neuroscience

(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.

- **Heterosynaptic** 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.

In the previous figure: homosynaptic and heterosynaptic mechanisms for long-term plasticity. a) The plastic changes that underlie long-term memory follow a homosynaptic rule, i.e., 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 or a decrease. 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 or in a decrease in synaptic strength.

### The Hippocampus as Model System to Study Plasticity

Hippocampus is a model system of learning and memory. The role of Hippocampus in learning and memory has been shown with rat experiments with the Morris Water Maze (MWM). MWM is a 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.

### Neural Plasticity in the Hippocampus

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 aminophosphonovaleric acid (AP5) does not affect synaptic transmission in the hippocampus, but prevents the LTP following brief high-frequency stimulation.

### Non-Hebbian Plasticity – Towards the Behavior Timescale

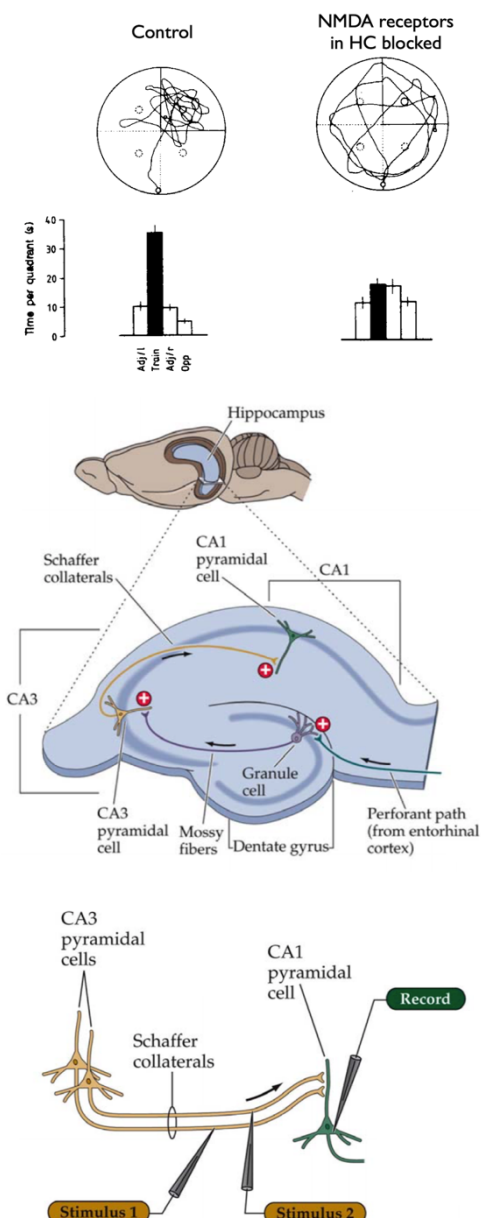
Hippocampus neurons learn spatial representations.

Paper: “**Behavioral time scale synaptic plasticity underlies CA1 place fields**”.

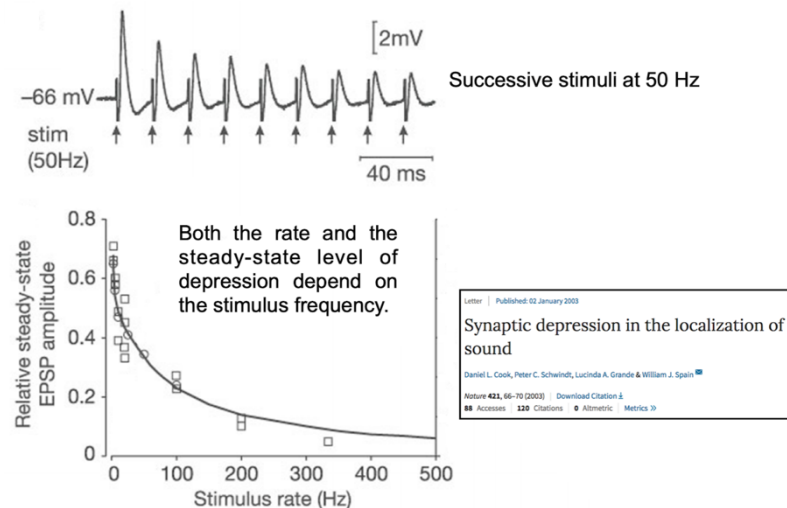
### Most Studied Synapse in Hippocampus: CA3 → CA1

The main pyramidal cell layers in Hippocampus are the CA1-4 regions (principally CA1 and CA3) and the dentate gyrus. The **Schaffer Collateral / Associational Commissural Pathway** is derived from axons that project from the CA3 region of the hippocampus to the CA1 region. The axons either come from neurons in the same hippocampus (ipsilateral) or from the other hippocampus (contralateral). These latter fibers are termed commissural fibers, as they cross from one hemisphere of the brain to the other. This pathway is utilized very extensively to study NMDA receptor-dependent LTP and LTD.

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



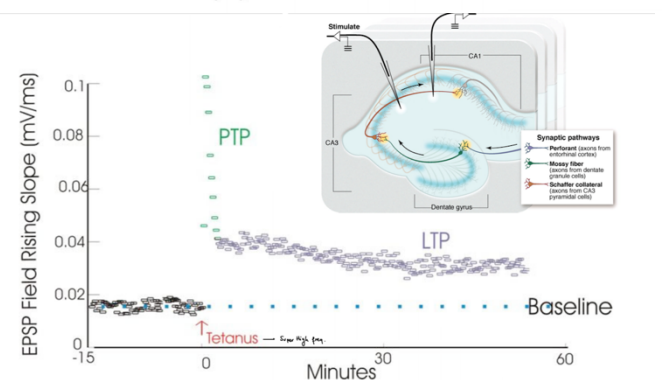
increased spiking strength as there is Residual  $\text{Ca}^{2+}$  in the cell. Short-term depression at about 40ms time frame can be observed if the CA3 to CA1 pathway is stimulated at 50hz it leads to a reduction in the EPSP which is dependent on the frequency of activation. LTP is measured in the hippocampus. The CA3 pathway is given a fast stimulus of (range 50 – 200 hz) 100 hz known as tetanus. This leads to a stronger post tetanic potentiation caused by the accumulation of  $\text{Ca}^{2+}$  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. (Estimated through in-vitro recordings).



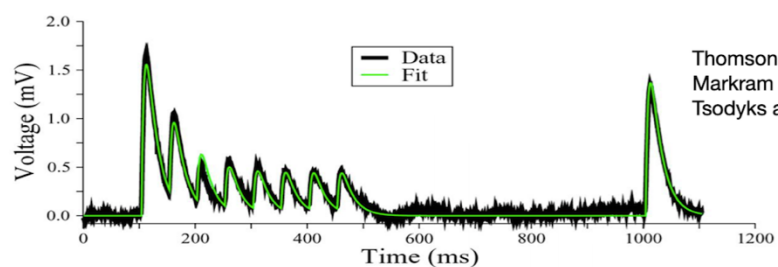
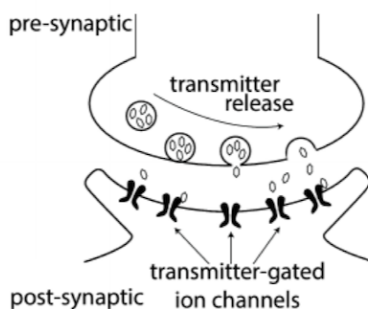
### Short-Term Synaptic Facilitation/Depression

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.
- Synapses in different cortical areas can have varied forms of plasticity, being either STD-dominated, STF-dominated, or showing a mixture of both forms.



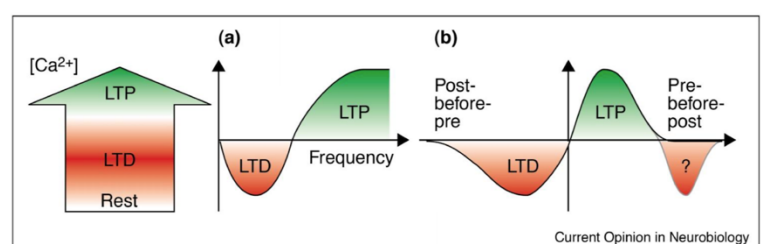
PTP believed to be caused by a large accumulation of  $\text{Ca}^{2+}$  in the terminal caused by a high frequency tetanic stimulation.



Thomson et al., 1993  
Markram et al., 1998  
Tsodyks and Markram 1997

### Synaptic Plasticity Strongly Depends on Calcium Levels

1. Level and timing of  $\text{Ca}^{2+}$  rise in spine determines LTD or LTP.
2. Low frequency synaptic firing (approx. 5 Hz) produces LTD, high-frequency synaptic firing (approx. 50 to 100 Hz) produces LTP.
3. The same  $\text{Ca}^{2+}$  rules may underlie "spike-timing-dependent synaptic plasticity" (STDP).

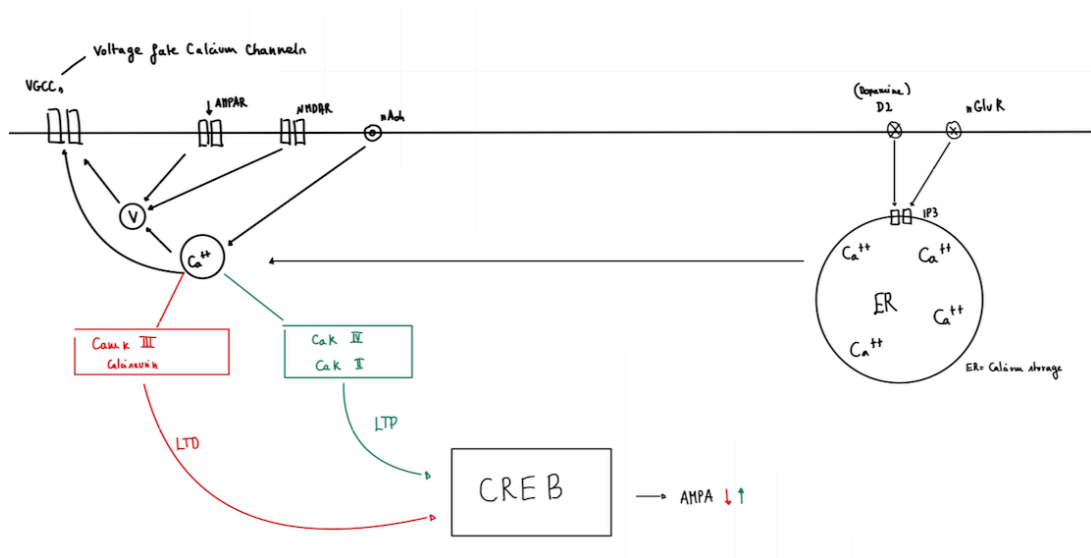




## Intracellular Plasticity Signaling Pathways

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 depolarized or hyperpolarized the cell becomes.

- **What controls LTP and LTD:**
  - CREB is controlled by many pathways that are dependent 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 depolarization opening voltage gated Ca channels as well as NMDA channels that further depolarize the cells. Dopamine D2 when binds in leads to Ca<sup>2+</sup> increase from the ER, which leads to increased Ca.
- **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 lead to Camkinse 2 and Calmodulin that reduces the phosphorylation (activation) of CREB thus AMPA receptors.



## Other Forms of Non-Synaptic (Intrinsic) 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 be summed up leading to AP, the synapse location will also modulate the excitability nearer the soma will be higher as there isn't a loss of charge.

## Training Methods for Deep ANNs – Lecture 03 – Benjamin Grewe

### Why is this Topic important?

Papers: “Cognitiva 85” & “Learning Representations by Back-Propagating Errors”.

- 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.

### Content of the Lecture:

- Recap: The Backpropagation of the Error Method (BP).
- Feedback Alignment (FA).
- Variants of FA (Direct, Indirect, Bi-Directional).
- Target Propagation and its Variants.
- Local (layer wise) Training for Deep Neuronal Networks.
- The Deep Feedback Control Method.

### Recap: The Backpropagation of the Error Method (BP)

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**. 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. 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:

$$\mathbf{h}^j = \sigma^j(W^j \cdot \mathbf{h}^{j-1}) = \sigma^j(z^j), \quad j = 1, \dots, L,$$

Where  $\mathbf{h}^j$  is the state of the  $j$ -th hidden layer and  $z_i^j = \sum_{k=0}^{N^j} w_{ik}^j h_i^{j-1}$  the input for the  $i$ -th neuron in the  $j$ -th hidden layer  $\mathbf{h}^L$  is the output layer and  $\mathbf{h}^0 = \mathbf{x}$  the input layer. The forward-mapping is defined by the non-linear activation function  $\sigma^j$  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  $W^j = (w_0^j, w_1^j, \dots, w_N^j)$  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  $\sigma^j$ , 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  $\theta_w^{j,l} = W^k, k = j+1, \dots, l$ , we can write the  $l$ -th layer  $\mathbf{h}^l$  as a function of the  $j$ -th layer  $\mathbf{h}^j$ , given the parameters  $\theta_w^{j,l}$ :

$$\mathbf{h}^l = \mathbf{h}^l(\mathbf{h}^j; \theta_w^{j,l})$$

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

$$\mathcal{L}_{MSE}(\mathbf{h}^L(\mathbf{x}; \theta_w^{0,L}), \mathbf{y}) = \frac{1}{2D} \sum_{i=1}^D (\mathbf{h}^L(\mathbf{x}_i; \theta_w^{0,L}) - y_i)^2$$

And the **Cross-Entropy Loss** (CE) for classification with C classes:

$$\mathcal{L}_{CE}(\mathbf{h}^L(\mathbf{x}; \theta_w^{0,L}), \mathbf{y}) = - \sum_{c=1}^C y_c \log h_c^L(\mathbf{x}_c; \theta_w^{0,L})$$

If the backpropagation algorithm is used, updating the weights is done by taking the derivatives of the loss with respect to all weights:

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

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

### Backpropagation

Backpropagation (BP) is a widely used algorithm in training feedforward neural networks for supervised learning. 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 naïve 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.

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  $z_j^l$ :

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

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

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

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

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

We note, that the gradient can also be written with a dependency to the weight:

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

Thus, we arrive at the recursive form:

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

Which gives us the weight update equation as follows:

$$w_k'^l = w_k^l - \eta \Delta w_k^l = w_k^l - \eta \frac{\partial \mathcal{L}}{\partial w_k^l} = w_k^l - \eta h_k^{l-1} \delta_j^l$$

Where  $\eta$  is the learning rate.

### Biological Plausibility Issues

- The backpropagation computation is purely linear, whereas biological neurons interleave linear and non-linear operations.
- If the feedback paths were used to propagate credit assignment by backpropagation, they would need precise knowledge of the derivatives of the non-linearities at the operating point used in the corresponding feedforward computation.
- Similarly, these feedback paths would have to use exact symmetric weights (with the same connectivity, transposed) of the feedforward connections.
- Real neurons communicate by (possibly stochastic) binary values (spikes).
- The computation would have to be precisely clocked to alternate between feedforward and backpropagation phases.
- It is not clear where the output targets would come from.

### Feedback Alignment

Papers: “Random Synaptic Feedback Weights Support Error Backpropagation for Deep Learning” & “Bio-Inspired Computer Vision: Towards a Synergistic Approach of Artificial and Biological Vision”.

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**. 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 example (such as MNIST), the method has been shown to work almost as well as regular back-propagation. We note the change in weight update:

### Backpropagation

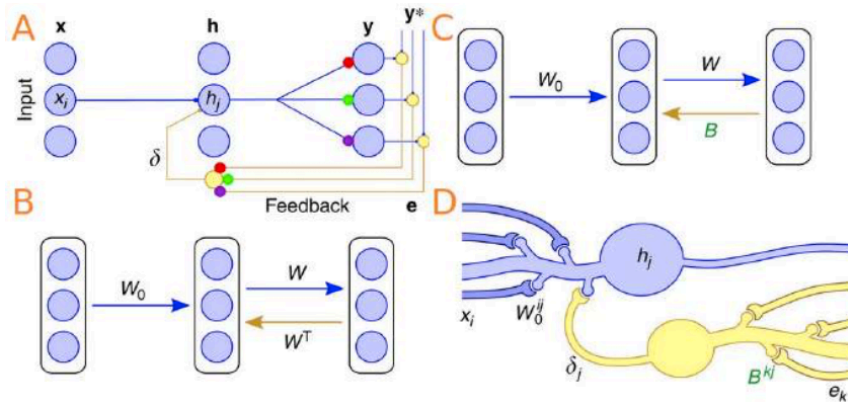
$$\Delta w_k^l = h_k^{l-1} \delta_{BPj}^l = h_k^{l-1} \sum \delta_{BPj}^{l+1} w_{jk}^{l+1} \sigma'(z_j^l)$$

### Feedback Alignment

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

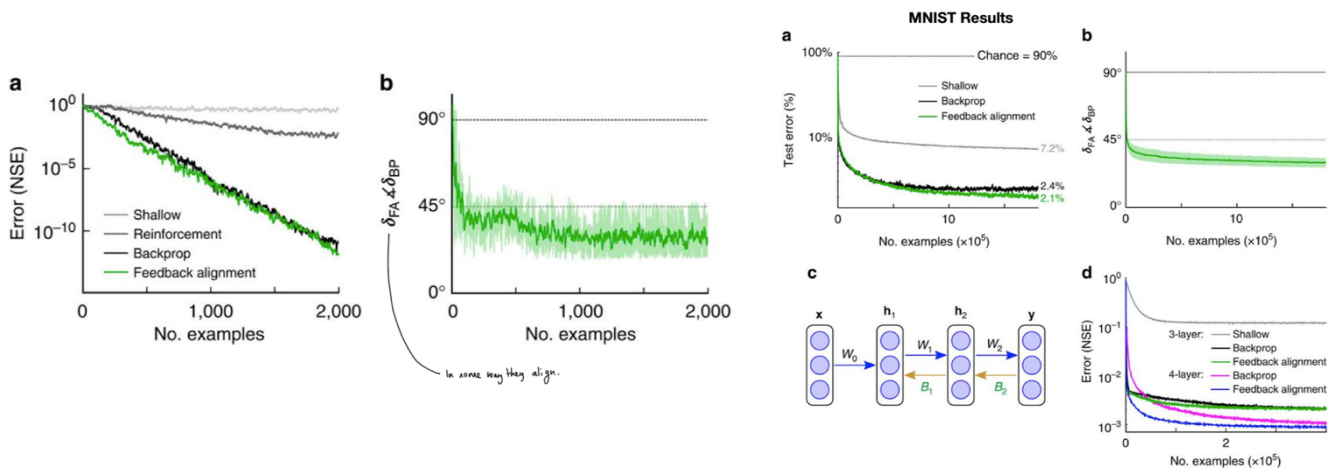
Where  $\beta^l$  is a random matrix with fixed weights (does not get updated) belonging to the l-th layer.





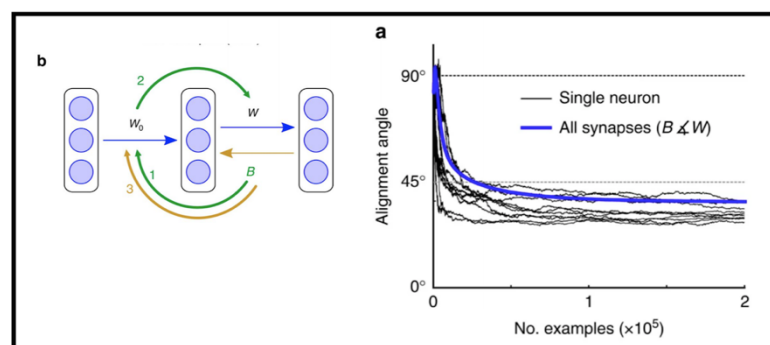
- (A) The Backprop learning algorithm requires that neurons know each others' synaptic weights, for example, the three coloured synapses on the feedback cell at the bottom must have weights equal to those of the corresponding coloured synapses in the forward path.
- (B) Backprop computes teaching, or modulator, vectors by multiplying the error vector  $e$  by the transpose of the forward weight matrix  $W$ , that is,  $\delta_{BP} = W^T e$ .
- (C) Our feedback alignment method replaces  $W^T$  with a matrix of fixed random weights,  $B$ , so that  $\delta_{FA} = B e$ . 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 unit (matrix superscripts denote single synapses). 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.

## FA is on par with BP for Linear Classification Problems & Works in Multilayer Networks



### Why Does Feedback Alignment Work?

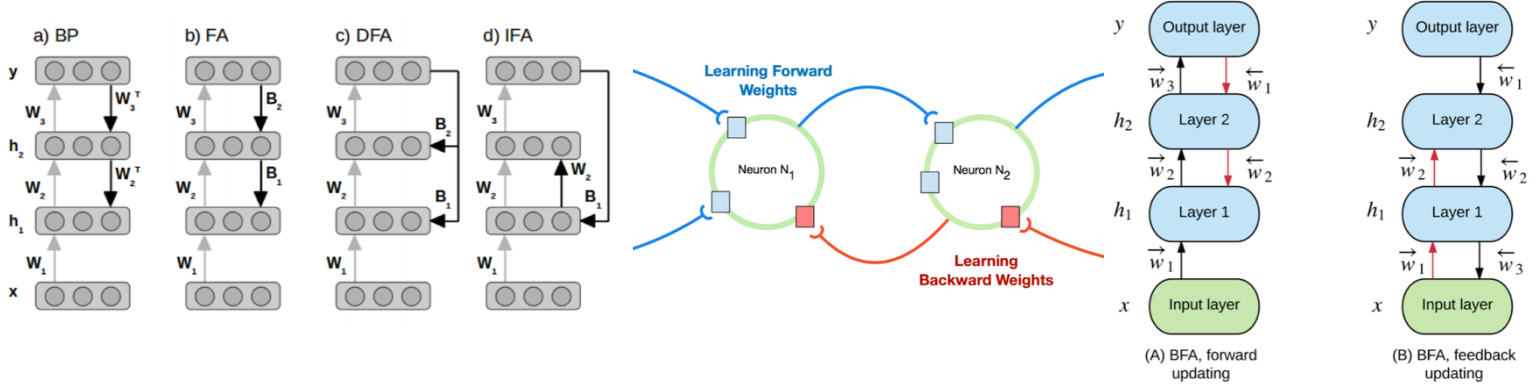
Some mathematical reasoning and the convergence proof is shown in the original paper and stated again in a follow-up paper. Intuitively: for FA the feedback weights are fixed, but if the forward weights are adapted, they will approximately align with the pseudo-inverse of the feedback weights in order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



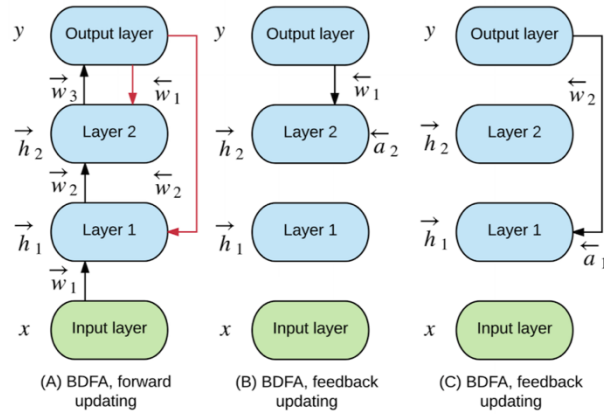
## Variations of Feedback Alignment

Papers: “*Direct Feedback Alignment Provides Learning in Deep Neural Networks*” & “*Adaptive Bidirectional Backpropagation: Towards Biologically Plausible Error Signal Transmission in Neural Networks*”.

There are variations of feedback alignment, which show to be useful especially for deeper network architectures. The variations shown below are Feedback Alignment (FA), Direct Feedback Alignment (DFA), Indirect Feedback Alignment (IFA), Bi-directional Feedback Alignment (BFA) and Bi-directional Direct Feedback Alignment (BDFA).

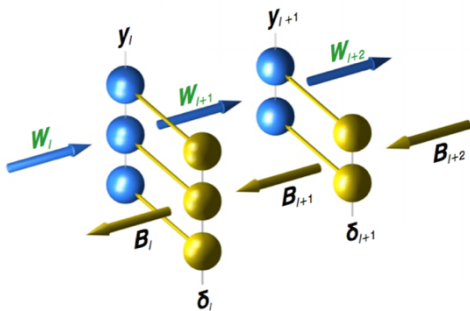


The figure to the left gives an overview of different error transportation configurations. Grey arrows indicate activation paths and black arrows indicate error paths. Weights that are adapted during learning are denoted as  $W_i$ , and weights that are fixed and random are denoted as  $B_i$ . The figure to the right describes the Bi-directional Feedback Alignment. Black arrows represent forward activation paths. Red arrows indicate error (gradient) propagation paths. The figure below introduces the last discussed variation, i.e., the Bi-directional Direct Feedback Alignment.



## Deep Learning without Weight Transport

Papers: “*Deep Learning without Weight Transport*”.



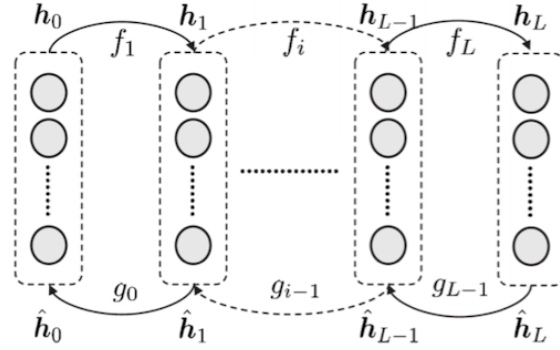
$$\delta_l = \phi'(y_l) B_{l+1} \delta_{l+1}$$

$$\Delta W_{l+1} = -\eta_W \delta_{l+1} y_l^T$$

$$\Delta B_{l+1} = \eta_B \delta_l \delta_{l+1}^T$$

## Target Propagation as a BP Alternative

The concept of Target Propagation (targetprop) goes back to Lecun (1986). The intuition is simple: 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. 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 the biological plausibility issues described before.



### Forward pass

$$\mathbf{h}^j = f^j(\mathbf{h}^{j-1}) = \sigma^j(W^j \mathbf{h}^{j-1}) = \sigma^j(\mathbf{z}^j), \quad j = 1, \dots, L, \quad (14)$$

### Backward pass

$$\hat{\mathbf{h}}^j = g^j(\hat{\mathbf{h}}^{j+1}), \quad j = L-1, \dots, 0, \quad (15)$$

where  $\hat{\mathbf{h}}^j$  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:

$$\hat{\mathbf{h}}^L = \mathbf{h}^L - \eta \mathbf{e}^L \quad (16)$$

$$\mathbf{e}^L = \frac{\partial \mathcal{L}}{\partial \mathbf{h}^L} \quad (17)$$

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

$$\Delta W^j = -\eta_{f^j} \frac{\partial \mathcal{L}^j(\mathbf{h}_{true}^j, \hat{\mathbf{h}}^j)}{\partial W^j} \quad (18)$$

**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  $Q^j$ :

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

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

$$\mathcal{L}_{rec}^j(g^j(f^{j+1}(\mathbf{h}^j)), \mathbf{h}^j) = \|g^j(f^{j+1}(\mathbf{h}^j)) - \mathbf{h}^j\|^2 \quad (20)$$

$$\Delta Q^j = -\eta_{g^j} \frac{\partial \mathcal{L}_{rec}^j}{\partial Q^j} \quad (21)$$

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

$$\mathcal{L}^j = \mathcal{L}_{local}^j(\mathbf{h}_{true}^j, \hat{\mathbf{h}}^j) + \mathcal{L}_{rec}^j(g^j(f^{j+1}(\mathbf{h}^j)), \mathbf{h}^j) \quad (22)$$

### 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<sup>[27]</sup> approached this problem by proposing a correction term in the propagated target  $\hat{\mathbf{h}}^j$  that should help in getting rid of the reconstruction error:

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

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.

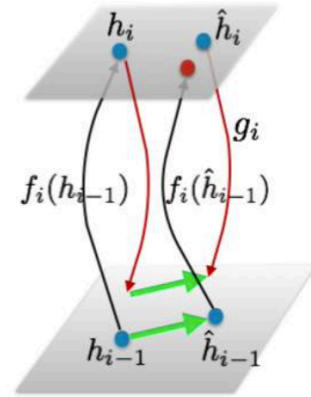


Figure 20: How to compute a target in the lower layer via difference target propagation.  $f^i(\hat{\mathbf{h}}^{i-1})$  should be closer to  $\hat{\mathbf{h}}^i$  than  $f^i(\mathbf{h}^{i-1})$

### Difference Target Propagation

[arxiv.org/abs/1412.7525](https://arxiv.org/abs/1412.7525)

Dong-Hyun Lee<sup>1</sup>, Saizheng Zhang<sup>1</sup>, Asja Fischer<sup>1</sup>, and  
Yoshua Bengio<sup>1,2</sup>

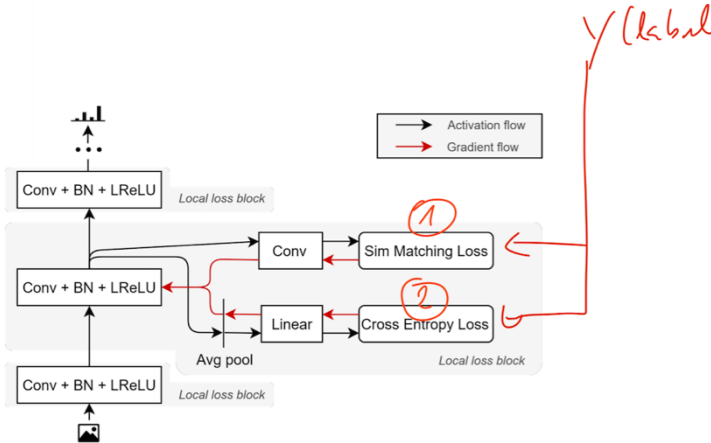
<sup>1</sup> Université de Montréal, Québec, Canada

<sup>2</sup> CIFAR Senior Fellow

## Local Learning

Papers: “Training Neural Networks with Local Error Signals”.

There are approaches that train neuronal networks using Local Layer error signals. 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).



### 1. Similarity Matching Loss (sim)

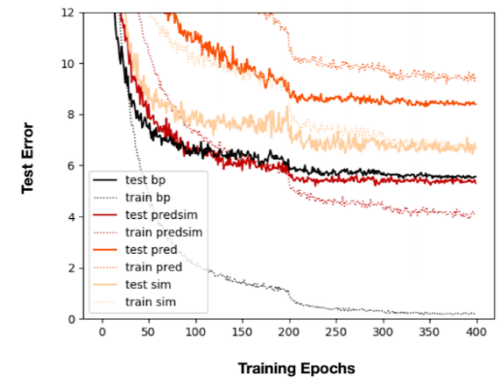
$$L_{sim} = \|S(\text{NeuralNet}(H)) - S(Y)\|_F^2 = \text{L2\_loss}(\text{hidden activations}, \text{labels})$$

### 2. Cross Entropy Loss (pred)

$$L_{pred} = \text{CrossEntropy}(Y, W^T H)$$

## Optimization vs. Generalization

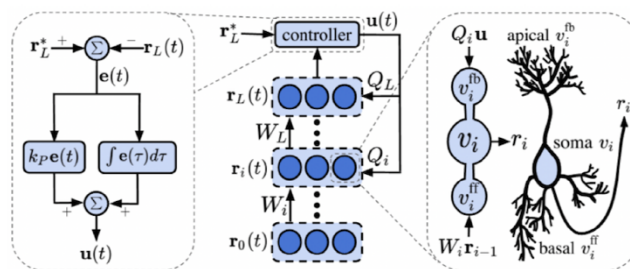
- Back-prop has fastest & lowest drop in training error.
- Local learning is competitive with back-prop in terms of test error.
- Local learning generalizes well.
- But: both **pred-** and **sim-** losses help optimization in a complementary way.
- Indeed, **similarity matchings** helps standard BP.



## Bio-Plausible Deep Learning Through Control

A novel, Bio-plausible Network Learning Algorithm: “Deep Feedback Control”.

It is based on a recurrent loop that stops when the output error is 0.



- $u(t)$  is the error.
- $Q_i$  and  $Q_L$  are set to be the inverse of the feedforward weights.
- The error is multiplied by  $Q_i$  and  $Q_L$  in the different layers to update the weights.
- The gradient is implicit.
- Hebbian type learning rule for  $Q$  that it updates to the transpose of the weight.

Advantages of the Deep Feedback Control (DFC) Algorithm:

- Supports continuous/asynchronous updates.
- Hidden layer activity does not need to be stored (no extra memory required).
- Highly parallelizable, but requires custom hardware.

- Very simple learning rule that is local in space and time solely based on the neuron's activity and effectively implements a delta rule.
- Absence of phases or back-propagation of errors (e.g., as in standard BP).
- The optimization approach "Gauss-Newton" which is fundamentally different from BP and standard gradient descent learning.

## Quiz

### 1. What is true regarding backpropagation update rule?

- a) it is also called generalized delta rule
- b) error in output is propagated backwards only to determine weight updates
- c) there is no feedback of signal at any stage
- d) all of the mentioned

### 2. There is error feedback only at the final stage of the network?

- a) yes
- b) no

### 3. What are general limitations of back propagation?

- a) local minima / saddle point problem
- b) slow convergence
- c) scaling
- d) all of the mentioned

### 4. What are the general tasks that are performed with backpropagation algorithm for training NNs?

- a) pattern mapping
- b) function approximation
- c) prediction
- d) all of the mentioned

## Learning Rules – Lecture 04 – Benjamin Grewe

### Content of the Lecture

- Why (local) Neuronal Learning Rules are Important?
- The Perceptron Learning Rule
- ADALINE and the Delta Rule
- Recap of Hebbian Learning
- Oja's Rule
- The Covariance Rule
- Sanger's Rule
- The Calcium Rule
- The Bienenstock-Cooper-Monroe (BCM) Rule.
- The Triplet Rule (Pfister&Gerstner)
- Extensions of Hebbian Learning Rules (Neo-Hebbian)

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.

## Error Minimization Rules

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

### 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 threshold linear transformation. Given an input vector of values, the output falls into two cases depending on whether the linear transformation is above or below threshold. Specifically, for an input vector  $x$ , corresponding weights  $w$ , and a threshold  $b$ , the output can be denoted as:

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

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:

#### Algorithm: Perceptron Learning Algorithm

```

P ← inputs with label 1;
N ← inputs with label 0;
Initialize w randomly;
while !convergence do
    Pick random x ∈ P ∪ N;
    if x ∈ P and w.x < 0 then
        w = w + x;
    end
    if x ∈ N and w.x ≥ 0 then
        w = w - x;
    end
end
//the algorithm converges when all the
inputs are classified correctly

```

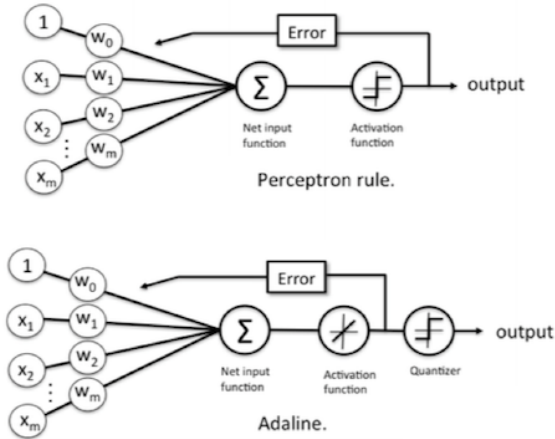
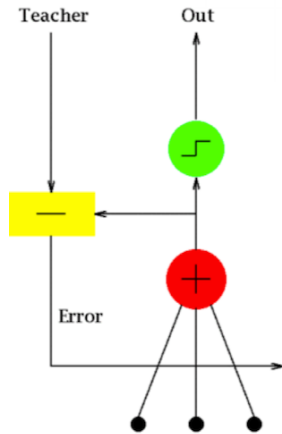
Target      Perceptron Out

$$\Delta w_{ji} = \lambda (t_i - y_i) \cdot x$$

## ADALINE (ADaptive Linear NEuron) Learning Rule

This can be viewed as a slightly modified instance of the perceptron learning rule. In the perceptron rule, the threshold result of the weighted sum of inputs is used for updating the weights in each iteration. In ADALINE, the weighted sum of inputs itself is used to update the weights in training.

- They are both classifiers for binary classification.
- Both have a linear decision boundary.
- Both can learn iteratively, sample by sample (the Perceptron naturally, and ADALINE via stochastic gradient descent).
- Both use a threshold function.



## DELTA Learning Rule

The Delta Rule uses the difference between target activation (i.e., target output values) and obtained activation to drive learning. The weight updates from this equation aims 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:

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

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:

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

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  $\alpha$  and the derivative  $g'$  of the activation function  $g$ . 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  $\alpha$  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.

### DELTA Rule vs Perceptron Learning Rule

We have seen that the DELTA rule and the Perceptron learning rule for training single-layer Perceptrons have a similar weight update equation. However, the two algorithms were obtained from very different theoretical starting points. The Perceptron learning rule was derived from a consideration of how we should shift around the decision hyper-planes for step function outputs, while the DELTA rule emerged from a gradient descent minimization of the Sum Squared Error for a linear output activation function. The Perceptron learning rule will converge to zero error and no weight changes in a finite number of steps if the problem is linearly separable, but otherwise the weights will keep oscillating. On the other hand, the DELTA rule will (for sufficiently small learning rates) always converge to a set of weights for which the error is a minimum, though the convergence to the precise target values will generally proceed at an ever decreasing rate proportional to the output discrepancies.

### Biologically Plausible Rules

The second category of learning rules we will discuss are those that draw inspiration from Neuroscience and Biology.

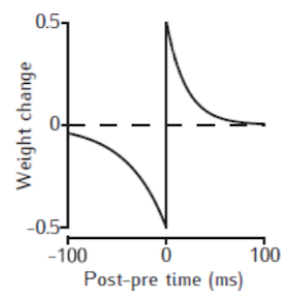
### Hebbian Learning Rule and STDP

*“When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place on one or both cells such that A’s efficiency as one of the cells firing B, is increased”.*

**Spike-Timing Dependent Plasticity (STDP)** 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 (LTP)**. 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 (LTD)**. We can start by writing a simple Hebbian weight update as the product of the input and output with some scaling factor:

$$w = \alpha \times x \times y = \alpha \times x \times [w \cdot x]$$

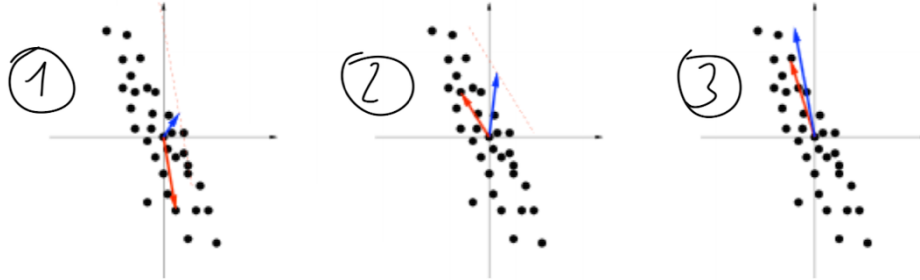
The more closely aligned  $w$  and  $x$  are, the larger  $y = w^T x$  is, and by definition of the dot product  $y = 0$  when  $w$  is orthogonal to  $x$ . This leads to the weight vector gradually pointing towards the input vector, or the cloud of input data in a dataset. We can mitigate this issue 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:



$$\nabla w = \alpha \times x \times y = \alpha(w \cdot x)x = \alpha(x \cdot x^T) \cdot w$$

Side note: “When the input  $x$  and output  $y$  are correlated, their product is positive, which results in a positive update to the weight (increases). When they are uncorrelated, the product is close to 0, which results in a small or no-update to the weight”.

where the last step involves a transformation of the inner product into an outer product.



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

$$\nabla w = \alpha(x \cdot x^T) \cdot w = \alpha \times C \cdot w$$

which leads to the following update over time:

$$\frac{\partial w}{\partial t} = C \cdot w$$

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  $\lambda$  is positive the weight vectors will continue to increase and blow up.

### Oja's Rule

Paper: “A Simplified Neuron Model as a Principal Component Analyzer”.

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.

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

### 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.

$$\begin{aligned} \nabla w_{cov} &= \alpha[y - \langle y \rangle][x - \langle x \rangle] \\ &= \alpha[y \times x - \langle y \rangle \times x - y \times \langle x \rangle + \langle y \rangle \times \langle x \rangle] \end{aligned}$$

$$\begin{aligned} \langle \nabla w_{cov} \rangle &= \alpha[\langle y \times x \rangle - \langle \langle y \rangle \times x \rangle - \langle y \times \langle x \rangle \rangle + \langle \langle y \rangle \times \langle x \rangle \rangle] \\ &= \alpha[\langle y \times x \rangle - \langle y \rangle \times \langle x \rangle - \langle y \rangle \times \langle x \rangle + \langle y \rangle \times \langle x \rangle] \\ &= \alpha[\langle y \times x \rangle - \langle y \rangle \times \langle x \rangle] \end{aligned}$$

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, weights 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.

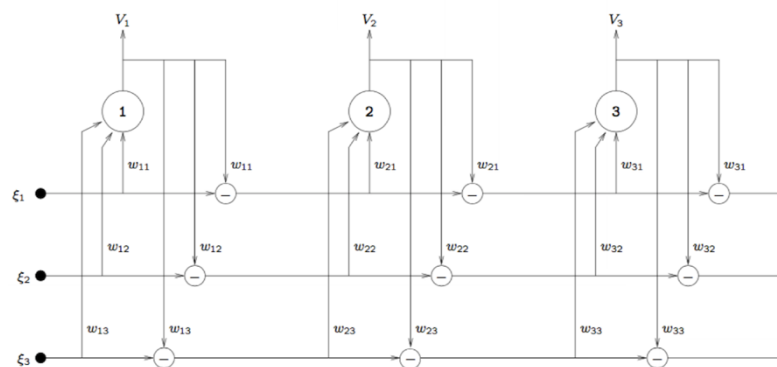
### Sanger's (PCA) Rule

PCA Recap: It is a tool from statistics for data analysis. It can reveal structure in high- $N$ -dimensional data that is not otherwise obvious. Like Hebbian learning, it discovers the direction of maximum variance in the data. But then in the  $(N - 1)$ -dimensional subspace perpendicular to that direction, it discovers the direction of maximum remaining variance, and so on for all  $N$ . The result is an ordered sequence of principal components. These are equivalently the eigenvectors of the correlation matrix  $C$  for zero-mean data, ordered by magnitude of eigenvalue in descending order. They are mutually orthogonal.

The idea is that we use a single Hebbian neuron that points in the direction of maximum variance, as described previously, and we 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).

We have seen that Hebbian learning, with appropriate provisions for preventing blow up, extracts the largest principal component. Let's take a look at two different neural network architectures capable of extracting more of them:

- Cascading Multiple Hebbian Neurons.
- Autoencoder Networks (we have already seen them before).



Algorithm:

1. Subtract the contribution of the first principal component.
2. Drive the difference into another Hebbian neuron.
3. This extracts the next principal component.
4. Subtract its contribution. Go to step 2.

With  $N$  Hebbian neurons, we will get all  $N$  principal components.

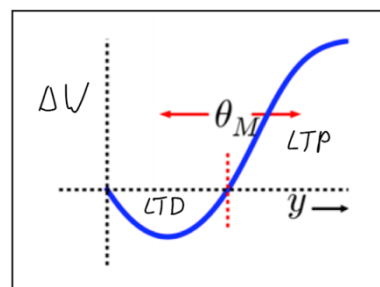
### Sejnowski's Infomax Network (ICA) Rule

Paper: "An Information-Maximisation Approach to Blind Separation and Blind Deconvolution".

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

### Bienenstock-Cooper-Monroe 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  $\theta_M$  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 10Hz the weight updates were negative, while above this stimulation frequency they were positive.

### Linear BCM

$$y = \sum_i w_i x_i$$

$$\frac{dw_i}{dt} = y(y - \theta_M)x_i - \epsilon w_i$$

$$\theta_M = E^P[(y/y_o)]$$

### Non-Linear BCM (I-BCM)

$$y = \sigma\left(\sum_i w_i x_i\right)$$

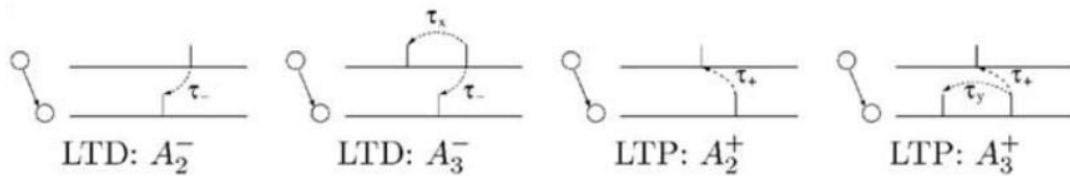
$$\frac{dw_i}{dt} = y(y - \theta_M)x_i \sigma'(y)$$

$$\theta_M = E[y^2]$$

Energy Function:

$$R = -\frac{1}{3} E[y^3] + \frac{1}{4} E^2[y^2]$$

### The Triplet Rule



Papers: “Triplets of Spikes in a Model of Spike Timing-Dependent Plasticity” & “A Triplet Spike-Timing-Dependent Plasticity Model Generalizes the BCM rule to Higher-Order Spatiotemporal Correlations”.

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 above in the figure, LTD generally occurs when a presynaptic spike occurs just after a postsynaptic spike, even if (as is  $A_3^-$ ) another presynaptic spike preceded the postsynaptic spike. Analogously, LTP tends to occur when a postsynaptic spike preceded the 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.

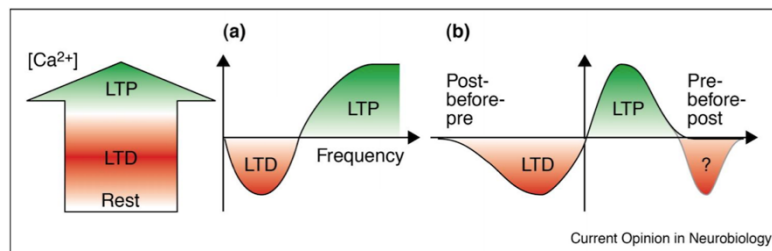
Note that if we set  $A_3^+ = 0$  and  $A_3^- = 0$ , the model becomes a classical pair-based STDP model (BCM).

$$w(t) \rightarrow w(t) - o_1(t)[A_2^- + A_3^- r_2(t - \epsilon)] \text{ if } t = t^{\text{pre}}.$$

$$w(t) \rightarrow w(t) + r_1(t)[A_2^+ + A_3^+ o_2(t - \epsilon)] \text{ if } t = t^{\text{post}}.$$

### The Calcium Rule

Paper: “Calcium-Based Plasticity Model Explains Sensitivity of Synaptic Changes to Spike Pattern, Rate and Dendritic Location”.



As shown in the previous picture, 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. The threshold is represented by the frequency of synaptic firing, indeed, low frequencies of synaptic firing (approx. 5Hz) produce LTD, while high frequencies of synaptic firing (approx. 50 to 100Hz) produce LTP. This rule results in the same basic STDP profile as before, and there are threshold parameters that can be changed to affect the dynamics.

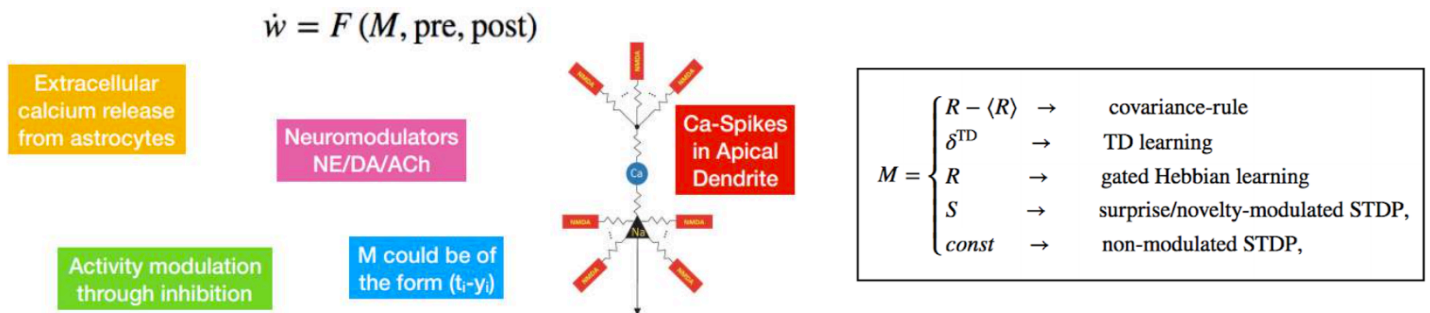
### Hebbian Learning: Unsupervised

Papers: “*The Ups and Downs of Hebb Synapses*” & “*Unsupervised Learning of Digit Recognition Using Spike-Timing-Dependent Plasticity*” & “*Local Plasticity Rules Can Learn Deep Representations Using Self-Supervised Contrastive Predictions*”.

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

### Hebbian Learning: Three-Factor Rules

Paper: “*Neuromodulated Spike-Timing-Dependent Plasticity, and Theory of Three-Factor Learning Rules*”.

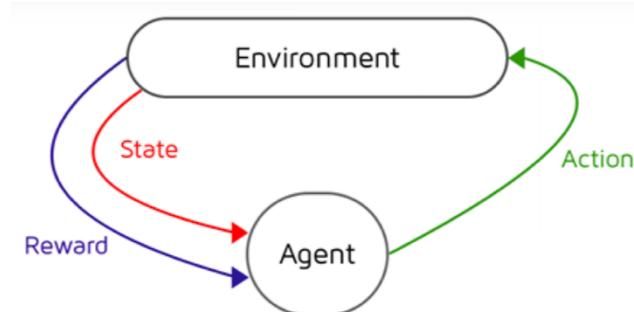


Three-factor Hebbian learning rules integrate the pre- and postsynaptic firing with a third factor,  $M$ , which includes values such as the covariance-rule, TD learning, gated Hebbian learning, surprise-modulated STDP, etc. For a biological neuron, this  $M$  factor may be viewed in a variety of ways, as shown in the picture above. 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 (an 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.

# Reinforcement Learning – Lecture 05 – Benjamin Grewe

## Content of the Lecture

- Introduction to Reinforcement Learning
  - Reinforcement Learning and the Brain.
  - Example of RL Neuroscience Experiments.
  - The Rescorla-Wagner Rule.
  - The Temporal Difference Rule.
  - The Basic Idea of Q-Learning.
- Reinforcement Learning in Machines
  - Basic Components of RL.
  - Policy and Value Function Based RL.
  - MC, MRP and MDPs.
  - The Problem of Finding the Optimal Policy.
  - The Bellman (Expectation) Equation.
  - RL and Deep Q-Learning.

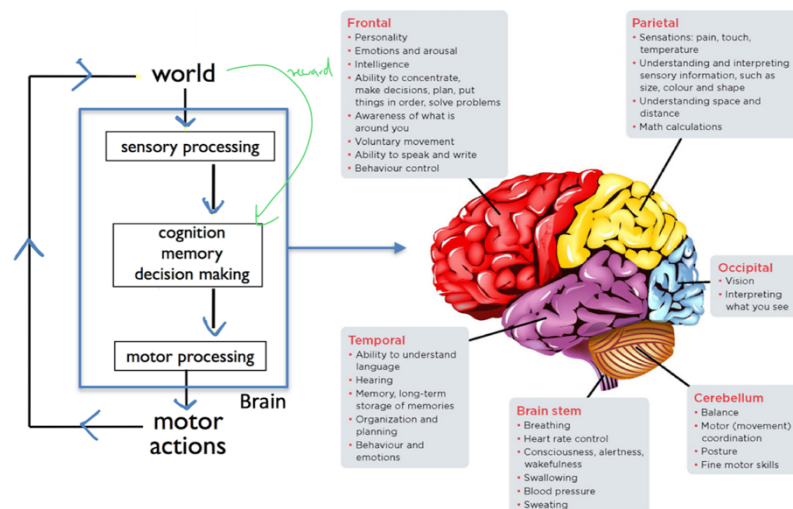


## What is Reinforcement Learning?

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.

## What makes reinforcement learning different from other machine learning paradigms?

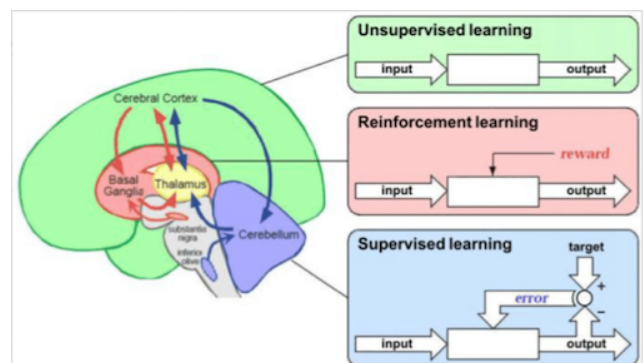
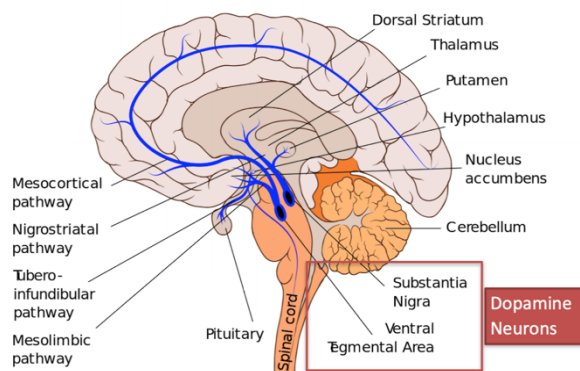
- There is only a reward/supervision signal after each action.
- Feedback is often delayed and not instantaneous.
- Time needs to be taken into account.
- The agent's actions affect the subsequent data it receives.



## Dopamine: Reward Prediction Error

Papers: “Predictive Reward Signal of Dopamine Neurons”

From Schultz (89): “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 neurons show phasic activations [...] reward-predicting [...] However, only few phasic activations follow aversive (causing avoidance of a thing) stimuli. By signalling rewards according to a prediction error, dopamine responses have the formal characteristics of a teaching signal postulated by reinforcement learning theories.”

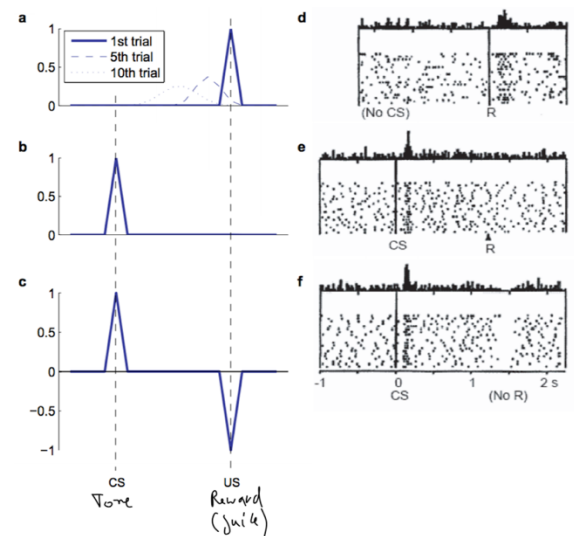


If the neocortex mostly performs unsupervised learning why does the VTA strongly project to almost all cortical areas and what is the effect of DA on a cortical neuron?

- Ventral structure does not project to dorsal (where we assume RL is 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.

The figure to the right pictures 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 prediction errors 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. Paper: *“Impact of Size and Delay on Neural Activity in the Rat Limbic Corticostriatal System”*.
- Adaptation to a New Situation: We found that midbrain dopamine neurons rapidly adapted to the information provided by reward-predicting stimuli. Responses shifted to the expected reward value, and the gain adjusted to the variance of reward value. Paper: *“Adaptive Coding of Reward Value by Dopamine Neurons”*.

## Models of Learning Reward Prediction

Even though these models here are called predicting models, we are looking at update rules which means the system changes over time – it learns. We might connect one of these learning rules to a MDP or RL to find an optimal behavior function for our agent.

### Rescorla Wagner Rule

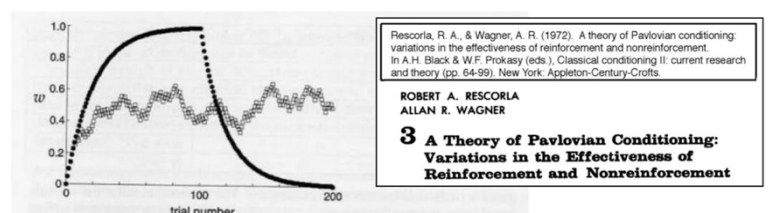
Model of classical conditioning in which learning is conceptualized in terms of associations between conditioned and unconditioned stimuli. Change in value  $V(s_t)$  is proportional to the difference between actual and predicted reward.

$$V(s_t) \leftarrow V(s_t) + \eta[R - \text{sum}(V(s_t))]$$

where:  $s_t$  is the stimulus,  $V(s_t)$  is the associative strength of conditioned stimulus  $s_t$ ,  $R$  is the reward,  $\eta$  is the learning rate,  $\text{sum}(V(s_t))$  is the sum of associative strengths of all conditioned stimuli (including  $s_t$ ) that are presented on this trial (the n-th trial) and  $|[R - \text{sum}(V(s_t))]|$  is the surprise.

Two assumptions/hypotheses:

- Learning is driven by the error signal (formalized as notion of surprise).
- Summation of predictors is linear.





## Temporal Difference (TD) Rule to Q-Learning

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 only at immediate rewards but future rewards should be taken into consideration as well on a discounted valuation. We assume that the path our agents takes to navigate the system is given.

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

where  $V(s_t)$  is the previous estimate,  $r_{t+1}$  is the next reward,  $\gamma V(s_{t+1})$  is the discounted value on the next step and  $r_{t+1} + \gamma V(s_{t+1})$  represents the TD target.

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  $Q(s, a)$

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

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 between old and new Q value seen in the literature:

$$\begin{aligned} Q(s, a) &\leftarrow (1 - \alpha_t)Q(s, a) + \alpha_t(r + \gamma \max_{a'}[Q(s', a')]) \\ Q(s, a) &\leftarrow Q(s, a) - \alpha_t Q(s, a) + \alpha_t r + \gamma \alpha_t \max_{a'}[Q(s', a')] \\ Q(s, a) &\leftarrow Q(s, a) + \alpha_t(r + \gamma \max_{a'}[Q(s', a')] - Q(s, a)) \\ Q(s, a) &\leftarrow Q(s, a) + \alpha_t(r + \gamma \max_{a'}[Q(s', a')] - Q(s, a)) \end{aligned}$$

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 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 deduced from the optimal value function  $V^*$ :

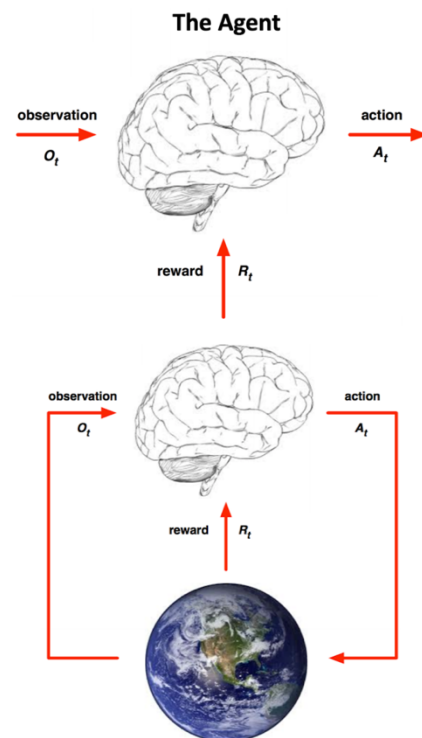
$$V^*(x) = \max_a Q^*(x, a)$$

## 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.

In the pictures: influences from and to the agent in RL to the surrounding world. At each step  $t$ , the agent executes an action  $A_t$  which the environment receives. The agent receives an observation  $O_t$  of the environment, for example through 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.** A reward  $R_t$  is a scalar feedback signal. It indicates how well the agent is doing at step  $t$ . The agent's job is to maximize cumulative reward.

Example rewards:

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

The **Goal** is to select actions that maximize total future rewards:

- Actions may have long-term consequences.
- Reward may be delayed.
- It may be better to sacrifice immediate reward to gain more long-term reward.
- For example: a financial investment may take months to mature, re-fuelling an helicopter might prevent a crash in several hours, blocking an opponent move might help winning chances many moves from now.

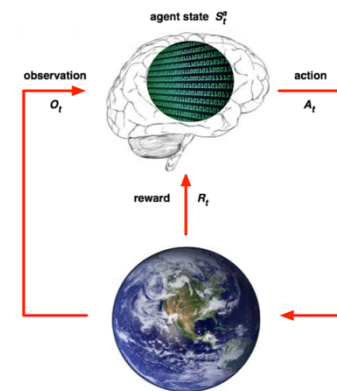
The fact that reward presented to the agent is not always immediate leads to the **exploration/exploitation dilemma**. An agent does not intrinsically know the future implications of its actions, or the dynamics of the environment.

### The Agent State

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 **history** ( $H_t$ ) of previous actions, observations and rewards:

$$H_t = A_1 O_1 R_1, \dots, A_t O_t R_t$$

- The agent state  $S_t$  is the agent's internal representation:
  - Whatever information the agent uses to pick the next action.
  - It is the information used by RL algorithms.
- It can be any function of the history:  $S_t = f(H_t)$ .



For the reasons explained above:  $P(S_{t+1}|S_t) = P(S_{t+1}|S_{1:t})$ . An RL agent may compute different functions on top of its state.

An RL agent may include one or more of these components:

- **Policy:** The agent's behavior function.
- **Value Function:** How good is each state and/or action?
- **Model:** The agent's representation of the environment.

### Policy

The agent's behavior function called policy maps from state  $s$  to action  $a$ . The policy may be stochastic or deterministic:

- Deterministic Policy:  $a = \pi(s)$
- Stochastic Policy:  $\pi(a|s) = P(A_t = a|S_t = s)$

## Value Function

The value function is a prediction of future rewards and does so by assigning a number to every state  $s$ , it is used to evaluate the goodness/badness of states. It depends on a policy  $\pi$  to determine where the agent could go and a probability distribution  $p(X'|\pi(X), X)$ . We compute this value as an expectation over the joint distribution:  $p(S_1, S_2, S_3, \dots)$ .

The value function is defined as:

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

In the lecture slides is formulated as:

$$v_{\pi}(s) = E_{\pi} (R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | S_t = s)$$

## Model

A model predicts what the environment will do next. From the previous section we see that we require a probability distribution that depends on direct actions  $a$  or a policy returning action  $\pi$ :

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

That predicts the next (immediate) reward:

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

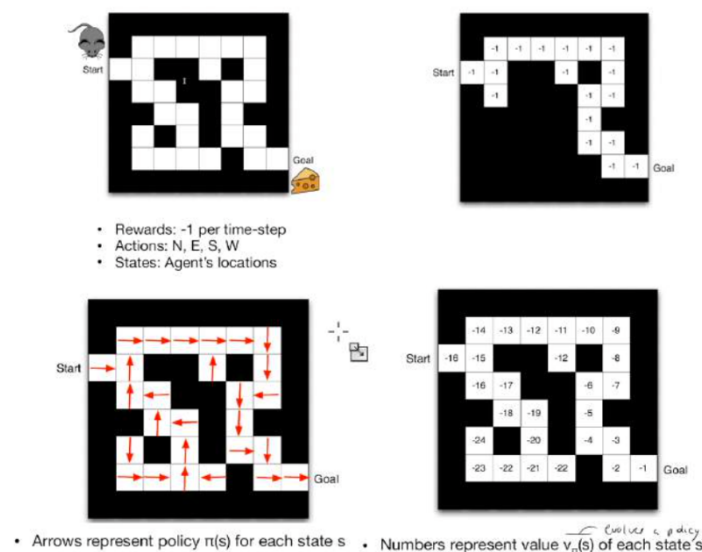
In the lecture slides is formulated as:

$$P_{ss'}^a = P[S_{t+1} = s' | S_t = s, A_t = a]$$

$$R_s^a = E[R_{t+1} | S_t = s, A_t = a]$$

where  $P$  predicts the next state and  $R$  predicts the next (immediate) reward.

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).



In the image, 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.

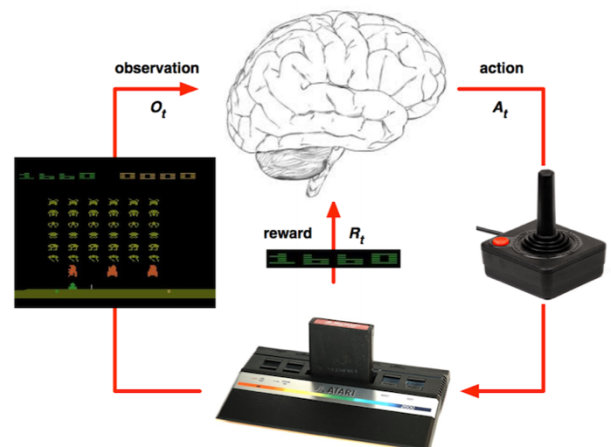
- Agent may have an internal model of the environment.

- Dynamics: How actions change the state.
- Rewards: How much reward from each state.
- The model may be imperfect.
- Grid layout represents transition model  $P_{s_t, s_{t+1}}^a$
- Numbers (top right) represent immediate rewards  $R_{s_t}^a$  from each state  $s$ .

We have 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 vs Policy Based**
  - **Value Based**
    - No Policy (Implicitly given by value function, see Bellman Eq.)
    - Value Function
  - **Policy Based**
    - Policy
    - No Value Function (Even though values can be computed off a policy).
- **Model Free vs Model Based**
  - **Model Free**
    - Policy and/or Value Function
    - No Model distribution needed
  - **Model Based**
    - Policy and/or Value Function
    - Model distribution needed.

A more sophisticated example is represented by the Atari Games. The Atari video-gaming platform provides an 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.



## 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 than solving it, i.e., we ignore the model. In a planning scenario, we can query the future through the emulator. We can therefore play/plan ahead to find the optimal policy by tree search. As already mentioned, this might not be possible even for simple games. RL on the other hand can be as well referred to as “trial-and-error” learning. However, obviously we try to guide the agent to lose the least amount of reward that is possible.

## 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.

## 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 graphic models (PGM) (Bayesian Networks) first. They are used from here on, but were not introduced explicitly in the lecture.

## 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:

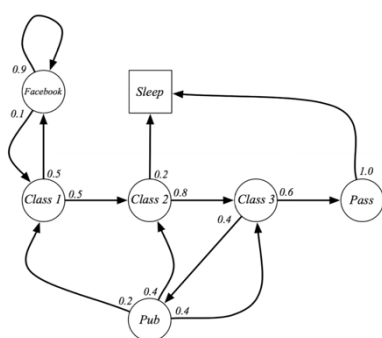
$$P(S_{t+1}|S_t) = P(S_{t+1}|S_{1:t})$$

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  $n = s_{t+1}$ :

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



State transition matrix

|       | 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

(Left) An example Markov Chain showing all state transition probabilities next to its node. (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.

## Markov Reward Process (MRP)

A Markov reward process is a stochastic process which extends a Markov chain by adding a reward rate to each state. Definition: A Markov Reward Process is a tuple  $(S, P, R, \gamma)$  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  $\gamma$  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.

$$G_t = R_{t+1} + \gamma R_{t+2} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

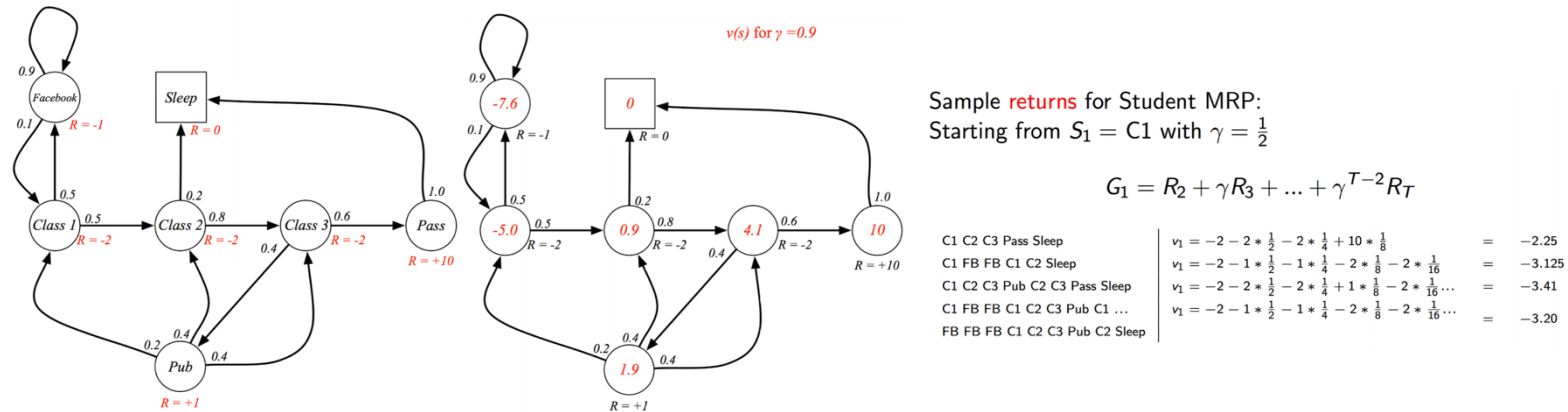
If we just look at  $G_t$  we must assume that we know the chain of events that lead to the specific rewards  $R$ , however the MRP is a stochastic process. Therefore we may compute the conditional value function given that we know where we start ( $s_t$ ). We have seen this function before in the RL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

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

which is just

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

Hence, the state value function  $v(s)$  of an MRP is the expected return starting from state  $s$ . It gives the long-term value of state  $s$ .



## 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  $(S, A, P, R, \gamma)$  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  $\gamma$  is a discount in the interval  $(0,1)$ .

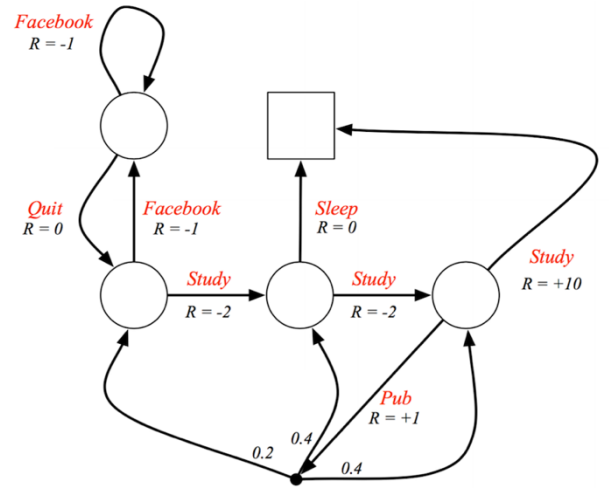
Markov Decision Processes formally describe an environment for Reinforcement Learning where the environment is fully observable. Almost all RL problems can be formalized as MDPs.

- Partially observable problems can be converted into MDPs.
- One Armed Bandits are MDPs with one state.



The MC seen before extended to be a MRP is here 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 we were in the pub, 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?



### 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 when the environment is known  $\rightarrow$  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.

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

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 through the sum and the equation becomes:

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

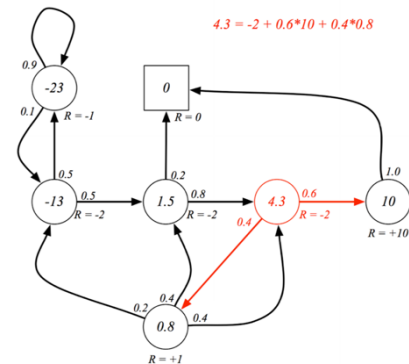
In figure, we have an example computation of the value of the red node in an MRP.

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.

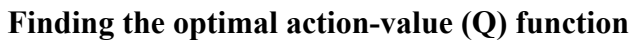
- The state-value function  $v_{\pi}(s)$  of an MDP is the expected return starting from state  $s$ , and then following policy  $\pi$ .
- The action-value function  $q_{\pi}(s, a)$  is the expected return starting from state  $s$ , taking action  $a$ , and then following policy  $\pi$ .



And if we again shift the reward:

$$\begin{aligned} q_\pi(s_t, a_t) &= \mathbb{E}_\pi[G_t | S_t = s_t, A_t = a_t] \\ &= \mathbb{E}_\pi[R_{s_{t+1}} + \gamma q_\pi(S_{t+1}, A_{t+1}) | S_t = s, A_t = a] \\ &\stackrel{!}{=} R_{s_t}^{a_t} + \gamma \sum_{s_{t+1} \in S} P_{s_t, s_{t+1}}^{a_t} v(s_{t+1}) \\ &= R_{s_t}^{a_t} + \gamma \sum_{s_{t+1} \in S} P_{s_t, s_{t+1}}^{a_t} \sum_{a_{t+1} \in A} \pi(a_{t+1} | s_{t+1}) q_\pi(s_{t+1}, a_{t+1}) \end{aligned}$$

In the figure we have an example of policy based computation of the value of the red node in an MDP.


$$v^*(s) = \max(v_\pi(s))$$
$$q^*(s, a) = \max(q_\pi(s, a))$$
$$\pi^*(a|s) = \begin{cases} 1, & \text{if } a = \operatorname{argmax}_{a \in A}(q^*(s, a)) \\ 0, & \text{otherwise} \end{cases}$$
$$q_{\pi}^*(s, a) = R_t^a + \gamma \sum_{s' \in \mathcal{S}} P_{s,s'}^a v^*(s')$$
$$v^*(s) \leftarrow \max_a [R_s^a + \gamma \sum_{s' \in S} P_{s,s'}^a v^*(s')]$$

37

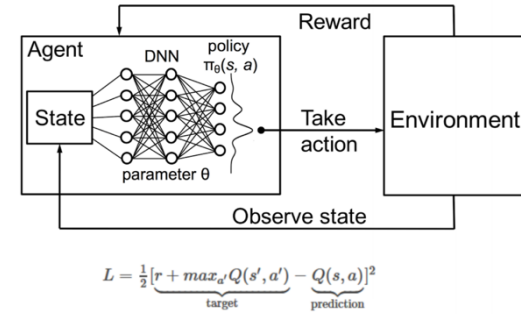
Bellman figured out that our policy being optimal means that we can be greedy w.r.t these optimality equations. If we update over and over again, we will converge to a fixed point which is the optimality policy.

- The Bellman Optimality Equation is non-linear.
- No closed form solution (in general).
- Many iterative solution methods.
  - Iterative Policy Evaluation.
  - Iterative Value Evaluation.
  - Q-Learning.

## Deep Reinforcement (Q) Learning

Paper: “Human-Level Control Through Deep Reinforcement Learning”.

In the previous sections, distribution 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 have 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.



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

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

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

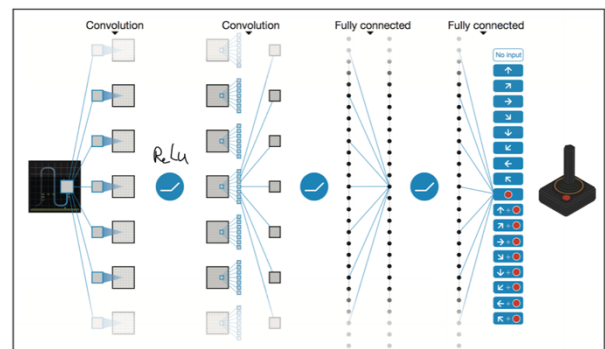
and then minimize the loss w.r.t  $\theta$ . Comment: Often, we approximate  $\mathbb{E}$  by sampling. The following represents the loss function to update the Q-learning rule:

$$L_i(\theta_i) = \mathbb{E}_{s,a,r,s' \sim U(D)} \left[ \left( r + \gamma \max_{a'} Q(s', a' | \theta_i^-) - Q(s, a | \theta_i) \right)^2 \right]$$

This is again the temporal difference rule.

In the picture we have a deep RL system trained directly from input (Atari game video output) to actions of the controlling joystick.

**On-policy methods** estimate the value of a policy while using it for control. In **Off-policy methods**, the policy used to generate behavior, called the behavior policy, may be unrelated to the policy that is evaluated and improved, called the estimation policy.



From the Sutton book: “The on-policy approach in the preceding section is actually a compromise – it learns action values not for the optimal policy, but for a near-optimal policy that still explores. A more straightforward approach is to use two policies, one that is learned about and that becomes the optimal policy, and one that is more exploratory and is used to generate behavior. The policy being learned about is called the target policy, and the policy used to generate behavior is called the behavior policy. In this case we say that learning is from data “ $o \leftarrow$ ” the target policy, and the overall process is termed “ $o \leftarrow$ -policy learning”.

# Un- and Self-Supervised Learning – Lecture 06 – Benjamin Grewe

## Content of the Lecture

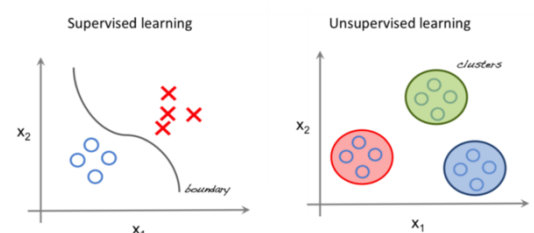
- What is Unsupervised Learning?
- Unsupervised Learning in the Brain
  - Unsupervised Learning in the cortex – Theory by K. Doya and D. Marr
  - Unsupervised Learning Experiments in Neuroscience
- Unsupervised Machine Learning
  - Non-Probabilistic UL Methods
    - Sparse Coding (Olshausen and Fields)
    - Simple Network Solutions: PCA, ICA (Infomax)
    - Autoencoders and Supervised Autoencoders
    - Contracting Autoencoders
    - Denoising and Sparse Autoencoders
    - The “Homomorphism Autoencoder”
    - Competitive Network Learning
  - Probabilistic (Generative) UL Methods
    - Boltzmann Machines
    - Contrastive Divergence (Hinton)
  - Self-Supervised Learning
    - PIXEL-RNN, PRED-NET

## Motivation

Geoffrey Hinton (*“Learning Representations by Back-Propagating Errors”*) suggest networks should be able to become intelligent on their own, unsupervised, without backprop.

- **Does not need labelled data.** In practice we often assume a fixed number of labels and call them cluster centers.
- **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 lower dimension space.
- **Is able to generate new data examples that are consistent with the statistics of the training data.**
- **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.
- **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.
- **Can facilitate future supervised learning.**
- **Can be used to cluster the data.**
- **Can be used to de-noise data.** We lose 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.
- **Can be used to encode data in a particular way (e.g., for data compression).**
- **We assume that humans have learned the world in an almost unsupervised way.**

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.



## Unsupervised Learning

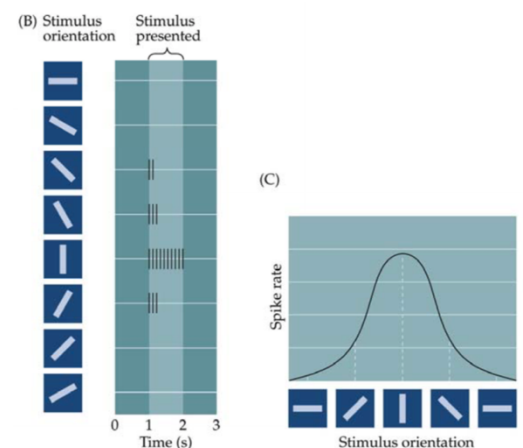
- **Non-Probabilistic Models:**
  - Sparse Coding
  - Autoencoders
  - KNN, K-Means
- **Probabilistic (Generative) Models:**
  - **Tractable Models**
    - Fully Observed Belief Nets
    - Pixel RNNs
  - **Non-Tractable Models**
    - Boltzmann Machine
    - Variational Autoencoders
    - Helmholtz Machines
    - Many others
  - Generative Adversarial Networks (GAN)
  - Moment-to-Moment Networks (not covered)

## Unsupervised Learning in the Brain

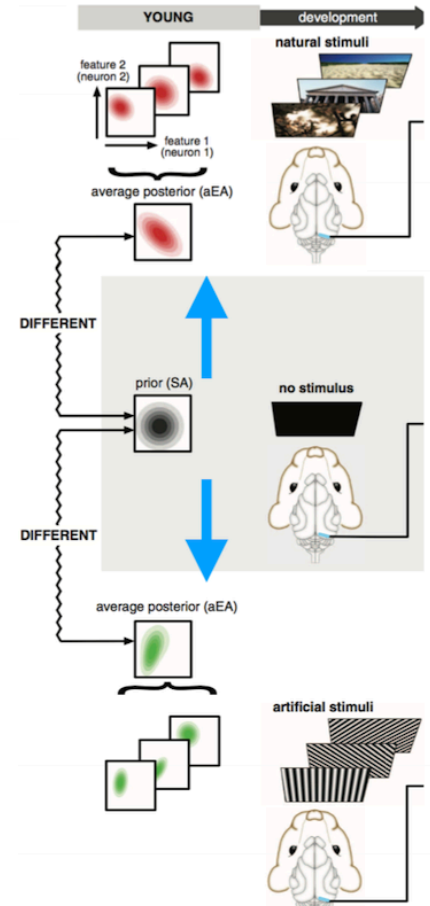
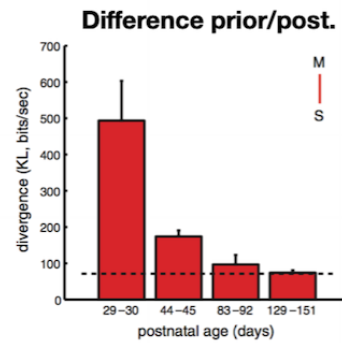
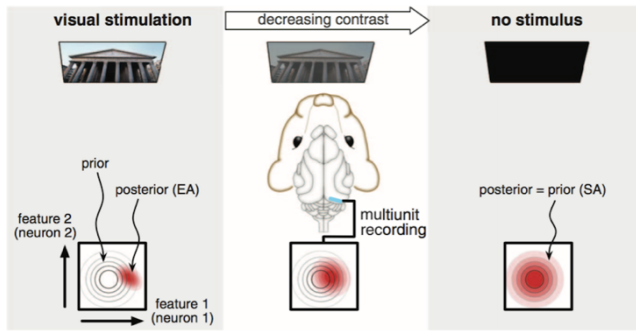
Papers: *“Unsupervised Yearning”*, *“Complementary Roles of Basal Ganglia and Cerebellum in Learning and Motor Control”*, *“Development of the Brain Depends on the Visual Environment”*. We are aware of the fact that the brain as well uses unsupervised learning. Several examples were collected for the lecture. In 1970 an experiment was conducted on cats. The kittens were housed from birth in a completely dark room, but from 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. 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.

In the picture we have 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 a 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. Paper: *“Spontaneous Cortical Activity Reveals Hallmarks of an Optimal Internal Model of the Environment”*.







In the figures: Notation: Evoked and spontaneous (dark) neural activity (EA and SA). Multi-neural EA (aEA).

In the top-left figure, the posterior distribution represented by EA is increasingly dominated by the prior distribution as brightness or contrast is decreased. In the right figure, 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 is unclear what the conditions are to trigger an adaptation. Another experiment (*"Stimulating Timing-Dependent Plasticity in Cortical processing of Orientation"*) 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  $\pm 40$ 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 systems 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.

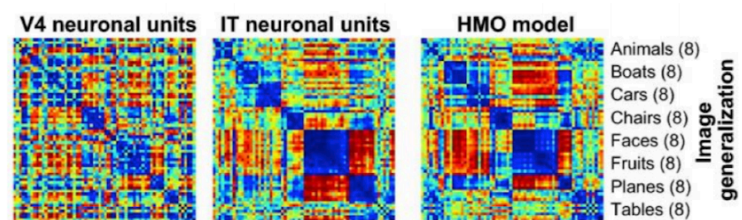
1<sup>st</sup> Experiment: The statistics of the neuronal activity has adapted to represent the input data statistics (spatial).

2<sup>nd</sup> Experiment: The statistics of the neuronal activity has adapted to represent the input temporal data statistics.

In the picture we have the confusion matrix of V4 neuronal units and units in artificial networks from a comparable depth.

### 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:  $I(x, y) \approx \sum_i a_i \phi_i(x, y) = \hat{I}$ .



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

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

Presented in the lectures was a set of known regularizers  $\mathbb{S}$ :  $S \in \mathbb{S} = \{\exp^{x^2}, \log(1 + x^2), \|x\|\}$

For each image presentation  $E$  is minimized with respect to  $a_i$ . Thus, for a given image, the  $a_i$  are determined from the equilibrium solution to the differential equation:

$$\frac{\partial E}{\partial a_i} = \dot{a}_i = b_i - \sum_j c_{ij} a_j - \frac{\lambda}{\tau} S'\left(\frac{a_i}{\tau}\right)$$

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

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

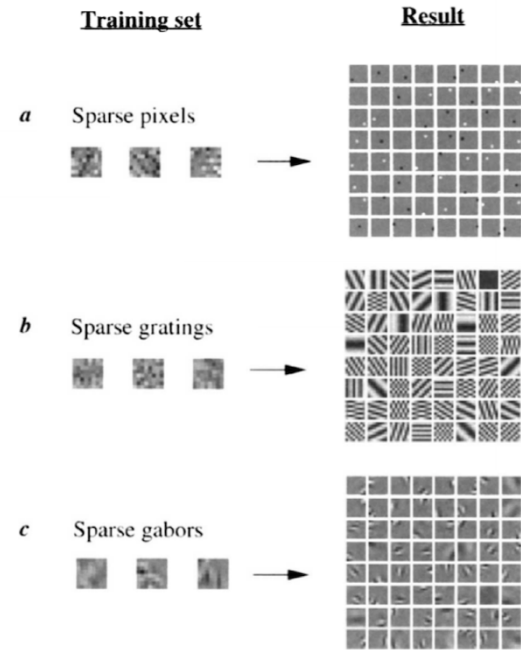
The  $\phi_i$  then evolve by gradient descent on  $E$  averaged over many image presentations. The learning rule for updating  $\phi$  is then:

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

Comments on Parameters and Operations:

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

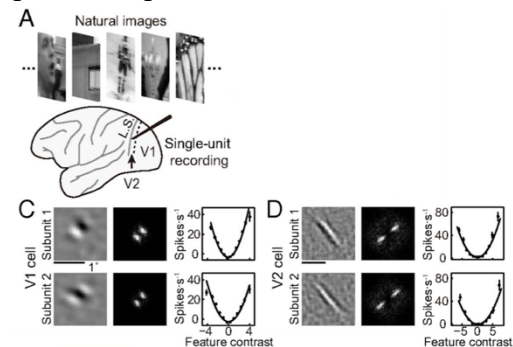
In the figure we have representative training images are shown at the left and the resulting basis functions that were learned from these examples are shown at the right. In a, images were composed of sparse pixels: each pixel was activated independently according to an exponential distribution. In b, images were composed similarly to a, except with gratings instead of pixels (i.e., sparse pixels in the Fourier domain). In c, images were composed of sparse, non-orthogonal Gabor functions with the methods described by Field. In all cases, the basis functions were initialized to random initial conditions. The learned basis functions successfully recover the sparse components from which the images were composed.



## Relation to Neuroscience

Paper: "Spatial Structure of Neuronal Receptive Field in Awake Monkey Secondary Visual Cortex (V2)".

This paper 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.



## Infomax (ICA)

Paper: "An Information-Maximisation Approach to Blind Separation and Blind Deconvolution". 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 (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:

$$\langle X_1, X_2 \rangle = \langle X_2 \rangle \langle X_1 \rangle = 0$$

Or

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

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:

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

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:

$$\begin{aligned} I(X; Y) &\equiv H(X) - H(X|Y) \\ &\equiv H(Y) - H(Y|X) \\ &\equiv H(X) + H(Y) - H(X, Y) \\ &\equiv H(X, Y) - H(X|Y) - H(Y|X) \\ I(X; Y) &= D_{\text{KL}}(P_{(X,Y)} \| P_X \otimes P_Y) \end{aligned}$$

We use the entropy:  $H(X) = - \sum_{i=1}^n P(x_i) \log_b P(x_i)$

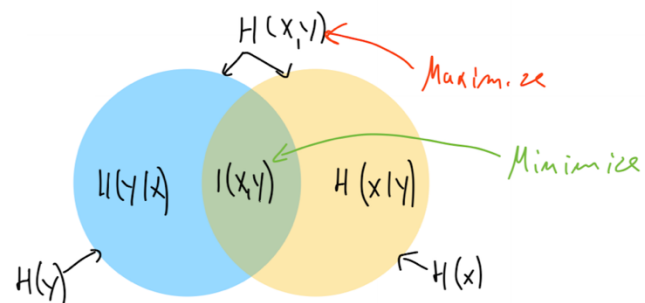
Idea:  $X, Y$  independent if:  $p(X, Y) = p(X)p(Y)$

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

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

In the picture we have a Venn diagram of the Infomax objective: We want to maximize the entropy and minimize the mutual information. Entropy maximization forces the network to generalize.

- $H(x)$  = Entropy of Distribution of the First Neuron's Output.
- $H(x|y)$  = Conditional/Marginal Entropy.
- $H(x, y)$  = Joint Entropy.
- $I(x, y)$  = Mutual Information.



We want to ensure that the outputs  $y_i$  are maximally independent. This is identical to requiring that the mutual information be small or alternatively that the joint entropy is large. Gradient ascent in

this objective function is called **INFOMAX** (maximize the enclosed area representing both information quantities).

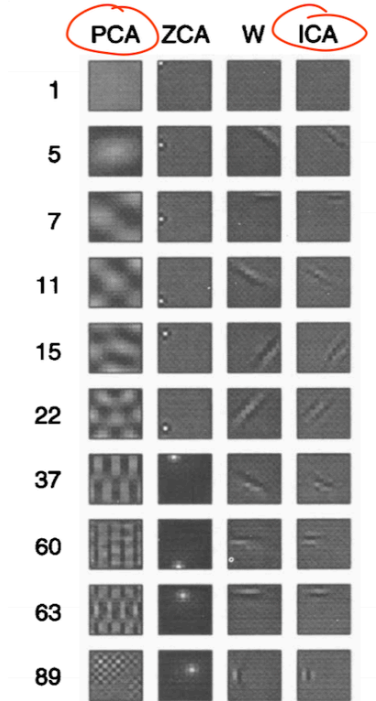
How do we actually implement Infomax? We can think of Infomax as a one layer linear neural network that produces:  $\mathbf{y} = \mathbf{W}\mathbf{x}$  such that outputs  $y_i$  that are maximally independent:

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

but keep in mind we also maximize  $I(\mathbf{X}; \mathbf{Y})$ .

In the picture 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 are 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 to be orthogonal.

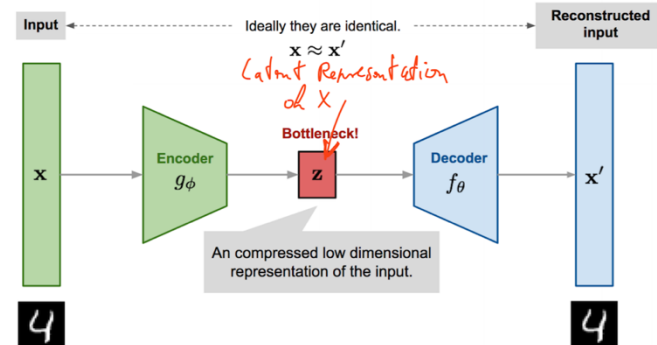


## Autoencoders

- $g\phi$ : encoding function.
- $f\phi$ : decoding function.
- $\phi$ : parameters of encoding function (trainable).
- $\theta$ : parameters of decoding function (trainable).
- MSE Loss:

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

The autoencoder is trained by gradient descent.



## Semi-Supervised Autoencoder

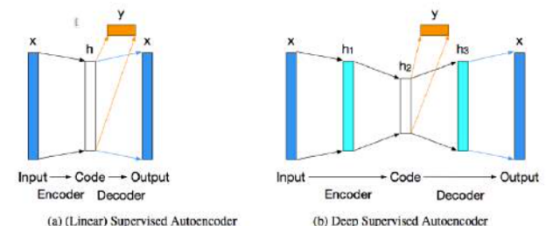
Paper: "Supervised Autoencoders: Improving Generalization Performance with Unsupervised Regularizers".

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).

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

where:

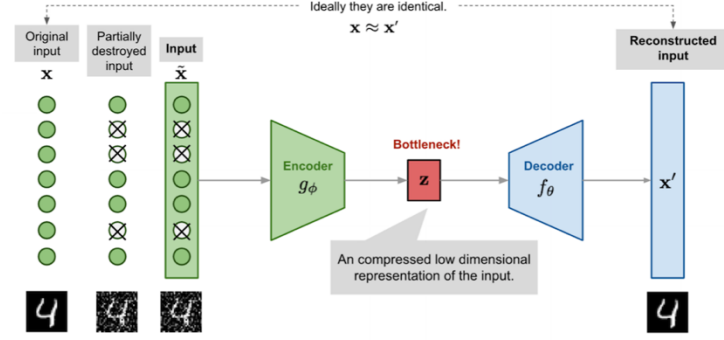
- $L_P$  is the loss of label  $y_i$
- $L_v$  is the loss of reconstruction  $\hat{x}_i$
- $\mathbf{W}_{1:2}$  are the weights of encoder (that produce label  $y$ )
- $\mathbf{W}_{1:4}$  are the weights of encoder + decoder.



## De-Noising Autoencoder (DEA)

Paper: “*Extracting and Composing Robust Features with Denoising Autoencoders*”

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. Note: In the experiment of the original DAE paper, the noise is applied in that a fixed portion of input dimensions are selected at random and their values are forced to 0. This is very similar to Dropout but the denoising autoencoder was proposed in 2008, 4 years before the dropout paper (Hinton, et al. 2012).



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

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

## 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.

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

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  $a_j^{(l)}(\cdot)$ ,  $j = 1, \dots, s_l$ . The fraction of activation of this neuron  $\rho^j$  is expected to be a small number  $\rho$ , known as sparsity parameter; a common config is  $\rho = 0.05$ .

$$\begin{aligned} L_{\text{SAE}}(\theta) &= L(\theta) + \beta \sum_{l=1}^L \sum_{j=1}^{s_l} D_{\text{KL}}(\rho \| \hat{\rho}_j^{(l)}) \\ &= L(\theta) + \beta \sum_{l=1}^L \sum_{j=1}^{s_l} \rho \log \frac{\rho}{\hat{\rho}_j^{(l)}} + (1 - \rho) \log \frac{1 - \rho}{1 - \hat{\rho}_j^{(l)}} \end{aligned}$$

Keep in mind that we specify our desired target distribution that is  $\rho$ . Common activation functions include sigmoid, tanh, relu, leaky relu, etc. A neuron is activated when the value is close to 1 and inactive with a value close to 0.

## 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:

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



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

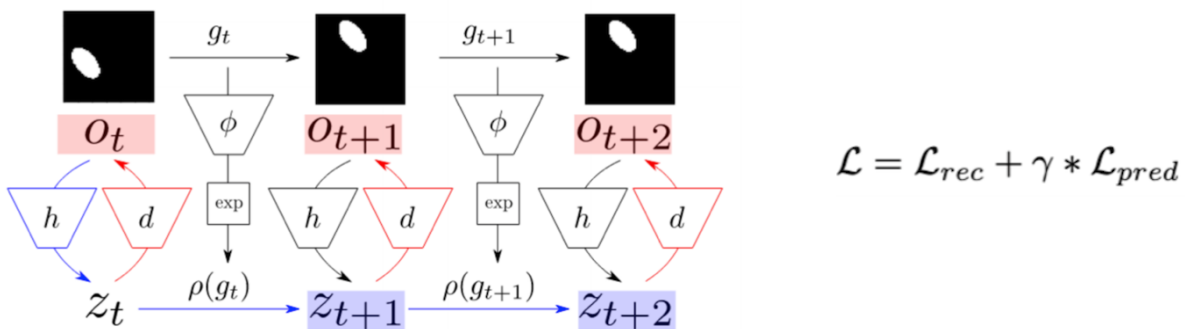
This penalty term is the sum of squares of all partial derivatives of the learned encoding with respect to input dimensions. The authors claimed that empirically this penalty was found to carve a representation that corresponds to a lower-dimensional non-linear manifold, while staying more invariant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

### Homomorphism Autoencoder (HomoAE)

Paper: “Homomorphism Autoencoder – Learning Group Structured Representations from Observed Transitions”.

A Homomorphism Autoencoder (HomoAE) is a type of autoencoder that is trained to preserve the homomorphism (structure-preserving) properties of the input data in its encoded representation. This is achieved by adding a homomorphism constraint to the standard autoencoder loss function. The constraint ensures that the encoded representation preserves certain properties of the input, such as symmetry or commutativity. The decoder then tries to reconstruct the original input based on the encoded representation, which should also possess the same homomorphism properties. The result is a neural network that can learn to preserve structural information in the data and can be used for tasks such as dimensionality reduction, data generation, and anomaly detection.

From the paper: “How can we acquire world models that vertically represent the outside world both in terms of what is there and in terms of how our actions affect it? Can we acquire such models by interacting with the world, and can we state mathematical desiderata for their relationship with a hypothetical reality existing outside our heads? As machine learning is moving towards representations containing not just observation but also interventional knowledge, we study these problems using tools from representation learning and group theory. Under the assumption that our actuators act upon the world, we propose methods to learn internal representations of not just sensory information but also of actions that modify our sensory representations in a way that is consistent with the actions and transitions in the world. We use an autoencoder equipped with a group representation linearly acting on its latent space, trained on 2-step reconstruction such as to enforce a suitable homomorphism property on the group representation. Compared to existing work, our approach makes fewer assumption on the group representation and on which transformations the agent can sample from the group. We motivate our method theoretically, and demonstrate empirically that it can learn the correct representation of the groups and the topology of the environment. We also compare its performance in trajectory prediction with previous methods.”



### Competitive Learning

Paper: “Feature Discovery by Competitive Learning”

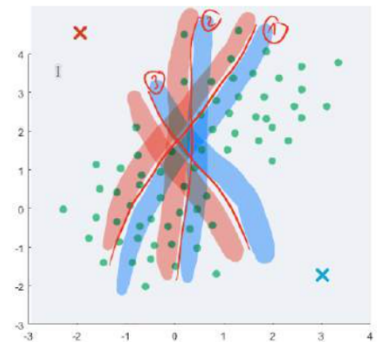
Competitive learning is a form of unsupervised learning in artificial neural networks, in which nodes compete for the right to respond to a subset of the input data. A variant of Hebbian learning, competitive learning works by increasing the specialization of each node in the network. It is well suited 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 its 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.

## The CL Algorithm

The competitive learning algorithm (two clusters):

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.



In the figure we have an 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.

## CL with Neural Networks

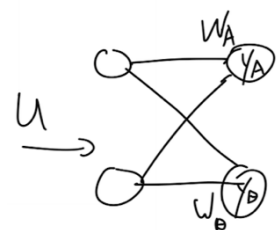
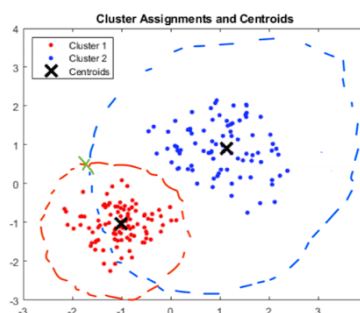
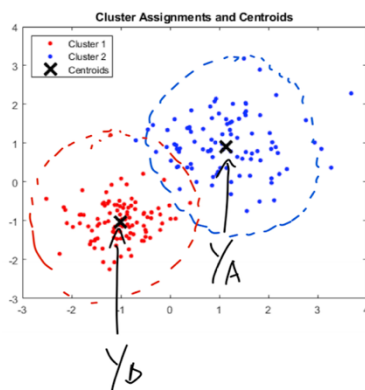
We ask for the neuron with the closest weight vector:

$$\|u - w_i\| = u^T u - 2w_i^T u + w_i^T w_i = \|u\|^2 + \|w_i\|^2 - 2y_i$$

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

We update our weights accordingly:  $w_A(t) = \frac{1}{t} \sum_{o=1}^t u_t$

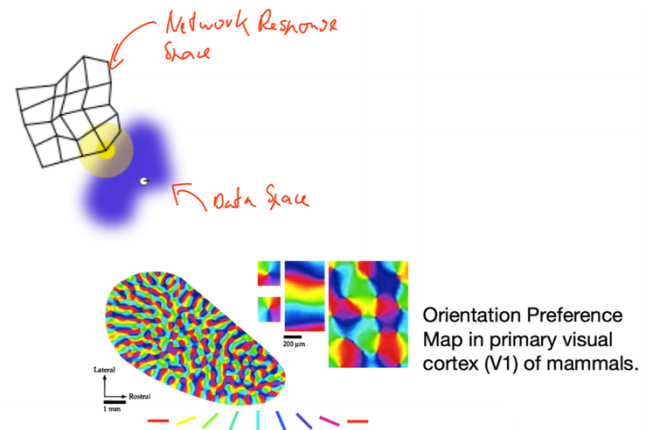
$$\nabla w_A = \eta(u_t - w_A)$$



The figure above shows Competitive NN: the position of the two neurons after convergence (left). A new datapoint and the data points equidistance line to the cluster-center neurons (center). The network structure (right).

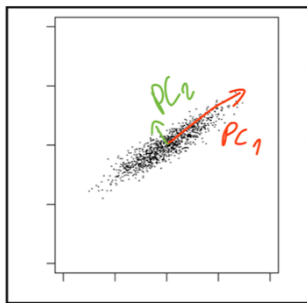
## Self-Organizing Maps

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.

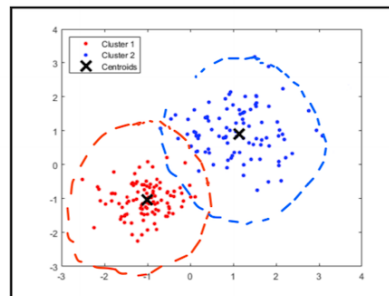


## Summary Plots

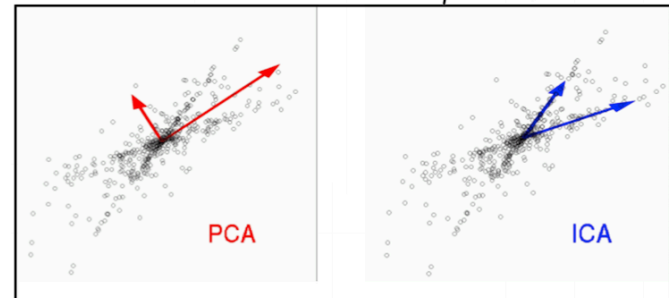
### PCA/Hebbian/Covariance Rule



### Competitive Learning



### PCA vs. ICA (h/omax)



## Probabilistic Generative Models

*“When one understands the causes, all vanished images can easily be found again in the brain through the impression of the cause. This is the true art of memory...” - Rene Descartes*

We know that our data has some causes  $v$ , but it is hard to specify. 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:

$$p(u, G) = \sum_v p(u|v, G)p(v, G)$$

$$G = (\mu_v, \tau_v, y_v)$$

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  $p(z)$ .

In the figure we have the difference between a 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. Indeed, interpolation between the classes is possible with a generative model.

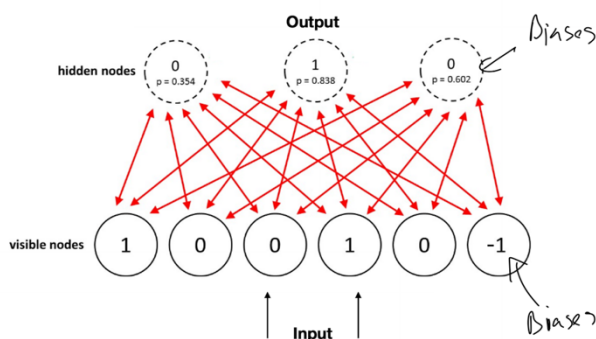
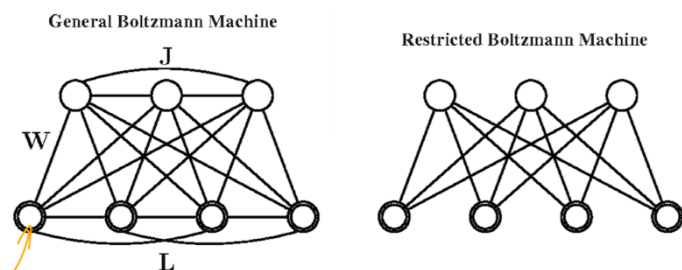


### The (restricted) Boltzmann Machine (RBM)

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 non between units of the same type (no hidden-hidden, nor visible-visible connections). Although learning is impracticable 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.

In the figure we have the 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.



In the (right) figure 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.

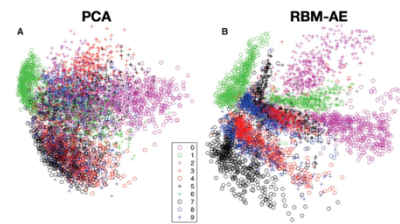
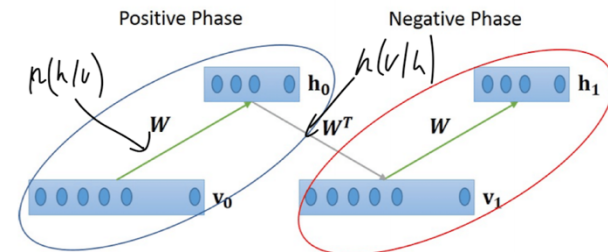
## Training the Restricted Boltzmann Machine (Contrastive Divergence)

Paper: “Training Products of Experts by Minimizing Contrastive Divergence”, “Reducing the Dimensionality of Data with Neural Networks”.

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:  $\nabla w = \eta(v_0 h_0^T - v_1 h_1^T)$ .



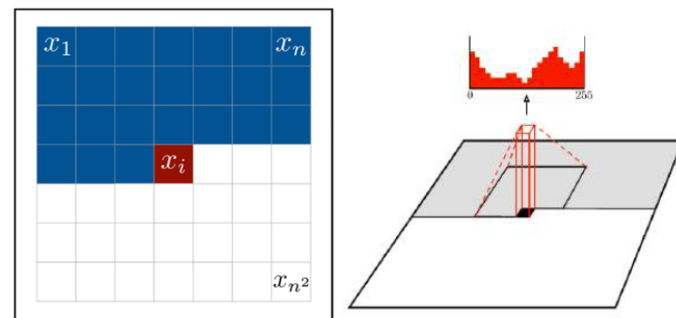
## Pixel-RNNs

Paper: “Pixel Recurrent Neural Networks”.

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).

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

In the figures we have the distribution over color space of a single pixel in the generative process of the PixelRNN.



## Alternative GM approaches not covered in this lecture:

- Helmholtz Machine & the Wake-Sleep algorithm.
- Variational Autoencoders & the Re-Parametrization trick.
- General Adversarial Networks (GANs).
- Flow Nets (Estimating Image Flow).
- Wave Nets (Used for Speech Generation).
- Expectation Maximization (EM).
- Moment-to-Moment Networks.

## Meta-Learning – Lecture 07 (a) – Benjamin Grewe

### Content of the Lecture

- Meta-Learning with ANNs
  - What is Meta-Learning?
  - Different Types of Meta-Learning Approaches.
  - Optimization Based (MAML).
  - Model Based (Meta and Hyper Networks).
- Meta-Learning in the Brain
  - Evidence for Meta-Learning in the Brain.
  - The Prefrontal-Cortex as Meta-Reinforcement Learner.
  - The Role of Neuromodulation in Meta Learning.

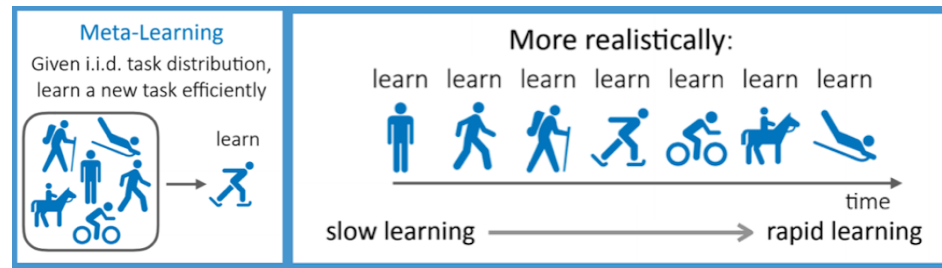


Paper: “Human-Level Concept Learning Through Probabilistic Program Induction”.

In the first lecture we listed the standing challenges in deep learning research, from which we now want to discuss continual learning 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 datasets (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.

The Principle of Learning the Learn



### 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:

- Learn an efficient distance metric (**metric-based**).
- Use (recurrent) network with external or internal memory (**model-based**).
- Optimize the model parameters explicitly for fast learning (**optimization-based**).

|                                                              | Model-based                 | Metric-based                                                                    | Optimization-based                        |
|--------------------------------------------------------------|-----------------------------|---------------------------------------------------------------------------------|-------------------------------------------|
| <b>Key idea</b>                                              | RNN; memory                 | Metric learning                                                                 | Gradient descent                          |
| <b>How <math>P_{\theta}(y \mathbf{x})</math> is modeled?</b> | $f_{\theta}(\mathbf{x}, S)$ | $\sum_{(\mathbf{x}_i, y_i) \in S} k_{\theta}(\mathbf{x}, \mathbf{x}_i) y_i$ (*) | $P_{g_{\phi}(\theta, S^L)}(y \mathbf{x})$ |

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:

$$\theta^* = \arg \min_{\theta} \mathbb{E}_{\mathcal{D} \sim p(\mathcal{D})} [\mathcal{L}_{\theta}(\mathcal{D})]$$

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  $S$  for learning and a prediction set  $B$  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  $f_{\theta}$  is the “learner” model, trained for operating a given task.
2. In the meantime, an optimizer  $g_{\phi}$  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  $\theta$  and  $\phi$  to maximize:

$$\mathbb{E}_{L \subset \mathcal{L}} \left[ \mathbb{E}_{S^L \subset \mathcal{D}, B^L \subset \mathcal{D}} \left[ \sum_{(\mathbf{x}, y) \in B^L} P_{g_{\phi}(\theta, S^L)}(y|\mathbf{x}) \right] \right]$$

## 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 weights sum of label of support set samples. The weight is generated by a kernel function  $k_\theta$ , measuring the similarity between two data samples.

$$P_\theta(y|\mathbf{x}, S) = \sum_{(x_i, y_i) \in S} k_\theta(\mathbf{x}, \mathbf{x}_i) y_i$$

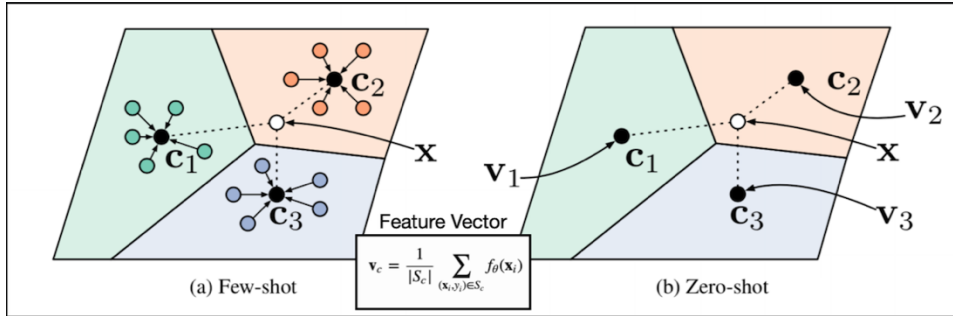
To learn a good kernel is crucial to the success of a metric-based meta-learning model. Metric learning is well aligned with this intention, as it aims to learn a metric or distance function over objects. The notion of a good metric is problem-dependent. It should represent the relationship between inputs 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** (“*Prototypical Networks for Few-shot Learning*”)
- **Siamese Networks**
- **Matching Networks** (“*Matching Networks for One Shot Learning*”)
- **Relation Networks** (“*Learning to Compare: Relation Network for Few-Shot Learning*”)

### Prototypical Networks

They use an embedding function  $f_\theta$  to encode each input into a M-dimensional feature vector. A prototype feature vector is defined for every class  $c \in C$ , as the mean vector of the embedded support data samples in this class.

$$\mathbf{v}_c = \frac{1}{|S_c|} \sum_{(x_i, y_i) \in S_c} f_\theta(x_i)$$



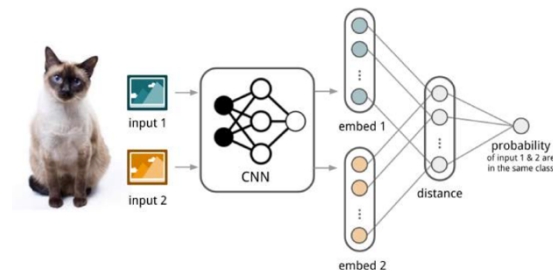
The distribution over classes for a given test input  $\mathbf{x}$  is a *softmax* over the inverse of distances between the test data embedding and prototype vectors.

$$P(y = c|\mathbf{x}) = \text{softmax}(-d_\varphi(f_\theta(\mathbf{x}), \mathbf{v}_c)) = \frac{\exp(-d_\varphi(f_\theta(\mathbf{x}), \mathbf{v}_c))}{\sum_{c' \in C} \exp(-d_\varphi(f_\theta(\mathbf{x}), \mathbf{v}_{c'}))}$$

where  $d_\varphi$  can be any distance function as long as  $\varphi$  is differentiable. In the paper, they used the squared Euclidean distance. The loss function is the negative log-likelihood:  $\mathcal{L}(\theta) = -\log P_\theta(y = c|\mathbf{x})$

### Siamese Networks

They 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.



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 an embedding function  $f_\theta$  which contains a couple of convolutional layers.
2. The L1-distance between two embeddings is  $|f_\theta(\mathbf{x}_i) - f_\theta(\mathbf{x}_j)|$
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.

$$p(\mathbf{x}_i, \mathbf{x}_j) = \sigma(\mathbf{W}|f_\theta(\mathbf{x}_i) - f_\theta(\mathbf{x}_j)|)$$

$$\mathcal{L}(B) = \sum_{(\mathbf{x}_i, \mathbf{x}_j, y_i, y_j) \in B} \infty_{y_i=y_j} \log p(\mathbf{x}_i, \mathbf{x}_j) + (1 - \infty_{y_i=y_j}) \log(1 - p(\mathbf{x}_i, \mathbf{x}_j))$$

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:

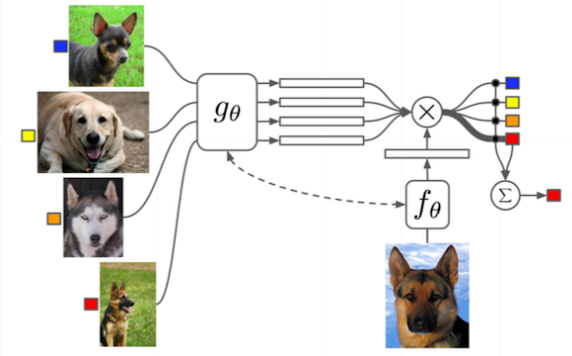
$$\hat{c}_S(\mathbf{x}) = c(\arg \max_{\mathbf{x}_i \in S} SP(\mathbf{x}, \mathbf{x}_i))$$

where  $c(\mathbf{x})$  is the class label of an image  $\mathbf{x}$  and  $\hat{c}(\cdot)$  is the predicted label.

### Matching Networks

They aim at learning a classifier  $c_S$  for any given (small) support set  $S = \{\mathbf{x}_i, y_i\}_{i=1}^k$  (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  $\mathbf{x}_i$ .

$$c_S(\mathbf{x}) = P(y|\mathbf{x}, S) = \sum_{i=1}^k a(\mathbf{x}, \mathbf{x}_i) y_i, \text{ where } S = (\mathbf{x}_i, y_i)_{i=1}^k$$



The attention kernel depends on two embedding functions,  $f$  and  $g$ , for decoding the test sample and the support set samples respectively. The attention weight between two data points is the cosine similarity,  $\cosine(\cdot)$ , between their embedding vectors, normalized by *softmax*:

$$a(\mathbf{x}, \mathbf{x}_i) = \frac{\exp(\cosine(f(\mathbf{x}), g(\mathbf{x}_i)))}{\sum_{j=1}^k \exp(\cosine(f(\mathbf{x}), g(\mathbf{x}_j)))}$$

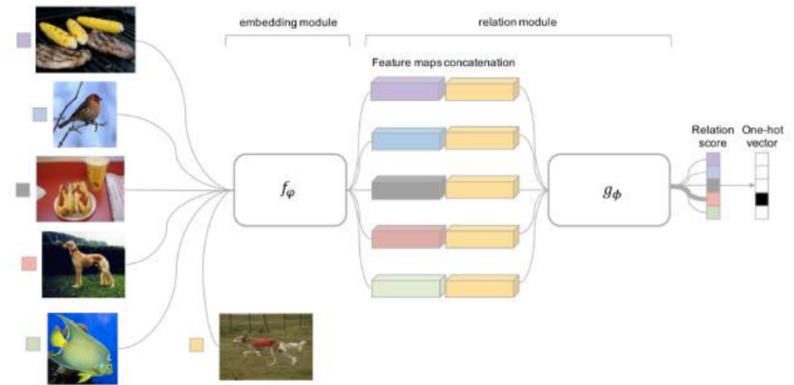
The embedding has to be chosen carefully. In a simple version, an embedding function is a neural network with a single data sample as input. Taking a single data point as input might not be enough to efficiently gauge the entire feature space. Therefore, the Matching Network model further proposed to 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 Networks

They are similar to Siamese Networks 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  $g_\phi$ . The relation score between a pair of inputs,  $\mathbf{x}_i$  and  $\mathbf{x}_j$ , is  $r_{ij} = g_\phi([\mathbf{x}_i, \mathbf{x}_j])$  where  $[\cdot, \cdot]$  is a 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:  $\mathcal{L}(B) = \sum_{(x_i, x_j, y_i, y_j) \in B} (r_{ij} - \mathbf{1}_{y_i=y_j})^2$ .

In the figure we have a Relation Network architecture for a 5-way 1-shot problem with one query example.



## Model-Based ML

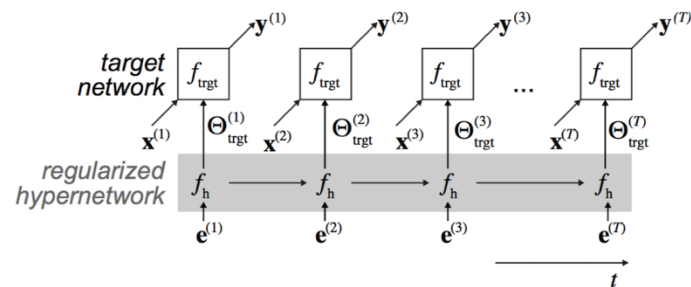
Model-based meta-learning models make no assumption on the form of  $P_\theta(y|x)$ . Rather it depends on a model designed specifically for fast learning – a model that updates its parameters rapidly with a few training steps. This rapid parameter update can be achieved by its internal architecture or controlled by another meta-learner model.

## Hypernetworks

Paper: “Continual Learning in Recurrent Neural Networks”, “Continual Learning with Hypernetworks”, “Meta-Learning with Latent Embedding Optimization”.

They 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 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.

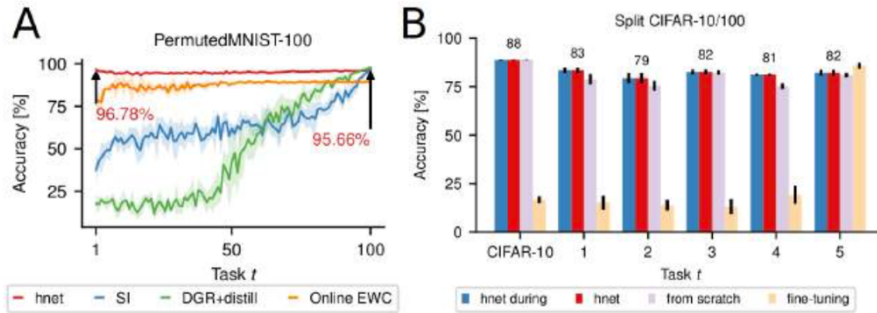
Commonly, the parameters of a neural network are directly adjusted from data to solve a task. Here, a weight generator termed hypernetwork is learned instead. Hypernetworks map embedding vectors to weights, which parametrize a target neural network. In a continual learning scenario, a set of task-specific embeddings is learned via backpropagation. Embedding vectors provide task-dependent context and bias the hypernetwork to particular solutions.



## Few-Shot Meta-Learning with Hypernetworks

In a few-shot meta-learning setting, a base network is trained on a set of tasks, and then the parameters of this base network are used as inputs to a hypernetwork, which generates the task-

specific weights for the base network. When presented with a new task, the base network's parameters are passed through the hypernetwork again, generating the weights for the base network to use on the new task. The key idea behind this approach is that the base network's parameters contain information about how to solve a wide variety of tasks, and the hypernetwork learns to generate task-specific weights based on this information. This allows the base network to quickly adapt to new tasks with very little data, because it can leverage its previous experience to quickly learn the new task.



In the figure we have two 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), online EWC (in orange) and Deep Generative Replay (DGR+distill, in green) methods are shown for comparison. **(B)** Split CIFAR-10/100 continual learning benchmark. Test set accuracies (mean  $\pm$  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).

### Optimization-Based ML

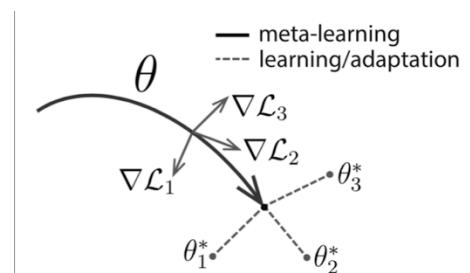
Deep Learning models learn through backpropagation of gradients. However, the gradient-based optimization is neither designed to cope with a small number of training sample, nor to converge within a small number of optimization steps. Is there a way to adjust the optimization algorithm so that the model can be good at learning with a few examples? This is what optimization-based approach meta-learning algorithms intend for. Look at **Model-Agnostic Meta-Learning (MAML)** and **LSTM Meta-Learner, Reptile** for further information (Not covered in this class).

### Model-Agnostic Meta-Learning (MAML)

This is a fairly general optimization algorithm, compatible with any model that learns through gradient descent. Let's say our model is  $f_\theta$  with parameters  $\theta$ . Given a task  $\tau_i$  and its associated dataset ( $\mathcal{D}^{(i)}$  train,  $\mathcal{D}^{(i)}$  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). 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  $\theta^*$  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.

$$\theta^* = \arg \min_{\theta} \sum_{\tau_i \sim p(\tau)} \mathcal{L}_{\tau_i}^{(1)}(f_{\theta'}^i) = \arg \min_{\theta} \sum_{\tau_i \sim p(\tau)} \mathcal{L}_{\tau_i}^{(1)}(f_{\theta} - \alpha \nabla_{\theta} \mathcal{L}_{\tau_i}^{(0)}(f_{\theta}))$$

$$\theta \leftarrow \theta - \beta \nabla_{\theta} \sum_{\tau_i \sim p(\tau)} \mathcal{L}_{\tau_i}^{(1)}(f_{\theta} - \alpha \nabla_{\theta} \mathcal{L}_{\tau_i}^{(0)}(f_{\theta})); \text{ updating rule}$$

#### Algorithm 1 Model-Agnostic Meta-Learning

**Require:**  $p(\mathcal{T})$ : distribution over tasks

**Require:**  $\alpha, \beta$ : step size hyperparameters

```

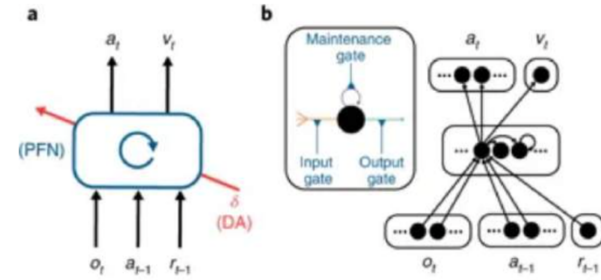
1: randomly initialize  $\theta$ 
2: while not done do
3:   Sample batch of tasks  $\mathcal{T}_i \sim p(\mathcal{T})$ 
4:   for all  $\mathcal{T}_i$  do
5:     Evaluate  $\nabla_{\theta} \mathcal{L}_{\mathcal{T}_i}(f_{\theta})$  with respect to  $K$  examples
6:     Compute adapted parameters with gradient descent:  $\theta'_i = \theta - \alpha \nabla_{\theta} \mathcal{L}_{\mathcal{T}_i}(f_{\theta})$ 
7:   end for Note: the meta-update is using different set of data.
8:   Update  $\theta \leftarrow \theta - \beta \nabla_{\theta} \sum_{\tau_i \sim p(\mathcal{T})} \mathcal{L}_{\tau_i}(f_{\theta'_i})$ 
9: end while

```

### Meta-Learning in the Brain

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 deal with a wider range of observations.

In the picture we have a **Meta-RL architecture 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 connects 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  $\sigma$  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 stimulations.



### Meta-Learning via Neuromodulation

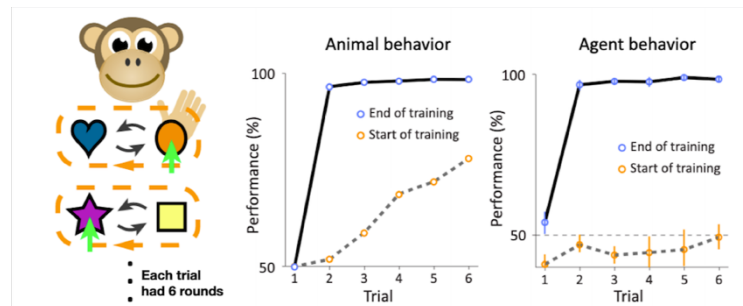
Neuromodulators play an important role in meta-learning in the brain. Some of the key modulators are listed below. Neuromodulatory systems can be seen to mediate the global signals that regulate the distributed learning mechanisms in the brain. Based on the review of experimental data and theoretical 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.
- **Serotonin** 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.

The paper “*Reinforcement Learning, Fast and Slow*” presents a framework for understanding the difference between two types of reinforcement learning algorithms: “fast” RL and “slow” RL. Fast RL algorithms, such as Q-learning, learn quickly but are prone to overfitting and instability. Slow

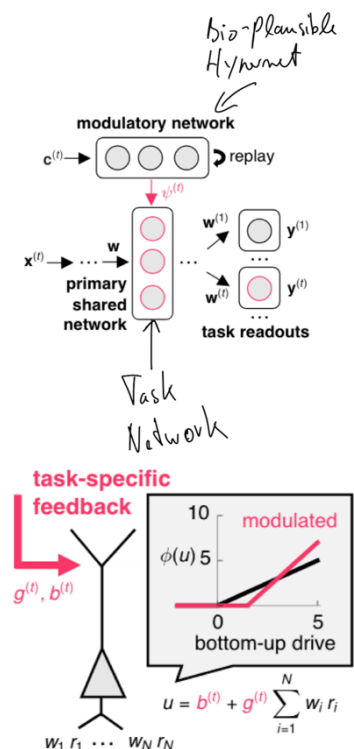
RL algorithms, such as Policy Gradient methods, learn more slowly but are more stable and less prone to overfitting. The paper argues that a combination of fast and slow RL can lead to better performance in a variety of tasks. Additionally, the paper also suggest that human learning can be understood in terms of these two types of RL, with fast RL corresponding to trial-and-error learning and slow RL corresponding to more deliberate, goal-directed learning.

The Harlow experiment is a study conducted by psychologist Harry Harlow in the 1950s, which aimed to investigate the role of learning in the development of complex problem-solving abilities. The experiment used rhesus monkeys as subjects, and it consisted of two parts. In the first part, the monkeys were presented with a series of problems, such as reaching through a hole in a barrier to retrieve food. The monkeys were



allowed to explore the problems and find solutions through trial and error. In the second part of the experiment, the monkeys were presented with a new set of problems that were more difficult than the ones they had encountered before. The monkeys were able to use the knowledge and skills they had acquired during the first part of the experiment to solve the new problems more quickly and effectively. This experiment demonstrated that the experience of solving problems through trial and error could lead to the development of problem-solving skills and strategies, which can be applied to new and more complex problems. This experiment was important in supporting the idea that learning to learn is possible, and that this type of learning can be achieved through experience and exposure to different challenges.

The concept of "Bio-plausible Modulatory Networks" is a method that attempts to mimic the way the brain continually learns. This approach is based on the idea that the brain uses a combination of different neural networks, each with a specific function, to process and learn from information. These networks work together and communicate with each other through modulatory signals, which can adjust the activity of different networks depending on the task or context. In this approach, the continual learning model is composed of several networks, each specialized in a specific task, and these networks are modulated by a central controller network. The central controller network is responsible for adapting the activity of the specialized networks depending on the task to be solved, and this allows the model to continue to learn new tasks without forgetting the previous ones.



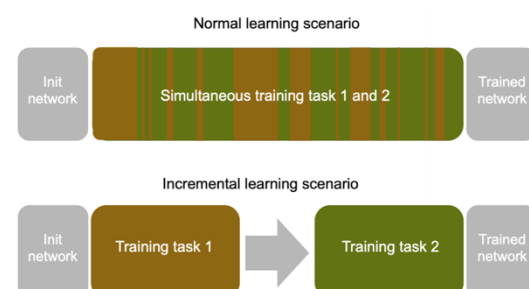
## Continual Learning – Lecture 07 (b) – Martino Sorbaro

### Content of the Lecture

- What do we mean by *catastrophic forgetting* and *continual learning*?
- How do we build models that are more resilient to forgetting?
- How does this relate to what happens in a biological brain?
- More in-depth study of a few algorithms.

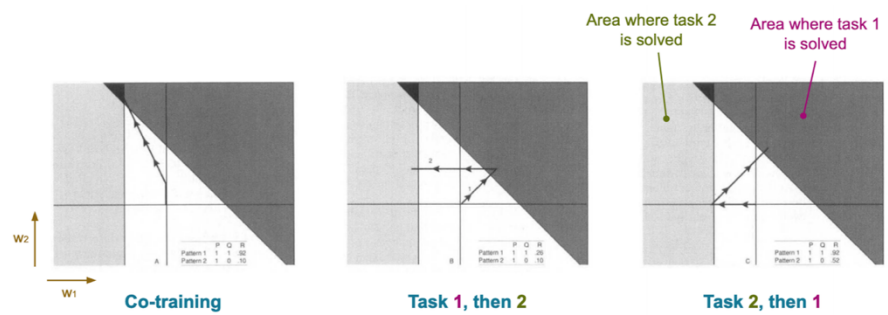
### Learning Arithmetics & Word Associations

“Catastrophic interference in connectionist networks: the sequential learning problem”. In front of a 2-task incremental learning scenario, Humans can still perform decently on the first task after



having learned the second one, while neural nets immediately forget the first when they start learning the second.

In the picture we have the view of the parameter space.



## Continual Learning “Scenarios” and Benchmarks

There are three scenarios:

- **Task-IL**, solve tasks so far, task-ID provided.
- **Domain-IL**, solve tasks so far, task-ID not provided.
- **Class-IL**, solve tasks so far and infer task-ID.

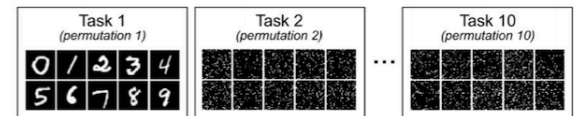


Figure 2: Schematic of permuted MNIST task protocol.

Table 3: Permuted MNIST according to each scenario.

|                  |                                           |
|------------------|-------------------------------------------|
| <b>Task-IL</b>   | Given permutation $X$ , which digit?      |
| <b>Domain-IL</b> | With permutation unknown, which digit?    |
| <b>Class-IL</b>  | Which digit <i>and</i> which permutation? |

## Strategies for Continual Machine Learning

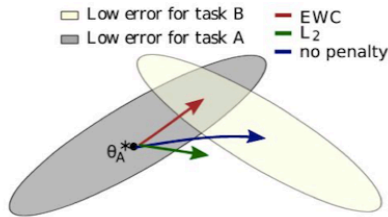
- **Architectural**
  - Use only part of a network.
  - Select subnetwork based on task.
  - Or, add new parts to a network when new tasks are learnt.
  - **Progressive Networks.**
- **Regularization-Based**
  - Add loss terms that try to keep the network constant:
    - In behavior or
    - In weights.
    - **Elastic Weight Consolidation & Synaptic Intelligence.**
- **Data Replay-Based**
  - Store, or learn to generate some of the previous data (Experience Replay).
  - Keep training on them while you train on new data (Generative Replay).
  - **Deep Generative Replay.**

## An Example of Architectural methods (Progressive Networks)

"*Progressive Neural Networks*" is a paper published by Google Brain team in 2017, that describes a method for incremental learning, which allows neural networks to learn new tasks without forgetting the previous ones. The paper propose a technique called Progressive Networks (PN), which is based on the idea of growing the neural network incrementally as new tasks are encountered. The PN approach consists of a multi-task neural network, where each task is associated with a specific sub-network called a "column". Each column is trained to perform a specific task, and new columns can be added as new tasks are encountered. The new columns are connected to the previously learned columns, allowing the network to transfer knowledge from previous tasks to new ones. The paper shows that the PN approach can learn a wide range of tasks, with different levels of difficulty, and it can also achieve better performance compared to other methods for incremental learning, such as fine-tuning or freezing the previous layers. The Progressive Network approach is useful in scenarios where the number of tasks or the amount of data is not known in advance, and it can be useful in applications such as lifelong learning, online learning and online adaptation.

## Regularization Methods

Regularization methods are used in continual learning to prevent catastrophic forgetting, which occurs when a model forgets previously learned tasks when learning new tasks.



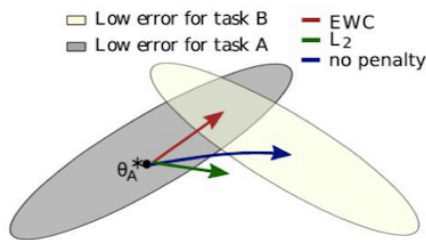
$$\mathcal{L}(\theta) = \boxed{\mathcal{L}_B(\theta)} +$$

Loss on task B

Regularization term that tries to preserve parameters important for task A

## Elastic Weight Consolidation (EWC)

Elastic Weight Consolidation (EWC) aims to keep the parameters of a neural network that are important for previous tasks fixed while allowing the parameters that are important for new tasks to change. This is done by adding a penalty term to the loss function based on the difference between the current parameters and the parameters that were optimal for the previous tasks. The main idea of EWC is to keep the parameters of a neural network that are important for previous tasks fixed while allowing the parameters that are important for new tasks to change. This is done by adding a penalty term to the loss function based on the difference between the current parameters and the parameters that were optimal for the previous tasks. The importance of each parameter is measured by the Fisher information matrix, which quantifies the amount of information that the parameter contains about the task.



$$\mathcal{L}(\theta) = \boxed{\mathcal{L}_B(\theta)} + \sum_i \frac{\lambda}{2} \boxed{F_i} (\theta_i - \boxed{\theta_{A,i}^*})^2$$

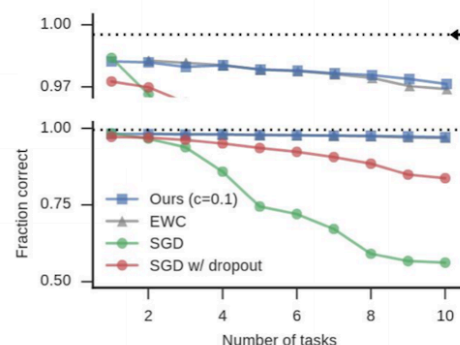
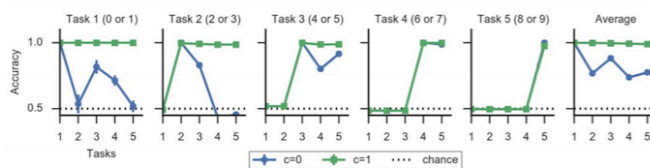
Loss on task B

Parameters learnt on task A

Fisher information relative to  $\theta_i$  (curvature of parameter space)

## Synaptic Intelligence (SI)

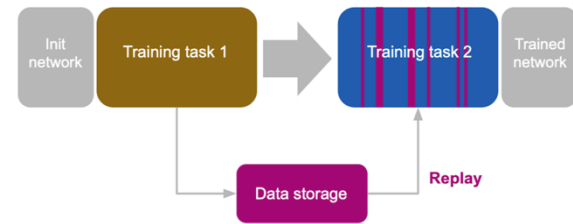
Synaptic Intelligence (SI) aims to keep the parameters that were important for previous tasks fixed by adjusting the learning rate of each parameter based on how much it has changed during previous tasks. The main idea of SI is to adjust the learning rate of each parameter in a neural network based on how much the parameter has changed during previous tasks. The authors propose a measure of the "importance" of each parameter, which is based on the magnitude of the gradient of the parameter with respect to the loss function during previous tasks. Parameters that have had a large gradient in the past are considered more important and have a lower learning rate, while parameters that have had a small gradient in the past have a higher learning rate. The authors test SI on a variety of image classification tasks and show that it outperforms EWC and other baselines. The paper also introduces a novel method for evaluating the performance of continual learning algorithms called "learning progress".





Data Replay methods in continual learning involve storing previously seen data and reusing it to help the model retain information from previous tasks when learning new tasks. This can be done in several ways, however the most common are:

- **Experience Replay:** this method involves storing a dataset of previous experiences, such as input-output pairs, and randomly sampling from this dataset during training on new tasks to provide additional examples for the model to learn from.
  - **Generative Replay:** this method involves training a generative model, such as a Generative Adversarial Network (GAN) or Variational Autoencoder (VAE), on the previous task data, and then using the generative model to generate new examples from the previous task to be used during training on new tasks.
- 
- ```
graph LR; Task[Task] --> DS[Data storage]; DS --> Model[Model]
```

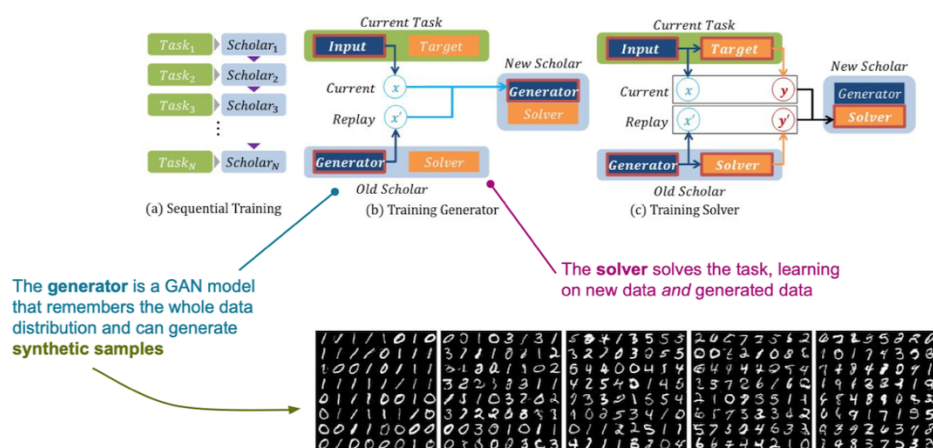


## Deep Generative Replay

Data Generative Replay is a method in continual learning that uses a generative model to generate new examples from previous tasks to be used during training on new tasks.

The general process of Data Generative Replay is as follows:

1. A generative model, such as a Generative Adversarial Network (GAN) or Variational Autoencoder (VAE), is trained on data from a previous task.
2. The generative model is then used to generate new examples from the previous task.
3. These generated examples are added to the training set for the new task, along with the examples from the new task.
4. The model is then trained on the combined dataset (new task examples + generated examples from previous task)
5. Steps 1-4 are repeated for each new task.



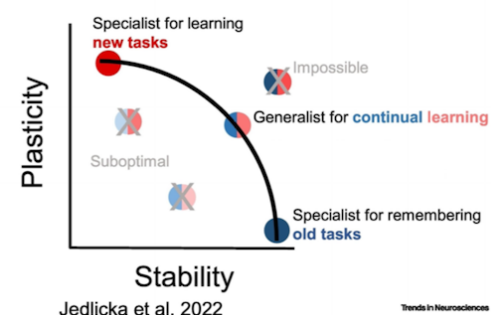
## How Does the Brain Do Continual Learning?

## May these ML methods be relevant for the brain?

## The Stability – Plasticity Dilemma

In ML, we can either:

- Speed up the learning rate, so we learn task 2 faster – but we shall also forget faster.
- Slow down the learning rate, so we forget task 1 slower – but we shall also learn task 2 less.





Similarly the brain needs to strike a balance between:

- More **stable** synapses, less sensitive to learning.
- More **plastic** synapses, which are more unstable.

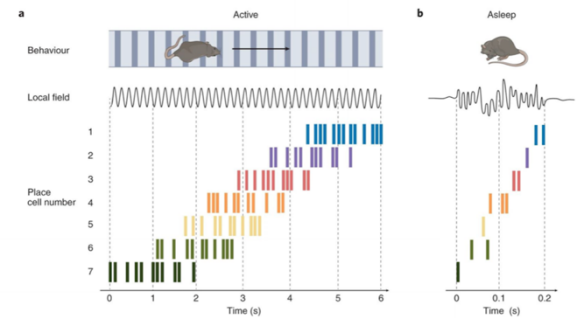
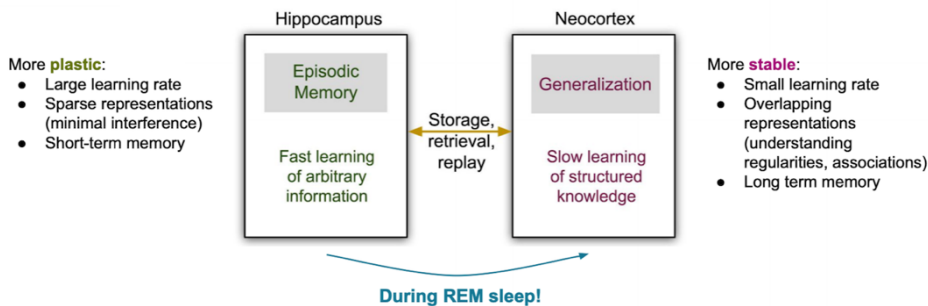
For example, children have more plastic brains, and both learn and forget faster than adults.

How can we solve this dilemma?

- Two complementary Learning Systems
- Neurogenesis
- Metaplasticity

## Two Complementary Learning Systems

As evidenced by experiments in rats, memories stored in the Hippocampus are replayed in the same order during sleep for consolidation.



## Neurogenesis

The process of growth of **new neurons**. It is known to happen during **development** in small children at a high rate. **Adult** neurogenesis:

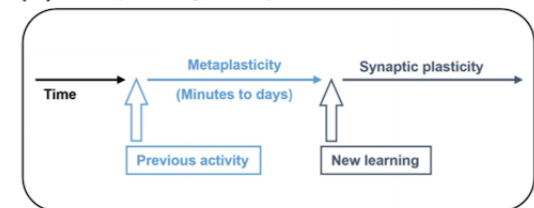
- Slow rates.
- Limited to certain areas of the brain.

The **Hippocampus** is one of the areas with adult neurogenesis. We could guess new neurons to be **plastic**, and older neurons to be more **stable**. Research on the role of neurogenesis in the formation of new memories is **inconclusive** so far.

## Metaplasticity

We usually talk about synaptic strength, or “weight”, which is changed by plasticity. But previous activity could also change **how easily a synapse undergoes plasticity**. There are mechanisms to regulate the **stability** or **plasticity** of the single synapse, acting on a **longer timescale**. This is called **metaplasticity**.

### (A) Metaplasticity concept



Aspects of continual learning here in a general sense, not just “incremental”

|                                         | Key features            |                                    |                            |                        |                 |                                        |
|-----------------------------------------|-------------------------|------------------------------------|----------------------------|------------------------|-----------------|----------------------------------------|
|                                         | Transfer and adaptation | Overcoming catastrophic forgetting | Exploiting task similarity | Task-agnostic learning | Noise tolerance | Resource efficiency and sustainability |
| Neurogenesis                            | ●                       | ●                                  | ●                          | ●                      |                 | ●                                      |
| Episodic replay                         |                         | ●                                  |                            | ●                      |                 | ●                                      |
| Metaplasticity                          |                         | ●                                  |                            | ●                      |                 | ●                                      |
| Neuromodulation                         | ●                       | ●                                  |                            |                        |                 |                                        |
| Context-dependent perception and gating | ●                       | ●                                  | ●                          | ●                      | ●               |                                        |
| Hierarchical distributed systems        |                         |                                    | ●                          |                        | ●               |                                        |
| Cognition outside the brain             | ●                       |                                    | ●                          |                        | ●               |                                        |
| Reconfigurable organisms                | ●                       | ●                                  | ●                          |                        |                 |                                        |
| Multisensory integration                |                         |                                    | ●                          |                        | ●               |                                        |

# Why Spikes? – Lecture 08 – Benjamin Grewe

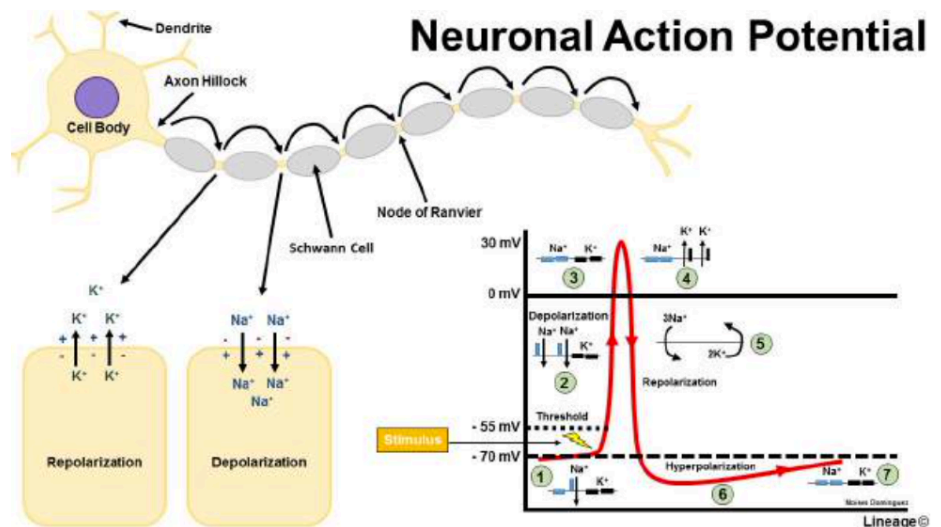
## Content of the Lecture

- What is a Neuronal Spike?
- (Dis)advantages of Digital (spike) vs. Non-Digital Communication.
- Non-Spiking Biological Systems and their Properties.
- Analogue Communication in the Retina, C. Elegans, Locust.
- The Different Types of Action Potentials.
- How Can We Measure Spiking Activity in a Biological Neuron?
- Temporal Coding Schemes with Spikes.
- Deep Learning with “Time to First Spike”.
- Introduction to Neuronal Spiking Dynamics.
- Intracellular Integration of Incoming Spikes.
- Action Potential Generation.
- Hodgkin-Huxley Model.

## 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.

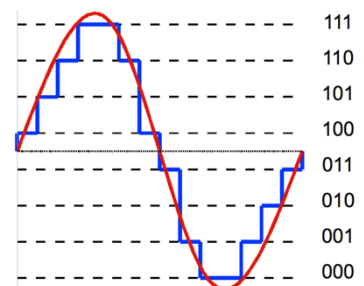
- Neuron is at resting potential (-70 mV) which is determined by the balance between extracellular and intracellular  $K^+$ ,  $Na^+$  and  $Cl^-$  concentrations.
- Neuron receives inputs at its dendrites which raises the membrane potential. If the membrane potential is below the threshold potential (-55 mV), 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 of 40/50 mV.
- 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 hyperpolarization. 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.



## (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 (as shown in the figure). As we can clearly see, 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, non-linear quantization techniques and schemas exist. One can take advantage of alternative quantization techniques to digitize an analogue signal

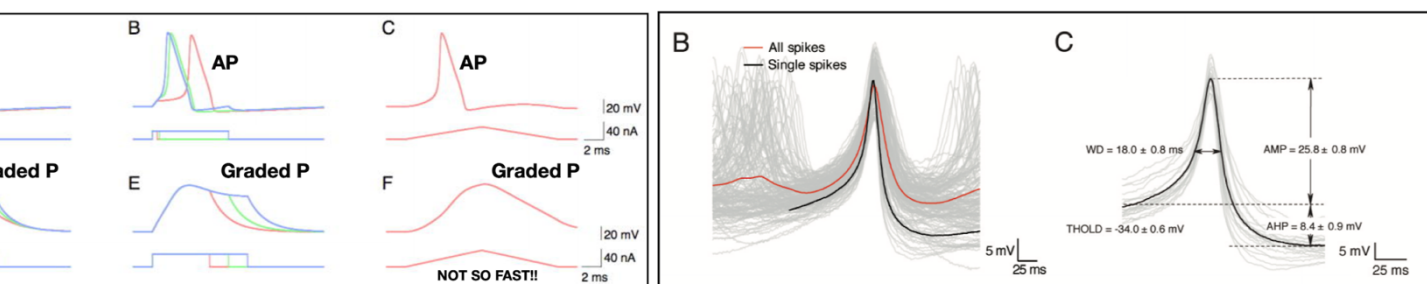
that has a different entropy, or more information concentrated at lower values at the cost of reducing the resolution in higher values. The most striking argument for analogue activity is given by Shannon's Information Capacity. A calculation done for Crab neurons demonstrates that an analogue channel transfers 2500-6000 bits/s while a digital channel would have 50-220 bits/s: an



order of magnitude inferior to an analogue channel. On the other side, the advantage of digital information is that we experience no signal loss during signal transmission across long distances.

### 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 (Photoreceptors, Horizontal Cells, Olfactory Granule Cells) where neurons don't utilize spiking. An entire textbook (*"Neurons without Impulses – their significance for vertebrate and invertebrate nervous systems"* – Alan Roberts) exists to cover these examples, but the lecture focuses just on one case: C. Elegans. C. Elegans have 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.



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.

An **Action Potential** is a rapid, all-or-nothing change in the electrical potential across the membrane of a nerve cell or muscle cell. It is triggered by a threshold stimulus, and once it is initiated, it propagates along the cell membrane without decreasing in amplitude.

A **Graded Potential**, on the other hand, is a change in the electrical potential across the membrane of a cell that varies in amplitude and duration. They are triggered by stimuli that do not reach the threshold level required to initiate an action potential, and their amplitude decreases with distance from the point of stimulation. Graded potentials can be either excitatory or inhibitory and can summate together. In summary, an action potential is an all-or-nothing, rapid change in membrane potential, triggered by a threshold stimulus and propagates without decrease in amplitude. A graded potential is a change in membrane potential that varies in amplitude, triggered by stimuli that do not reach threshold level, and decreases with distance from the point of stimulation.

### Why Spikes- 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 Process – 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 amoeba uses Mechanosensitive Calcium channels to generate a spike which causes local contraction/compression to generate movement away from an object.
- Calcium spikes regulates homeostatic processes –  $\text{Ca}^{2+}$  intracellular signaling cascade/pathways are the foundations of molecular biology. Seeing  $\text{Ca}^{2+}$  influx follows an action potential, it would be wasteful to have to come up with a different mechanism to mediate  $\text{Ca}^{2+}$  levels. Spiking does this as a byproduct.

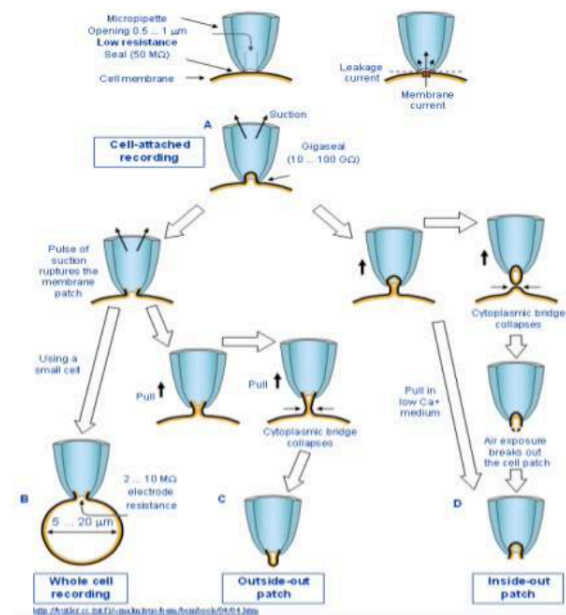
- Cells send information across long distances – Any analogue wavelength would decay over time, and be vulnerable to noise fluctuations. Neurons have tricks to reduce this (myelination, increasing axon diameter in sea squids), but ultimately spikes guarantee an intact and reconstructable signal being delivered.
- Energy Efficiency – Self Explanatory. A human uses 100 Watts, the brain takes up to 20 W. Compare that to the power supply unit of a standard desktop (300W) or a high-end rig (800W).

## How To Measure Spiking Activity in a Biological Neuron?

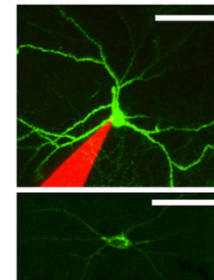
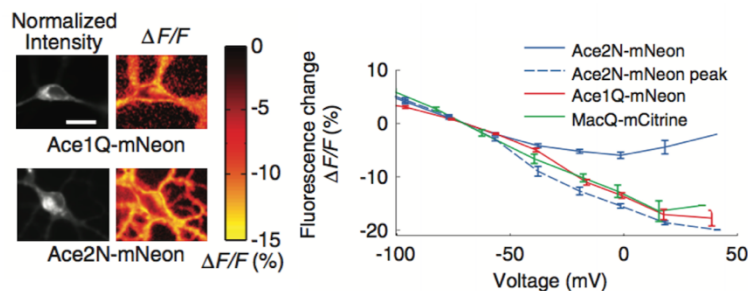
How would one measure a voltage level change of anything?

Using a Voltmeter. The only caveat to this is probe design.

Simply placing the tip of the probe in contact with the membrane is usually inefficient (especially at the 1 or 2 micrometer level), thus there are several techniques to properly “clamp” the membrane.



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 fluorescence techniques (Voltage Indicators) – Ace1Q-mNeon & Ace2N-mNeon are examples of them.



GEVIs undergo a conformational change in response to a voltage change which changes their fluorescence levels. 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.

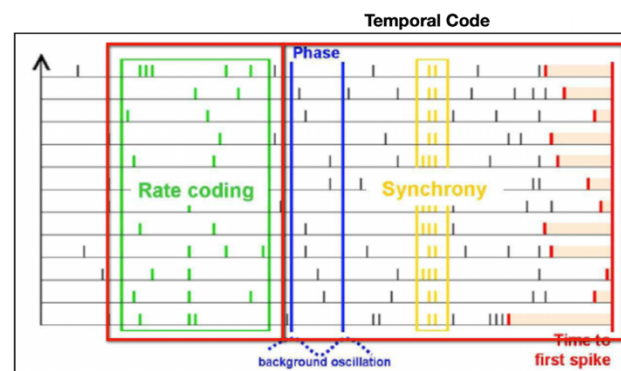
Advantages of Imaging:

- Allows to simultaneously record populations of neurons.
- Allows cell type (gene specific) labelling and imaging.
- Enables repeated (chronic) imaging over many months.

## Temporal Coding Schemes with Spikes

Now that we have 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. These can be divided into two primary categories:

- **Rate Coding**, it refers to information being carried by the firing rate. It is often argued, or assumed, that firing rate captures essentially all relevant information.
- **Temporal Coding**, it may refer to several quite different ideas:





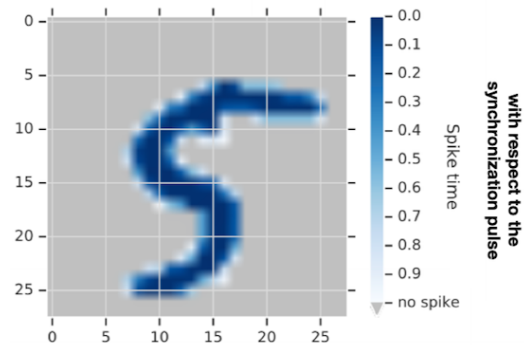
- Much of the information may be transmitted by a neuron during certain small intervals of time.
- Synchronous, or what I would call quasi-synchronous, firing of neurons within and across ensembles may carry important information.
- The precise timing, or pattern, of spikes may carry information.

In the lecture, several examples were given:

- Spike Rate coding in the Cat V1 encodes direction selectivity. (Cat)
- Phase Coding in Place Cells in the Hippocampus to determine position. (Rat)
- Temporal (Latency) Coding to determine Interaural Time Difference (ITD). (Owl)

### Deep Learning with “Time to First Spike”

An Artificial Spiking Neural Network has been trained to utilize the time to first spike scheme. The (MNIST) 2D image information was encoded as shown in the picture. “*Temporal Coding in Spiking Neural Networks with Alpha Synaptic Function*”, this paper modelled spikes via the Alpha Function. And it can be trained to solve Boolean tasks (AND, OR, XOR), as well as MNIST classification. The alpha synaptic function is a mathematical model that describes the dynamics of synaptic neurotransmitter release. It is characterized by a time constant and a maximum conductance, and it can be used to model the behavior of different types of synapses. In the context of learning with backpropagation, the alpha synaptic function can be used to improve the accuracy of the network by allowing for more precise control of the timing of spikes. By using the alpha synaptic function, the network can more accurately represent temporal patterns in the input data and perform temporal coding.



### Hodgkin-Huxley Model

Mathematically models neuronal firing.

Properties of the Sodium channel:

- Depolarize neuron when open (-70mV → +40mV)
- Establish a positive **feedback** loop (Voltage increase → Na<sup>+</sup> channels open → Voltage increase).

Properties of the Potassium channel:

- K-channels open → Voltage decrease.
- K-channels are Voltage-dependent.
- K-channels are slow. (Voltage increase → triggers fast Na<sup>+</sup> and positive feedback → Voltage increase → one millisecond later K<sup>+</sup> open → Voltage decrease).

### Hodgkin-Huxley type equations

$$C \frac{dv}{dt} = I_{app} - \sum I_{ion}(v, m_i, h_i)$$

$$\frac{dm_i}{dt} = \frac{m_{\infty}(v) - m_i}{\tau_m(v)}$$

$$\frac{dh_i}{dt} = \frac{h_{\infty}(v) - h_i}{\tau_h(v)}$$

$$I_{Na} = \bar{g}_{Na} m_{Na}^p(v) h_{Na}^q(v) [v - V_{Na}]$$

$$I_K = \bar{g}_K m_K^r(v) h_K^s(v) [v - V_K]$$

$$I_L = \bar{g}_L [v - V_L]$$

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



# Deep Learning With Spikes – Lecture 09 – Friedemann Zenke

## Content of the Lecture

- **Recap:** Spiking Neuron Models
- **Aim:** Supervised Learning in Multi-Layer Spiking Networks (Sequence-to-Sequence task)
- **Problems:**
  - Spikes  $\rightarrow$  Ill-defined Derivative
  - Temporal Credit Assignment
- **Solutions:**
  - Brief Review Existing Smoothing Solutions.
  - Derivation of Bio-Plausible Surrogate Gradients Learning Rule: SuperSpike.
- Explore Capabilities of Spiking Networks with Fewer Approximations.
- Take a Brief Journey Through a Spiking Networks Optimization Landscape.
- Look at Sensitivity to Surrogate Nonlinearity,
- **Approach:** BackPropagation Through Time & Surrogate Gradients (Classification Tasks).

## Motivation

Neurobiology mostly uses spiking neural networks. Neurons output spikes, which are binary events and localized in time. So how do hidden units learn?

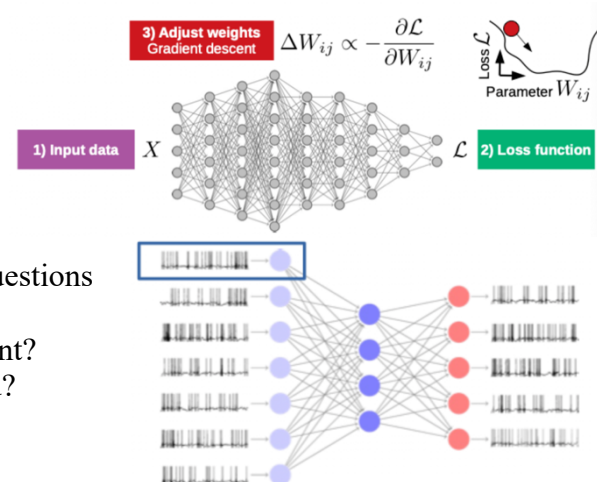
## Bottom-Up Approach:

- Start with a random network model.
- Include data driven plasticity model.
- Observe function  $\rightarrow$  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 a function in mind.
- Derive suitable plasticity rules.
- Build functional network models.



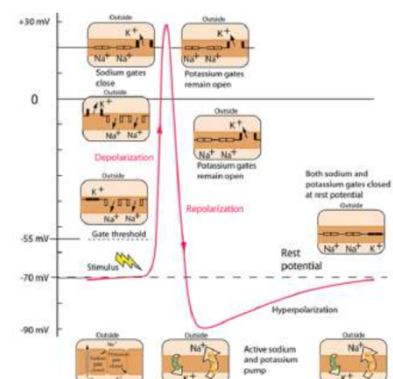
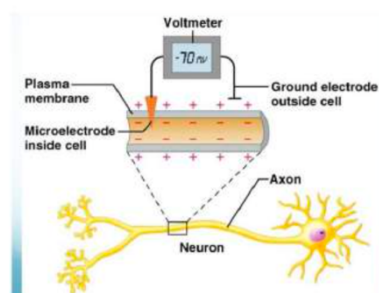
Deep Learning is an example of a top-down framework. Two questions remain:

- **The Algorithmic Question:** How to compute the gradient?
- **The Conceptual Question:** Which functions are learned?

## Recap: Spiking Neuron Models

- Spiking Networks consist of Spiking Neurons.
- Network Modelling largely relies on Simplified Neuron Models.

## Biophysics of Neuronal Signal Transmission



## From Biophysical to Reduced Neuron Models

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

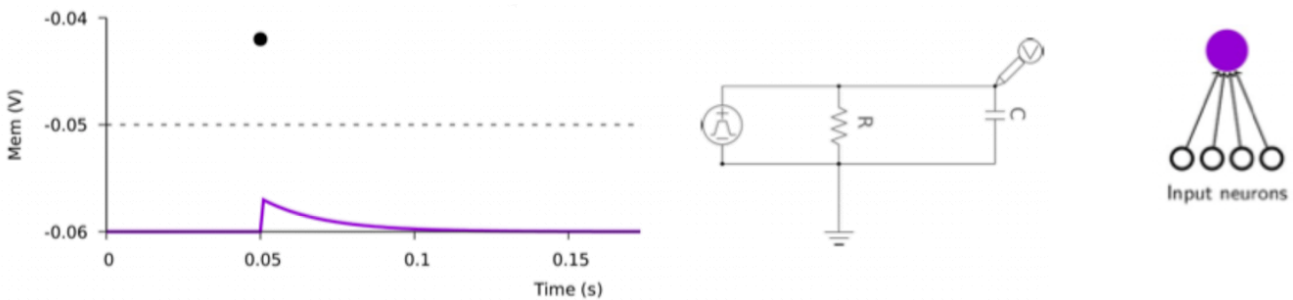
Neuron models where action potentials are described as events are called “Integrate-and-Fire” models. No attempt is made to describe the shape of an action potential. Integrate-and-Fire models have two separate components that are both necessary to define their dynamics:

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:

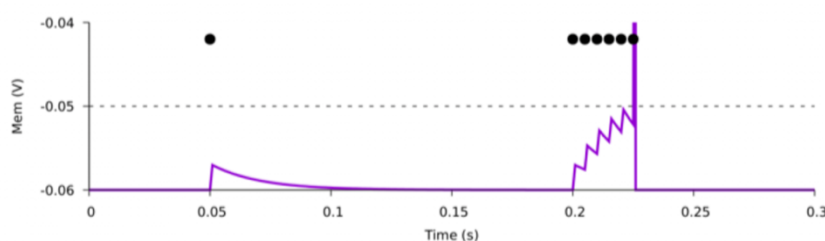
$$\tau \frac{d}{dt} U(t) = -(U(t) - U_{rest}) + R_{in} I(t)$$

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



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  $U_{rest} < \theta$ :

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



## 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 is 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  $I(t)$ :

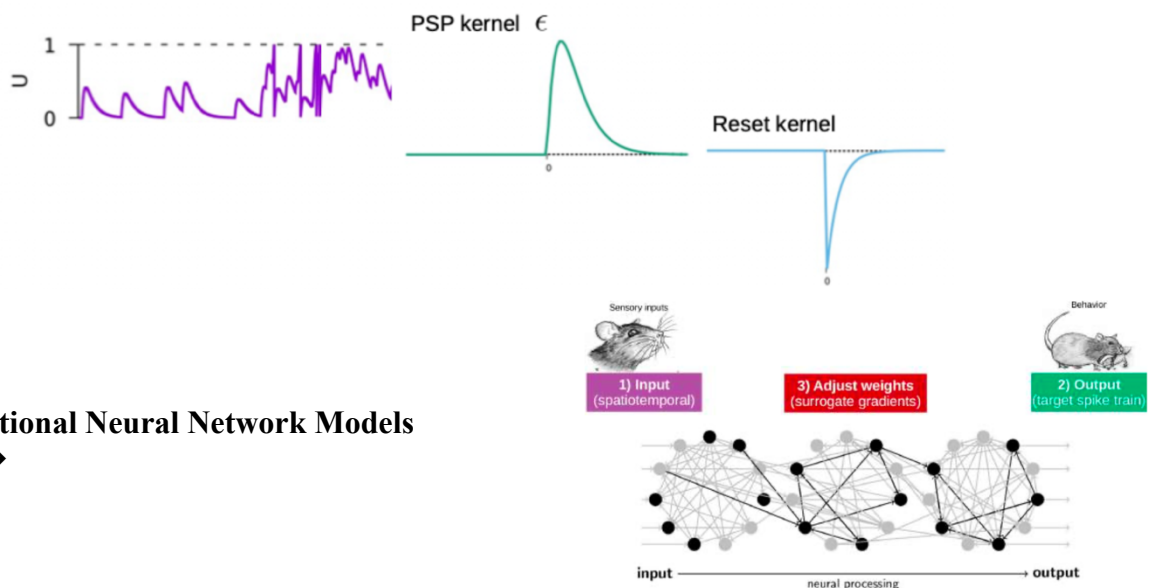
$$\frac{d}{dt}I(t) = -\frac{I(t)}{\tau_{\text{syn}}} + S(t)$$

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

## The Spike Response Model (SRM0)

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

$$U_i(t) = \sum_j w_{ij} (\epsilon * S_j)(t) + (\eta * S_i)(t) + U_{\text{rest}}$$



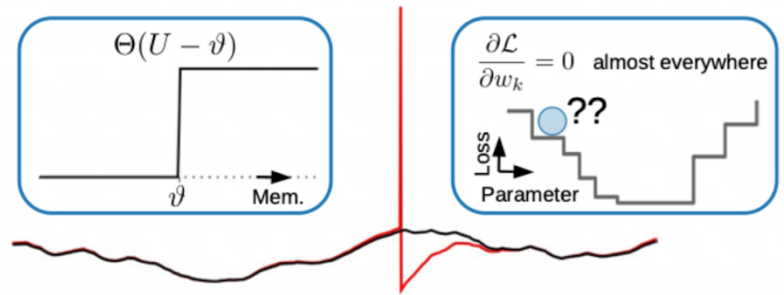
## Towards Functional Neural Network Models

We want this →

## Dealing with the Vanishing Gradient Problem

### Defining the Problem

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



### Van Rossum Distance between Output and Target Spike Trains

There are 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:

$$S(t) = \sum_i^M \delta(t - t_i)$$

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:

$$S(t) = \sum_i^M H(t - t_i) \exp -(t - t_i)/t_c$$

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

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

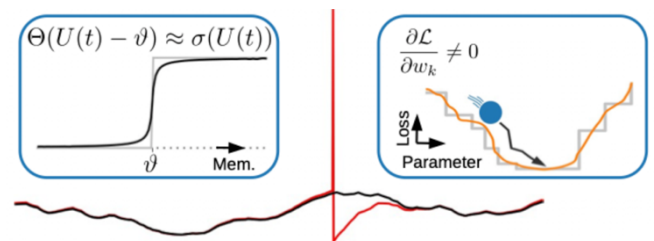
The problem with our spike model and such a loss becomes evident when we try to differentiate the loss with respect to the single weights:

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

The second partial 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.

### A History of Struggle

- Noise Injection
- Differentiate Firing Times
- Make Spikes Differentiable
- Force Hidden Units "On Target"
- Firing-Rate Approaches



### Surrogate Gradients & SuperSpike

Idea: Replace the non-differentiable Heaviside function with the differentiable sigmoid function  $\sigma$ , 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". This procedure leads to the replacements:

$$\frac{\partial S_i}{\partial w_{ij}} \rightarrow \sigma'(U_i) \frac{\partial U_i}{\partial w_{ij}}$$

If now the membrane potential  $U_i(t)$  is written in the integral form as a spike response model (SRM0)

$$U_i(t) = \sum_j w_{ij} (\epsilon * S_j)(t) + (\eta * S_i)(t) + U_{\text{rest}}$$

where  $\epsilon$  is the causal membrane kernel (corresponding to the postsynaptic potential) and  $\eta$  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:

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

Here,  $r$  is the learning rate,  $e_i(s) = \lambda * (\hat{S}_i - S_i)$  the error signal and  $\lambda$  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).  $\sigma'$  is the voltage nonlinearity, thus the learning rule is voltage based.

## Hidden Layers

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

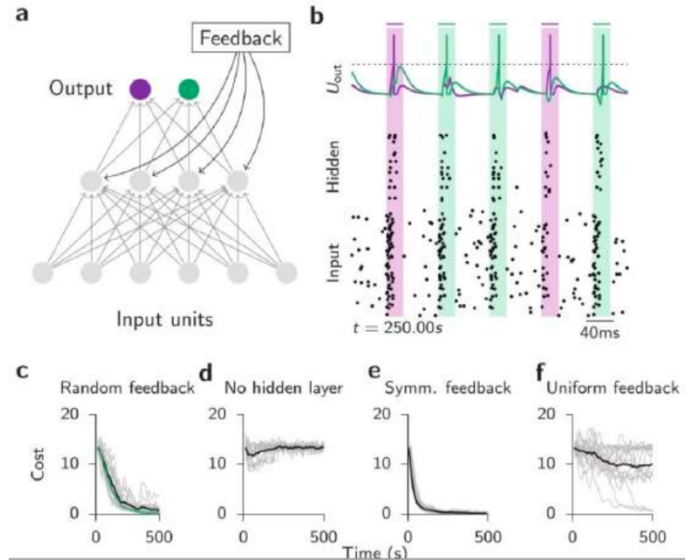
$$\frac{\partial w_{ij}}{\partial t} \equiv \sum_k e_k(t) \epsilon * [w_{ki} \epsilon * (\epsilon * S_j(t) \sigma'(U_i)) \sigma'(U_k)]$$

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. Not that all quantities computed online. Temporal credit assignment through dynamics at the synaptic level (eligibility trace).

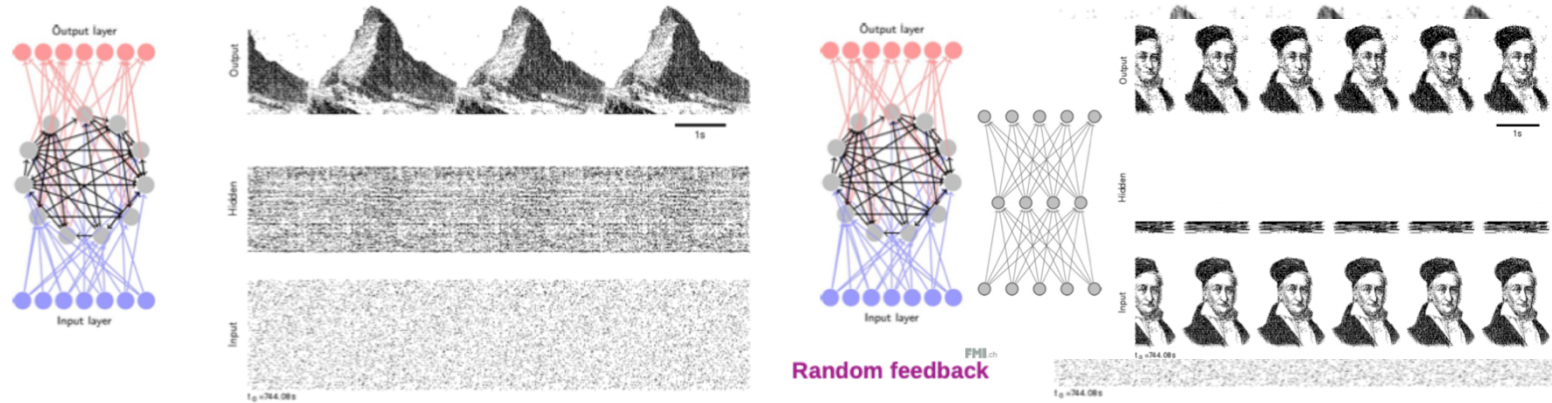
In the figure: Network trained to solve a non-linearly separable classification problem with noisy input neurons. (a) Sketch of network layout with two output units and four hidden units. (b) Snapshot of network activity at the end of training with random feedback. Four input patterns from two non-linearly separable classes are presented in random order (shaded areas). In between stimulus periods, input neurons spike randomly with 4Hz 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 is shown in green. (d) Same as panel c but for symmetric feedback. (e) Same as panel c but for uniform (“all ones”) feedback connections.





## 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 a 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.

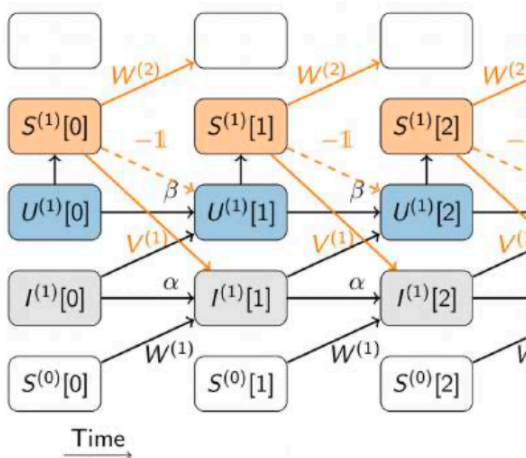


## What about Unsupervised Learning? A Spiking Auto-Encoder with “Gaussian” Input

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.

## 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 state 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 the figure below.



$$S_i^{(1)}[n] = \Theta \left( U_i^{(1)}[n] - \vartheta \right) \quad \text{Surrogate gradient}$$

$$U_i^{(1)}[n+1] = \beta U_i^{(1)}[n] + I_i^{(1)}[n] - S_i[n]$$

$$I_i^{(1)}[n+1] = \underbrace{\alpha I_i^{(1)}[n]}_{\text{exp. current decay}} + \underbrace{\sum_j W_{ij} S_j^{(0)}[n]}_{\text{feed-forward input}}$$

In the figure: 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  $\alpha$  in each time step and fed into the

membrane potentials  $U$ . The  $U$  are similarly decaying over time as characterized by  $\beta$ . 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 Backpropagation 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  $W(1)$  and recurrent weights  $V(1)$ , on which the standard BP applies:

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

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

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

Applying BP to an unrolled network is referred to as Back-Propagation Through Time (BPTT).

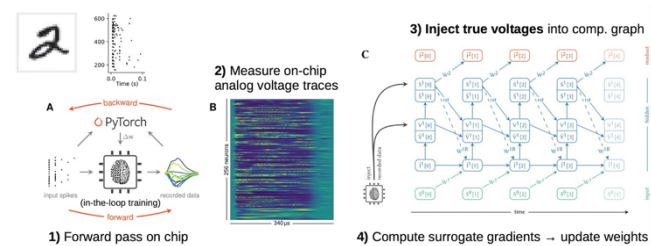
## Neuromorphic Engineers Build Hardware that Seeks to Emulate Neural Networks Instead of Simulating Them

To make things more efficient, we would like to create a hardware that does most of the simulation through physical properties rather than simulating them through software. One major issue of these hardware is related to **Device Mismatch** (a stumbling block for widespread use of analog neuromorphic hardware), i.e., a slight difference in membrane potential among chips due to manufacturing variability, which is not found in software simulations. So, can **analog neuromorphic substrates** self-calibrate through surrogate gradient learning and overcome device mismatch? To study this question we used **BrainScaleS-2** analog neuromorphic hardware system and if you record from a neuron in this chip when a current is injected you can capture the analog voltage through an oscilloscope.

### In-the-loop Surrogate Gradient Training

Forward-pass on chip and backward pass in software.

1. Forward pass on chip.
2. Measure on-chip analog voltage traces.
3. **Inject true voltages** into comp. graph.
4. Compute surrogate gradients  $\rightarrow$  update weights.



Functional spiking neural networks trained on accelerated **analog neuromorphic hardware**. We show that on the MNIST example training loss goes basically to zero. Surrogate gradient learning self-calibrates the **analog neuromorphic substrate**.

- Surrogate gradients with BPTT are a powerful tool to build functional spiking neural networks.
- Can instantiate sparsely active networks.
- Voltage-aware in-the-loop training with surrogate gradients self-calibrates neuromorphic hardware.

# Learning in Recurrent Neuronal Networks – Lecture 10 – Pau Aceituno

## Content of the Lecture

- Unstructured Recurrence
  - Circuit-level Recurrence
    - Anatomical Evidence
    - Functional Evidence
  - Training Recurrent Nets
    - Backpropagation Through Time
    - Reservoir Computing (Computation and Dynamics)
    - Biological Learning Rules (Hopfield Nets)
    - SORN
- Hierarchical Recurrence
  - Evidence in Primates
    - Processing Time as Evidence
  - LSTMs
    - Implementation
    - Functions
  - Potential Uses
    - Predictive Coding
    - Training Feedforwardness
  - Challenges

## Motivation: Why Is It So Important to Understand RNN Learning?

- The mammalian cortex is highly recurrent – it will help us to better understand the brain.
- 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.
- 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).
- RNNs perform exceptionally well in language-modelling, the task of predicting the probability of the next word in a sequence.

## RNNs in the Brain

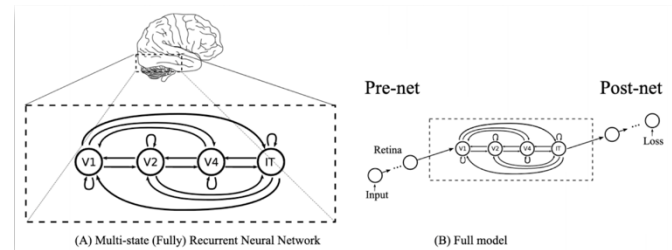
There are claims that networks of the AlexNet type successfully predict properties of neurons in visual cortex. Thus, one natural question arises: how similar is an ultra-deep residual network to the primate cortex? A notable difference is the depth. While a residual network has many as 1202 layers, 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 100Hz). The evolutionary advantage of having fewer layers is apparent: it supports rapid (100ms 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.

### Recurrent Projections in the Cat Brain

Paper: “*A Quantitative Map of the Circuit of Cat Primary Visual Cortex*”.

By mapping the circuit of cat primary visual cortex (V1), it is evident, that there are recurrent projections involved.

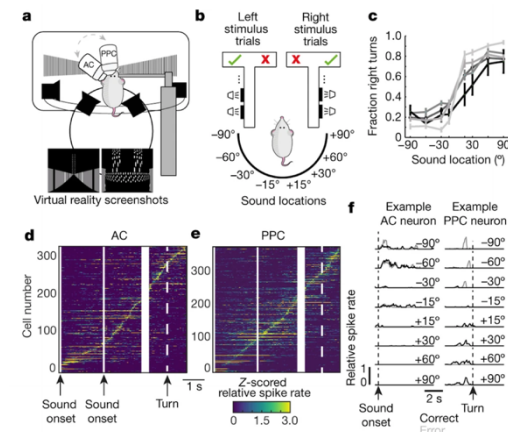


### Anatomical Evidence for RNNs in the Rodent Brain

Paper: “*Distinct Timescales of Population Coding Across Cortex*”.

(Train a mice to turn left or right depending on sound location and Record neural activity in Auditory Cortex and Posterior Parietal Cortex).

Similarly, it has been shown in rodents, that the communication between columns is organized by multiple highly specific horizontal projection patterns. **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.



### Anatomical Evidence for RNNs in the Primate Brain

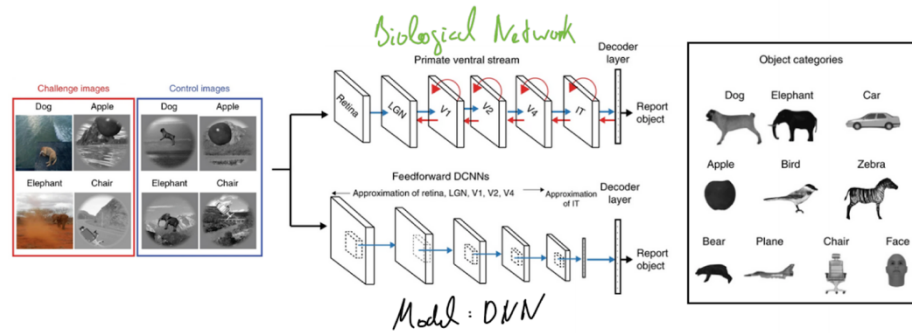
The brain has both a feed-forward structure and recurrent pathways. Information can get sent back from one area to a previous one or echo around the same area multiple times. Studies suggest this extra processing helps the brain interpret challenging visual information, such as objects that are occluded or viewed from unusual angles. A recent study 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 0.1s 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 computes 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 recurrent processing is an important contributor to computation in the visual system.

In the image below: Both primates and feedforward DCNNs were tasked to identify which object is present in each test image (1320 images). Top: the stages in the primate ventral visual pathway



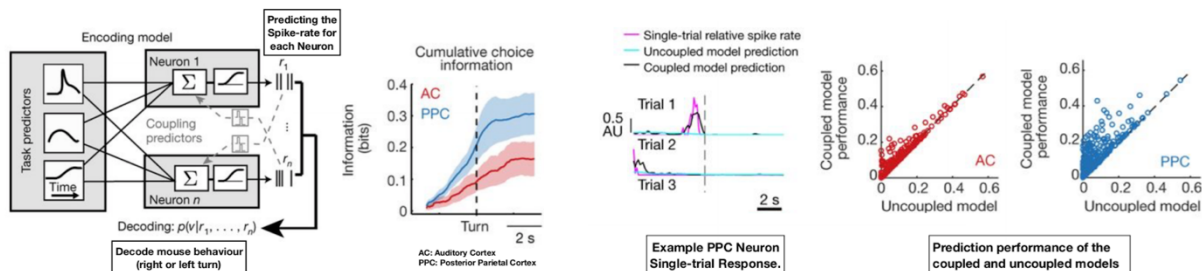
(retina, LGN, V1, V2, V4, and the IT cortex), which is implicated in core object recognition. We can conceptualize each stage as rapidly transforming the representation of the image and ultimately yielding the primates' behavior (i.e., producing a behavioral report of which object was present). The blue 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.

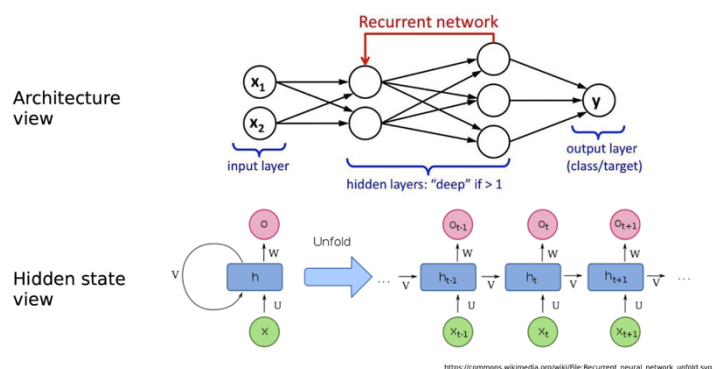
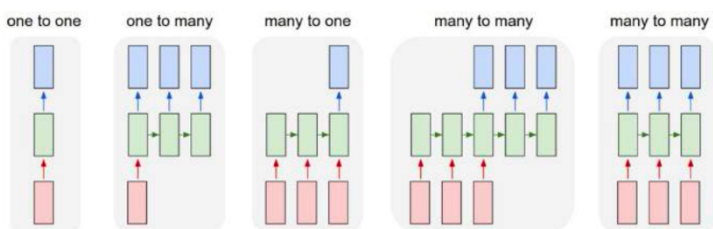


## Functional Evidence

Generate two models of neural activity incorporating any variable we can think of with a Generalized Linear Model (GLM). The predictors can be trained in isolation (**uncoupled**) or dependent on previous neuron activity (**coupled**). The Coupled model performs much better for PPC, ergo we assume the recurrence is important. For AC both perform similarly, but AC is less recurrent than PPC.



## RNNs in Machine Learning



Main advantage: Memory

In the left figure: 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. 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

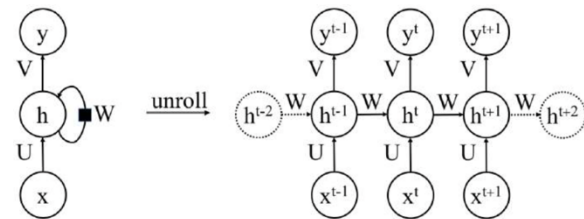


sequences because the recurrent transformation (green) is fixed and can be applied as many times as we like.

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, are shown in the previous figure. 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 chosen 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:

$$\begin{aligned} y(t) &= g_y(W_y h(t) + b_y) \\ h(t) &= g_h(W_x x(t) + W_h h(t-1) + b_h) \\ &= g_h((W_{hh} W_{xh}) \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}) \\ &= g_h(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}), \end{aligned}$$

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 to the right.



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 over all times steps (Not to be confused with all epochs).

### Back-Propagation Through Time (BPTT)

The unfolding shown in the figure above 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 algorithm to the unrolled network to calculate all weight updates.

There are several drawbacks to BPTT:

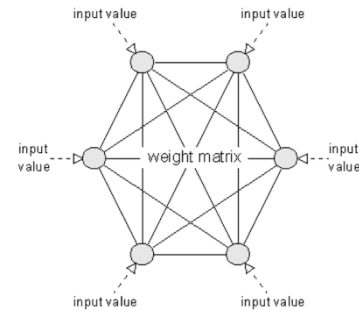
- **Costly Parameter Update:** Especially for long sequences, the parameter update for a shallow layer is the same as updating a parameter in an extremely deep feedforward network. One way to fix this is using a truncated BPTT algorithm. It processes the sequence one timestep at a time, and every  $k1$  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.
- **Exploding Gradients:** The gradients coming from the deeper layers have to go through continuous matrix multiplications because of the chain rule, and as they approach the earlier layers. If they have large value ( $>1$ ) 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.

- **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.

## RNNs in Theoretical Neuroscience

### Hopfield Network

A Hopfield network is a form of recurrent artificial neural network popularized by John Hopfield in 1982, 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).



A Hopfield network has various units, which have a binary state (1/0). The units update asynchronously or synchronously with the following rule:

$$S_i = +1 \quad \text{if} \quad x_i = \sum_{j \neq i} w_{ij} \cdot S_j > \theta_i$$

$$S_i = -1 \quad \text{if} \quad x_i = \sum_{j \neq i} w_{ij} \cdot S_j < \theta_i$$

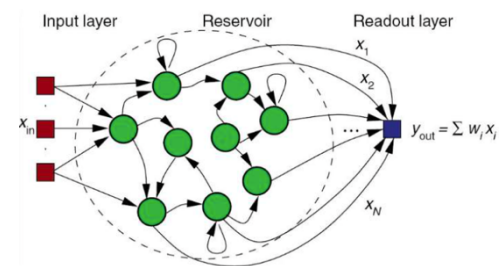
Here,  $S_i$  is the  $i$ -th unit of the Hopfield network and  $\theta_i$  is the threshold. One can define an energy term as:

$$E = -\frac{1}{2} \sum_{i,j} w_{ij} S_i S_j + \sum_i \theta_i S_i$$

With each update step, the energy either stays constant or decreases.

### 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 an 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. A really cool thought is, that basically every (abstract or physical) dynamical system can be used as the reservoir, including a water tank, an electronic circuit or parts of the brain itself.

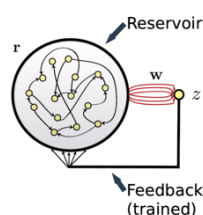


- **Echo State Network:** Recurrent neural network with a random and sparsely connected (1%) hidden layer / reservoir, works in discrete time, different activity update modes.
- **Liquid State Machine:** Biologically plausible, spiking RNN reservoir, randomly connected, continuous in time, asynchronous integration, uses linear discriminate units.

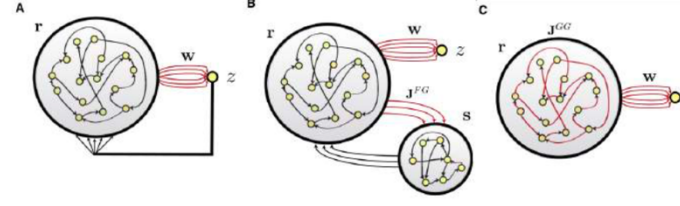
Where is the memory? In the dynamic traces of activity.

### Learning Dynamics/Algorithms

- Learning via Backpropagation Through Time (**BPTT**)
- Learning via Real Time Recurrent Learning (**RTRL**)
- **Reservoir Computing:** reservoir capture features of the dynamics  $\rightarrow$  can be used to generate them.
- **FORCE:** First-Order, Reduced and Controlled Error (FORCE) learning. In all three cases shown in the following figure, a recurrent generator network with firing rates  $\mathbf{r}$  drives a linear readout unit with output  $z$  through weights  $\mathbf{w}$  (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 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, 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 internal to the network.



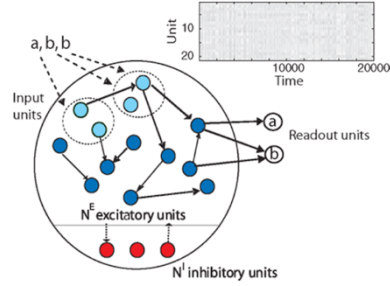
### Self-Organizing Recurrent Networks (SORN)

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. The SORN learns to encode information in the form of trajectories through its high-dimensional state space reminiscent of recent biological finding 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.

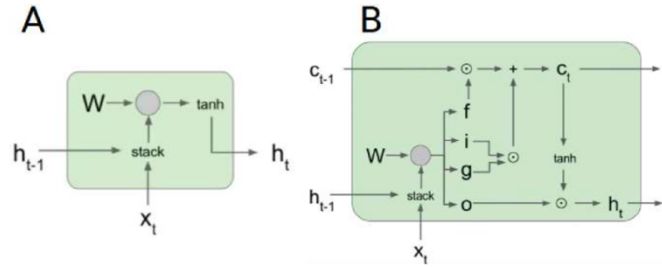
Process:

1. Start with random network.
2. Add biological-like learning rules.
3. Train readout layer linearly.



### 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. A common LSTM unit is composed of a cell, an input gate **i** (whether to write to cell), an output gate **o** (how much to reveal cell), a forget gate **f** (whether to erase cell) and a gate gate **g** (how much to write cell). 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 (a comparison between a normal RNN cell A and a LSTM cell B).



The gate vector can be written as:

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

where  $W = (W_i, W_f, W_o, W_g)$ . The cell state is defined as the following:

$$c_t = f \odot c_{t-1} + i \odot g$$

$$c_t = \sigma(W_{hf}h_{t-1} + W_{xf}x_t) \odot c_{t-1} + \sigma(W_{hi}h_{t-1} + W_{xi}x_t) \odot \tanh(W_{hg}h_{t-1} + W_{xg}x_t)$$

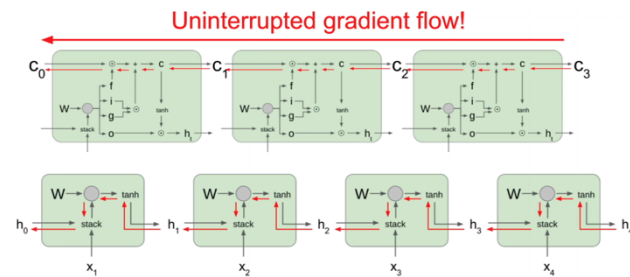
And the hidden state is a function of the cell state:

$$\begin{aligned} h_t &= o \odot \tanh(c_t) \\ &= \sigma(W_{ho}h_{t-1} + W_{xo}x_t) \odot \tanh(c_t) \end{aligned}$$

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 shown in the following figure. 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.

In the figure: Illustration of LSTM over many sequences. Red arrow denotes the gradient, which can flow uninterruptedly.



## Biological Plausibility

- **BackPropagationn Through Time**
  - The brain does not perform BPTT that we know of.
  - But BPTT can be used to train networks that behave like the brain.
- **Is Reservoir Computing Better?**
  - No one things the brain is as simple as RC...
  - But useful simple model.

## Across-Layer Recurrence for Learning

Multiple implementation of learning algorithms (usually backprop through time) require **feedback from higher layers**.

- Through symmetric connections.
- By differentiating inputs and errors.
- Reusing the Predictive Coding setting.
- Many more...

## Challenges

- Temporal Credit Assignment in Biological RNNs.
- Structural Priors for Better Computation.
- Bottom-up vs. Top-down Information.
- Integrating Multiple Timescales.
- Finding and Understanding Learning Rules.

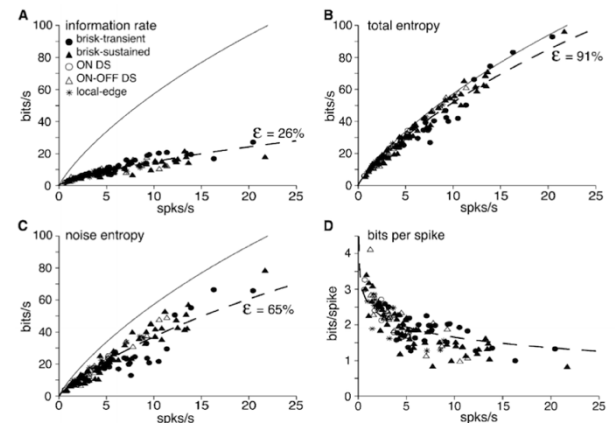
## Recap

- The Brain uses recurrent connections
  - Anatomically, recurrence is widely visible
  - Functionally, recurrent models capture neural activity better.
- Backpropagation Through Time is difficult
  - Vanishing/Exploding gradients, and temporal credit assignment.
- RNNs keep memory in their dynamics
  - Relationship memory-stability and edge of chaos.
- Biological learning rules can be used in RNNs
  - Hopfield Networks learn associations through Hebbian Learning
  - SORN learns sequences through STDP + Homeostasis.
- Recurrent btw layers can improve inference
  - Predictive Coding or by layer unrolling.
- Higher brain areas are more recurrent
  - Evidence: takes longer to process challenging tasks.
- LSTMs work as RNNs
  - Bypass the backprop problems with specialized memory units
  - More interpretable than standard RNNs.
- Recurrence across layers can be leveraged for learning.

# Predictive Coding – Lecture 11 – Pau Aceituno

## Content of the Lecture

- **Coding**
  - Principle
  - Evidence
- **Temporal Predictions**
  - Principle
  - Effects & Evidence
  - Artificial Networks
- **Predictive Coding**
  - Circuits
  - Learning
- **Problems with Predictive Coding**
  - Interpretation
  - Data
- **Bayesian Brain**
  - Main Ideas
  - Effects
  - Formalism & Free Energy



## Compression: Description Principle

- 4K display:  $3 \times 10 \text{ bit/pixel} \times 40000 \text{ pixel} \times (3840 \times 2160) \times 48 \text{ frame/s} = 5 \times 10^{14}$ .
- 4K compressed video: 44 – 85 Mbps (4K)
- Optic Nerve: <1 Mbps (guinea pig), hence there is compression going on.

## Estimation of information:

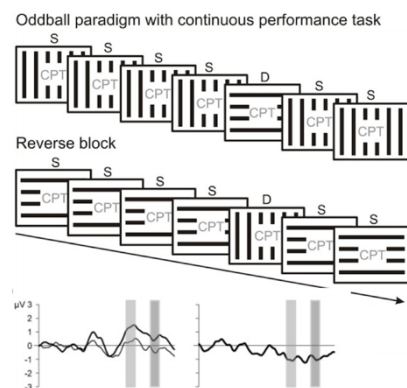
- How many spikes? Depends on neuron type.
- How many bits per spike? More spikes, less informative. (The rarer an event it is the more information it conveys).
- How much noise? By reproducibility.

## Temporal Compression: Engineering

The standard video-compression algorithms only send the unexpected information, i.e., the change in pixels within an image, rather than the full matrix of pixels composing an image.

## Temporal Compression: Mismatch Negativity (MMN)

(Oddball) Paradigm where you show a bunch of images with vertical lines and then one horizontal and viceversa, such image breaks the predictability pattern which elicits a strong response in the EEG.



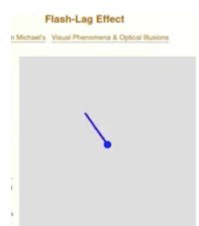
## Temporal Compression: Mismatch in Health

Such responses are studied by clinicians to understand diseases.

- **Dyslexia**: diminished MMN in auditory stimuli.
- **Alcoholism**: Decreased MMN response, but enhanced after rehab.
- **Schizophrenia**: Shorter MMN durations before psychotic episodes.
- **Autism**: Prolonged/reduced MMN latencies.

## Compression: Time Sequences Illusion

Time can be used to recognize a recurrence, complex images require more time to be processed than straightforward ones. E.g., Flash-Lag Effect cannot be predicted by the brain



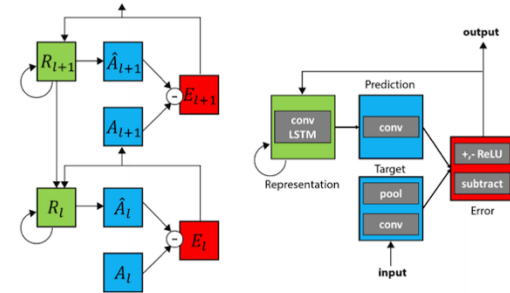


so it doesn't look collinear, also tennis player cannot be seeing the ball and must be predicting the trajectory.

### Compression: Application to RNNs (Deep Predictive Coding: Pred-Net)

Training RNNs to predict sequences automatically enforces “good” representations. One of the main problem of DL is that it requires a lot of labelled images, so they trained a network and a component that computes a predictive error. Their models are trained on minimizing such prediction error. They are compressing in time.

This network consists of a series of repeating stacked modules that attempt to make local predictions of the input to the module, which is then subtracted from the actual input and passed along to the next layer. In the figure: Left: Illustration of information flow within two layers. Each layer consists of representation neurons ( $R_l$ ), which output a layer-specific prediction at each time step ( $\hat{A}_l$ ), which is compared against a target ( $A_l$ ) to produce an error term ( $E_l$ ), which is then propagated laterally and vertically in the network. Right: Module operations for case of video sequences.



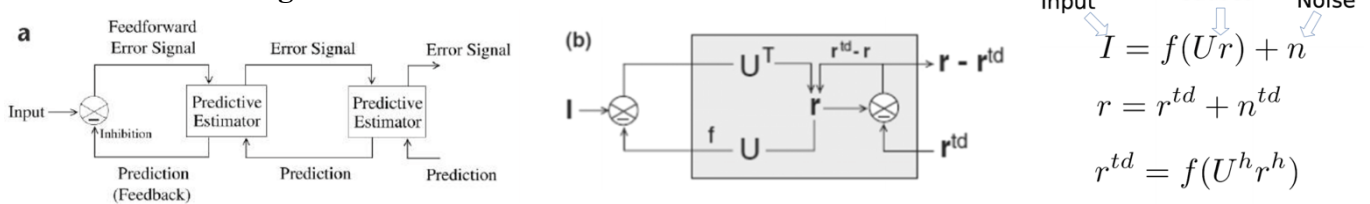
### Predictive Coding

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.

### Predictive Coding: Static

Identifying a letter in word that you already know allows you to be faster and more accurate, as you are exploiting predictive capabilities. So, if we have a “high-level” description (prior knowledge) of an object, we are better in describing it.

### Predictive Coding: Circuit



The left figure represents the inhibiting feedback prediction mechanism in the visual cortex. Rao & Ballard 1999 paper represents the foundational paper in predictive coding. They built this simple circuit to explain some of the effects involved in prediction and integration of top-down knowledge and bottom-up sensory stimuli. If your predictions correctly matches the input, the inhibitory connections make sure that they cancel out.

Handwritten notes from the Rao & Ballard 1999 paper. The notes include the following equations and text:

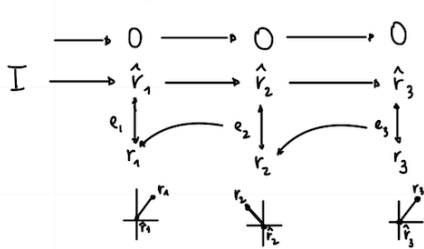
- $I = f(w_r) + n$  (causal, noise, top-down)
- $r = r^{td} + n^{td}$ ,  $r^{td} = f(w^h, r^h)$  (hierarchical)
- $n = \mathcal{N}(0, \sigma^2)$ ,  $n^{td} = \mathcal{N}(0, \sigma_d^2)$
- Energy or Difference between Input and Prediction has to be minimized E
- $E = \frac{1}{2} [I - f(w_r)]^T [I - f(w_r)] + \frac{1}{2} (r - r^{td})^T (r - r^{td})$
- $E = -\log(I/r)$
- $E = E + g(r) + h(w)$  (prior, r, prior, w)
- $g(r) = \sum_i \log(1 + r_i^2)$
- $\frac{dE}{dw} = -\frac{1}{2} \frac{\partial E}{\partial w} = \frac{1}{2} \frac{\partial}{\partial w} [I - f(w_r)]^T [I - f(w_r)] + \frac{1}{2} \frac{\partial}{\partial w} (r - r^{td})^T (r - r^{td})$
- $\frac{dE}{dw} = \frac{1}{2} \frac{\partial E}{\partial w} = \frac{1}{2} \frac{\partial}{\partial w} [I - f(w_r)]^T [I - f(w_r)] + \frac{1}{2} \frac{\partial}{\partial w} (r - r^{td})^T (r - r^{td})$
- $\frac{dE}{dw} = \frac{1}{2} \frac{\partial E}{\partial w} = \frac{1}{2} \frac{\partial}{\partial w} [I - f(w_r)]^T [I - f(w_r)] + \frac{1}{2} \frac{\partial}{\partial w} (r - r^{td})^T (r - r^{td})$

## Predictive Coding: Effects

Predictive Coding can be used to denoise images, which is similar to what can be done with an Autoencoder.

## Predictive Coding: Supervised Learning

(turning) Predictive Coding  $\xrightarrow{\text{into}}$  Supervised learning



Where:

$\hat{r}_i$  = estimated / FF

$r_i$  = "truth" (label)

$e_i$  = error

$$\frac{dr_i}{dt} = e_i - \sum_{j \in C(i)} e_j \frac{\partial \hat{r}_j}{\partial r_i}$$

change of coordinates

$$\frac{dW_i}{dJ} = e_i \cdot \frac{\partial \hat{r}_i}{\partial W_i}$$

layer-by-layer  $\frac{dW_L}{dJ} = (r_L - \hat{r}_L) \cdot \frac{\partial \hat{r}_L}{\partial W_L}$

$$\frac{dW_{L-1}}{dJ} = e_L^* \cdot \frac{\partial \hat{r}_L}{\partial W_{L-1}} = e_L^* \cdot \frac{\partial \hat{r}_L}{\partial \hat{r}_{L-1}} \cdot \frac{\partial \hat{r}_{L-1}}{\partial W_{L-1}}$$

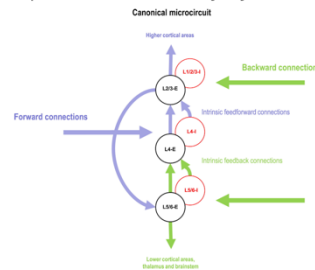
Back Propagation

"Instead of correcting the error in one layer, you correct it a bit in every layer."

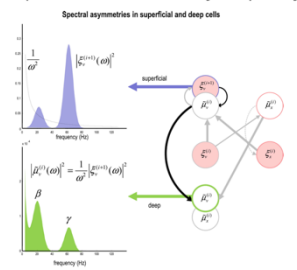
## Predictive Coding in Biology: Circuits

How can I take the Rao & Ballard circuit and map it into something in the cortex, knowing the connections between layers. The Thalamus fits into L4, then L4 fits into L2/3 and so forth... So, we can think of layer 4 as the "X" in the Rao & Ballard circuit that integrates from L5/6 and from the FF connection. L2/3 is the most recurrent part of the brain, which takes more time.

Split information flow by layer



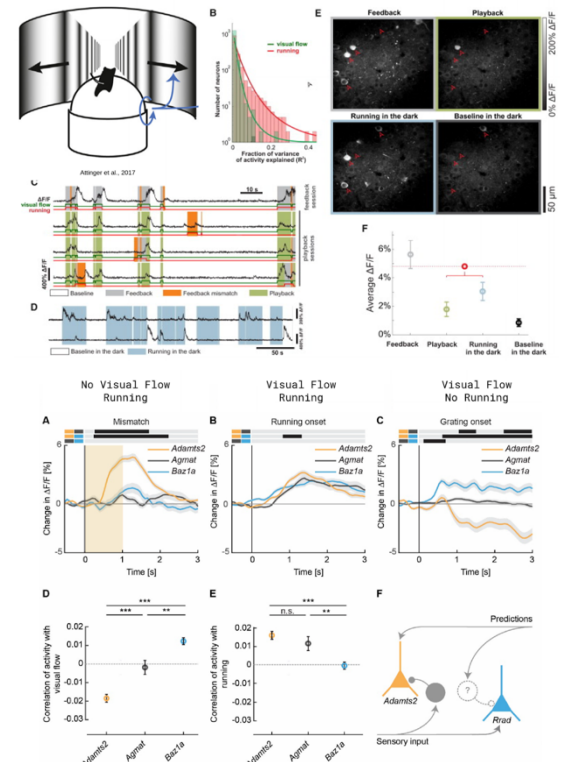
Split information flow by frequency



## Predictive Coding in Biology: Experiment

They took a mice and made him run through a VR setup where walls show lines (grates), they can adjust the lines to make the mice think he is running or keep them still so they move only when the mice is effectively running (mismatch), or also to make them move such that they look still even when the mice is running.

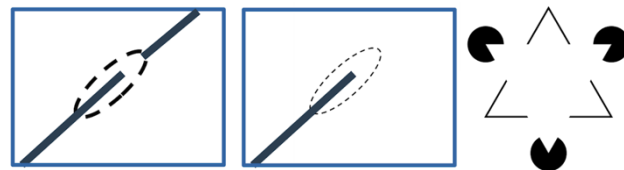
They put some markers in neurons, which allowed the identification of different types of neurons: the orange neuron seems to fire when there is no visual flow but the mice runs and, when "things" match, it shows a lower response. They also identified a neuron that fires in the presence of visual flow but no running. Then, they draw correlation plots which show that the orange neuron correlate negatively with visual flow, the black and grey neurons don't correlate with the visual flow, while the blue correlates positively. They proposed a circuit where you have sensory inputs (visual flow), predictions (running). And they evidence some predictions happening as the predictive coding circuit postulates.



## Predictive Coding in Biology: Problems

Problems:

- Errors, Representations and Interpretations
- Learning and Mismatch Negativity



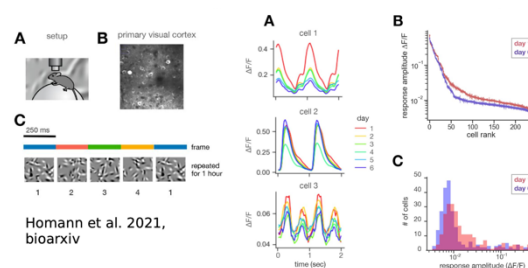
### Errors, Representations and Interpretations

Suppose you have a line and then you interrupt it, you would expect in Predictive Coding that the end of the line is an error. But it is difficult to disentangle between a neuron that signals an error and a neuron that signals the end of a line. Is the firing of the neuron a representation of the end of the line or is it a firing in response to an error in prediction?

**Kanizsa illusory triangle.** Neurons with receptive fields fire at the illusory lines. Are they perception neurons or error neurons? The triangle is not there, so the triangle that we perceive is the result of prediction errors or perception? It is hard to make a proper interpretation.

### Learning and Mismatch Negativity

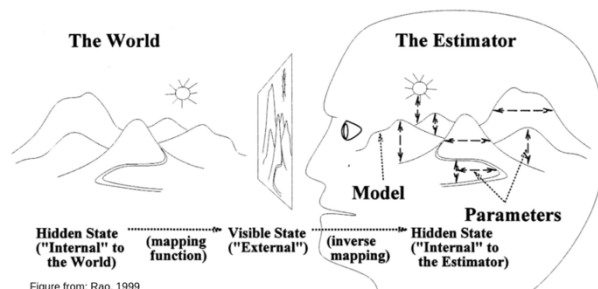
When you make an error in prediction, neurons fire to signal the error. In PC, we would expect that the most active neurons get suppressed over time. Because as we learn the mismatch negativity firing activity should go down as the predictions get better. The weird thing is that the activity does not go down in the most active neurons, it goes down in the neurons that are only slightly active.



### Bayesian Brain: Perception as Inference

The main idea behind the Bayesian brain is that we use prior knowledge to infer properties that are not explicitly shown by observation.

- Depth can be inferred from prior knowledge/an existing model.



### Bayesian Brain: Clinical Explanations

- Autistic people are “bombarded” by external stimuli.
- Schizophrenic people are “bombarded” by internal models. During psychotic episodes, schizophrenic people cannot distinguish between what’s reality and what’s in their mind.

### Free Energy Principle

Latent variable  $P(z|x)$  / Data/sensation  $Q(z)$  → Approximated by  $Q(z)$  (Tractable distribution of the variables)

By Bayes Theorem:  $P(z|x) = \frac{P(x|z)P(z)}{\int P(x|z')P(z')dz'}$

In order to do so, we minimize the KL divergence  $D_{KL}(Q||P)$  which is:  $D_{KL}(Q||P) = \int Q(z) \log \frac{Q(z)}{P(z|x)}$

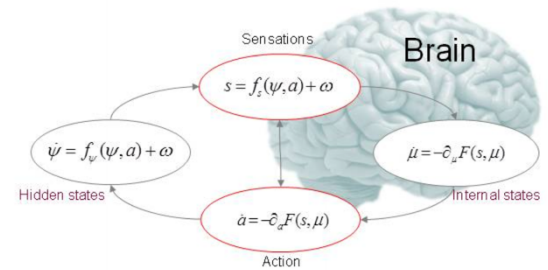
So, if we maximize  $\mathcal{L}(Q)$  or equivalently Minimize  $D_{KL}(Q||P)$

Then has been named free energy:  $F[(s, a, \mu)]$

By Bayes Theorem:  $\log P(x) = D_{KL}(Q||P) + \mathcal{L}(Q)$

We care about what we see in the external world. We cannot do variational inference in the “true world”, i.e., hidden states. In order to do that, we can use actions, sensations and internal states. “Animals want to reduce their uncertainty about the world”. “If I felt something on my back, I can update my internal model through sensation, but I can also turn around (action) which changes my sensation (from which the connection between sensations and actions). Therefore, I have a dual optimization goal, one is that in my internal model I should change the parameters to fit the best

data I have, but I cannot only passively receive sensations and update the model. I can also choose actions that would eventually lead me to have better data and a better model.” Actions are part of inference if you have an agent. This branch of research claimed that this would explain everything from cells to brains and minds. There are however a few problems: If I hear something on my back and I want to remove my uncertainty I can just turn around, but I can also do something else. If I am in a dark room there is no Free Energy.



“You want to maximize your surprise temporarily to have a better model that minimizes your surprise overall”. From a neuroscientific perspective, such theory is reductionist in saying that all we want to do is minimizing uncertainty.

## Recap

- Mismatch Negativity
  - Experiments and Measurements
  - Relationship to Compression.
- Rao & Ballard
  - Abstract Circuit
  - Dynamics and Learning (Math)
  - What is it good for: Denoising, Supervised Learning.
- Rao & Ballard in Biology
  - Receptive Fields
  - Problem of Interpretability: Perception vs Error
  - Identify Circuits that can correspond to Predictive Coding.
- Bayesian Brain
  - Understand the Idea
  - Relationship with Free Energy.

## Neuromorphic Intelligence – Lecture 12 – Giacomo Indiveri

### Artificial vs Neuromorphic Intelligence

- **Artificial Neural Networks**
  - Simulate abstract brain-inspired computing architectures on digital time-multiplexed computing substrates.
- **Neuromorphic Architectures**
  - Use the physics of electronic devices and circuits to **emulate** real neurons. They use physical (real) time and circuit dynamics to compute through their time evolution.
  - The physical hardware substrate **is** the algorithm.

### Synapses vs Neurons

- **In Biology**
  - Pyramidal neurons have thousands of synapses, mainly distributed on the distal dendritic arborization.
  - The excitatory drive from 10-40 inputs, discharging at an average rate of 100 spikes/s, should cause the postsynaptic neuron to discharge near 100 spikes/s.
- **In Neurons**
  - To first order approximation, a neuron has a **gain of less than 1/10**.
  - To produce one output spike, at least 10 input spikes should arrive in parallel to 10 different synapses.

- An input spike train arriving onto a single synapse at 1kHz should produce one output spike train at 100Hz.

### Dale's Principle

In 1935, Sir Henry Dale hypothesized that a neuron extends its metabolic activity from the soma to all of its processes. In 1954, Sir John Eccles reformulated this principle to say that “neurons release the same set of transmitters at all of their synapses”.

**Dale's Law:** In practice Dale's law requires neurons to be **only excitatory** or **only inhibitory**. A neuron cannot excite some targets and inhibit others. All projections of excitatory neurons can have only positive synaptic weights. All projections of inhibitory neurons can only have negative weights.

### Dendrites

The active electrical properties of dendrites shape neuronal input and output and are fundamental to brain function. They can implement analog signal processing, digital logic, and state dependent computations. These properties enable neocortical pyramidal neurons to classify linearly non-separable inputs – a computation conventionally thought to require multilayered networks. Recently it has even been hypothesized that there are multiple “Dendritic solutions to the credit assignment problem”.

### The Neocortex

The neocortex is the newest part of the cerebral cortex to evolve. It is a distinguishing feature of mammals. In humans, it is 90% of the cerebral cortex and 76% of the entire brain.

### The Canonical Micro-Circuit

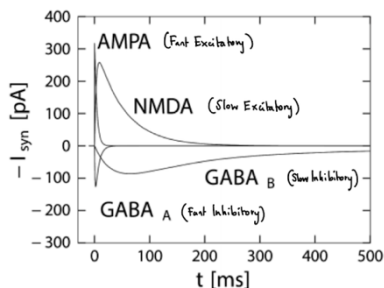
#### Across all areas of Neocortex

- Most cortical connections are local and excitatory.
- Most cortical excitatory synapses and almost all inhibitory ones originate from cortical neurons.
- Most of these synapses originate from neurons within the local cortical area.

#### Cortical Computation

- It is carried out almost entirely by local circuits.
- Recurrent circuits perform signal restoration.
- Control of **gain** is critical for cortical computation.

### Model of Synapses



#### First order approximation

$$\tau \frac{d}{dt} I_{syn}(t) + I_{syn}(t) = I_w \delta(t_{pre})$$

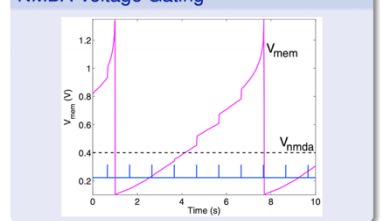
#### Alpha function approximation

$$\begin{aligned} \tau_{pos} \frac{d}{dt} I_{pos}(t) + I_{pos}(t) &= I_w \delta(t_{pre}) \\ \tau_{neg} \frac{d}{dt} I_{neg}(t) + I_{neg}(t) &= I_w \delta(t_{pre}) \\ I_{syn}(t) &= I_{pos}(t) - I_{neg}(t) \\ \text{with } \tau_{pos} < \tau_{neg} \end{aligned}$$

#### Synapse types

| Receptor          | Function                                            | $\approx \tau$ |
|-------------------|-----------------------------------------------------|----------------|
| AMPA              | Fast excitatory                                     | 5 ms           |
| NMDA              | Slow excitatory, with voltage gating mechanism      | 100 ms         |
| GABA <sub>B</sub> | Slow inhibitory                                     | 200 ms         |
| GABA <sub>A</sub> | Fast inhibitory, with shunting (divisive) mechanism | 10 ms          |

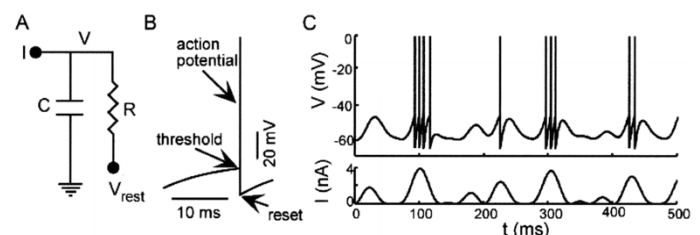
#### NMDA Voltage Gating



### The First Models of Neurons

- **Lapicque: The Integrate and Fire Model**

$$\begin{aligned} I(t) &= C \frac{d}{dt} V_m(t) \\ V_m(t) &= V_{rest} \text{ if } V_m \geq V_{th} \end{aligned}$$





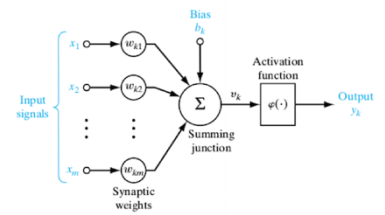
## - McCulloch & Pitts Neuron

“A logical calculus of the ideas immanent in nervous activity”.

Linear threshold units, binary inputs/outputs

$$y_k = \phi\left(\sum_{j=1}^N w_{kj} x_j\right)$$

where  $y_k$  is the  $k$ -th output unit,  $x_j$  is the  $j$ -th component of the input vector, and  $w_{kj}$  is the weight between them.



## - Hodgkin & Huxley Model

- Cell Lipid Bilayer (Membrane): Capacitance/Area
- Leak Channels: Linear Conductance
- Voltage-Gated Ion Channels: Conductance (V,t), voltage-dependent open-close probability.

HH equations

For each ionic current:

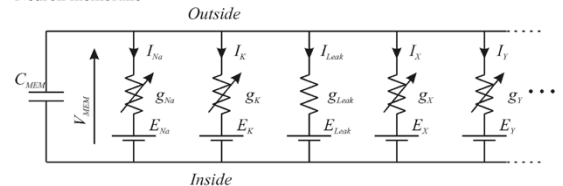
$$C_{mem} \frac{dV_{mem}}{dt} + \sum I_i + I_{syn} + I_{leak} = 0$$

$$I_i = g_i \cdot m^p \cdot h^q$$

$$\tau_m \frac{dm}{dt} = m_{\infty} - m$$

$$m_{\infty} = \frac{1}{1 + e^{\frac{offset_m - V_{mem}}{perim_m}}}$$

Neuron membrane



## Models of Neurons in Neural Networks

The McCulloch&Pitts model quickly became extremely popular, and dominated the Artificial Neural Network scene for decades. Why? **Isomorphism with calculus of logical propositions.**

In the hands of John Von Neumann, the McCulloch&Pitts model became the basis for the logical design of digital computers.

| Neuron model    | Representation  | Output  |
|-----------------|-----------------|---------|
| LaPicque        | Continuous time | Spiking |
| Rashevsky       | Continuous time | Binary  |
| McCulloch&Pitts | Discrete time   | Binary  |

## The Deep Network Revolution

Although the first successes of ANNs were first demonstrated in the 1980's they only started to outperform classical optimization and engineering approaches from 2009 on. In 2011, CNNs trained using **backpropagation** on GPUs achieved for the first time superhuman performance in a visual pattern recognition contest.

## Deep Networks Galore

- As CNNs and DNNs outperformed classical approaches, many research groups started to extend and optimize them.
- The AI field is now (mostly) dominated by attempts to improve accuracy on standard benchmarks, by scaling up network size and parameter count.
- GPT-3 is a network with 175-billion parameters. It's memory size is exceeding 350GB, and training it requires an estimated \$12 million.

## Problems and Limitations of AI

### Energy Intensive

At this pace, by 2025 the ICT industry will consume 20% of the entire world's electricity

[International Renewable Energy Agency, Internet of Things innovation landscape brief]

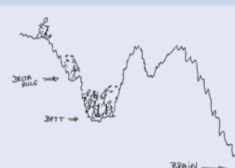
### High cost of data movement

DRAM access is at least 1500x more costly than a MAC operation in CNN accelerators. [Tu et al., 2018]

### Brittle and narrow AI

DNNs programmed to perform a limited set of tasks. They operate within a pre-determined, pre-defined range. [medium.com]

### Algorithmic limitations



[I am] deeply suspicious of **back-propagation**. I don't think it's how the brain works. The future depends on some graduate student who is deeply suspicious of everything I have said. [Geoff Hinton]

## **The Neuromorphic Engineering Approach**

### **Methods**

- Study **fundamental neuroscience**, principles of neuro-physiology, neuro-anatomy, theory of computation, electrical engineering, ...
- Exploit the physics of silicon to reproduce the bio-physics of neural systems, using **subthreshold analog** VLSI circuits.
- Develop distributed multi-core spiking architectures using **asynchronous digital** VLSI circuits.
- Build **real-time autonomous cognitive agents** able to carry out behavioral tasks in complex environment.

### **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<sup>3</sup> and manages to squeeze almost 1 million neurons in it. The energy per operation is approximated to 10–15J/spike. Functional principles used by these life-forms can be used in the development of neuromorphic agents. Some of these are:

- Exploiting physical space in the most efficient way possible.
- Let time represent itself.
- Use both analog and digital computing elements.
- Exploit non-linearities and temporal dynamics.
- Leverage noise, variability and stochasticity.
- Minimize wiring (maximize local connectivity).
- Maximize fine grain parallelism (no time-multiplexing).
- Co-localize memory and computation.
- Use physical time (no clocks).
- Match dynamics and time-constants.
- Optimize for processing complex spatio-temporal (dynamic and noisy) signals.
- Re-use computational principles for sensory processing, motor control and cognitive computing.

## **Brain-Inspired Computing: A Radical Paradigm Shift**

### **Exploit Physical Space**

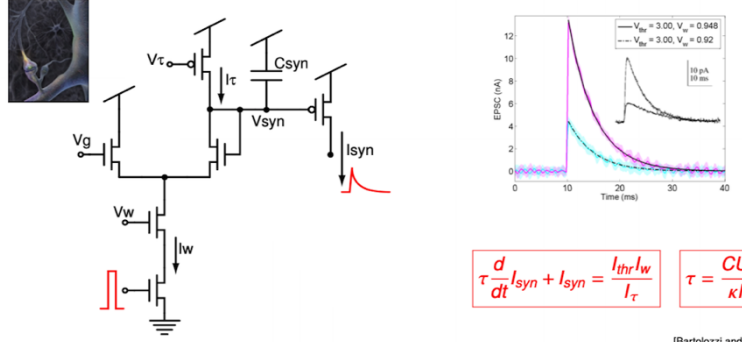
- Use parallel arrays of processing elements.
- Maximize fine grain parallelism (no time-multiplexing).
- To co-localize memory and computation.
- Use passive circuits, data-drive, in continuous time.
- Exploit all the properties of transistors and memristors.

### **Let Time Represent Itself**

- For interacting with the environment in real-time.
- To match the circuit time constants to the input signal dynamics.
- For inherently synchronizing with the real-world natural events.
- To process sensory signals efficiently.

## Synapse Analog Circuits

The DPI is a CMOS current-mode circuit that operates in the subthreshold regime integrating voltage pulses. However, rather than using a single p-FET to generate the appropriate  $I_w$  current, via the triangular 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 the picture below.



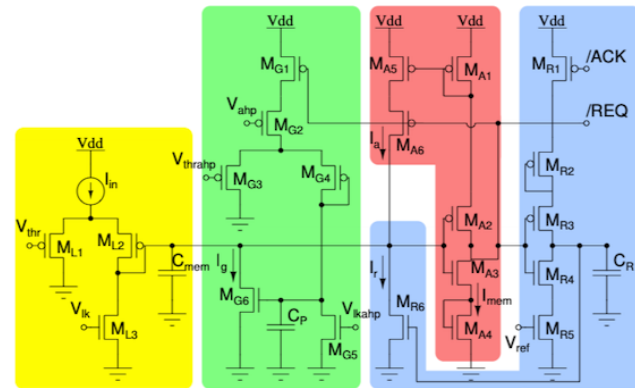
[Bartolozzi and Indiveri, 2007]

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. 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) and on larger 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 the following picture. 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:

$$\tau \frac{d}{dt} I_{mem} + I_{mem} \approx \frac{I_{thr} I_{in}}{I_t} - I_g + f(I_{mem})$$

$$\tau_{ahp} \frac{d}{dt} I_g + I_g = \frac{I_{thr} I_{ahp}}{I_{\tau_{ahp}}}$$

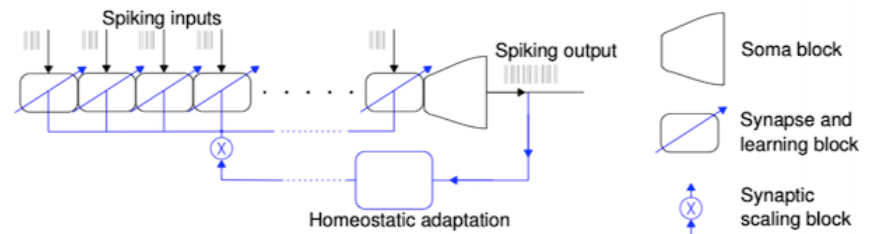
The state of the art version of this neuron circuit consumes one order of magnitude less power than the circuit described in the following figure and two orders of magnitude less power than the digital implementation of the I&F neuron. Given the exponential nature of the generalized IF neuro's non-linear term  $f(I_{mem})$ , the DPI-neuron implements an adaptive exponential IF model. 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.



## Neuromorphic Processors

Typical spiking neural network chips have the elements described in the figure below. 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
- Analog and digital co-design
- On-Chip inference and learning
- Reconfigurable architecture
- Distributed SRAM and TCAM memory cells
- Capacitors for state 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
- Reprogrammable network topology and connectivity
- Ideal for integration with binary, non-volatile resistive memory devices
- Ideal for integration with dynamic, volatile/non-volatile memristive devices
- Ideal for integration in 3D VLSI technology.



## Why Spikes?

- Background: Signal processing systems that need to process real-world sensory signals and interact with the environment in real-time.
- Requirements:
  - Low power and low-latency.
  - Local processing, multi-core architectures and distributed computing.
  - Robust communication of signals across long distances through noisy channels.
- Solution: Use Spikes! The optimal method that minimizes bandwidth and power consumption for achieving this goal, under these constraints, is **pulse-frequency modulation**.

## Why Analog?

### Advantages

- Exploit the full potential of **emerging memory technologies**
  - Control multi-level properties with analog pulse heights.
  - Exploit intrinsic non-linearities.
  - Exploit intrinsic stochasticity.
  - Exploit non-volatility properties.
- Avoid use of digital clock circuitry.
- Avoid large DAC/ADC overhead.
- Minimize power consumption.

### PCM-trace

Exploit the drift of PCM devices to implement long-lasting eligibility traces. These enable the construction of powerful learning mechanisms for solving complex tasks by bridging the synaptic (ms) and behavioral time-scales (minutes).

### Disadvantages

Membrane currents measured across 256 neurons, in response to the same inputs differ.

How to cope with mismatch? Integrate over space (use **populations** of neurons) and Integrate over time (use **mean rates**).

### False Myths about Analog Neural Responses

- Neural Responses are Slow (Population firing rates of neurons can reliably encode weak signal changes approx. 50 times faster than individual neurons, populations of noisy neurons have very fast response times).
- Neurons need to fire at high firing rates to achieve high precision (E-I Balanced populations can encode signals with high precision and low rates, sparse neural population activity can represent signals with high accuracy).
- Neurons need to be accurate to propagate precise information across population layers (Neurons need to be noisy to propagate neural activity reliably, to propagate signals using low-firing rates, it is necessary to use inhomogeneous populations of neurons).

### Neuromorphic Applications

- Extreme Edge Computing
  - Online Anomaly Detection
    - Industrial Predictive Maintenance (PM): PM involves the health monitoring of a degrading system. Vibration patterns yield valuable information about the health state of a running machine. PM is typically applied to large industrial tasks, but could be useful for small appliances and robots as well.
  - Online HFO Detection
    - High-Frequency Oscillations (HFO) are spontaneous EEG events in the frequency range between 80 and 500 Hz consisting of at least four oscillations that clearly stand out from the baseline. HFO are biomarkers for epileptogenic brain tissue.

### Neuromorphic vs Conventional Processors

| Pros                                                                                                                                                                           | Cons                                                                                                                                                                  |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"><li>• Low latency</li><li>• Ultra low-power (&lt;1mW)</li></ul>                                                                              | <ul style="list-style-type: none"><li>• Limited resolution (&lt;4bits)</li><li>• High variability, noisy</li></ul>                                                    |
| What are they good for?                                                                                                                                                        | What are they bad at?                                                                                                                                                 |
| <ul style="list-style-type: none"><li>• Real-time sensory-motor processing</li><li>• Sensory-fusion and on-line classification</li><li>• Low-latency decision making</li></ul> | <ul style="list-style-type: none"><li>• High accuracy pattern recognition</li><li>• High precision number crunching</li><li>• Batch processing of data sets</li></ul> |
| How to program a neuromorphic processor?                                                                                                                                       |                                                                                                                                                                       |
| <ul style="list-style-type: none"><li>• Configure network structure and parameters</li><li>• Train the network with different learning methods</li></ul>                       | <ul style="list-style-type: none"><li>• Define neural computational primitives</li><li>• Compose multiple primitives for context dependent processing</li></ul>       |



## Exploitation

### Technology Transfer and Applications

We are now entering the era of *neuromorphic intelligence* in which dedicated cognitive “chiplets” will be used to provide intelligence to a multitude of **edge-computing** devices.

### The Perfect “recipe” for Fabricating Neuromorphic Intelligence Devices

- “Mix” populations of mixed-signal silicon neurons and synapses.
- “Add” capacitors and volatile memristors for state dynamics and memory traces.
- “Sprinkle” distributed memory elements for parameter storage (SRAM, TCAM, non-volatile memristors).
- “Bake” with asynchronous digital circuits for event-based communication.
- “Serve” with always-on, on-line, self-supervised learning methods.

### Conclusion

- Conventional AI increasing power requirements are unsustainable.
- New emerging memory technologies can benefit from massively parallel processing architectures.
- Neuroscience and Machine Learning are uncovering powerful and robust neural processing methods.
- This is the perfect time to follow the neuromorphic engineering approach for starting a hardware revolution.

## Attention Is All You Need – Lecture 13 – Benjamin Grewe

### Content of the Lecture

- Attention in Neuroscience
  - Definition of Attention (Cognitive Neuroscience).
  - What is Selective Attention in Biology (Cognitive Neuroscience).
  - Neuronal Substrates of Attention (Cognitive Neuroscience).
- Attention Models in Machine Learning
  - Self-Attention.
  - Transformers.
  - GPT3 and ChatGPT.

### Selective Sensory Attention in Cognitive Neuroscience

*“It is the taking possession by the mind in clear and vivid form, of one out of what seem several simultaneously possible objects or trains of thought ... It implies withdrawal from some things in order to deal effectively with other, ...” – William James in Principles of Psychology (1890)*

*“Attention is the flexible control of limited computational resources” – Lindsay (2020)*

### Function of Selective Attention

- In primates,  $10^6$  retinal ganglion cell axons can carry about 1 MB/s of visual information (already compressed).
- This is too much to be handled in real-time by the brain (there are many other sensory domains that feed data into the brain).

 define attention in neuroscience

 In the field of neuroscience, attention is a cognitive process that allows individuals to selectively concentrate on a specific aspect of the environment while ignoring other stimuli. This process is mediated by the brain's attentional system, which is made up of various neural structures and pathways that work together to filter out irrelevant information and focus on relevant stimuli. The attentional system plays a crucial role in many everyday activities, such as reading, driving, and listening to someone speak. It is also important for higher-level cognitive processes, such as problem-solving and decision-making.

| Property                 | Bottom-Up                                                  | Top Down                      |
|--------------------------|------------------------------------------------------------|-------------------------------|
| Spatial specificity      | Throughout field of view                                   | Spatially proscribed (focal)  |
| Feature specificity      | Acts at all times and in all feature dimensions (saliency) | Can select specific attribute |
| Duration                 | Transient                                                  | Sustained (with effort)       |
| Task dependency          | No                                                         | Yes                           |
| Under volitional control | No                                                         | Yes                           |

- Some fraction of sensory input is selected for processing by higher-level, cognitive routines, while the non-attended portion of the input is processed at a reduced bandwidth.
- Attention is common to all animals.

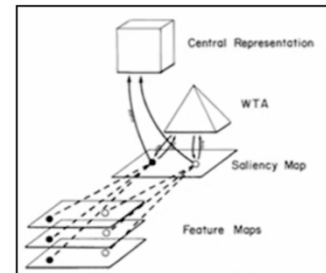
### Bottom-Up Attention

Something immediately draws our eye to the Toblerone and the Matterhorn. This something is **visual saliency**.



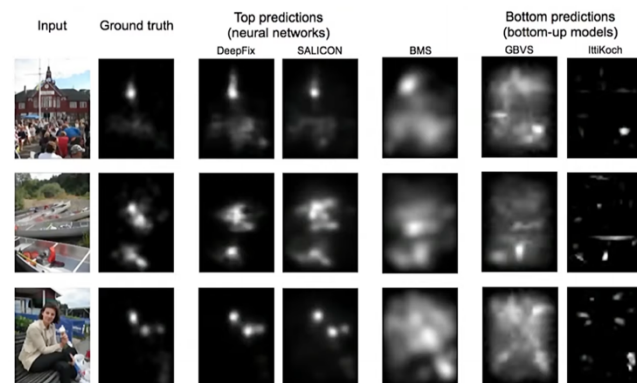
### Visual Saliency

- Single map that encodes saliency is feed by multiple feature maps. Which features draw the gaze independent of the task is unknown. Typically, these are faces, orientations, color and intensity.
- After detecting the most salient location, its salience is inhibited.
- Single cell evidence for salience map in the initial response of LIP and FEF neurons in primates.



Itty & Koch 2001

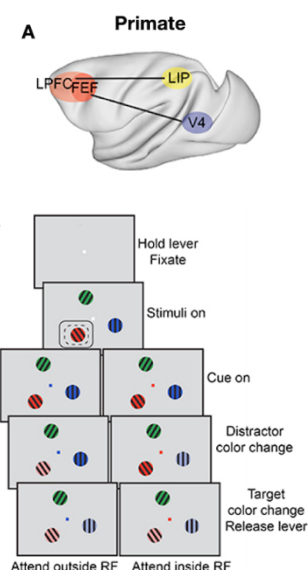
Nowadays DL is exploited to predict saliency spots in pictures. They take ground truth data and train neural networks to predict saliency points in images with higher accuracy than older models. Such models are also exploited to work in the other direction and help in designing images to convey attention towards specific points of the figure.



### Top-Down Attention

- Is flexible and task dependent.
- Can last indefinitely with effort.
- Spatial based, feature-based and object based are forms of top-down attention.
- A close relationship between neuronal mechanics controlling eye gaze and selective attention (LIP, FEF, V4, SC).

Selective Top-Down Attention Neural Signals have been measured in Primates. The primates watch a screen and fix the white dot. Then the three stimuli went on and the cue went on (the central dot that changes color). The researchers used electrophysiology to record the activity of individual neurons in the visual cortex of monkeys as they performed a visual search task. The goal of the task was to detect a target stimulus among a set of distractors. The researches found that attentional modulation was present in both feedforward and feedback signals in the visual cortex, with top-down attentional signals strengthening target-selective responses and suppressing responses to distractors. This study provides evidence for the role of top-down attentional signals in modulating both feedforward and feedback processing in the primate visual cortex, and suggests a mechanism by which attention can selectively enhance target representation and improve perceptual processing.



The paper “*Selective Top-Down Attention Modulates Feedforward and Feedback Neural Signals in Primates*” by Paperi et al. (2017) investigated the effect of top-down attention on feedforward and feedback signals in primate brains. The study found that when attention was directed towards a particular visual stimulus, both feedforward and feedback signals increased in

the regions of the brain responsible for processing that stimulus, demonstrating that top-down attention can modulate both feedforward and feedback signals. These results suggest that attention plays a crucial role in shaping neural processing in the visual system, and that top-down attention can dynamically influence feedforward and feedback signals to enhance processing of behaviorally relevant information.

## Attention in Deep Learning

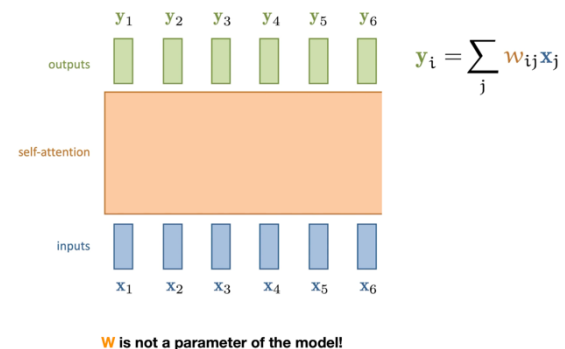
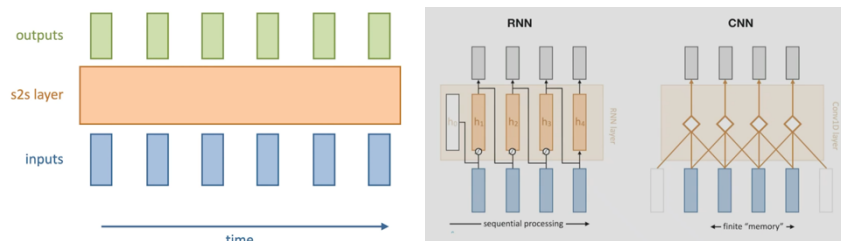
In the field of Machine Learning, attention is a mechanism used by certain types of neural networks to selectively focus on certain parts of an input when processing it. This is useful because it allows the model to automatically learn to focus on the most relevant parts of the input, which can improve its performance on a given task. For example, an attention mechanism might be used in a machine translation model to automatically focus on the words in the source sentence that are most relevant for generating the correct translation.

We'll focus here on time-series attention models.

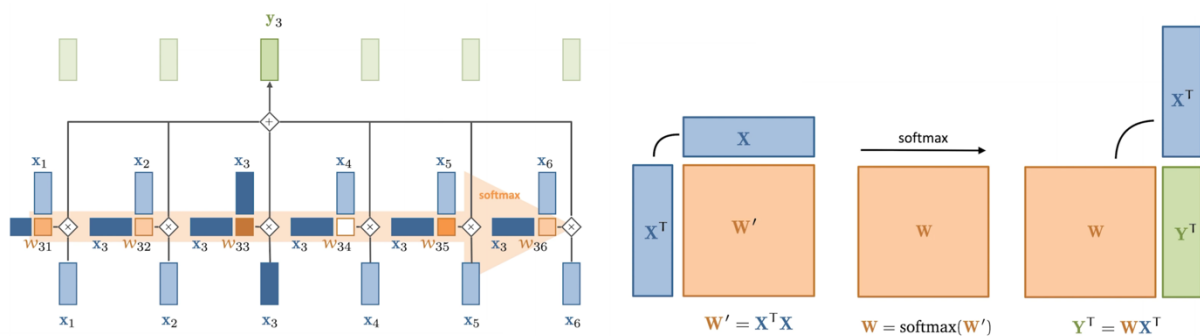
This is a sequence-to-sequence neural network. You want your architecture to produce an output for every timestamp.

And usually you do these through RNNs (LSTMs), which features hidden states and recalls previous states of the network, thus requires a sequential step-by-step calculation of the states. On the other hand we have CNNs which are great in parallelization, but the downside is their finite and fixed memory (loose flexibility). Can we combine these two architectures? That is, can we have the long dependencies and flexibility of RNNs together with the parallelization properties of CNNs? Here is where **Self-Attention (SA)** Models come in.

The outputs are calculate according to the formula in the right figure, however the key peculiarity is that  $W$  is not a parameter of the model that is learnt through gradient descent. It is a parameter that indicates how similar the inputs are similar between them and are calculated as shown in the following figure. These weights are passed through a Softmax such that the sum of all weights add up to 1.



$$w'_{ij} = \mathbf{x}_i^T \mathbf{x}_j \quad w_{ij} = \frac{\exp w'_{ij}}{\sum_j \exp w'_{ij}}$$



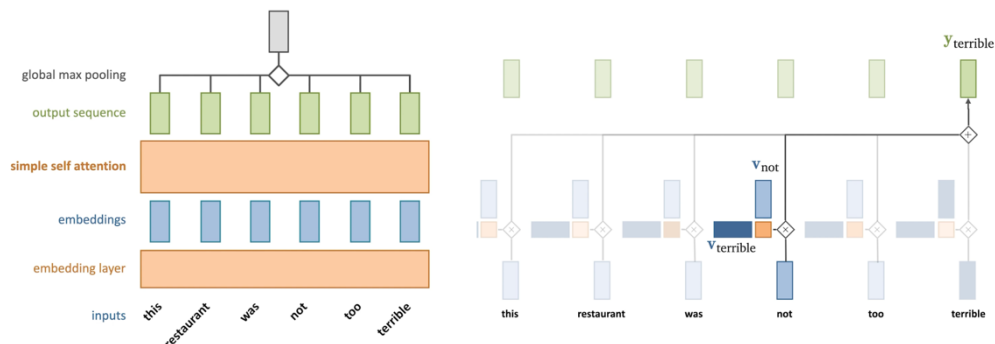
- In simple self-attention  $W_{ii}$ , i.e.,  $X_i$  to  $Y_i$  usually has the most weight. Which is not a big problem, but we will allow this to change later.

- Simple self-attention has no parameters. Whatever parametrized mechanism generates  $x_i$  (like an embedding layer) drives the self-attention.
- There is a linear operation between  $X$  and  $Y$ . Non-vanishing gradients through  $Y = WX^T$ , vanishing gradients through  $W = \text{softmax}(X^T X)$ .



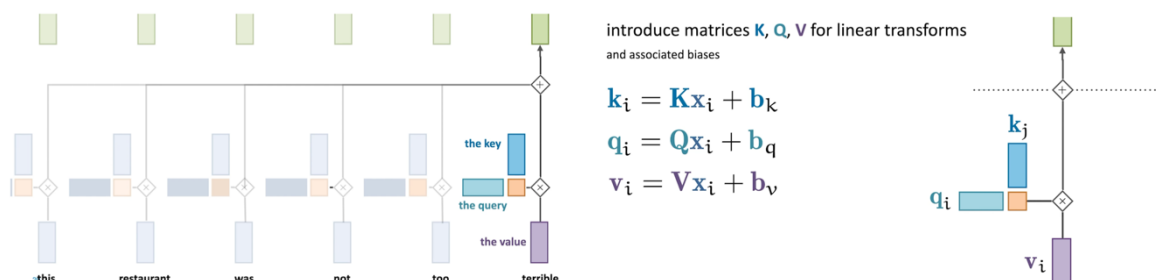
### Intuition how Self-Attention works: The Dot Product

Let's say we want to try a model to understand the sentiment of some restaurant reviews (like in the example below). The word "terrible" is something that probably we wouldn't like to have in a restaurant review, however if it comes after "not too" is probably not such a bad review. Hence, our model should pose attention also to these part of the sentence to understand the overall sentiment of the phrase, i.e., the weight between "not" and "terrible" should be a major one in our self-attention matrix. Which implies that the model should be capable of understanding the dependencies between the input "not" and "terrible" and how it modulates the meaning.

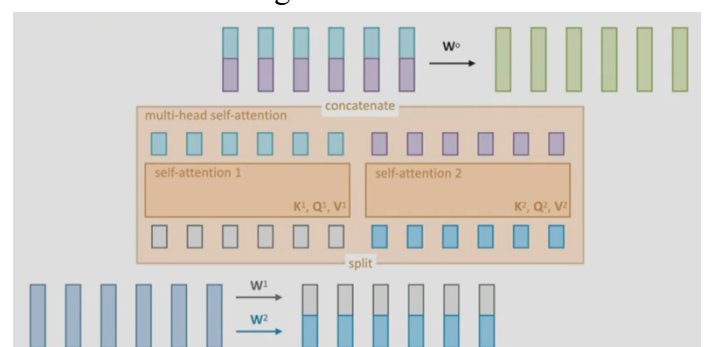
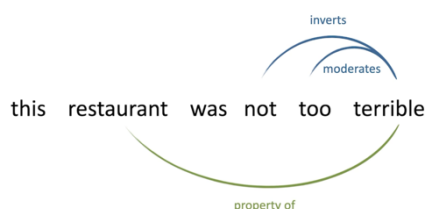


### Improving Self-Attention

- **Scaled Dot Product:** If we have longer and longer input vectors then  $W_{ij}$  gets larger and larger. In order to avoid that we scale it through a **normalization factor  $K$** .  $w'_{ij} = \frac{x_i^T x_j}{\sqrt{k}}$
- **Key, Value and Query** transformations. These is the main breakthrough towards transformers architectures: the Key value is not anymore just the input value, but rather a linear transformation of the input, i.e. the input value times  $K$  plus a bias term  $b$ . In this step we are actually allowing the network to learn these attentional weights because the parameters  $(K, Q, V, b_k, b_q, b_v)$  can be trained through gradient descent.



- **Multi-Head Attention:** The input vector is projected into some low-dimensional version, you concatenate them and you obtain multiple sets of attention weights that can be learnt. This allows the network to have a self-attention that can understand the relation between words in the sentence.



## Recap of Self-Attention

Self-Attention: Sequence-To-Sequence layer with **Parallel Computation** and **Perfect Long-Term Memory**. Fundamentally a set-to-set layer, no access to the sequential structure of the input. A large part of the behavior comes from the parameters upstream.

## Transformers

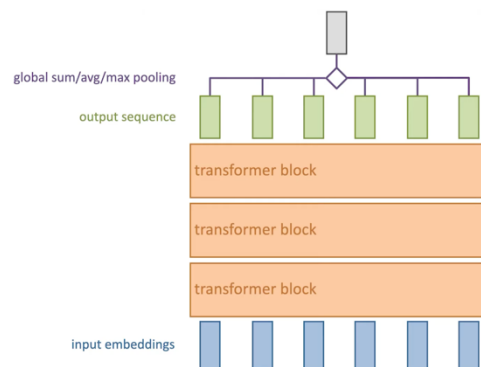
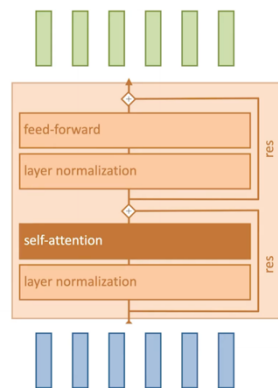
Any sequence-based model that primarily uses self-attention to propagate information along the time dimension. More broadly: any model that primarily uses self-attention to propagate information between the basic units of our instances.

- Pixels → Image Transformer.
- Graph Nodes → Graph Transformer.

The transformer architecture is based on blocks as the one showed below that are stuck on each other as shown in the right figure.

### TRANSFORMER BLOCK

```
class Block(nn.Module):  
    def forward(self, x):  
        y = self.layernorm(y)  
        y = self.attention(x)  
        x = x + y  
  
        y = self.layernorm(x)  
        y = self.linear(x)  
        return x + y
```



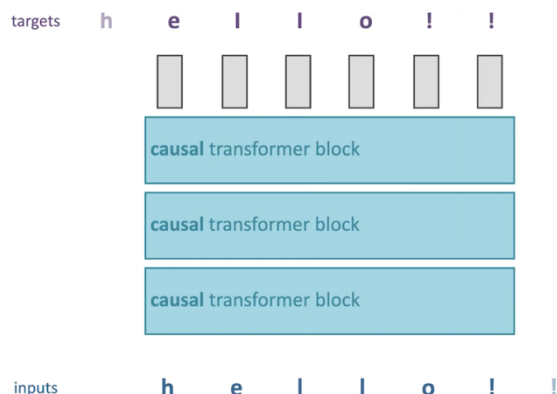
## The Auto-Regressive Transformer

One problem that we still have is making sure that the network cannot access information regarding upcoming targets from the previous layers. In order to do this we calculate the **Causal Self-Attention** weight matrix and all future values are set to minus infinity.

## Causal Self-Attention

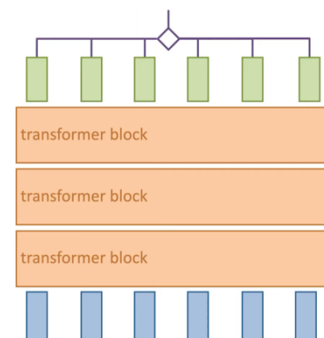
By using Causal Self-Attention we force the network to figure out which words are important in the previous sequence to generate the following letter or word. The blocks that feature such characteristic are named **Causal Transformer Blocks**.

## The Causal Autoregressive Transformer



This is not a real restaurant, it's a filthy burger joint.

This is not a filthy burger joint, it's a real restaurant.





## The Transformer: Position Embeddings

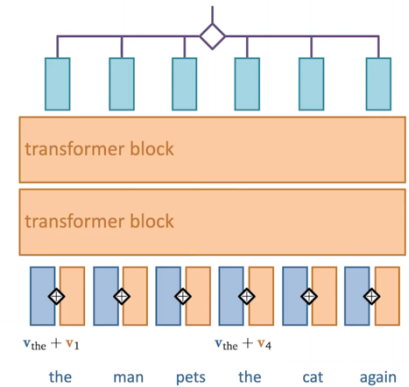
The last problem we have to address is “repetition of words” or again different meanings associated with the same word (e.g., “The”). In order to overcome this limitation, **Position Embeddings** are used, so that same words are considered differently by the network.

word embeddings:

$\mathbf{v}_{\text{the}}, \mathbf{v}_{\text{man}}, \mathbf{v}_{\text{pets}}, \mathbf{v}_{\text{cat}}, \mathbf{v}_{\text{again}}$

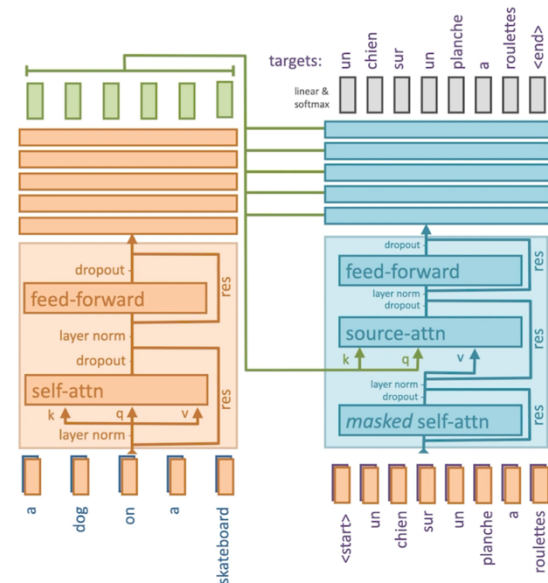
position embeddings:

$\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_4, \mathbf{v}_5, \dots$



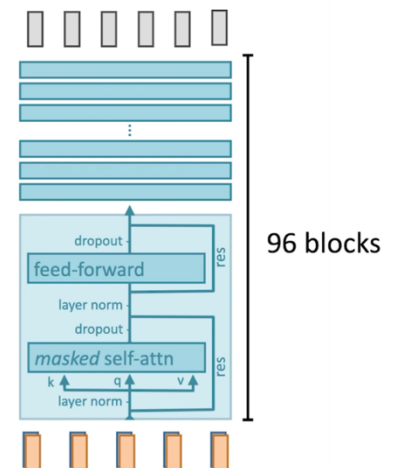
## The “Original Transformer” (ELMo)

Adding all these features together, we obtain the very first transformer ELMo from the paper “*Attention Is All You Need*” that was exploited for language translation. It was a **machine translation** model with no recurrent layers or convolutions, but rather an **encoder/decoder** configuration with positional encoding. It featured 512 dims, 8 heads, 2x6 blocks. It has been trained for 3.5 days on 8 GPUs.

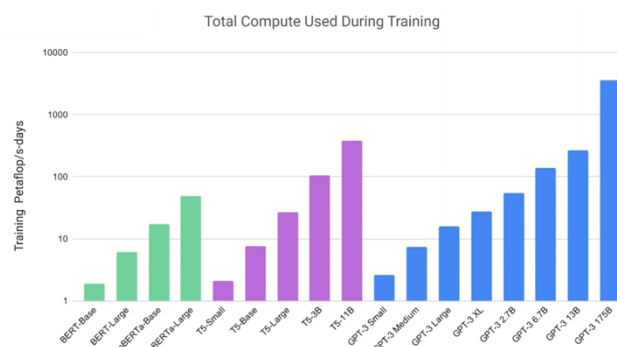


## The GPT3 Transformer

Autoregressive language model. Single stack of **causal trf blocks** with positional embeddings. 12288 dims, 96 heads, 96 blocks, sequence size 2048 and 175 billion total parameters. It has been trained on 10000 GPUs, likely in around 12 days for about \$4,6 million.



## Training Transformer Models



## Jeopardy Questions

- The value of the current state ( $V_s$ ) changes according to the “immediate reward” and its “estimated difference to the next state” in this rule. What is the Temporal Difference Rule?
- This system is used to express genes in a cell specific manner (in GMOs). What is the CRE (/ loxP) system?
- This man is the creator of a binary network with Hebbian learning rules that use attractors as memory. Who is John Hopfield?
- This well-studied brain area is most important for memory. What is the Hippocampus?
- This is what  $x_1$  and  $x_2$  are, given  $\langle x_1, x_2 \rangle - \langle x_1 \rangle \langle x_2 \rangle = 0$ . What is uncorrelated?
- This animal is rarely used as a model organism in neuroscience. What is [fill in animal name here] that is not one of the classical ones used?
- This property of SNNs makes it difficult to train them. What is spiking nonlinearity?
- The adaptive linear neuron was introduced in 1960 under this name. What is ADALINE?
- He postulated “What fires together, wires together”. Who is Donald O. Hebb?
- $\Delta w_{ij} = x_i y_j$  is commonly used to formalize the simplest form of this rule. What is Hebbian Learning Rule?
- These ions are responsible for generating an action potential. What is Sodium and Potassium?
- This method uses an infrared laser to image, e.g.,  $Ca^{2+}$  activities. What is 2-photon microscopy?
- This method generates random feature that do not require learning and acts like jelly. What is reservoir computing?
- This neuromodulator is thought to encode reward prediction error in the brain. What is Dopamine?
- Responding to glutamate, this receptor opens for sodium and calcium ions. What is the NMDA receptor?
- In short-term plasticity, this “gain” may precede depletion. What is facilitation/post-tetanic potentiation?
- CamKII is dependent on this tiny second messenger. What is Calcium ion?
- For consecutive action potentials, these properties tend to stay constant. What are amplitude and duration (shape)?
- This unsupervised method enforces independence of its components. What is Independent Component Analysis (ICA)?
- In this probabilistic Autoencoder the latent representation of an input is sampled from a set of means and standard deviations. What is a Variational Autoencoder?
- This network is trained via the “Wake/Sleep” algorithm. What is Helmholtz Machine? (Not covered).
- These are the methods that `pytorch.autograd.Function` has. What are the forward and backward methods?
- This signal can be detected when dopamine binds to dLight1.1. What is Fluorescence?
- This method/network has a short, counter-intuitive name and is used as an alternative to backprop through time. What is (an) LSTM?
- In the SuperSpike rule, this operation is used to generate eligibility traces. What is Convolution?
- In this rule, a presynaptic event followed by a postsynaptic event followed by a presynaptic event leads to local learning. What is the
- A counterargument for this method can be constructed by locking yourself up in a dark room. What is Free Energy Principle?
- Investigated in owls, this computation is used to determine direction. What is ITD?

- Updating weights with respect to an approximation of the feedforward activations instead of the true feedforward activations is the key difference between these two methods. What are Predictive Coding and Backpropagation?
- Experts commented on sales events from this company with (e.g.) “I don’t think there was anything revolutionary in the presentation”. What is Neuralink?

## How Can Biological Learning Be So Efficient? – Lecture 14 – Christoph Von Der Malsburg

### Content of the Lecture

- Deep Learning
- Ontogenesis
- Representation/Perception
- Learning

### Deep Learning

It represents a revolution in Computer Science! All attention on System Architecture instead of on specific algorithms. Can we say that the Game is Over? Is intelligence just about computational power and data? Shall we stop studying the brain? Shall we give up on understanding the (human) mind?

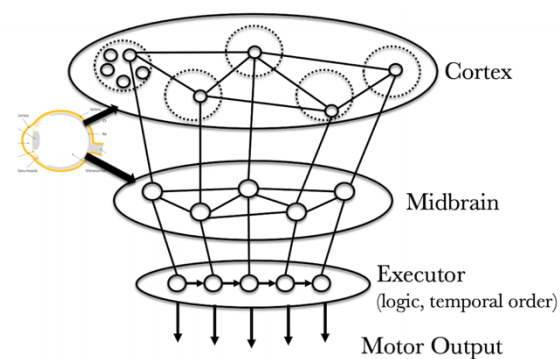
### Limitations of DL

- DL has always looked through human eyes, i.e., it is reliant on language and images that are processed by human understanding.
- DL cannot behave in natural environments (yet, mainly due to a problem in modelling corner situations).
- DL has no Situation Awareness (You are nobody for ChatGPT, Moravec’s paradox).
- DL has no Personality (The system cannot take Responsibility, Veracity, Ethics).
- DL has no Efficient Learning.

### Ontogenesis

It represents the process by which a human/animal is generated.

- Brain is wired to function at birth (e.g., ingestive, reproductive, defensive behavior).
- Reptile brain is inherited through eons.
- Expansion into cerebral hemispheres. (Homeomorphic Expansion: the Midbrain is hardwired genetically and the Cortex is a prestructured image of midbrain, subject to learning and self-organization).



### Information Content of the Brain

- $10^{10}$  neurons.
- $10^{14}$  synaptic connections.
- 33 bits address per connection.
- $10^{15}$  bytes to describe the brain’s wiring.

### Genetic Information

One gigabyte (3.3 billion nucleotides).

### Training Information

A few gigabytes of VR.

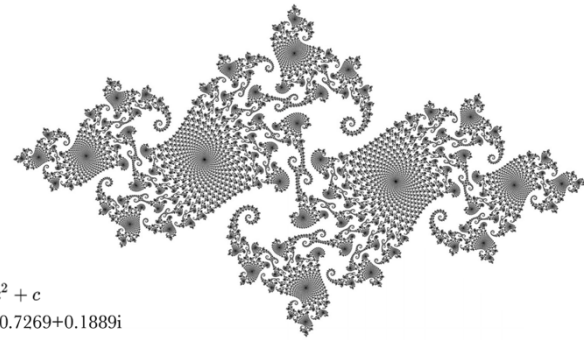
## Information Gap

- One gigabyte of genetic information.
- Some gigabytes to describe the learning environment.
- A petabyte to describe the brain's wiring.

Then where do 99% of the information come from?

## Kolmogorov Complexity

- The shortest algorithm to create a structure.
- Julia Set: image created on the base of an algorithm.
- The amount of information that is required to describe this structure in infinite.
- Highly Efficient Kolmogorov Algorithms
  - Can create a lot of structure.
  - But only certain structures can be created that way.
- The brain must have a very efficient algorithm in the development of its structure.

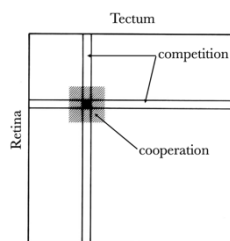
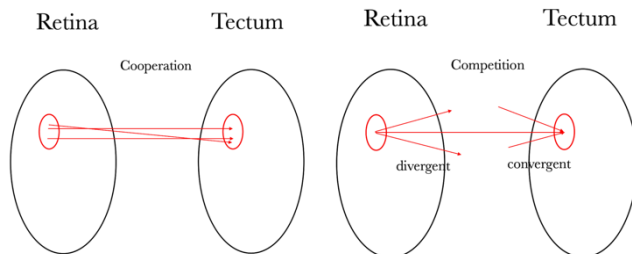
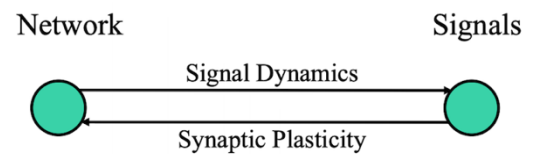


$$f_c(z) = z^2 + c$$

$$c = -0.7269 + 0.1889i$$

## The Brain's Kolmogorov Algorithm

- Low Kolmogorov Complexity.
- Powerfully reduced search space.
- Network Self-Organization.
- Attractor Networks: an universe of structured patterns.
- Ontogenesis of retinotopic fiber projections as example of network self-organization. (Development of wiring between the eye's retina and the optic tectum). During cooperation, neighboring cells help each other building stronger connections.



$$\dot{W}_{\tau\rho} = \alpha + W_{\tau\rho}F_{\tau\rho} - W_{\tau\rho}B_{\tau\rho}(\alpha + WF)$$

$W_{\tau\rho}$ : Connection  $\rho \rightarrow \tau$

Divergent and convergent inhibition between synapses:

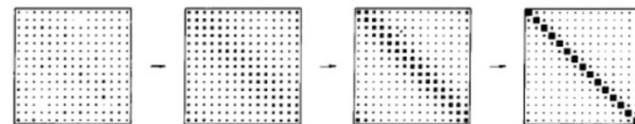
$$B_{\tau\rho}(X) = \frac{1}{2N} \left( \sum_{\tau'} X_{\tau'\rho} + \sum_{\rho'} X_{\tau\rho'} \right)$$

Cooperativity:

$$F_{\tau\rho} = \sum_{\tau'\rho'} C_{\tau\tau'}^T W_{\tau'\rho'} C_{\rho'\rho}^R$$

## Emergence of Self-Reinforcing Network

- One-dimensional retina and tectum.
- Periodic Boundary Conditions.
- Small symmetry-breaking component in initial state.
- Networks emerge on the basis of self-consistency of connectivity.



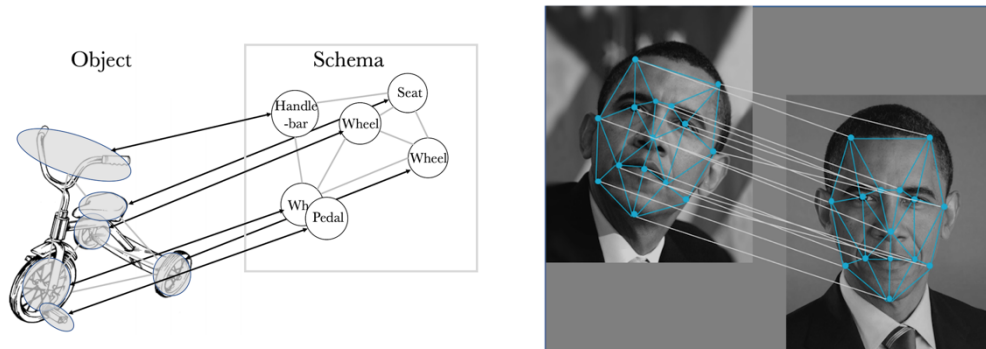
## Two Types of Nets

- Neural Fields
- Topological Mappings

## Representation and Perception

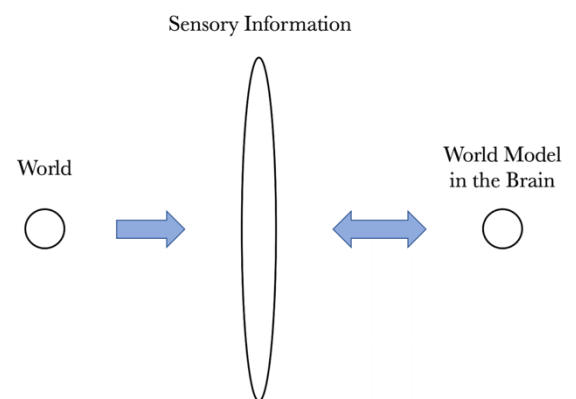
- Scene Representation/Perception
- Computer Graphics: Composing Modalities (Coloring, Material, Shading, Geometry, Edges, Silhouette). They generate images by putting together factors, a sort of ontogenesis of a scene.

- Perception (Computer Vision) is an Inverse Problem: going from images to the composing modalities.
- Object Recognition by Network Emergence. (To recognize a person, you need a model of the person in your cortex somewhere, there is an overlay of network fragments that in a given situation can be activated such that they together form a self-stabilizing network).



## Learning

- Perception precedes Learning!
- Perception must be there at Birth.
- Learning only within the focus of attention.
- The Focus of Attention is shaped by Behavior.
- Learning is shaped by Strategy.
- Learning is gated by Emotions. (Locus Coeruleus communicates through Epinephrin, which is thought to be crucial for synaptic plasticity, hence it may play a substantial role in the memory of emotion-related events).
- Learning refines Consistency of Sensory-Motor Coordination.
- Learning takes Snapshots of Perceived Structures.



## Conclusion

Efficient Learning needs:

- A Functional Initial State.
- Behavioral Drive.
- Embodiment.
- Scene Representation.
- Perception.
- Focal Attention.