Cognitively Interesting Topics in Artificial Neural Networks

Iwo Błądek

Poznan University of Technology

Poznań, 2018

Outline of the Presentation

Approaches to Studying Human Cognition

Eysenck and Keane [3] distinguished the following approaches to studying human cognition:

- ***Experimental cognitive psychology*** — understanding cognition by studying behavior.
- ***Cognitive neuroscience*** — understanding cognition by studying both behavior and corresponding brain activity.
- ***Cognitive neuropsychology*** — understanding cognition by studying behavior of patients with some brain injury.
- ***Computational cognitive science*** — understanding cognition by developing computational models, which takes into account knowledge of behavior and the brain.

Approaches are often combined in order to produce more convincing results.

Computational Cognitive Science

Understanding cognition by developing **computational models**, which takes into account knowledge of behavior and the brain.

There are two main types of computational cognitive models:

- **production systems**

  Model consists of IF ... THEN rules.

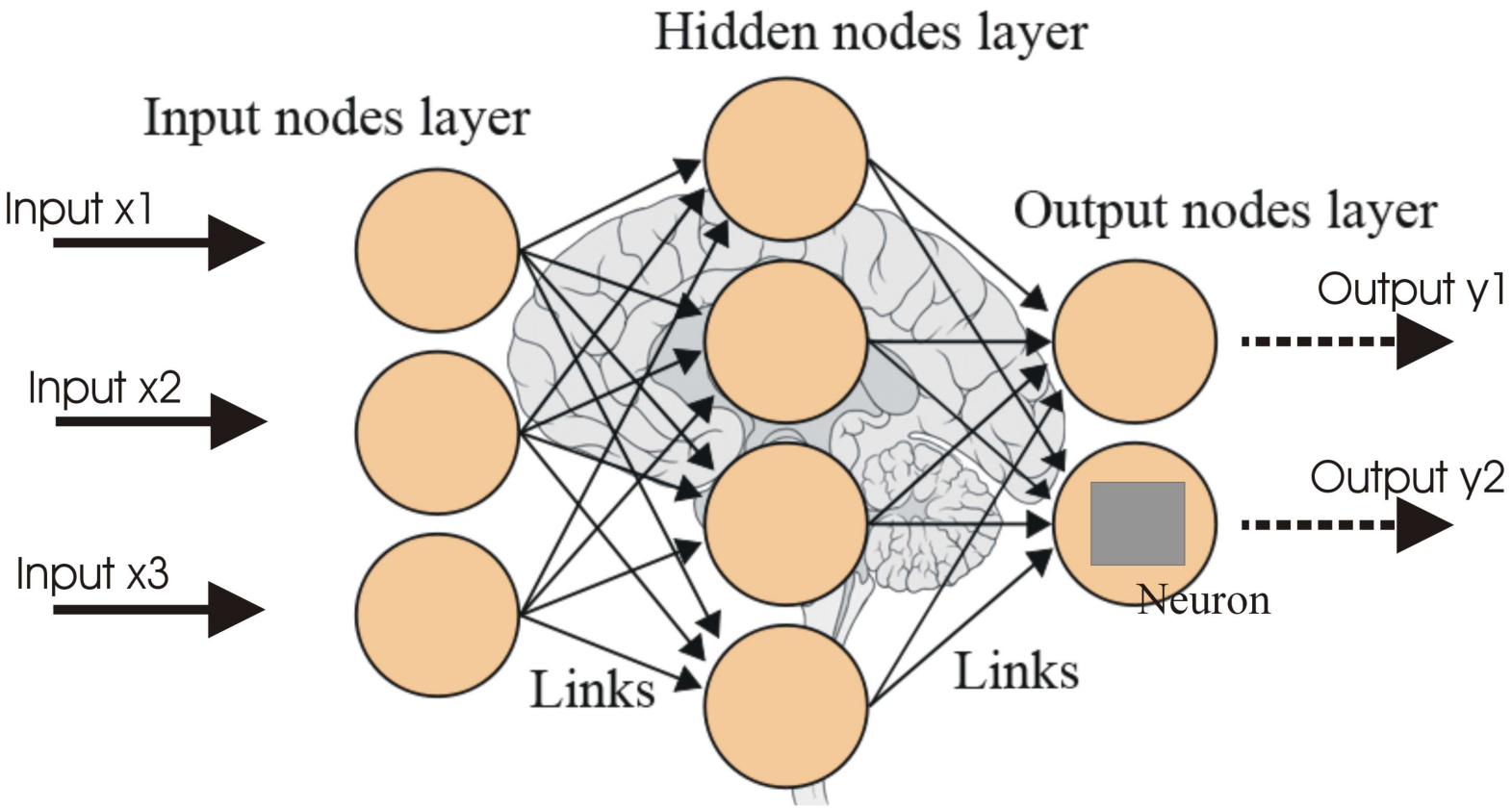
  Example: ACT-R.

- **connectionist networks**

  Model consists of many basic interconnected elements, which together produce complex behavior.

  Example: Nengo, artificial neural networks (ANN).

Artificial Neural Networks



Źródło: http://futurehumanevolution.com/artificial-intelligence-future-human-evolution/artificial-neural-networks

Artificial Neural Networks

- *ANN* $\equiv$ Artificial Neural Networks.
- In contrast to Nengo, neurons in ANN are sending not spikes, but real ($\mathbb{R}$) numbers.
- ANN were **not** created to simulate the brain, but to mimic its effectiveness on computational problems. However, ANN may be also considered as a very simplified model of biological networks.

Recursion

Recursion

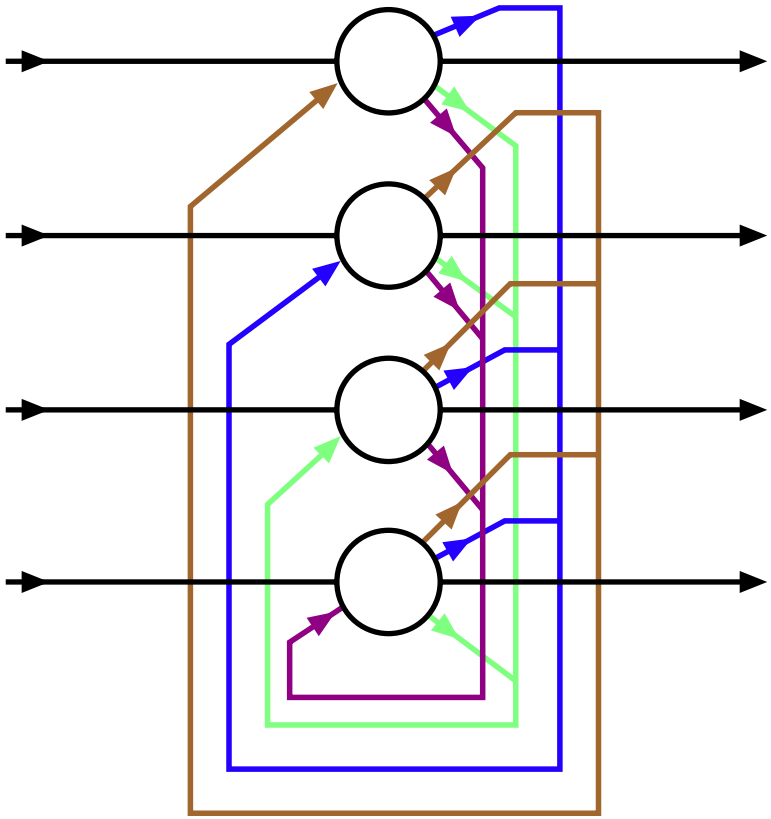
A kind of computations which involves executing itself for a modified input in order to reach final result. Presence of a stop condition is necessary for such computations to end.

Example:

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)
```

Recurrent neural network

Network with connections forming at least one directed cycle. In such a cycle a single activation may potentially travel without end.



Recurrent neural network

Network with connections forming at least one directed cycle. In such a cycle a single activation may potentially travel without end.

Some properties:

- Their dynamic is often very complicated.
- Hard to learn and analyze.
- Network may or may not settle in a certain final state.
- Output of the network varies in time.

Biological neural networks in the brain are recurrent.

Outline of the Presentation

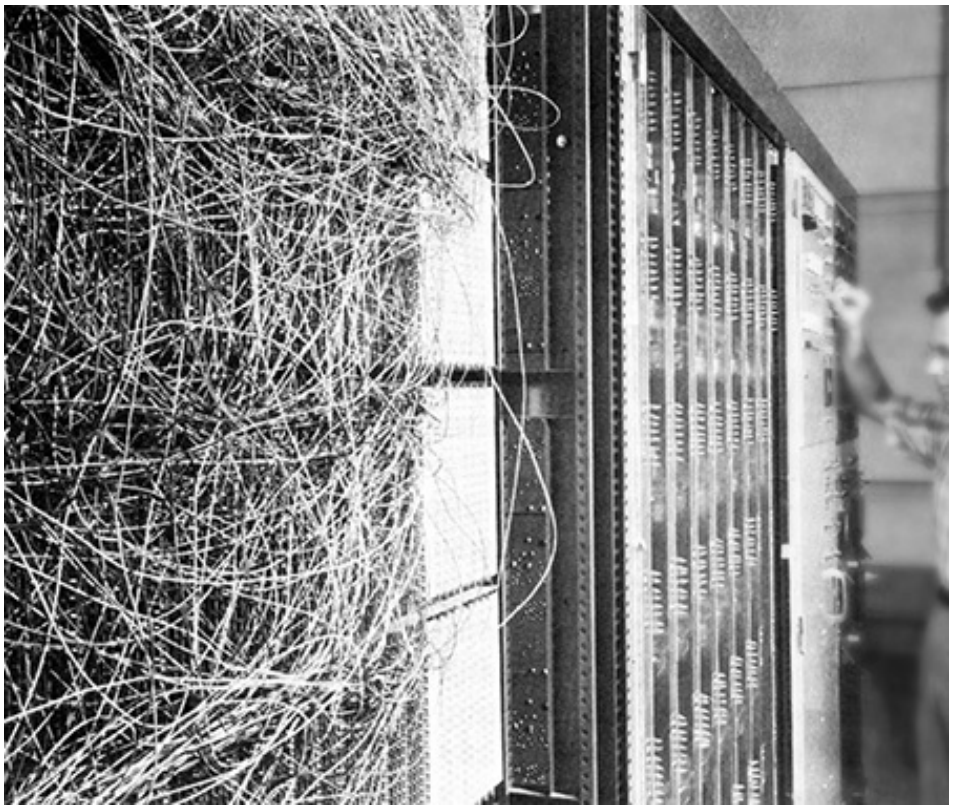
Brief History of ANN Research

History of research presented here was based mainly on [6] and [1].

- 1943 – McCulloch and Pitts showed that simple types of neural networks could, in principle, compute any arithmetic or logical function.
- 1949 – Hebb's learning rule. In his book entitled *"The Organization of Behaviour"*, Hebb presented the idea that classical psychological conditioning may be based on phonemena occuring in neurons.

Brief History of ANN Research

- 1951 – *Snark*, experimental neuro-computer by M. Minsky.
- 1958 – First really successful hardware implementation of neuro-computer, the *Mark I Perceptron*, by F. Rosenblatt.



Source: http://www.rutherfordjournal.org/article040101.html#chapter09

Brief History of ANN Research

- Generally, in 1950s and 1960s research on ANN had rather experimental emphasis and lacked rigor.
- Much enthusiasm and exagerrated claims, for example that artificial brains will be created in a matter of a few years.
- 1969 – M. Minsky and S. Papert published the book "Perceptrons", in which they showed limitations of this model of computation (e.g. that single-layered linear network cannot realize *xor* function).
- As a result, interest in ANN diminished. Many considered this field to be in a "dead end".

Brief History of ANN Research

- 1970s – "Quiet years", but research was continued under the headings of adaptive signal processing, pattern recognition, and biological modeling.

Brief History of ANN Research

- 1980s – Renewed enthusiasm for neural networks.
- 1982 – Hopfield net.
- 1986 – *Backpropagation algorithm* popularized by influential article in Nature by D. Rumelhart, G. Hinton, and R. Williams [5].
- 1986 – PDP (Parallel Distributed Processing).

Brief History of ANN Research

- 1990s – Despite successes in the 1980s funding for ANN research was still scarce.
- Some interesting results in deep learning, e.g. successful applications of convolutional networks.

Brief History of ANN Research

- 2000s – Major increase in the performance of computers and GPUs (Graphical Processing Units, used also to speed up some mathematical computations). This made it possible to learn bigger ANN.
- 2012 – Deep learning achieves outstanding results in the ILSVRC-2012 ImageNet competition.
- After that, ANN research, and especially deep learning research, returns to the mainstream.
- 2016 – AlphaGo, computer program for playing Go based on deep learning, wins 4:1 in a match with the Go master, Lee Sedol.

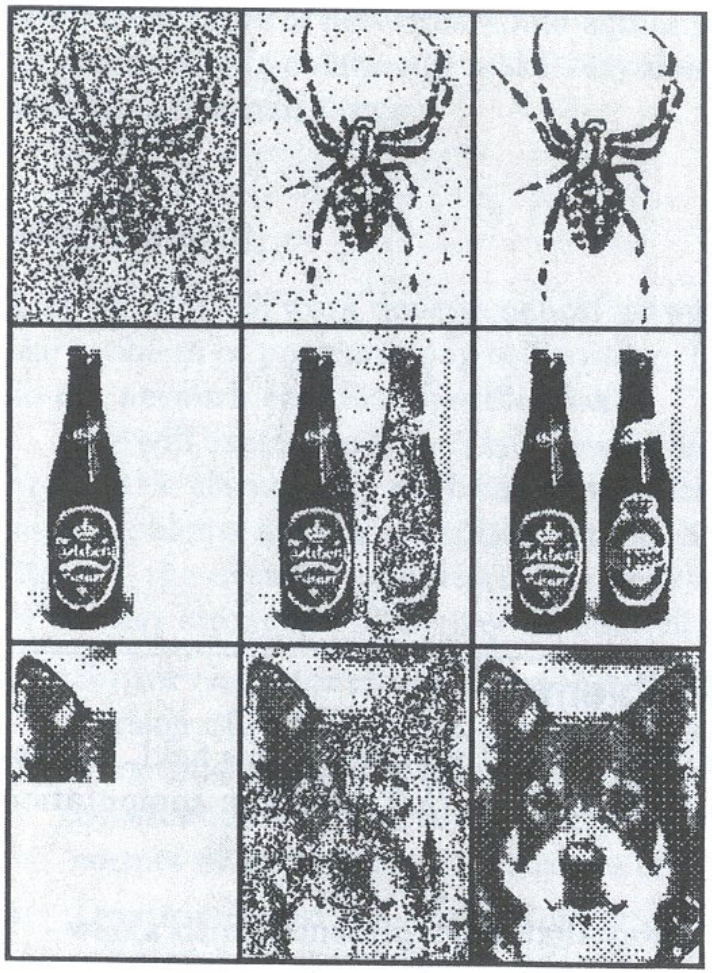
Outline of the Presentation

Hopfield Net

- First proposed in 1982 by John J. Hopfield in [4].
- Autoassociative memory realized by a recursive ANN.
- Memories are encoded by weights of the connections.
- Learning is *not* error-driven, as in usually considered feedforward networks.
- Outputs of the neurons constitute output of the whole network after certain stable state is reached (usually only a small fraction of neurons constitute a network's output).

What Is an Autoassociative Memory?

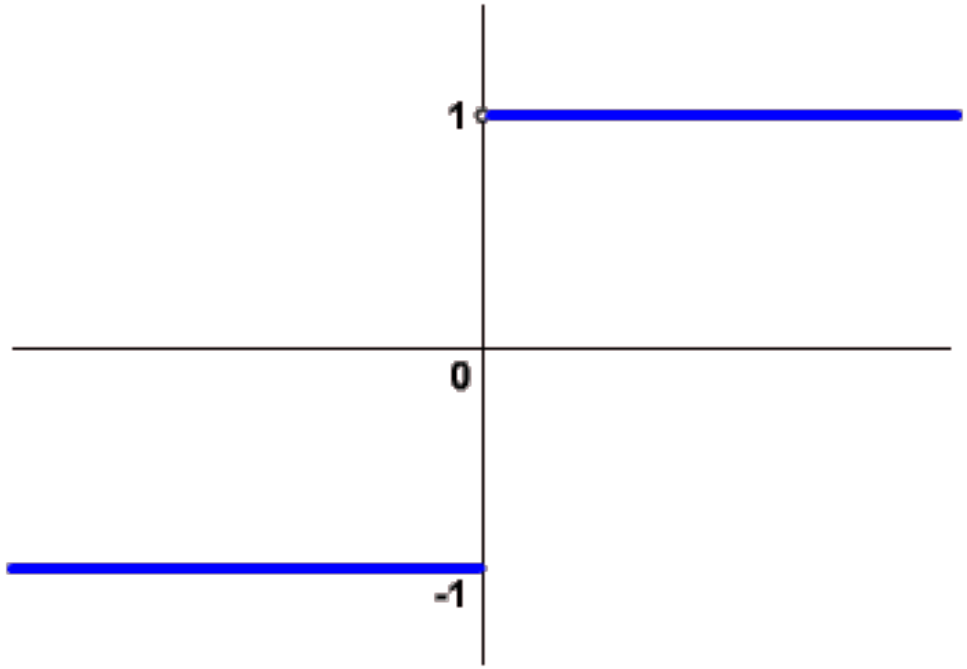
A memory, which returns saved data when provided with some part of it.



Source: http://science.slc.edu/~jmarshall/courses/2002/fall/cs152/lectures/intro/intro.html

Neurons

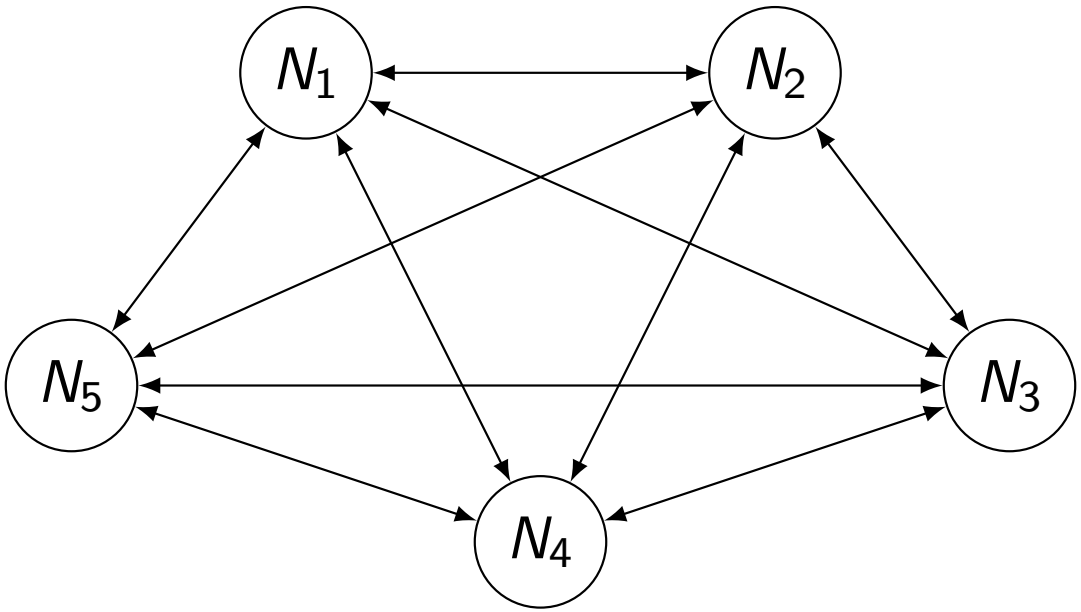
In the Hopfield net, a simple bipolar activation function is used.



$$s_i = \begin{cases} +1 & \text{if } \sum_j w_{ij} s_j \geq 0, \\ -1 & \text{otherwise.} \end{cases}$$

Architecture

In the Hopfield net, every neuron is connected to all other neurons.



Important constraints:

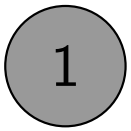
- symmetry: $w_{ij} = w_{ji}$
- no self connections: $w_{ii} = 0$

Updating Neuron States

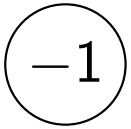
Two methods of updating network state:

- **Asynchronous update rule**
  We update neurons one at a time by summing inputs and applying activation function of a randomly chosen neuron.
- **Synchronous update rule**
  We compute sums of inputs for all neurons, and then simultaneously apply their activation functions.

Legend

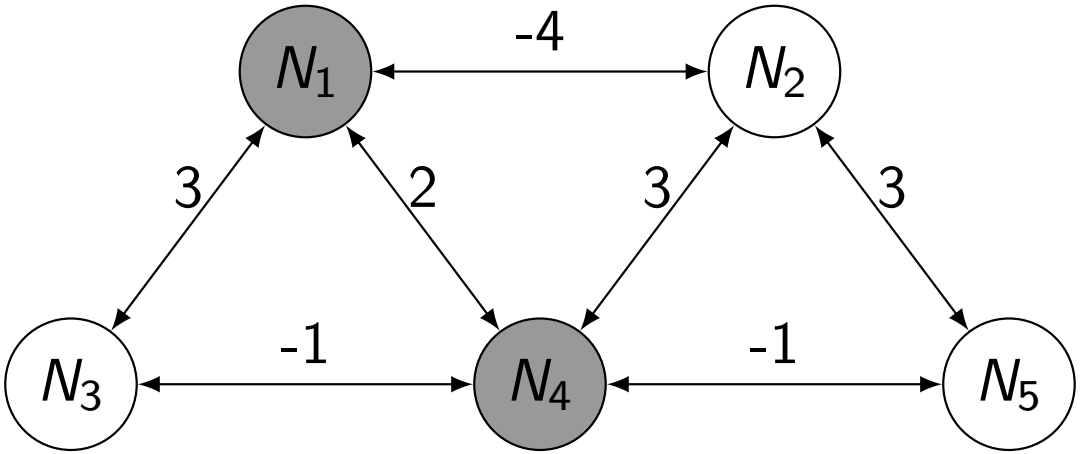


Neuron activated



Neuron not activated

Asynchronous Update



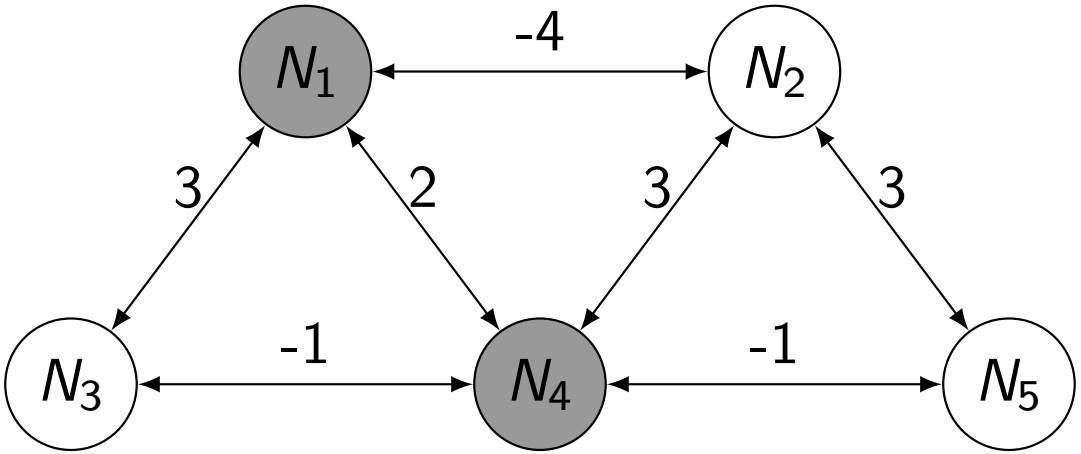
Update queue: N_1, N_2, N_3, N_4, N_5

Sum of inputs for N_1:

$$\sum = (-1) \cdot 3 + (-1) \cdot (-4) + 1 \cdot 2 = -3 + 2 + 4 = 3$$

Set state to 1 (because ≥ 0; in this case nothing changes).

Asynchronous Update



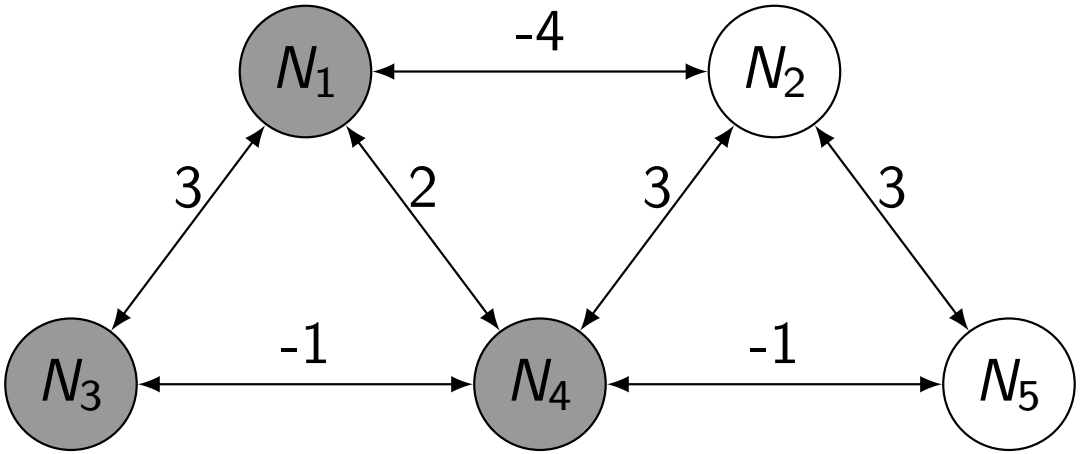
Update queue: N_1, N_2, N_3, N_4, N_5

Sum of inputs for N_2:

$$\sum = 1 \cdot (-4) + 1 \cdot 3 + (-1) \cdot 3 = -4 + 3 - 3 = -2$$

Set state to -1 (because < 0; in this case nothing changes).

Asynchronous Update



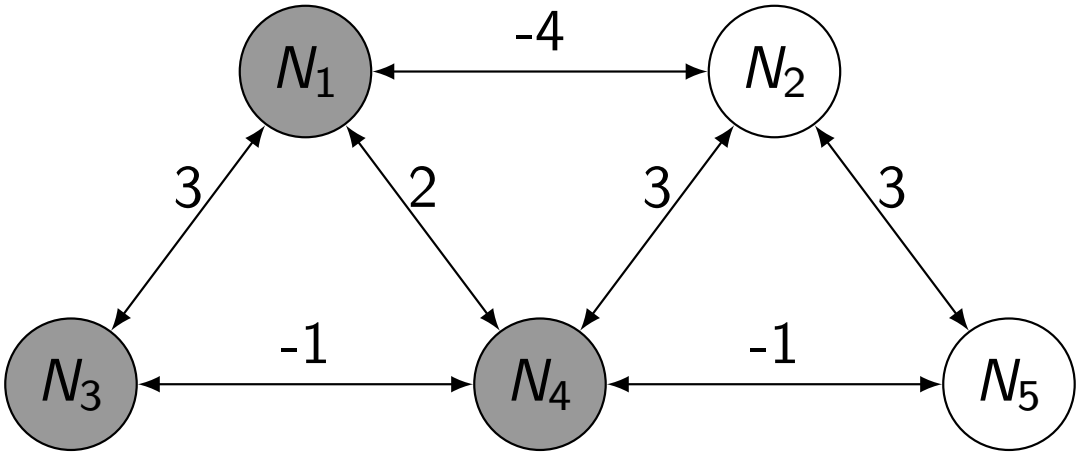
Update queue: N_1, N_2, N_3, N_4, N_5

Sum of inputs for N_3:

$$\sum = 1 \cdot 3 + 1 \cdot (-1) = 3 - 1 = 2$$

Set state to 1 (because ≥ 0).

Asynchronous Update



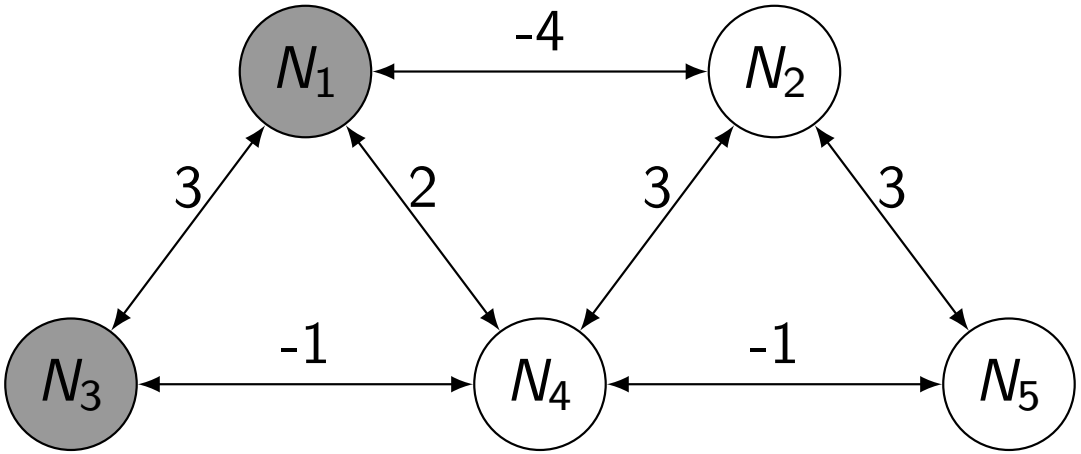
Update queue: N_1, N_2, N_3, N_4, N_5

Sum of inputs for N_4:

$$\sum = 1 \cdot (-1) + 1 \cdot 2 + (-1) \cdot (3) + (-1) \cdot (-1) = -1 + 2 - 3 + 1 = -1$$

Set state to -1 (because < 0).

Asynchronous Update



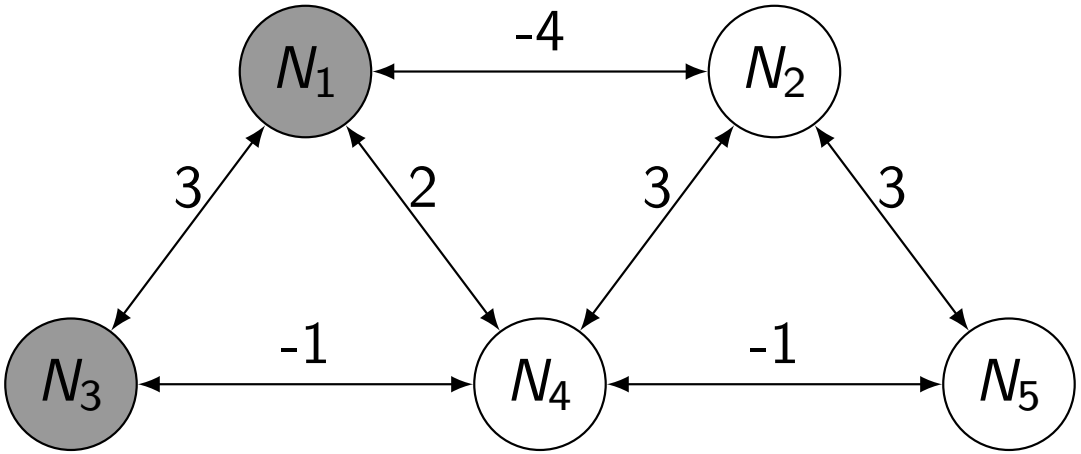
Update queue: N_1, N_2, N_3, N_4, N_5

Sum of inputs for N_5:

$$\sum = (-1) \cdot (-1) + (-1) \cdot 3 = 1 - 3 = -2$$

Set state to -1 (because < 0).

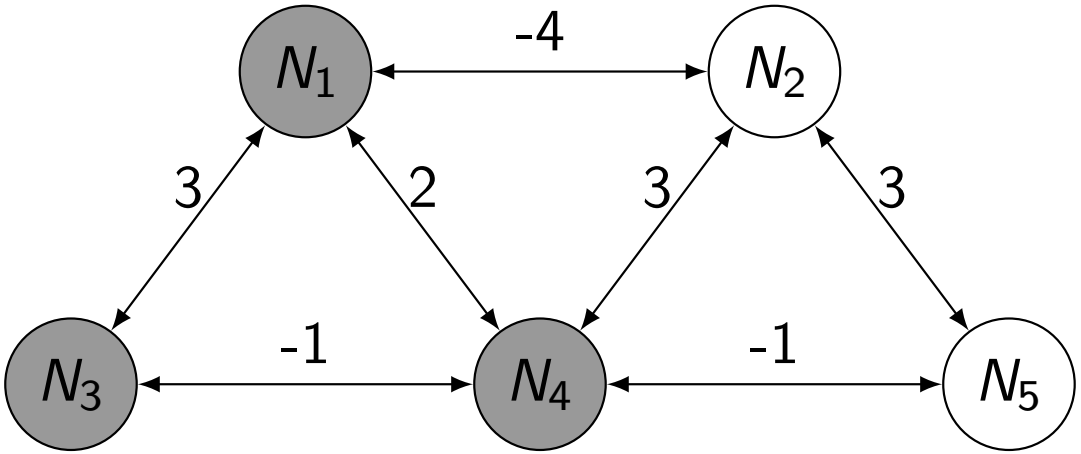
Asynchronous Update



We choose randomly another sequence of units and repeat procedure until there are no changes of neuron state.

New update queue: N_4, N_2, N_3, N_1, N_2

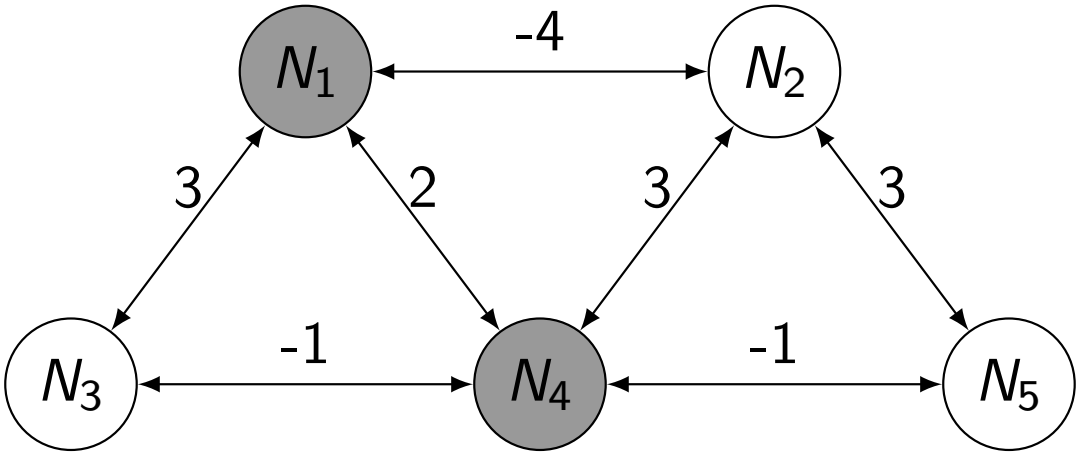
Asynchronous Update



Final stable state of the net (attractor). This is also the local minimum of the energy function.

New update queue: N_4, N_2, N_3, N_1, N_2

Synchronous Update



Iteration 1

Inputs for N_1: $\sum = -3 + 2 + 4 = 3$

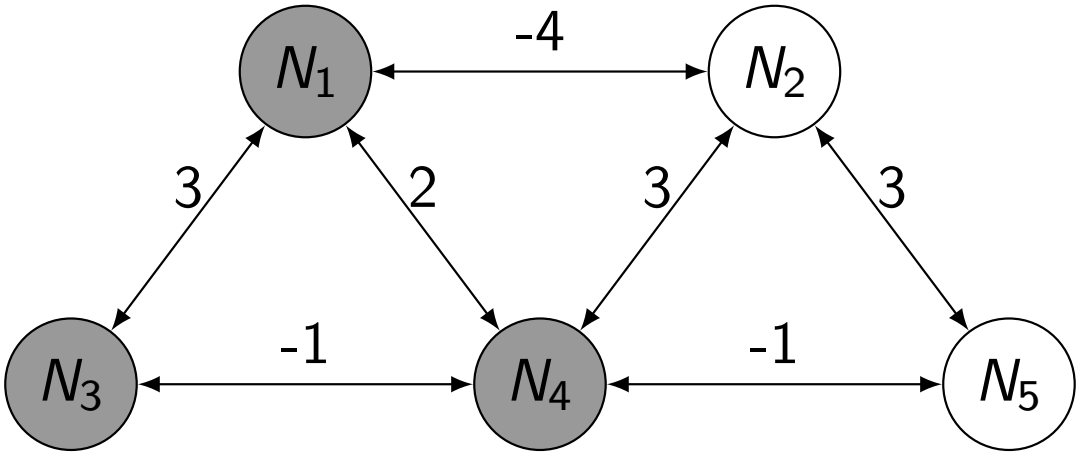
Inputs for N_2: $\sum = -4 + 3 - 3 = -4$

Inputs for N_3: $\sum = 3 - 1 = 2$

Inputs for N_4: $\sum = 1 + 2 - 3 + 1 = 1$

Inputs for N_5: $\sum = -1 - 3 = -4$

Synchronous Update



Iteration 2

Inputs for N_1: $\sum = 3 + 2 + 4 = 9$

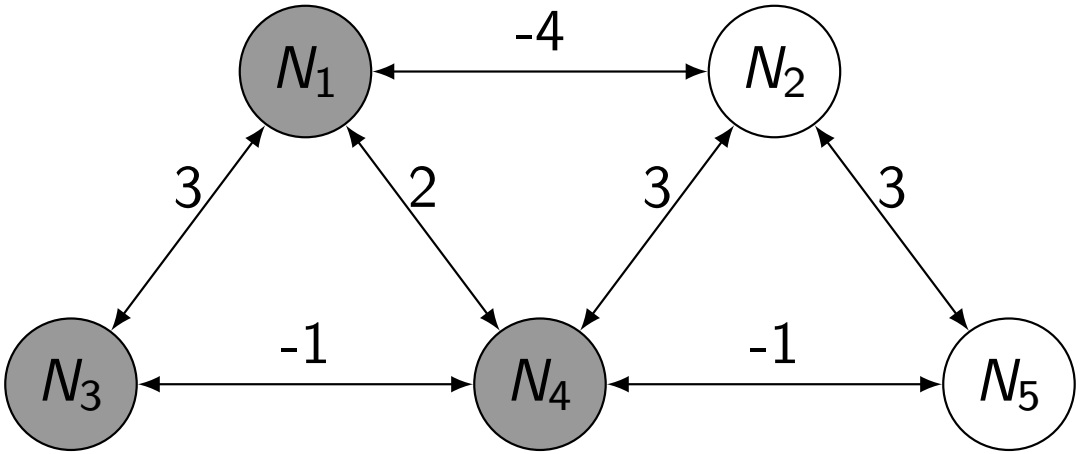
Inputs for N_2: $\sum = -4 + 3 - 3 = -4$

Inputs for N_3: $\sum = 3 - 1 = 2$

Inputs for N_4: $\sum = -1 + 2 - 3 + 1 = -1$

Inputs for N_5: $\sum = -1 - 3 = -4$

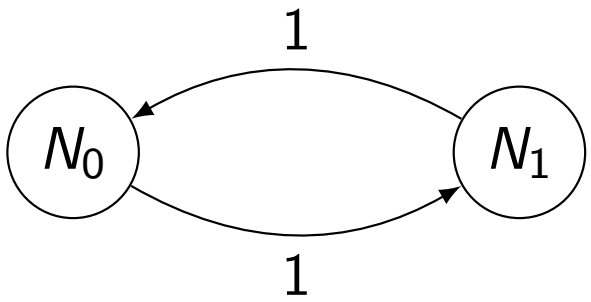
Synchronous Update



For this input, synchronous update rule ended in the same stable state as asynchronous update rule. Not always happens so.

Exercise 1

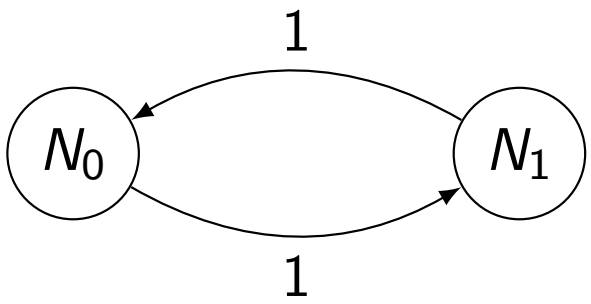
Simulate asynchronous and synchronous updating of these simple 2-neuron Hopfield networks for different initial states. States are represented as vectors with elements from set $\{-1, 1\}$. What is the difference between update strategies? What are the stable states network will settle into?



N_0	N_1	Async	Sync
-1	-1	?	?
-1	1	?	?
1	-1	?	?
1	1	?	?

Exercise 1

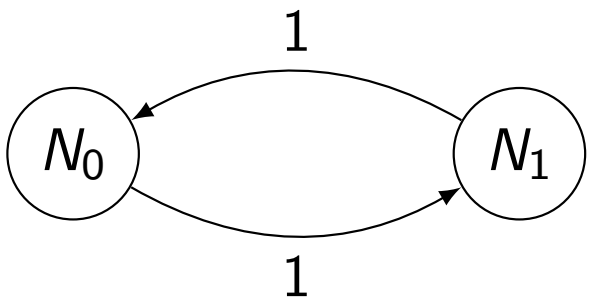
Simulate asynchronous and synchronous updating of these simple 2-neuron Hopfield networks for different initial states. States are represented as vectors with elements from set $\{-1, 1\}$. What is the difference between update strategies? What are the stable states network will settle into?

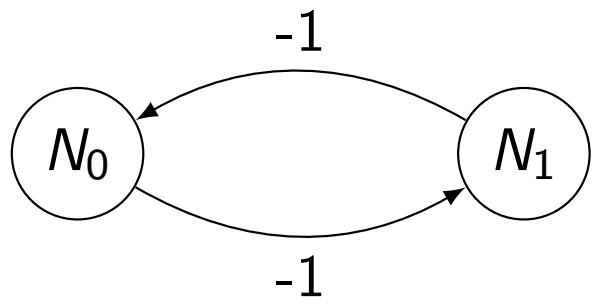


N_0	N_1	Async	Sync
-1	-1	$(-1, -1)$	$(-1, -1)$
-1	1	depends	∞
1	-1	depends	∞
1	1	$(1, 1)$	$(1, 1)$

Exercise 1

Simulate asynchronous and synchronous updating of these simple 2-neuron Hopfield networks for different initial states. States are represented as vectors with elements from set $\{-1, 1\}$. What is the difference between update strategies? What are the stable states network will settle into?



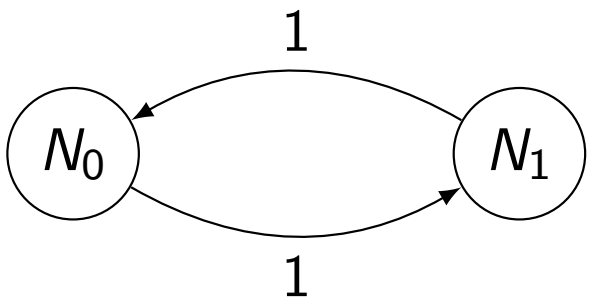


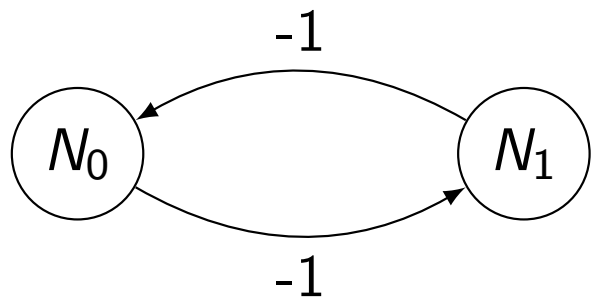
N_0	N_1	Async	Sync
-1	-1	$(-1,-1)$	$(-1,-1)$
-1	1	depends	∞
1	-1	depends	∞
1	1	$(1,1)$	$(1,1)$

N_0	N_1	Async	Sync
-1	-1	?	?
-1	1	?	?
1	-1	?	?
1	1	?	?

Exercise 1

Simulate asynchronous and synchronous updating of these simple 2-neuron Hopfield networks for different initial states. States are represented as vectors with elements from set $\{-1, 1\}$. What is the difference between update strategies? What are the stable states network will settle into?





N_0	N_1	Async	Sync
-1	-1	$(-1,-1)$	$(-1,-1)$
-1	1	depends	∞
1	-1	depends	∞
1	1	$(1,1)$	$(1,1)$

N_0	N_1	Async	Sync
-1	-1	*depends*	∞
-1	1	$(-1,1)$	∞
1	-1	$(1,-1)$	∞
1	1	*depends*	∞

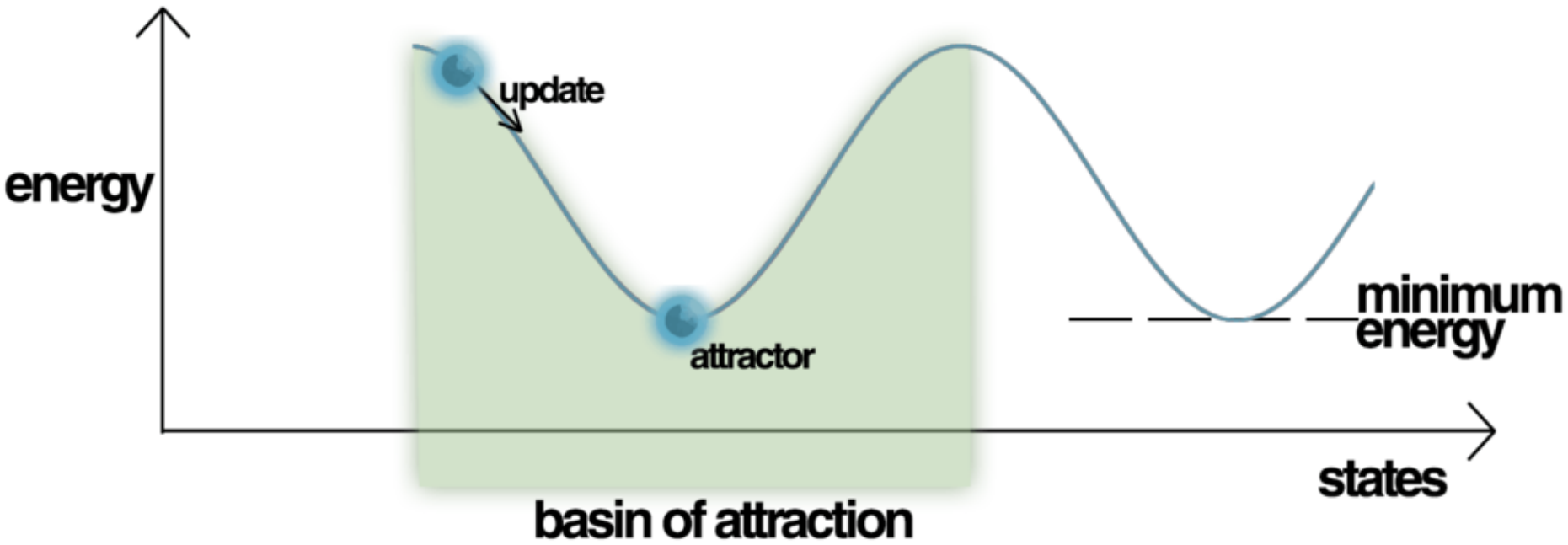
Global Energy Function

To each network's state we can assign certain energy value. Our remembered patterns are local minima in the such created energy landscape. Under asynchronous update rule we have a guarantee that we will eventually reach one of those minima.

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

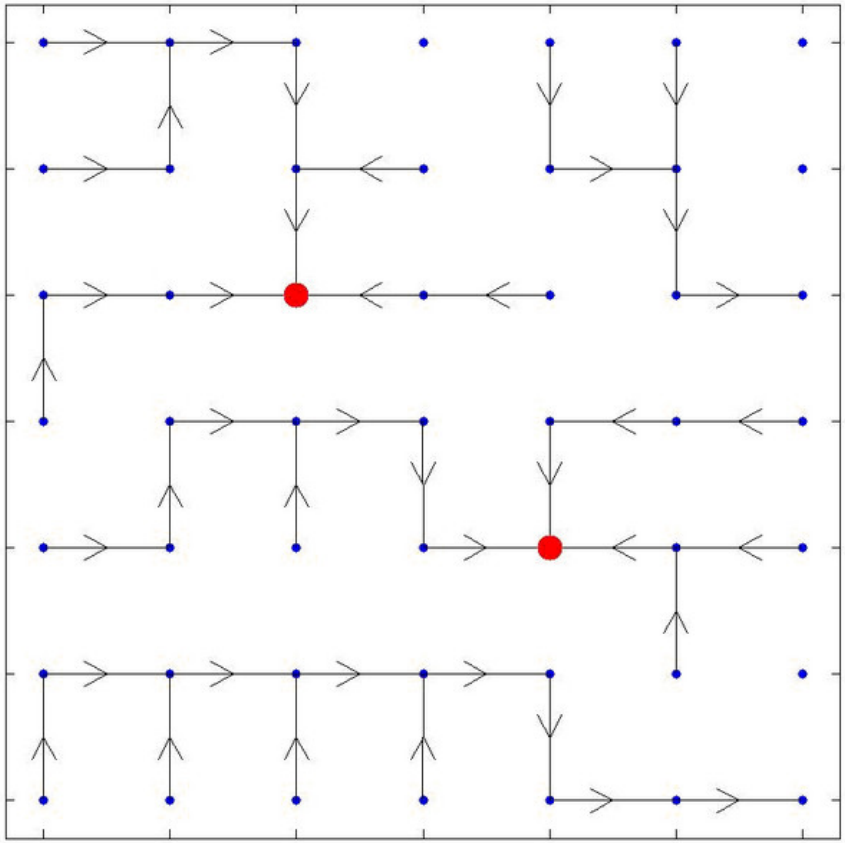
where θ_i is a bias of neuron (for simplification we did not mention it before – it is a constant value which may be provided as input to each neuron).

Energy Landscape



Source: https://en.wikipedia.org/wiki/Hopfield_network

Attractors in State Space

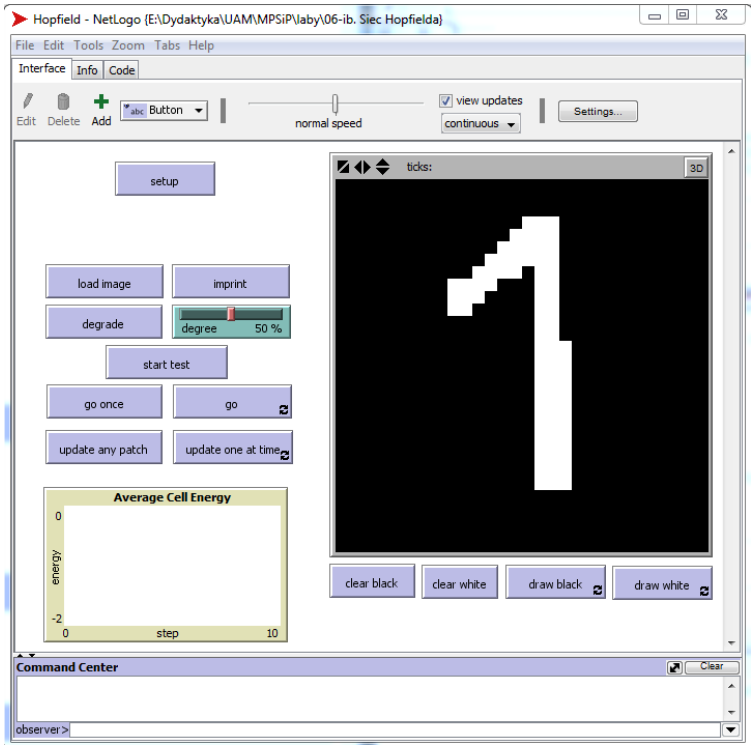


"Two-dimensional representation of motion in state space. Transitions to states outside this fragment are not indicated."

Source: http://www.scholarpedia.org/article/Hopfield_network

Exercise 2

1. Download NetLogo 4.1.3 (https://ccl.northwestern.edu/netlogo/4.1.3/).
2. Download Hopfield Network Simulator (Hopfield.nlogo): link
3. Run the Hopfield network model in NetLogo and experiment with the behavior of the network. `Imprint` – memorize pattern.

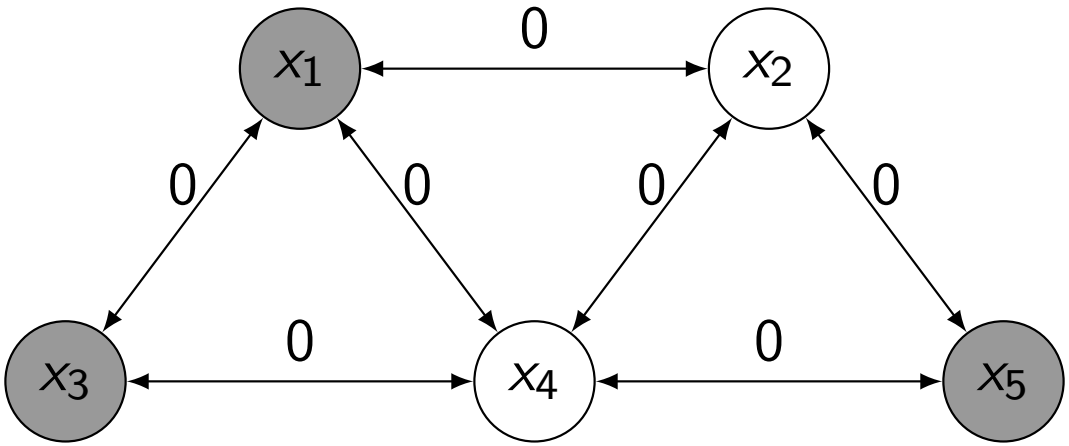


Learning – Hebbian Learning Rule

Learning a pattern $\vec{x} = (x_1, \ldots, x_n)$ is done by using a **Hebb rule**:

$$\Delta w_{ij} = x_i x_j$$

where x_i is a desired value of the neuron for the given pattern.



When we want to store additional patterns, we can simply add them to existing weights. Process of learning a Hopfield net is **incremental**.

Learning – Example

Our goal is to learn the following patterns:

- $\mathbf{A} = [1, -1, 1, -1, 1]$
- $B = [1, 1, 1, -1, -1]$

Changes of weights for A:

$\Delta w_{12} = x_1 \cdot x_2 = 1 \cdot (-1) = -1$
N_1 and N_2 will tend to have opposite values.

$\Delta w_{13} = x_1 \cdot x_3 = 1 \cdot 1 = 1$
N_1 and N_3 will tend to have similar values.

$\Delta w_{14} = x_1 \cdot x_4 = 1 \cdot (-1) = -1$

Learning – Example

Our goal is to learn the following patterns:

- $\mathbf{A} = [1, -1, 1, -1, 1]$
- $B = [1, 1, 1, -1, -1]$

Changes of weights for A:

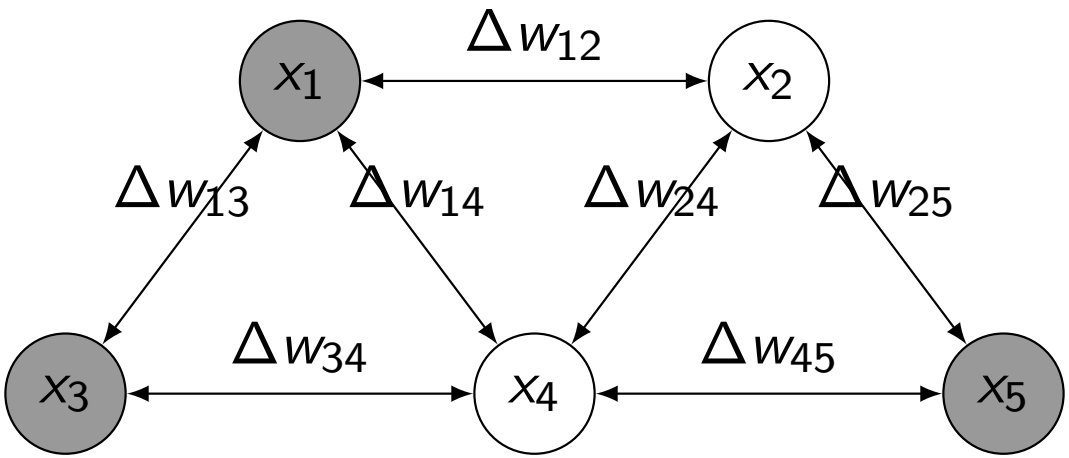
$$\Delta w_{12} = x_1 \cdot x_2 = 1 \cdot (-1) = -1$$
$$\Delta w_{13} = x_1 \cdot x_3 = 1 \cdot 1 = 1$$
$$\Delta w_{14} = x_1 \cdot x_4 = 1 \cdot (-1) = -1$$

$$\Delta w_{23} = x_2 \cdot x_3 = (-1) \cdot 1 = -1$$
$$\Delta w_{24} = x_2 \cdot x_4 = (-1) \cdot (-1) = 1$$
$$\Delta w_{25} = x_2 \cdot x_5 = (-1) \cdot 1 = -1$$

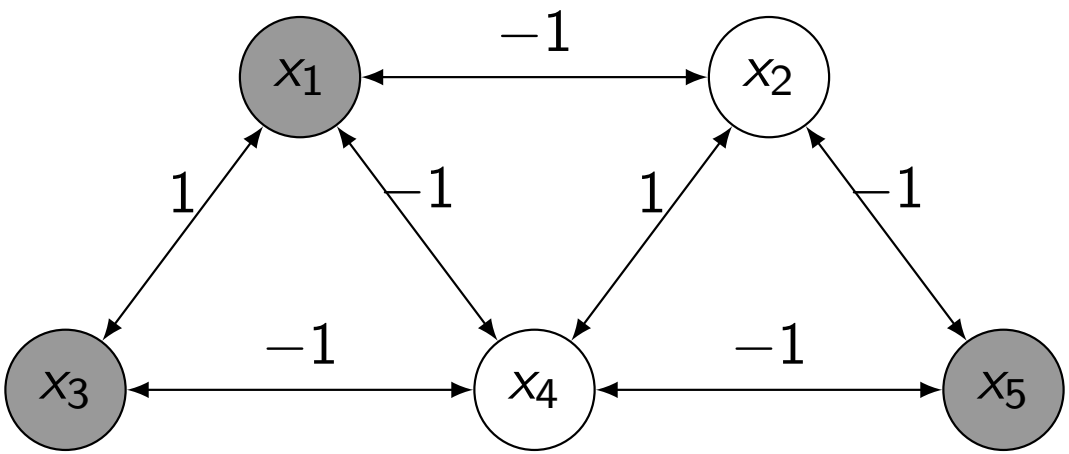
$$\Delta w_{34} = x_3 \cdot x_4 = 1 \cdot (-1) = -1$$
$$\Delta w_{35} = x_3 \cdot x_5 = 1 \cdot 1 = 1$$

$$\Delta w_{45} = x_4 \cdot x_5 = (-1) \cdot 1 = -1$$

Learning – Example



Learning – Example



Learning – Example

Our goal is to learn the following patterns:

- $A = [1, -1, 1, -1, 1]$
- $\mathbf{B} = [1, 1, 1, -1, -1]$

Changes of weights for B:

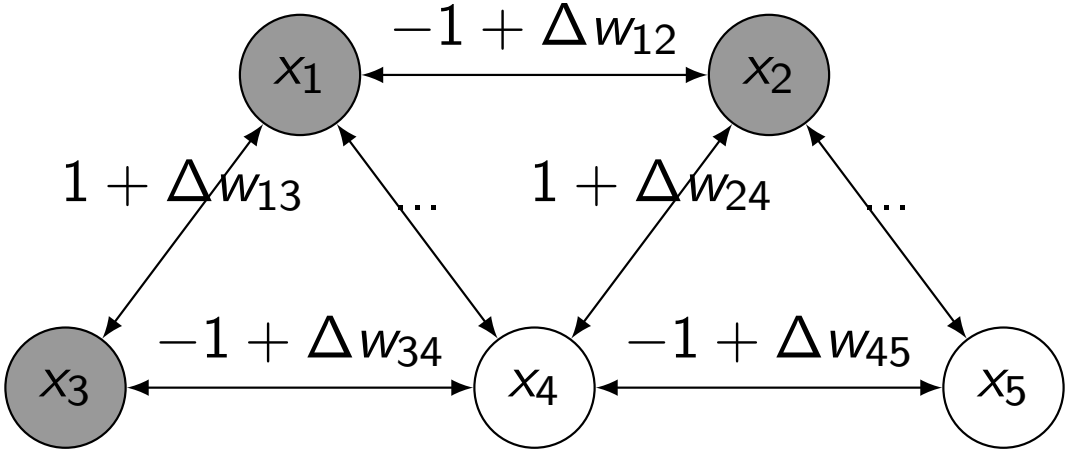
$$\Delta w_{12} = x_1 \cdot x_2 = 1 \cdot 1 = 1$$
$$\Delta w_{13} = x_1 \cdot x_3 = 1 \cdot 1 = 1$$
$$\Delta w_{14} = x_1 \cdot x_4 = 1 \cdot (-1) = -1$$

$$\Delta w_{23} = x_2 \cdot x_3 = 1 \cdot 1 = 1$$
$$\Delta w_{24} = x_2 \cdot x_4 = 1 \cdot (-1) = -1$$
$$\Delta w_{25} = x_2 \cdot x_5 = 1 \cdot (-1) = -1$$

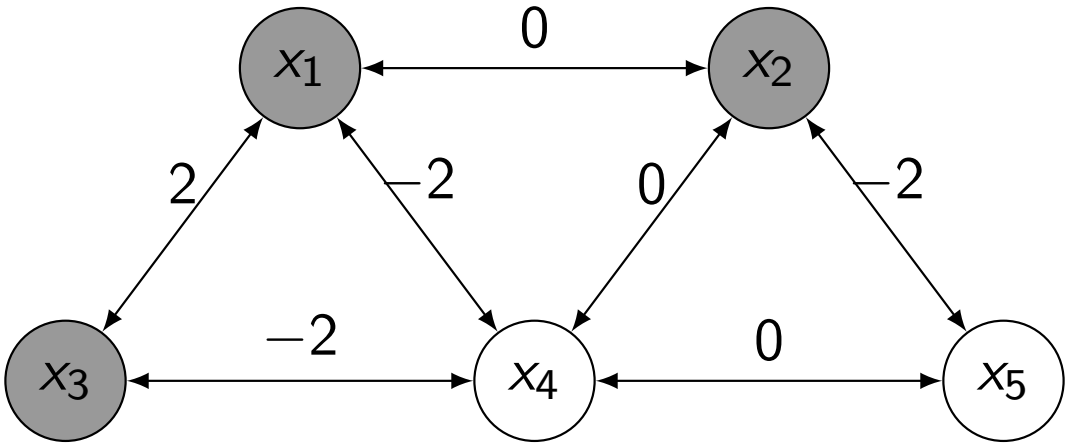
$$\Delta w_{34} = x_3 \cdot x_4 = 1 \cdot (-1) = -1$$
$$\Delta w_{35} = x_3 \cdot x_5 = 1 \cdot (-1) = -1$$

$$\Delta w_{45} = x_4 \cdot x_5 = (-1) \cdot (-1) = 1$$

Learning – Example



Learning – Example



Summary of Hopfield Networks

Advantages:

- An item can be retrieved by just knowing part of its content.
- Fast learning.
- Robust against "hardware" damage.

Disadvantages:

- Small memory capacity – interference of weights.

Outline of the Presentation

What Is Deep Learning?

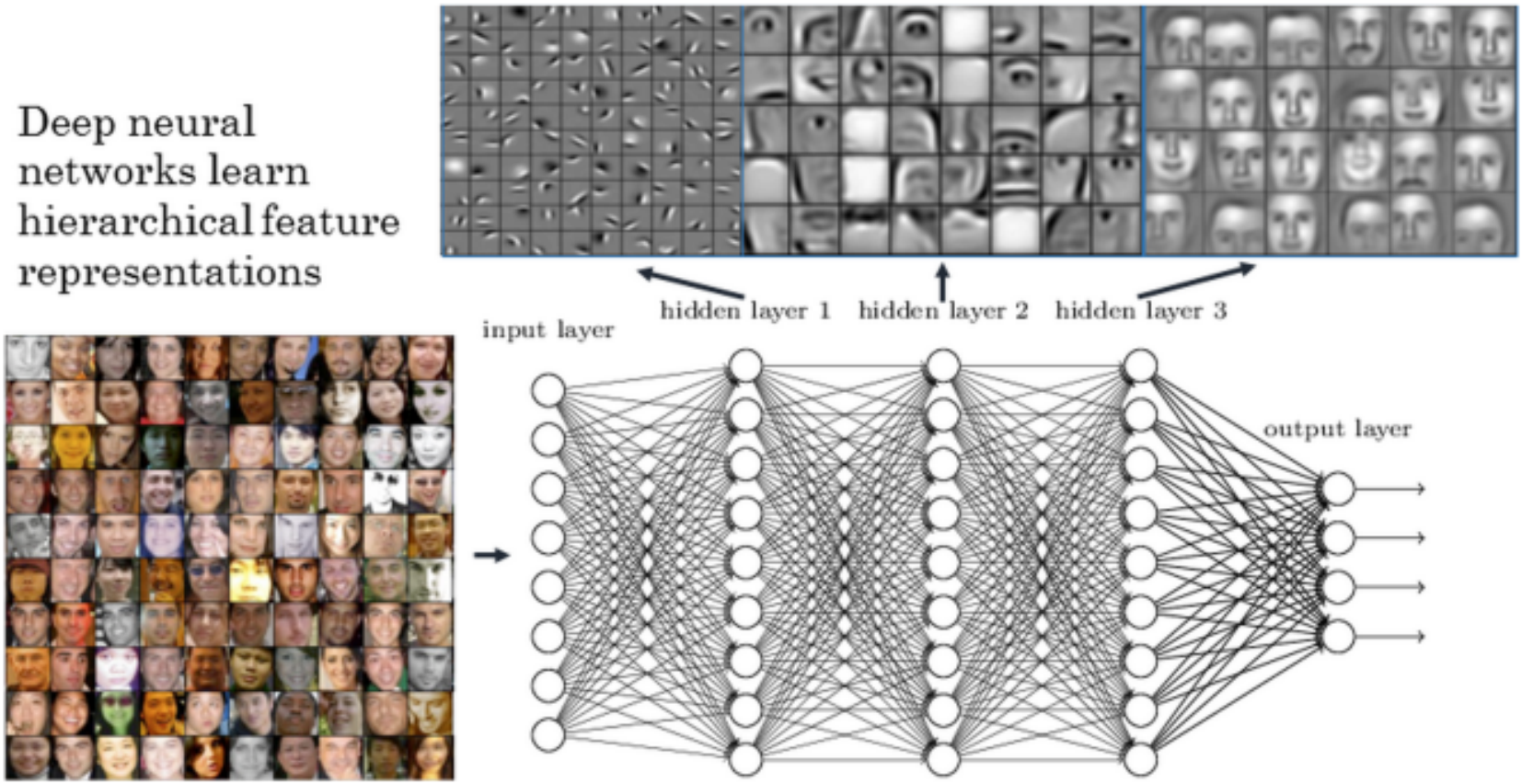
Deep neural network

An artificial neural network with many layers.

- Learning such networks was problematic due to the vanishing gradient problem in the backpropagation algorithm. Error propagated to the earliest layers was too weak to allow for effective learning.
- **Deep Learning** – A collection of techniques, rather than a single algorithm, to learn networks with many layers.

Deep Learning

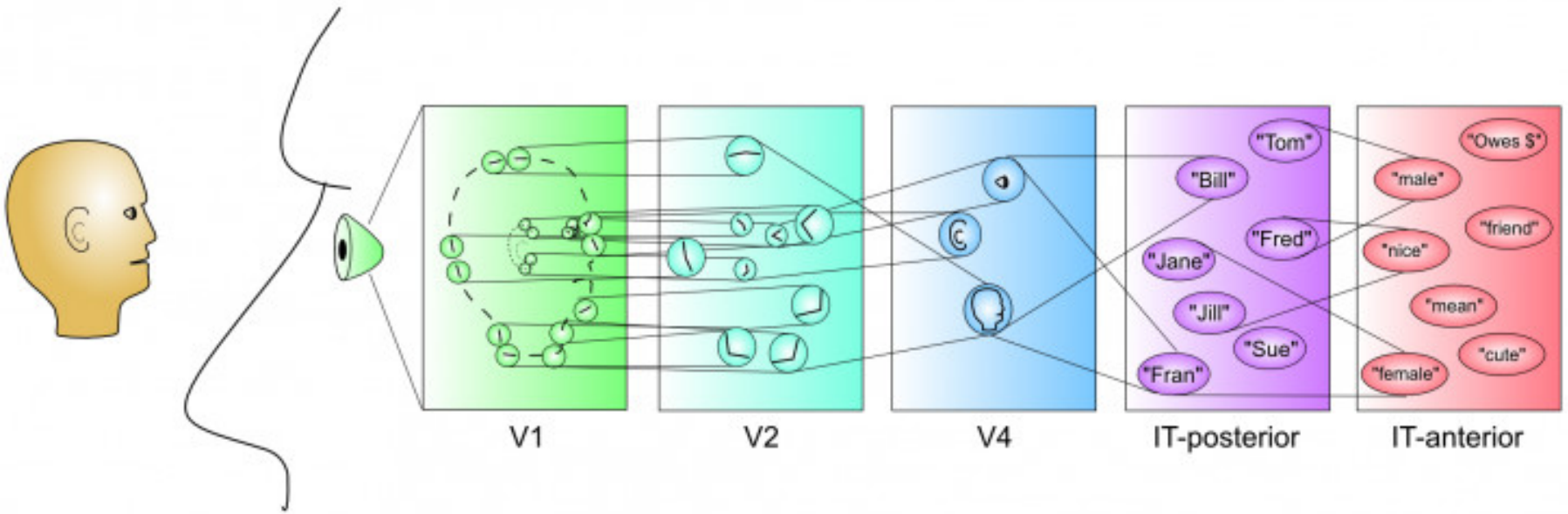
Core idea: **Hierarchical feature learning**



Source: https://nivdul.wordpress.com/2015/11/17/exploring-deep-learning-with-li-zhe/

Deep Learning

Core idea: **Hierarchical feature learning**



Processing of information in the visual cortex is done in the similar way.

Source: https://grey.colorado.edu/CompCogNeuro/index.php/CCNBook/Perception

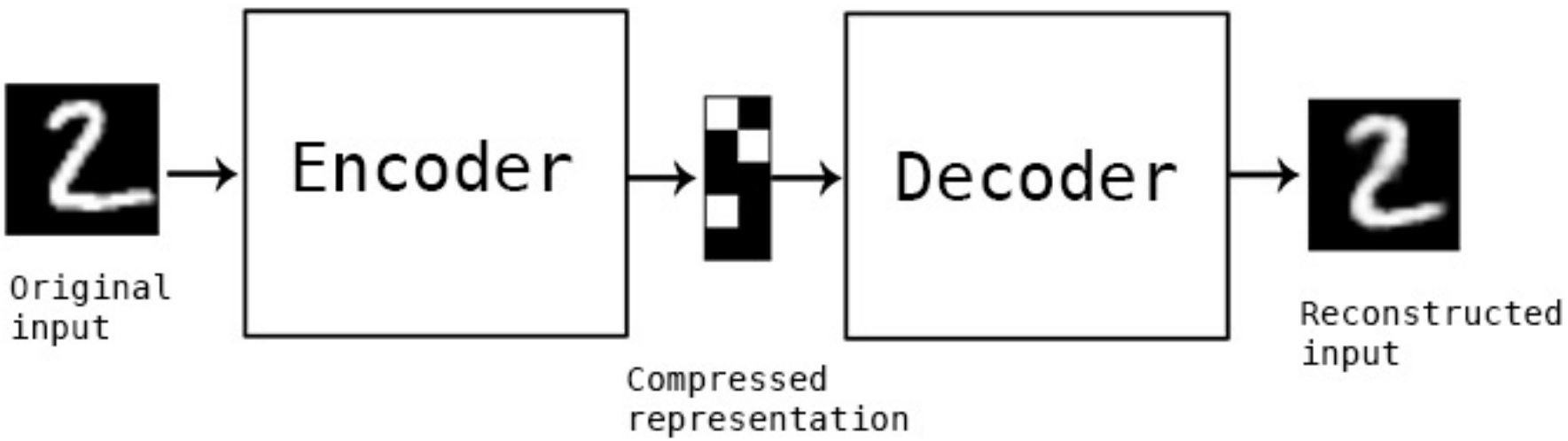
Types of Deep Machine Learning

Informal classification of deep machine learning approaches presented by Pedro Domingos in [2]:

1. Stacked autoencoders.
2. Based on Boltzmann machines.
3. Convolutional neural networks.

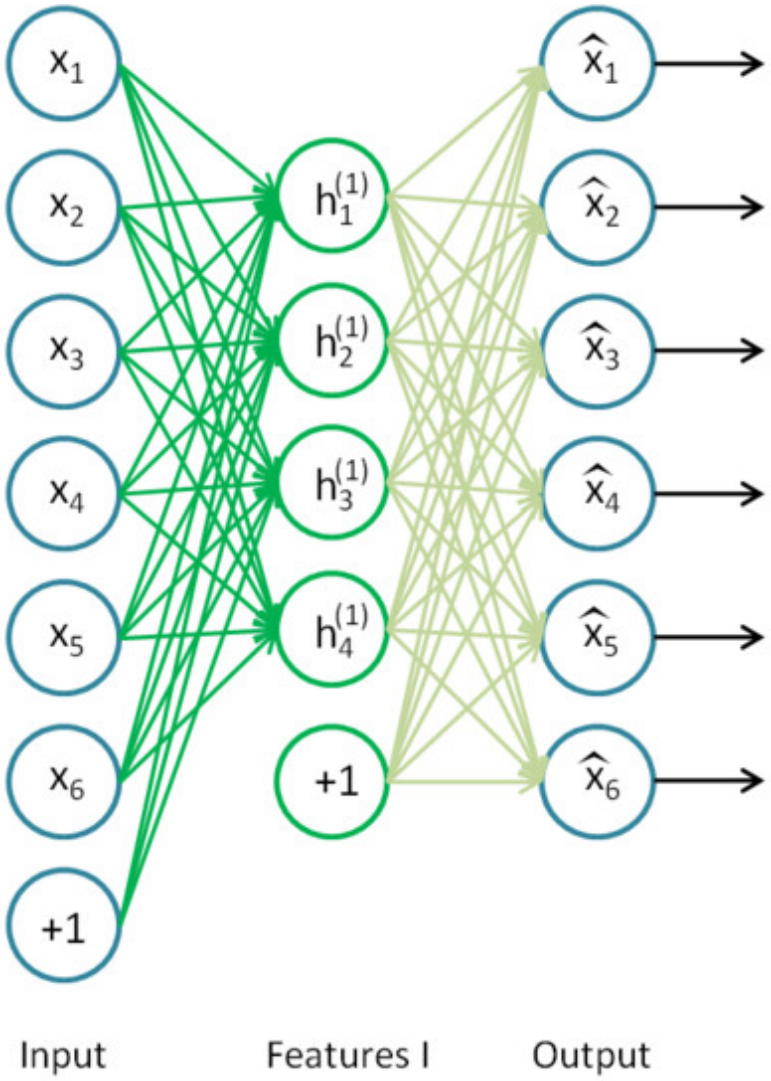
1. Autoencoder

- **Idea:** encode (compress) original data and then decode it.
- Network is learned by backprop to return the output similar to the original data.
- Autoencoder must find the most important information, so it suffices for reconstruction.



Source: https://blog.keras.io/building-autoencoders-in-keras.html

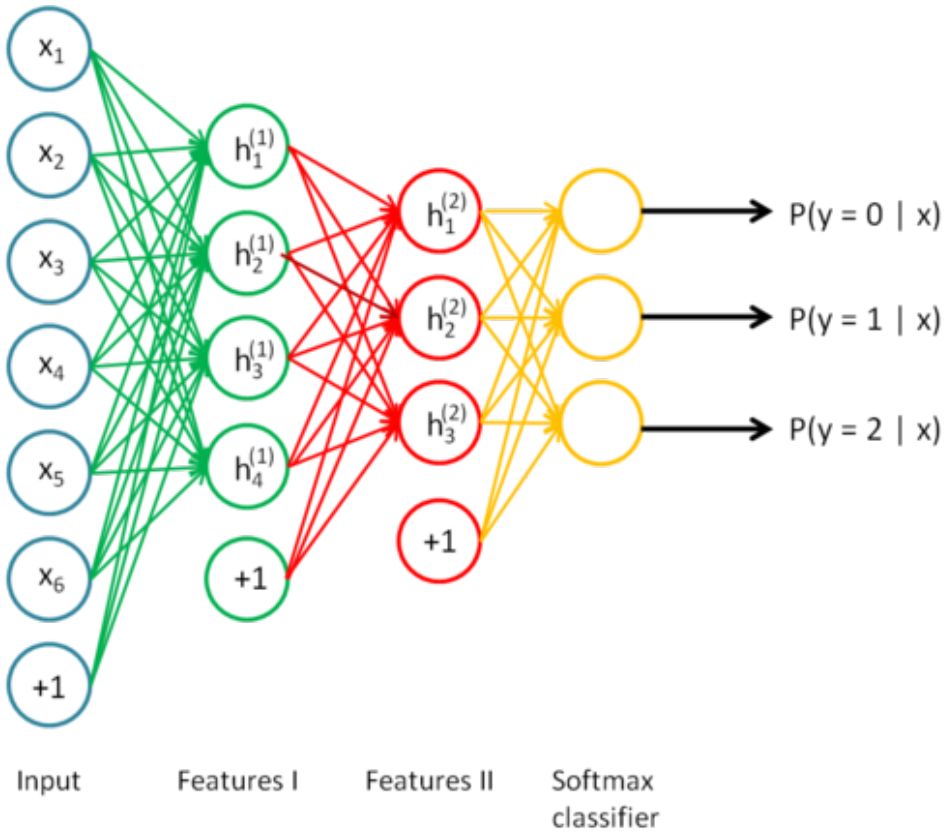
1. Autoencoder – Architecture



Source: http://ufldl.stanford.edu/wiki/index.php/Stacked_Autoencoders

1. Stacked Autoencoders

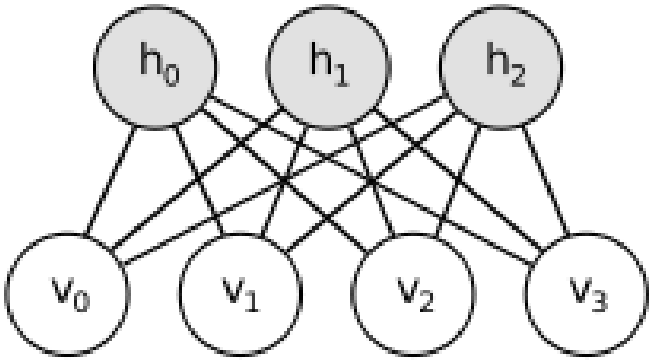
- Layers are being pretrained sequentially, one by one.
- Decoding layers are discarded.
- Softmax classifier transforms features into probabilities of decisions.
- After this *pretraining*, backprop is run on the whole network.



Source: http://ufldl.stanford.edu/wiki/index.php/Stacked_Autoencoders

2. Boltzmann Machines

- Stochastic version of a Hopfield net with additional hidden neurons.
- Name comes from the *Boltzmann probability distribution*.
- **Restricted Boltzmann Machines** – easier to learn version with distinguished *visible layer* and *hidden layer*. No two nodes in the same layer share a connection.



Source: http://deeplearning.net/tutorial/rbm.html

2. Boltzmann Machines – Learning Phases

- Learning in this approach works in 2 phases:
  1. **Forward pass** – information from visible layer is passed to the hidden layer. Neurons may or may not activate, depending on the probability function.
  2. **Backward pass** – information from hidden layer passed to the visible layer for reconstruction.

- Reconstruction is then compared with the original data (we want to minimize differences between them).
- Boltzmann Machines are used to extract the most important features of the data (hidden layer).

2. Deep Belief Nets

Deep Belief Net

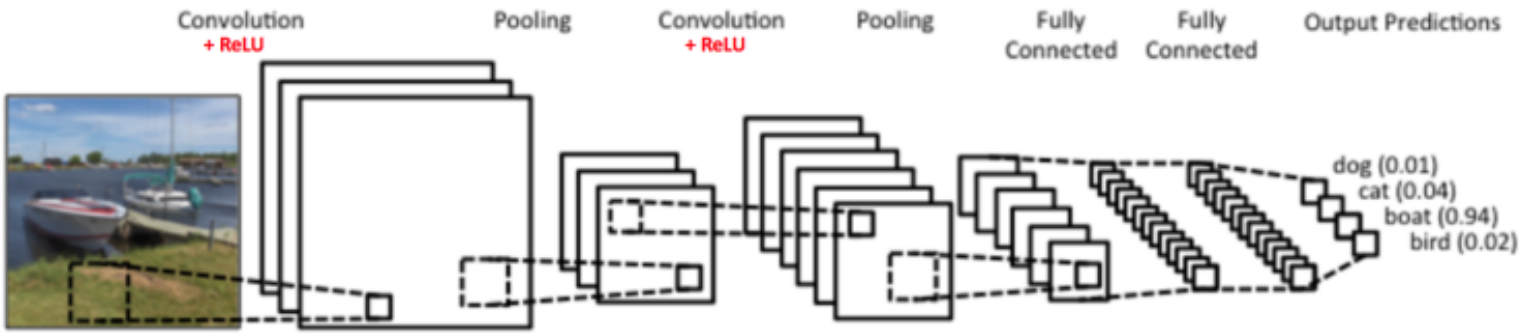
A network which combines several restricted Boltzmann machines.

Learning is done by considering hidden layer of one Boltzmann machine as a visible layer of the next machine on the stack (similar motive as with the stacked autoencoders).

After a network is trained, it is basically only learned to extract important features. In the next step, traditional backpropagation is used in the supervised scenario. Thus, Deep Belief Nets act as *initialization of the weights*.

3. Convolutional Neural Networks

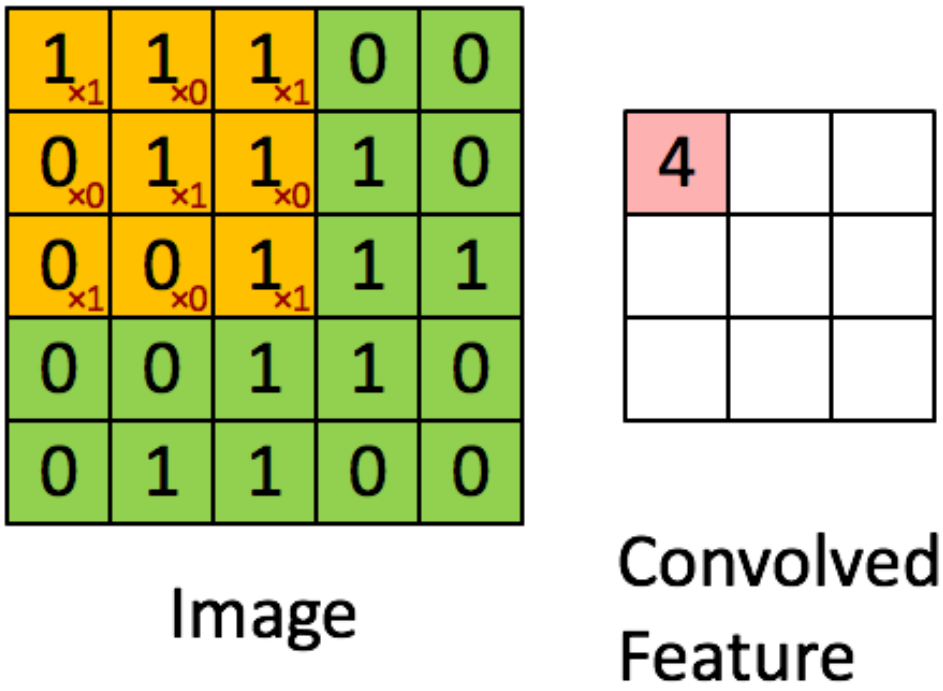
- Best for classifying images.
- Consist of 4 types of layers:
  - convolution layer
  - ReLU layer
  - pooling layer
  - fully connected layer (final classification)
- Convolution and pooling layers are often "stacked" on each other.



Source: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

3. Convolution Layer

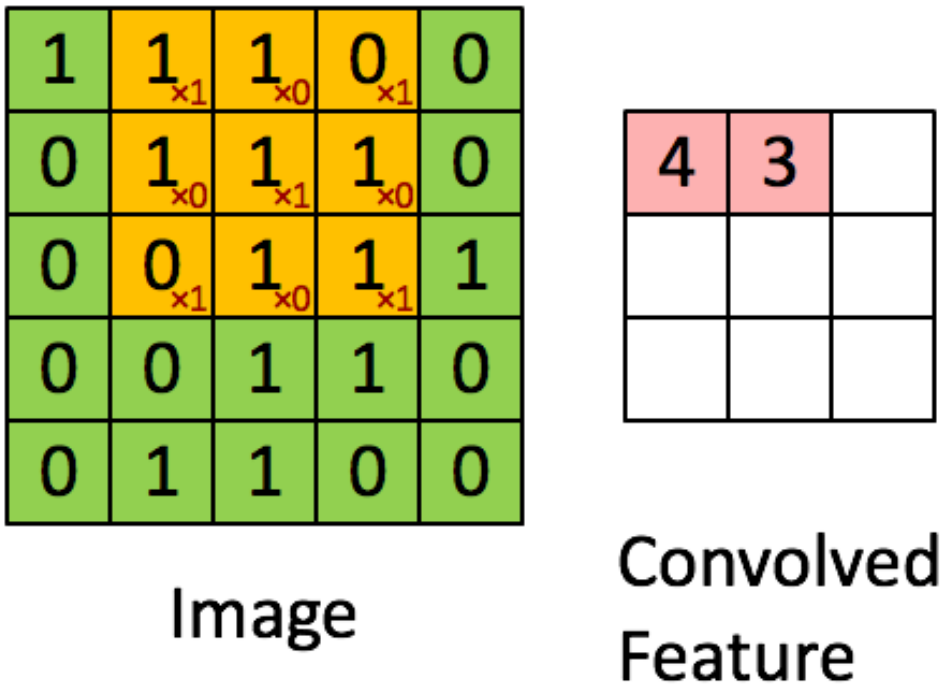
- *Convolution* is a mathematical operation, about which we talked a little bit.



Source: http://ufldl.stanford.edu/tutorial/supervised/FeatureExtractionUsingConvolution/

3. Convolution Layer

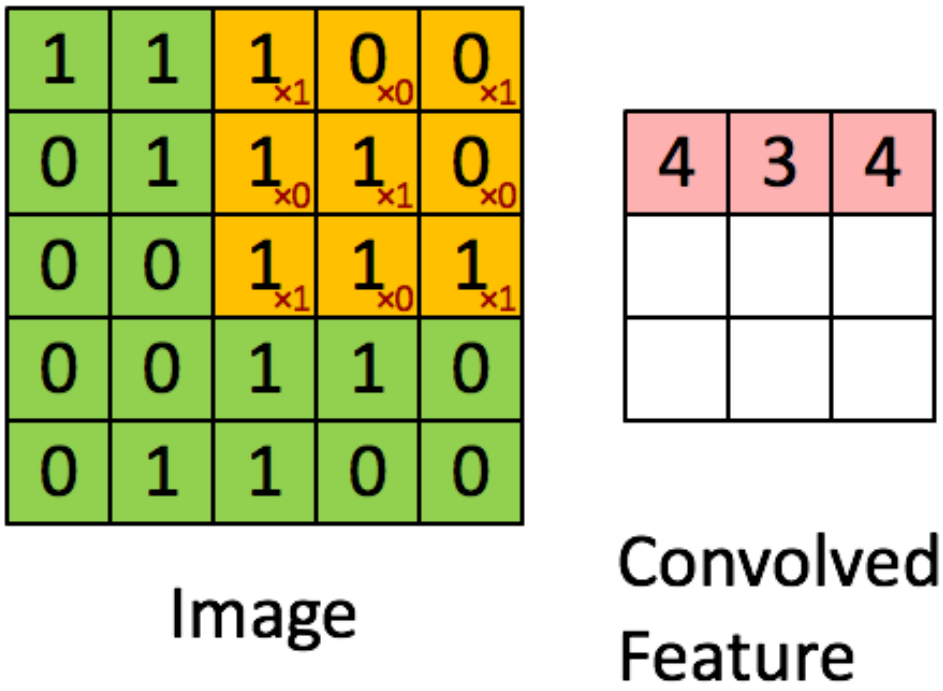
- *Convolution* is a mathematical operation, about which we talked a little bit.



Source: http://ufldl.stanford.edu/tutorial/supervised/FeatureExtractionUsingConvolution/

3. Convolution Layer

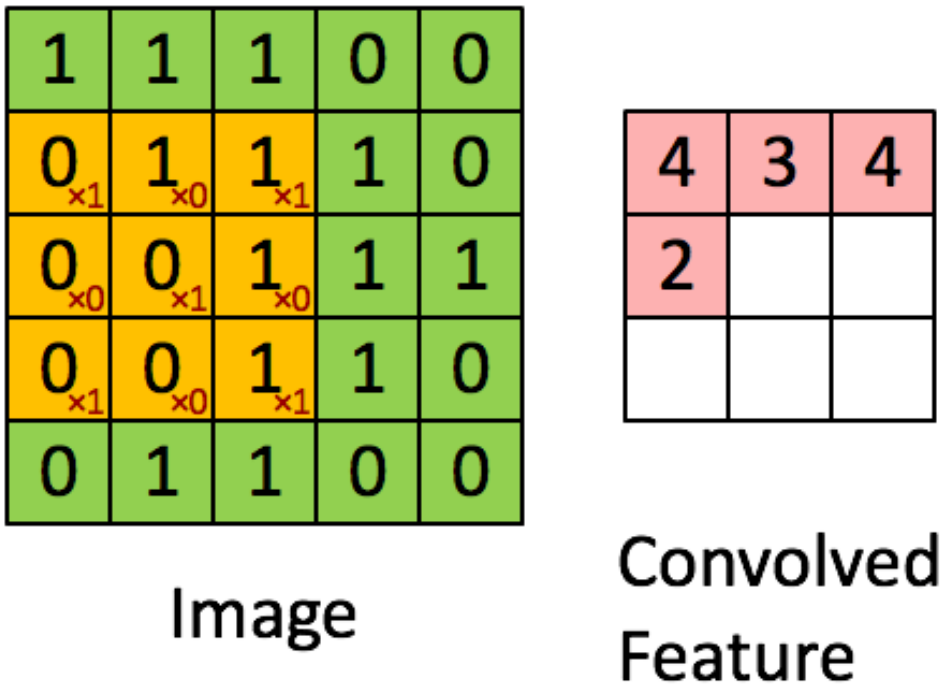
- *Convolution* is a mathematical operation, about which we talked a little bit.



Source: http://ufldl.stanford.edu/tutorial/supervised/FeatureExtractionUsingConvolution/

3. Convolution Layer

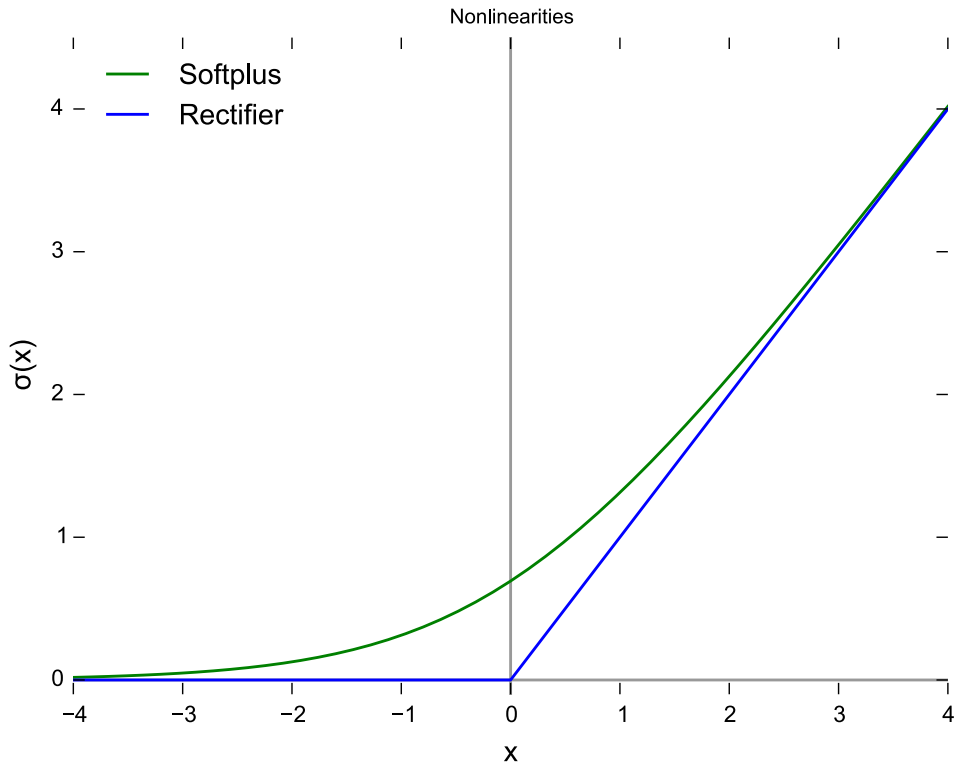
- *Convolution* is a mathematical operation, about which we talked a little bit.



Source: http://ufldl.stanford.edu/tutorial/supervised/FeatureExtractionUsingConvolution/

3. ReLU Layer

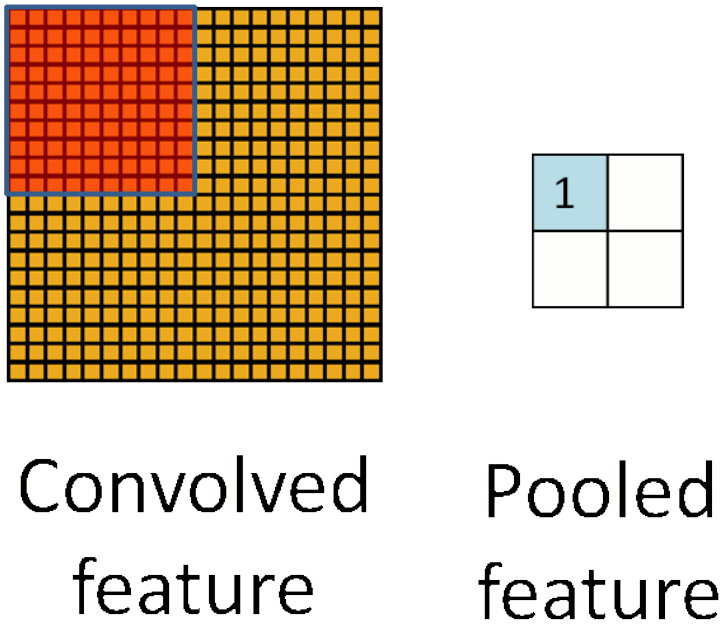
- Build from Rectified Linear Units (ReLU).
- Those units are much easier to train when in the deep layers.



Source: https://en.wikipedia.org/wiki/Rectifier_(neural_networks)

3. Pooling Layer

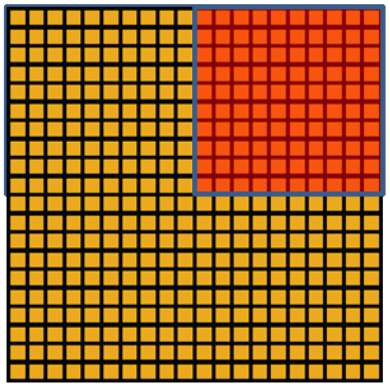
- Gathers statistics from the convolution layers.
- Usually used are max and mean pooling layers.



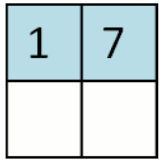
Source: http://ufldl.stanford.edu/tutorial/supervised/Pooling/

3. Pooling Layer

- Gathers statistics from the convolution layers.
- Usually used are max and mean pooling layers.



Convolved feature

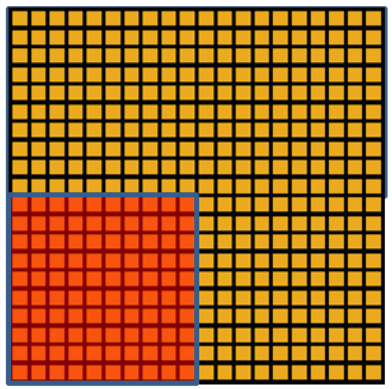


Pooled feature

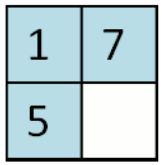
Source: http://ufldl.stanford.edu/tutorial/supervised/Pooling/

3. Pooling Layer

- Gathers statistics from the convolution layers.
- Usually used are max and mean pooling layers.



Convolved feature

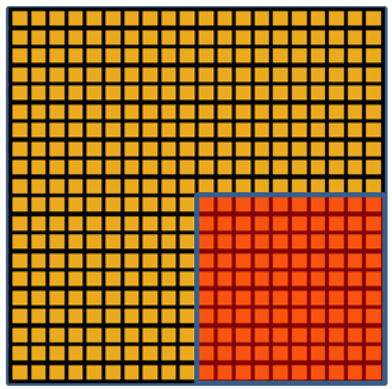


Pooled feature

Source: http://ufldl.stanford.edu/tutorial/supervised/Pooling/

3. Pooling Layer

- Gathers statistics from the convolution layers.
- Usually used are max and mean pooling layers.



Convolved feature

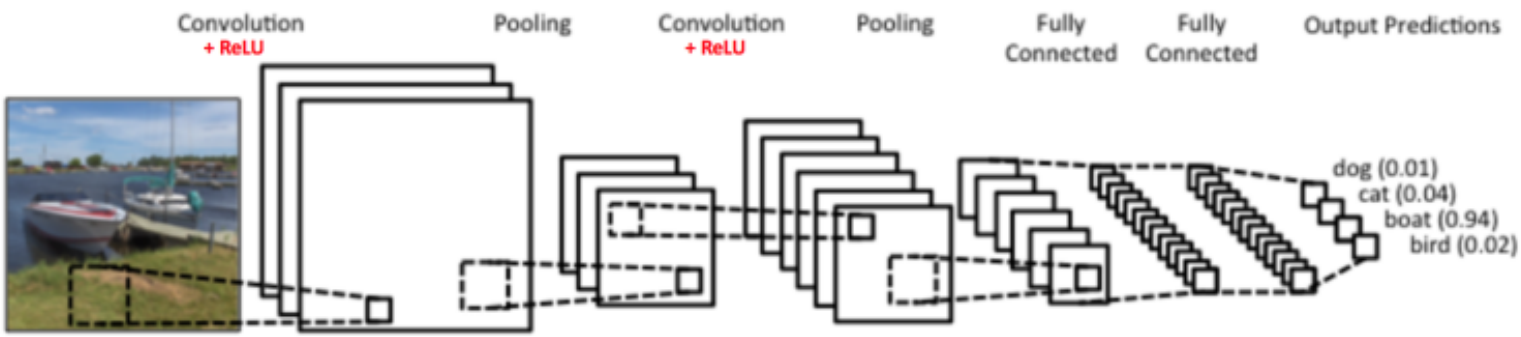
1	7
5	9

Pooled feature

Source: http://ufldl.stanford.edu/tutorial/supervised/Pooling/

3. Fully Connected Layer

- Standard multilayer perceptron layer used to perform classification on the extracted set of high-level features.



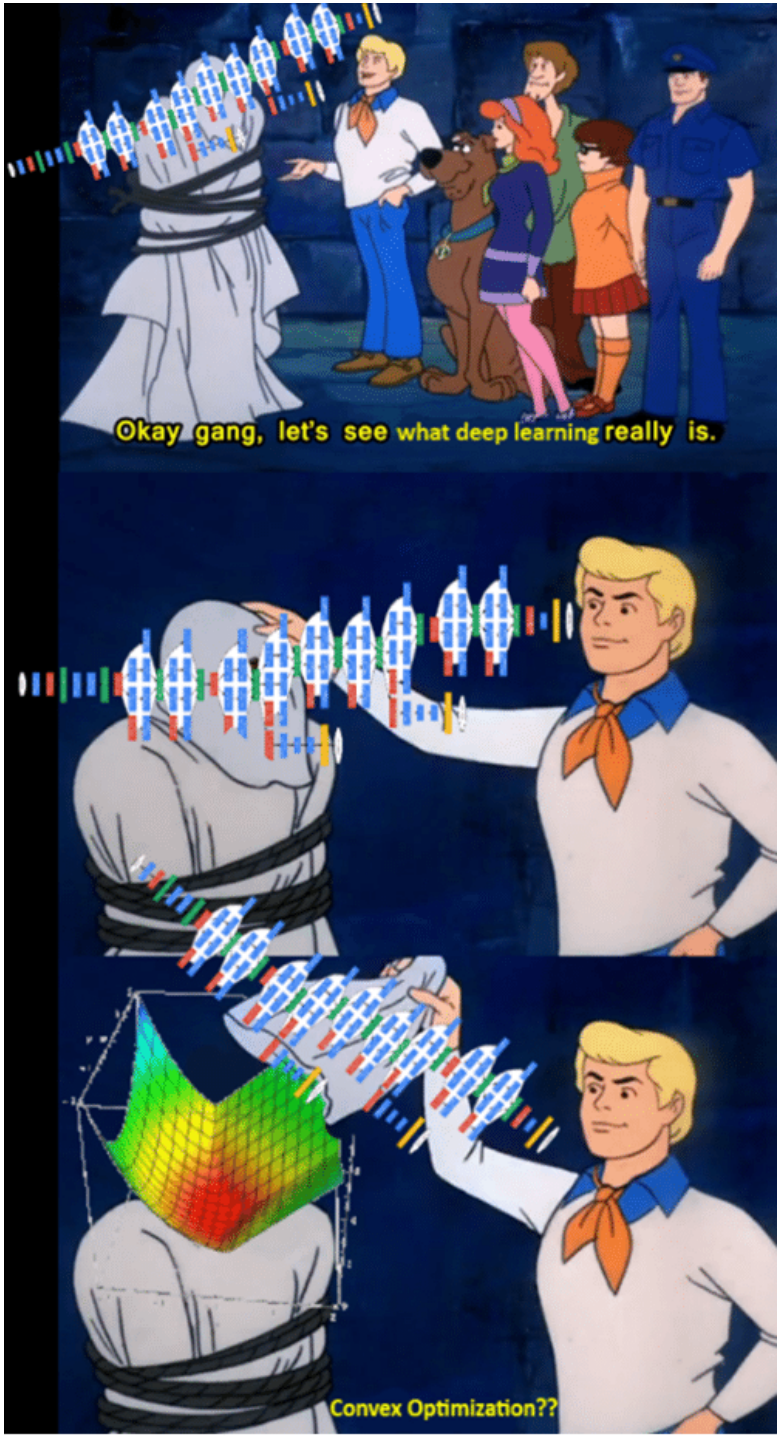
Source: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

Achievements – AlphaGo

- Computer program developed by Google DeepMind for playing Go.
- DeepMind was a start-up founded in 2010 and focusing on artificial intelligence. It was acquired by Google in 2014.

Achievements – AlphaGo

- 9–15.03.2016: AlphaGo played a five-game match against Lee Sedol, who is one of the top Go players in the world.
- Result: **(AlphaGo) 4 : 1 (Lee Sedol).**
- AlphaGo was trained using a database of around 30 millions moves. Professional Go players generally learn in a similar way, i.e. by studying games of the masters, however on a smaller scale. . .
- Go is considered to be very hard for artificial intelligence because of huge number of possible positions.


Okay gang, let's see what deep learning really is.
Convex Optimization??

Outline of the Presentation

Self-Organizing Maps (SOM)

- Other names: *Kohonen map*, *Kohonen network* (they were introduced by the Finnish professor Teuvo Kohonen in the 1980s).
- This type of networks realizes ***unsupervised learning***.
- Instead of error-driven learning, ***competitive learning*** is used, in which neurons compete for the right to respond to a subset of the input data.
- *Dimensionality reduction* of the original data (described on several attributes) to usually 2 or 3 dimensions (neuron's position in the grid). This grid of neurons is called ***'map'***.

- **Main idea:** Similar items from the training data will tend to be close to each other in the created map.

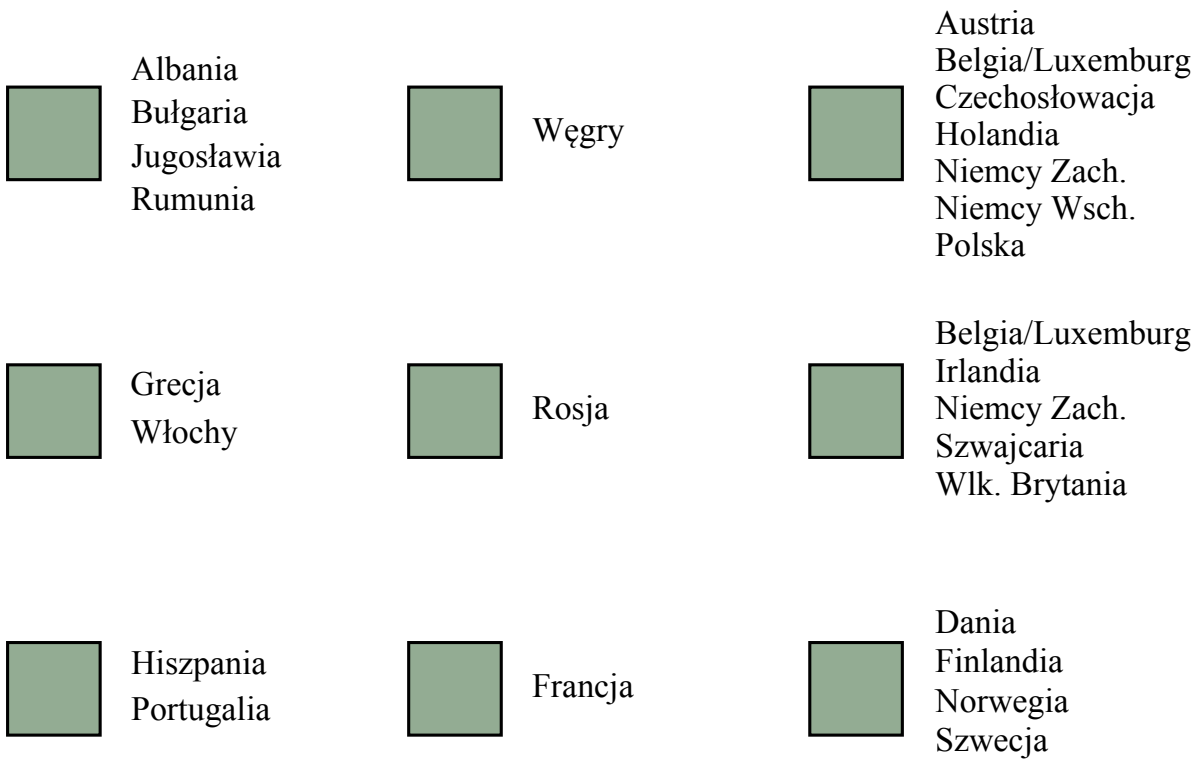
Self-Organizing Maps (SOM)

- Neurons in the SOM "imitate" inputs, i.e. they can be thought of as "artificial examples".
- Real examples (inputs) support neuron which is the most similar to them in terms of distance (here is when competition happens). This in turn makes neuron (aka "artificial example") even more similar to them, or more precisely: their average.
- The above process runs iteratively.
- More details after an example...

An Example of SOM

Exemplary application of SOM to the data of consumption of proteins of different origins in European countries (each described on 9 attributes).

Country name next to the neuron: distance < 0.5



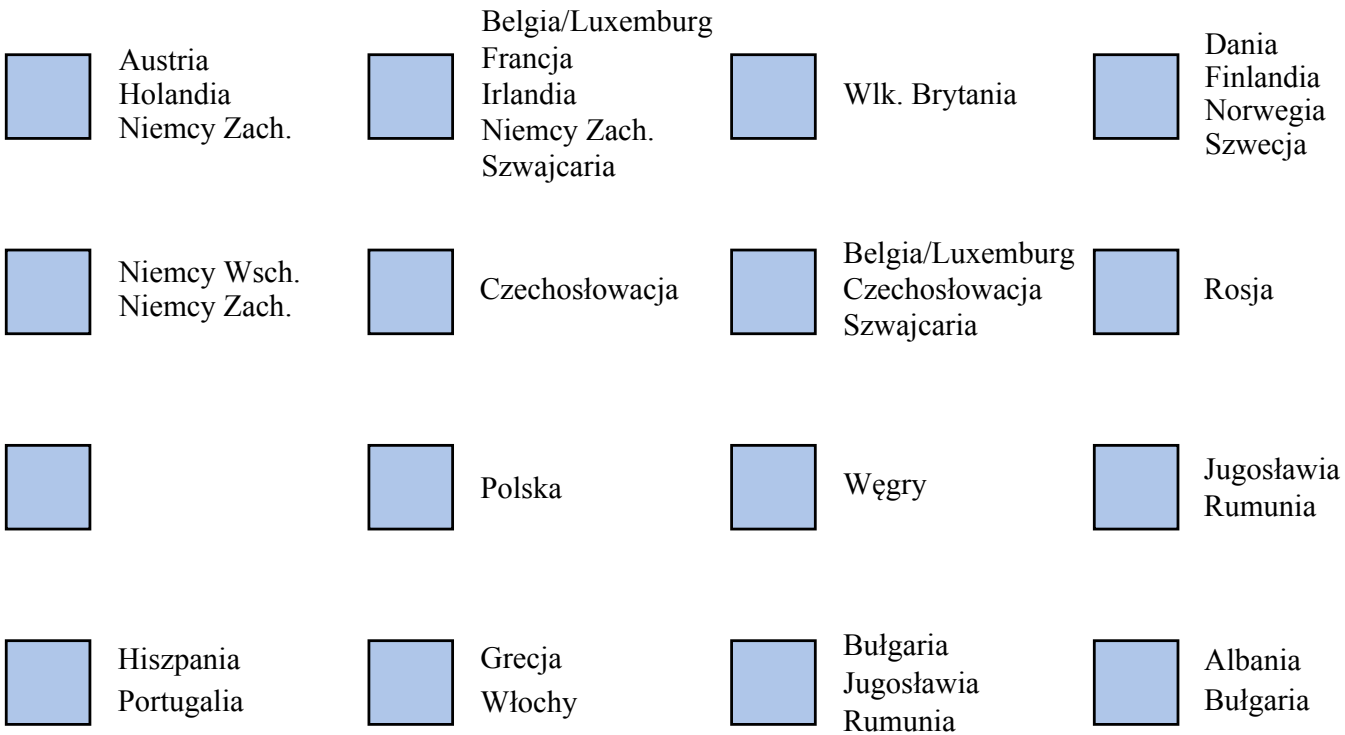
Topology 3x3.

Source: author's own work; report for the subject "Machine Learning and Neural Networks".

An Example of SOM

Exemplary application of SOM to the data of consumption of proteins of different origins in European countries (each described on 9 attributes).

Country name next to the neuron: distance < 0.5



Topology 4x4.

Source: author's own work; report for the subject "Machine Learning and Neural Networks".

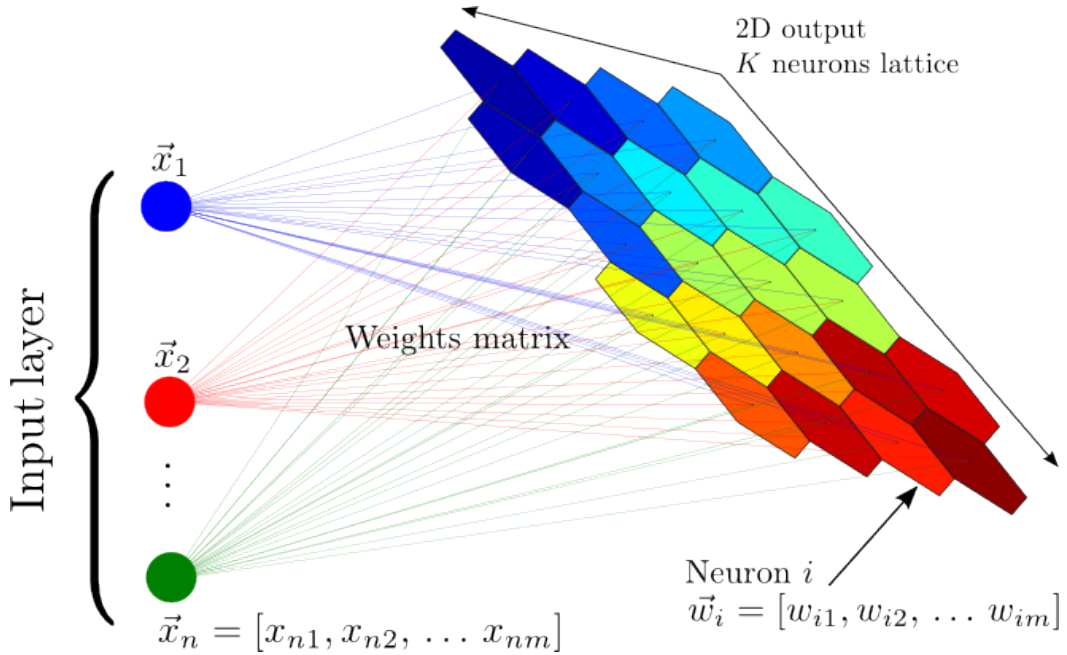
An Example of SOM – Observations

Observations:

- Each neuron in the grid specializes for certain inputs (countries).
- A single neuron can be active for multiple inputs (countries).
- A neuron can be also active for no inputs (by active we mean distance < 0.5 from the input).
- Neighboring neurons have bigger similarity than non-neighboring.

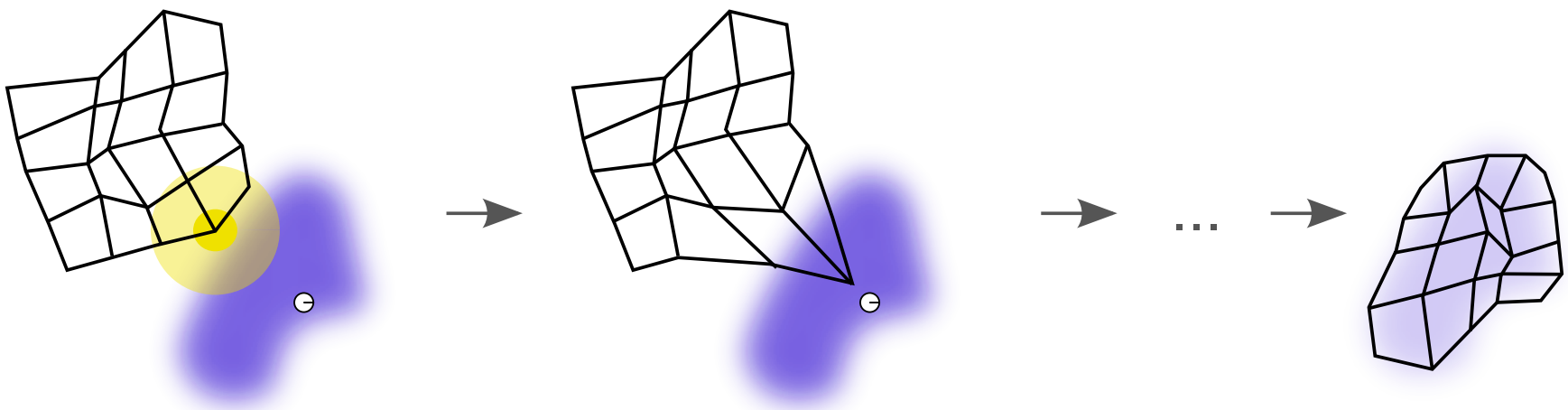
SOM Architecture

- Input layer consists of n units, where n is the number of records in the training data. Each such unit represents a *vector* of values of m attributes for a certain training example.
- Every input is connected to all neurons, so each neuron has n weights.
- Neurons are regularly arranged in a fixed topology.



Self-Organizing Maps – Learning

- **Aim of the learning:** create such a net of neurons that it's different parts will respond similarly to certain input patterns.
- **Competitive learning** – updated will be only weights of the neuron most similar to the currently considered pattern x_i and, in a lesser degree, weights of its nearest neighbors.



Self-Organizing Maps – Learning

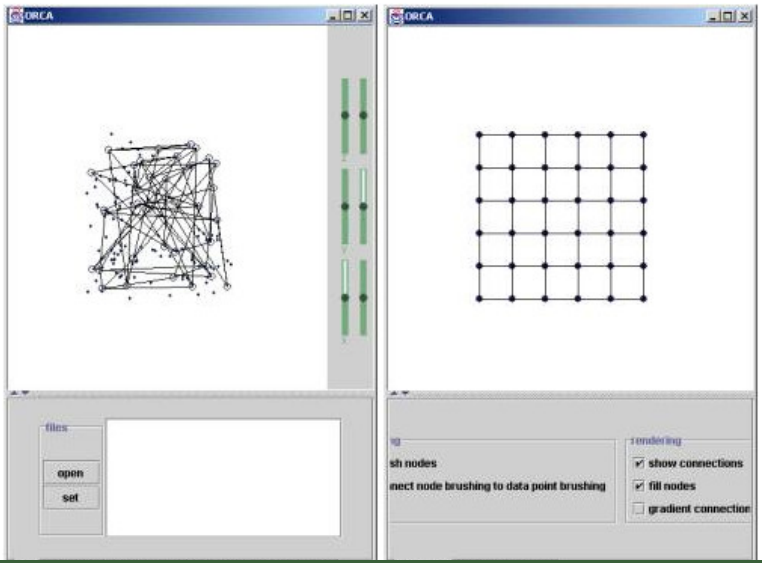
1. Randomize weight vectors for each node (neuron).
2. Take random pattern x_i.
3. Find such a node N_k that Euclidean distance between it and pattern x_i is minimal.
4. Move N_k and its neighbors (N_j) even closer towards x_i by updating weights:
   $w_j(t+1) = w_j(t) + \Theta(j,k,t) \cdot (x_i - w_j(t))$
   where:
   j – index of the currently considered node
   k – index of the closest node to x_i
   t – time
   $\Theta(j,k,t)$ – neighborhood function, which determines how strong is learning for N_j.
5. Go back to point 2 if time t lesser than a certain threshold.

Exercise 1

1. Download SOM Visualization Software (http://jsomap.sourceforge.net/ee547/somviz.zip).
2. To start program you must open a terminal and type:

   java -cp .;jsomap.jar;orca.jar;jama.jar examples.SOMExample data/DATAFILE.txt

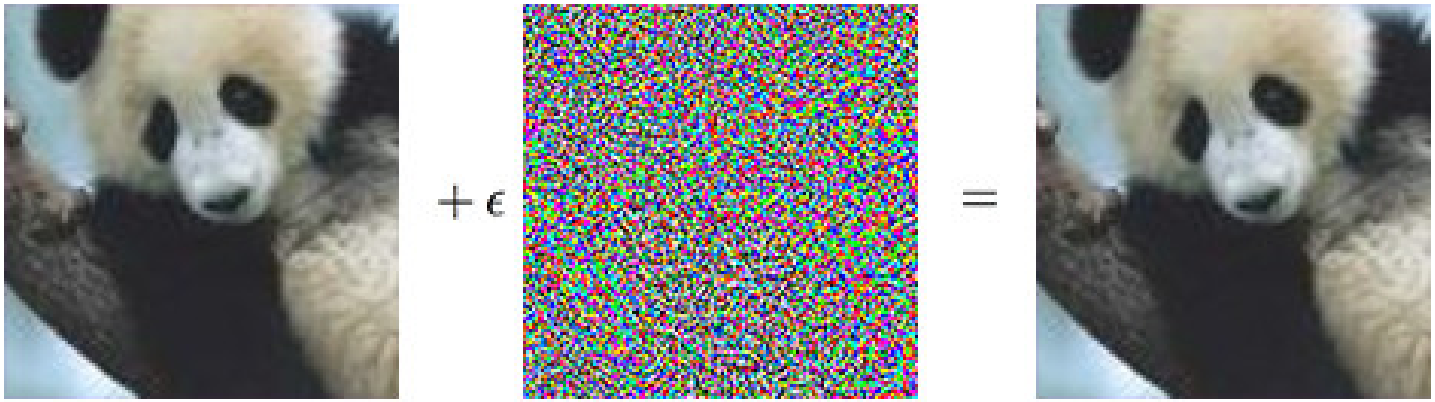
   On Linux replace ';' with ':'. DATAFILE should be replaced by a file name from the directory data/. Choose whichever one you like.
3. Read a short tutorial in the user manual (http://jsomap.sourceforge.net/ee547/UserManual.htm).



Outline of the Presentation

Adversarial Examples

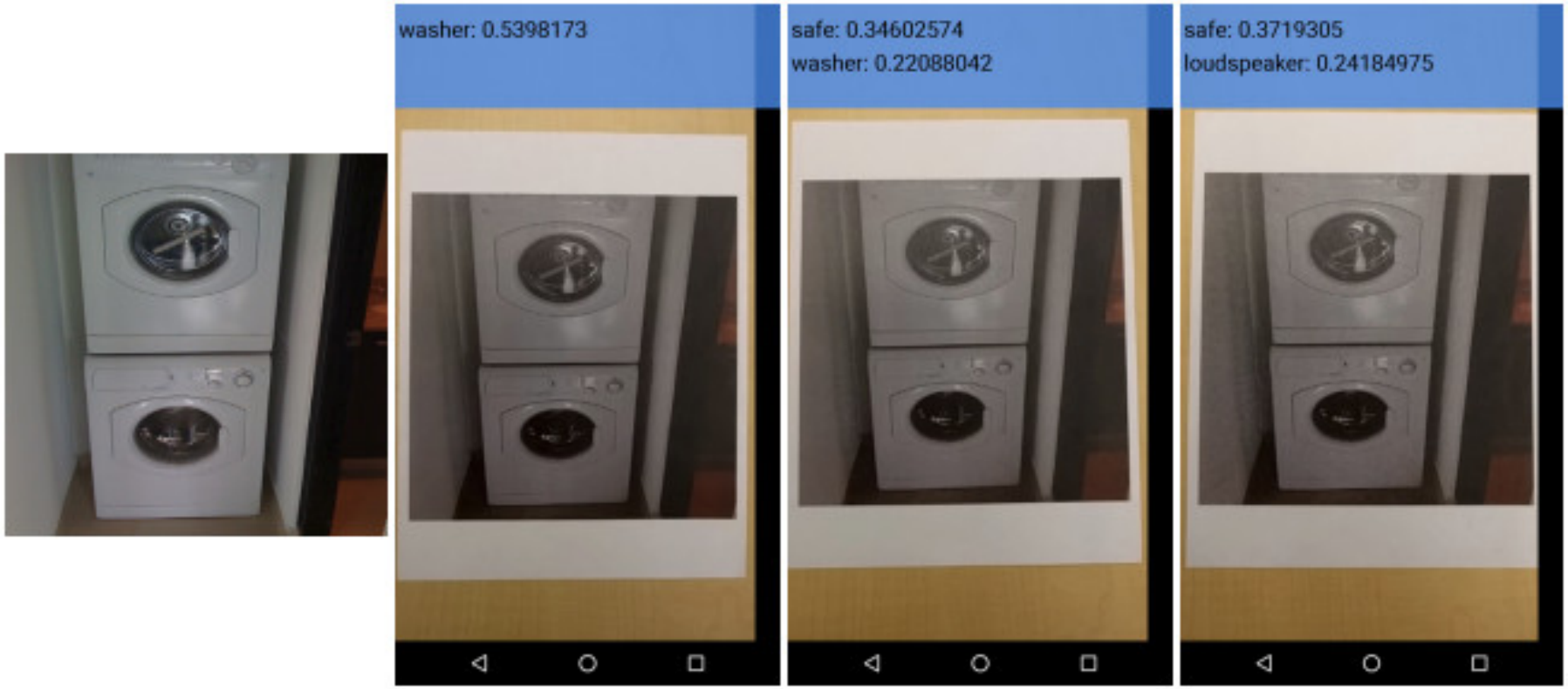
How robust are methods used in AI?



Source: https://blog.openai.com/adversarial-example-research/

Adversarial Examples

How robust are methods used in AI?



(a) Image from dataset (b) Clean image (c) Adv. image, $\epsilon = 4$ (d) Adv. image, $\epsilon = 8$

Source: https://blog.openai.com/adversarial-example-research/

Adversarial Examples

Adversarial Examples – How It Works

How adversarial examples are created:

Most adversarial example construction techniques use the gradient of the model to make an attack. In other words, they look at a picture of an airplane, they test which direction in picture space makes the probability of the "cat" class increase, and then they give a little push (in other words, they perturb the input) in that direction. The new, modified image is mis-recognized as a cat.

(https://blog.openai.com/adversarial-example-research/)

Adversarial Examples – Conclusions

- **Source of the problem:** too many degrees of freedom – space of possible inputs is just too big.
- You can read more at: https://blog.openai.com/adversarial-example-research/.
- Could such techniques potentially work against the brain? Most probably yes. After all, both the brain and deap learning networks are optimized for typical use cases.
- This is somewhat similar to optical illusions.

Final Words

Thank you for your attention.

Bibliography I

[1] *Deep Learning in a Nutshell: History and Training*. https://devblogs.nvidia.com/parallelforall/deep-learning-nutshell-history-training/. Accessed: 2016-04-13.

[2] P. Domingos. *The Master Algorithm: How the Quest for the Ultimate Learning Machine Will Remake Our World*. Penguin Books Limited, 2015. ISBN: 9780241004555.

[3] M. W. Eysenck and M. T. Keane. *Cognitive Psychology: a student's handbook*. 6th ed. Psychology Press, 2010.

[4] J. J. Hopfield. "Neural networks and physical systems with emergent collective computational abilities". In: *pnas* 79 (1982), pp. 2554–2558.

[5] D. Rumelhart, G. Hinton, and R. Williams. "Learning representations by back-propagating errors". In: *Nature* 323 (Oct. 1986), pp. 533–536.

Bibliography II

[6] N. Yadav, A. Yadav, and M. Kumar. *An Introduction to Neural Network Methods for Differential Equations*. 1st ed. Springer, 2015.