PV021: Neural networks

Tomáš Brázdil

Course organization

Course materials:

- **Main:** The lecture
- Neural Networks and Deep Learning by Michael Nielsen

  `http://neuralnetworksanddeeplearning.com/`

  (Extremely well written modern online textbook.)
- Deep learning by Ian Goodfellow, Yoshua Bengio and Aaron Courville

  `http://www.deeplearningbook.org/`

  (A very good overview of the state-of-the-art in neural networks.)

Course organization

Evaluation:

- Project
  - teams of two students
  - implementation of a selected model + analysis of real-world data
  - implementation either in C++, or in Java **without use of any specialized libraries for data analysis and machine learning**
  - real-world data means *unprepared*, cleaning and preparation of data is part of the project!
- Oral exam
  - I may ask about anything from the lecture **including some proofs that occur only on the whiteboard!**
- (Optional) This year we will try to organize a simple data analysis competition (to try the concept). It is up to you whether to participate, or not. During the competition, you may use whatever tools for training neural networks you want.

FAQ

Q: Why English?

A: Couple of reasons. First, all resources about modern neural nets are in English, it is rather cumbersome to translate everything to Czech (combination of Czech and English is ugly). Second, to attract non-Czech speaking students to the course.

Q: Why are the lectures not recorded?

A: Apart from my personal reasons, I want you to participate actively in the lectures, i.e. to communicate with me and other students during the lectures. Also, in my opinion, online lectures should be prepared in a completely different way. I will not discuss this issue any further.

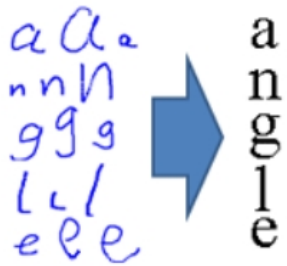
Q: Why we cannot use specialized libraries in projects?

A: In order to "touch" the low level implementation details of the algorithms. You should not even use libraries for linear algebra and numerical methods, so that you will be confronted with rounding errors and numerical instabilities.

Machine learning in general

- Machine learning = construction of systems that may learn their functionality from data

  (... and thus do not need to be programmed.)
  - spam filter
    - learns to recognize spam from a database of "labelled" emails
    - consequently is able to distinguish spam from ham
  - handwritten text reader
    - learns from a database of handwritten letters (or text) labelled by their correct meaning
    - consequently is able to recognize text



  - ...
  - and lots of much much more sophisticated applications ...

- Basic attributes of learning algorithms:
  - **representation**: ability to capture the inner structure of training data
  - **generalization**: ability to work properly on new data

Machine learning in general

Machine learning algorithms typically construct mathematical models of given data. The models may be subsequently applied to fresh data.

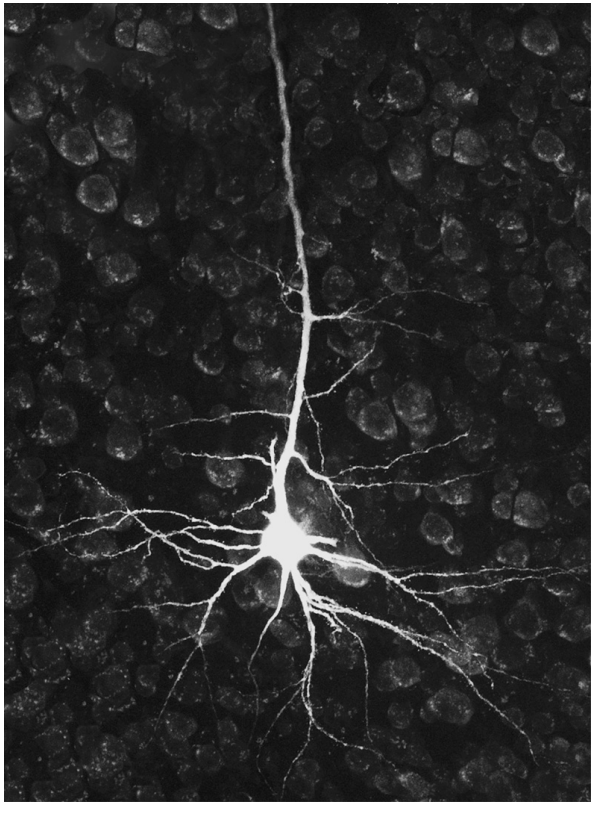
There are many types of models:

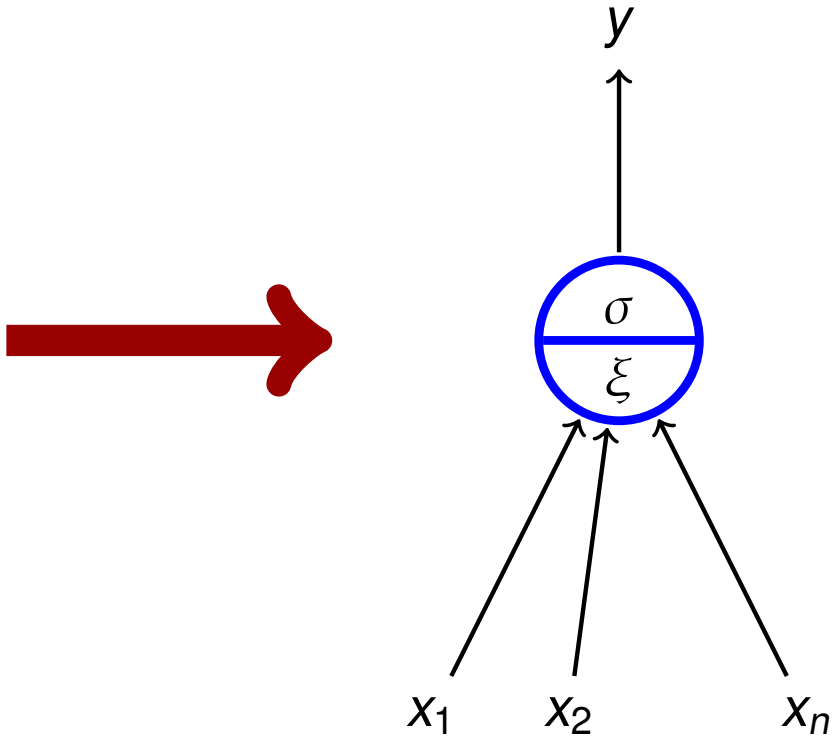
- decision trees
- support vector machines
- hidden Markov models
- Bayes networks and other graphical models
- **neural networks**
- ...

Neural networks, based on models of a (human) brain, form a natural basis for learning algorithms!

Artificial neural networks

- **Artificial neuron** is a *rough mathematical approximation* of a biological neuron.
- **(Aritificial) neural network (NN)** consists of a number of interconnected artificial neurons. "Behavior" of the network is encoded in connections between neurons.





Zdroj obrázku: `http://tulane.edu/sse/cmb/people/schrader/`

Why artificial neural networks?

Modelling of biological neural networks (computational neuroscience).

- simplified mathematical models help to identify important mechanisms
  - How a brain receives information?
  - How the information is stored?
  - How a brain develops?
  - ...
- neuroscience is strongly multidisciplinary; precise mathematical descriptions help in communication among experts and in design of new experiments.

I will not spend much time on this area!

Why artificial neural networks?

Neural networks in machine learning.

- Typically primitive models, far from their biological counterparts (but often inspired by biology).
- Strongly oriented towards concrete application domains:
  - decision making and control - autonomous vehicles, manufacturing processes, control of natural resources
  - games - backgammon, poker, GO
  - finance - stock prices, risk analysis
  - medicine - diagnosis, signal processing (EKG, EEG, ...), image processing (MRI, roentgen, ...)
  - text and speech processing - automatic translation, text generation, speech recognition
  - other signal processing - filtering, radar tracking, noise reduction
  - ...

I will concentrate on this area!

Important features of neural networks

- Massive parallelism
  - many slow (and "dumb") computational elements work in parallel on several levels of abstraction
- Learning
  - a kid learns to recognize a rabbit after seeing several rabbits
- Generalization
  - a kid is able to recognize a new rabbit after seeing several (old) rabbits
- Robustness
  - a blurred photo of a rabbit may still be classified as a picture of a rabbit
- Graceful degradation
  - Experiments have shown that damaged neural network is still able to work quite well
  - Damaged network may re-adapt, remaining neurons may take on functionality of the damaged ones

The aim of the course

- We will concentrate on
  - basic techniques and principles of neural networks,
  - fundamental models of neural networks and their applications.
- You should learn
  - basic models
    (multilayer perceptron, convolutional networks, recurent network (LSTM), Hopfield and Boltzmann machines and their use in pre-training of deep nets)
  - Standard applications of these models
    (image processing, speech and text processing)
  - Basic learning algorithms
    (gradient descent & backpropagation, Hebb's rule)
  - Basic practical training techniques
    (data preparation, setting various parameters, control of learning)
  - Basic information about current implementations
    (TensorFlow, CNTK)

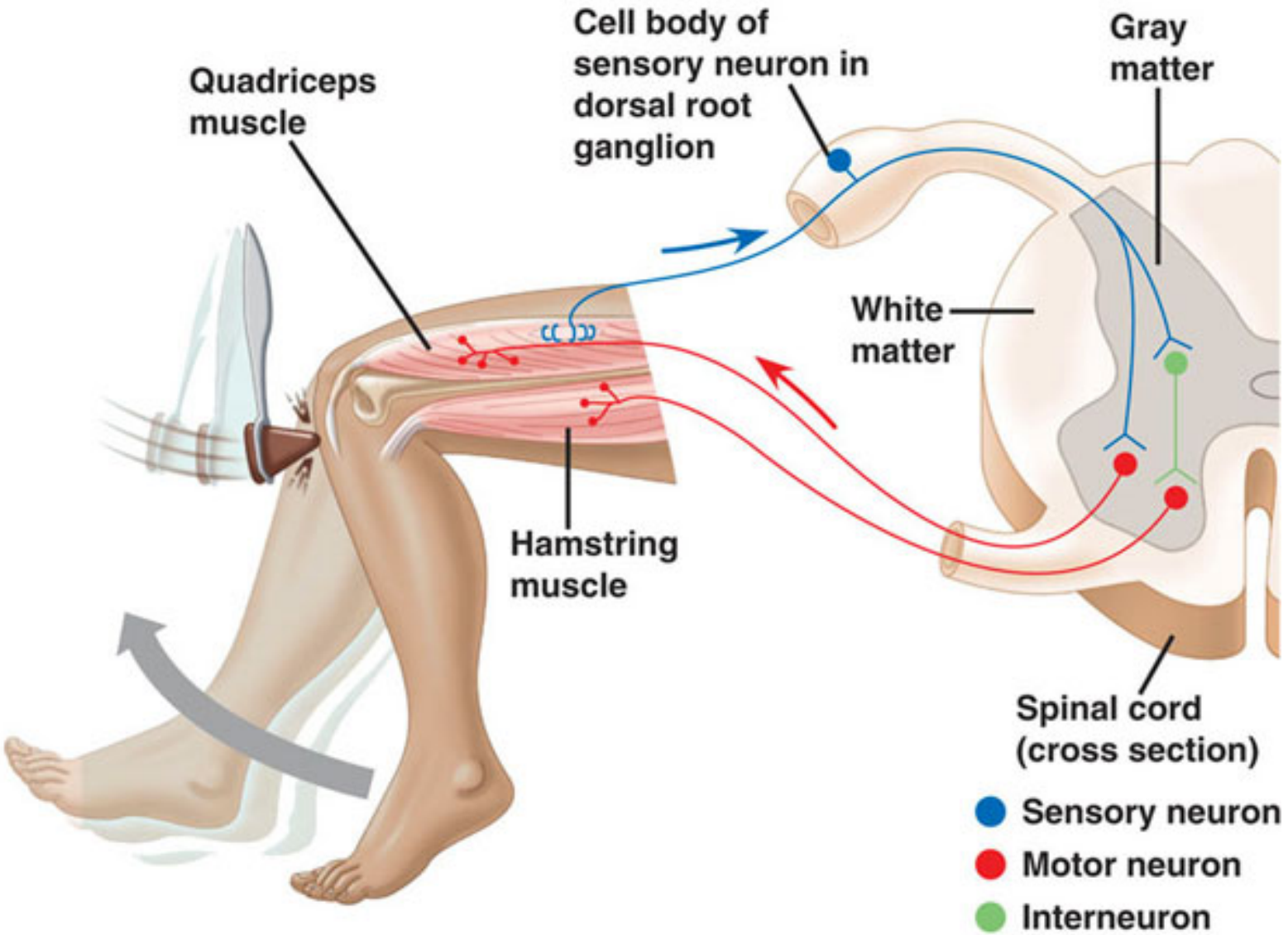
Biological neural network

- Human neural network consists of approximately 10^{11} (100 billion on the short scale) neurons; a single cubic centimeter of a human brain contains almost 50 million neurons.
- Each neuron is connected with approx. 10^4 neurons.
- Neurons themselves are very complex systems.

Rough description of nervous system:

- External stimulus is received by *sensory receptors* (e.g. eye cells).
- Information is futher transfered via peripheral nervous system (PNS) to the central nervous systems (CNS) where it is processed (integrated), and subseqently, an output signal is produced.
- Afterwards, the output signal is transfered via PNS to *effectors* (e.g. muscle cells).

Biological neural network



Zdroj: N. Campbell and J. Reece; Biology, 7th Edition; ISBN: 080537146X

Cerebral cortex

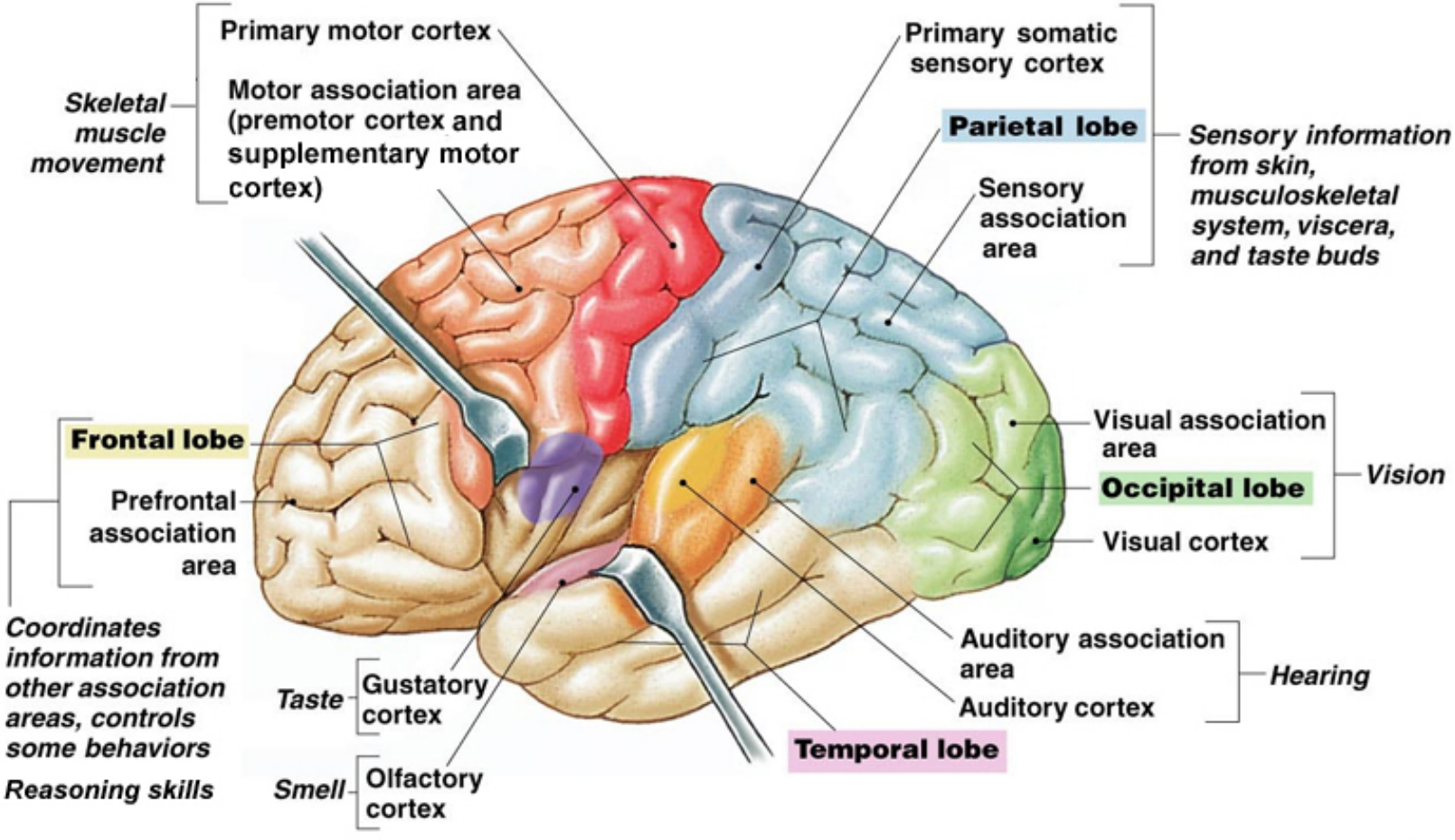
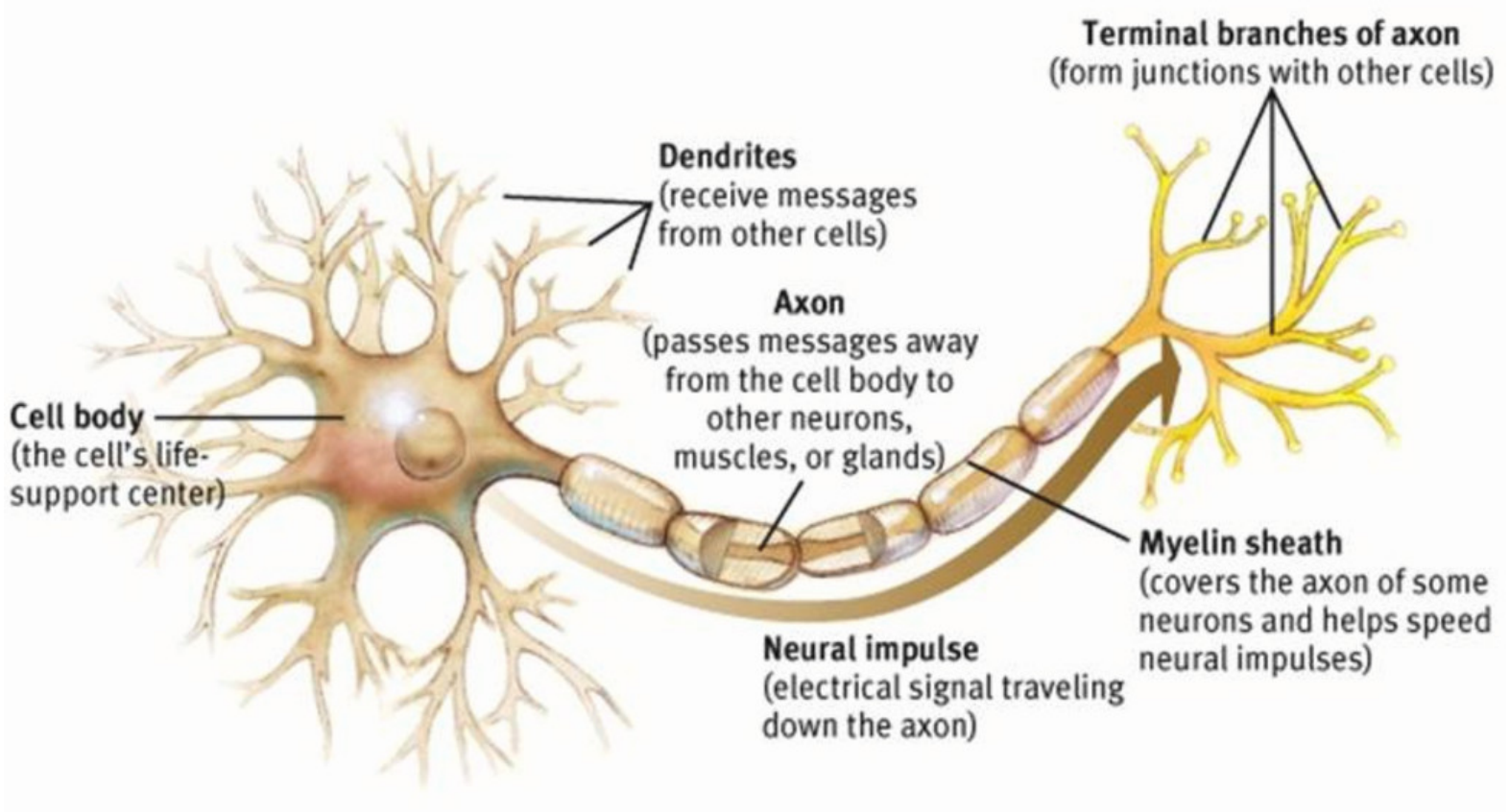


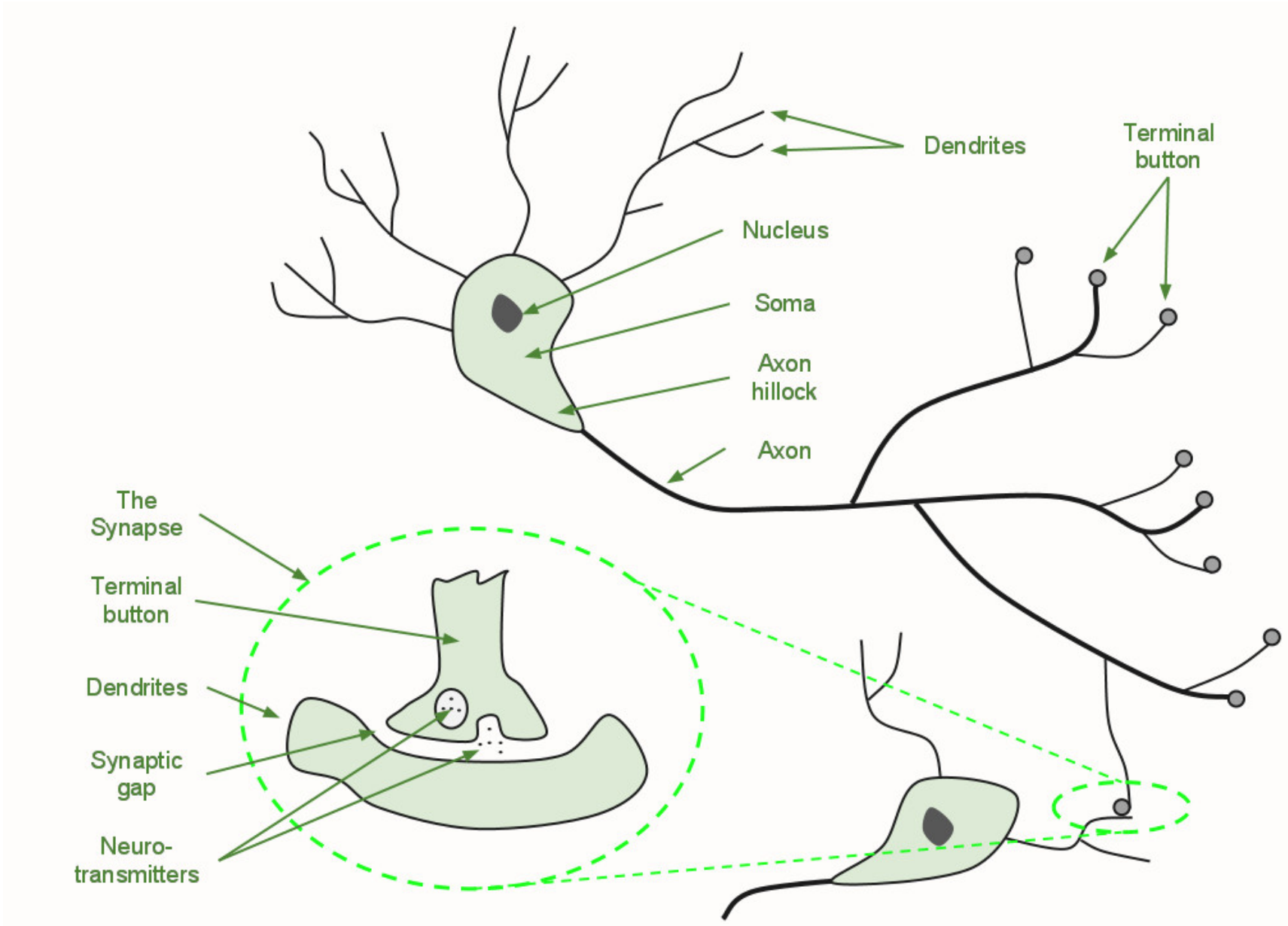
Fig. 9-15

Biological neuron

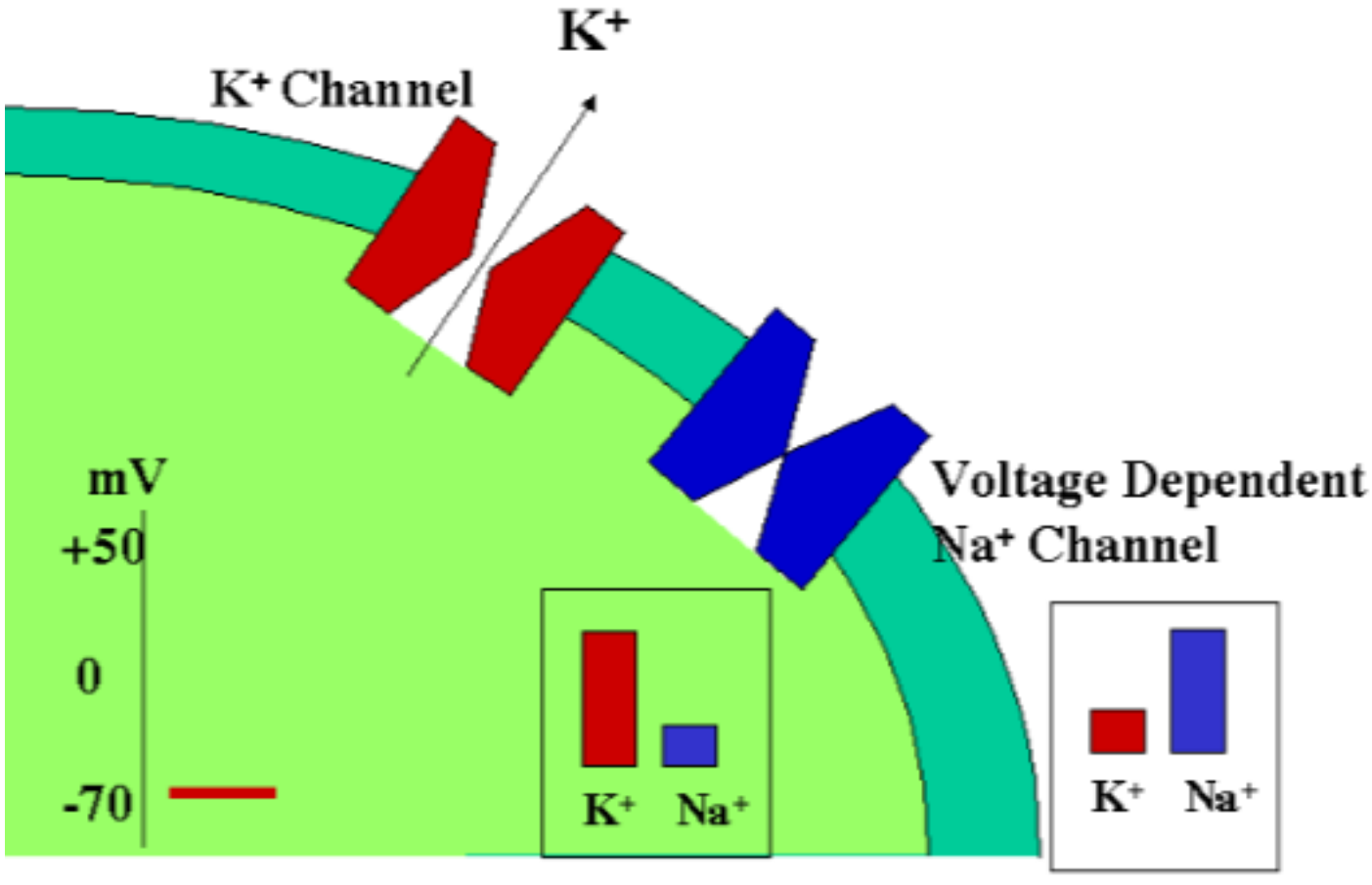


Zdroj: http://www.web-books.com/eLibrary/Medicine/Physiology/Nervous/Nervous.htm

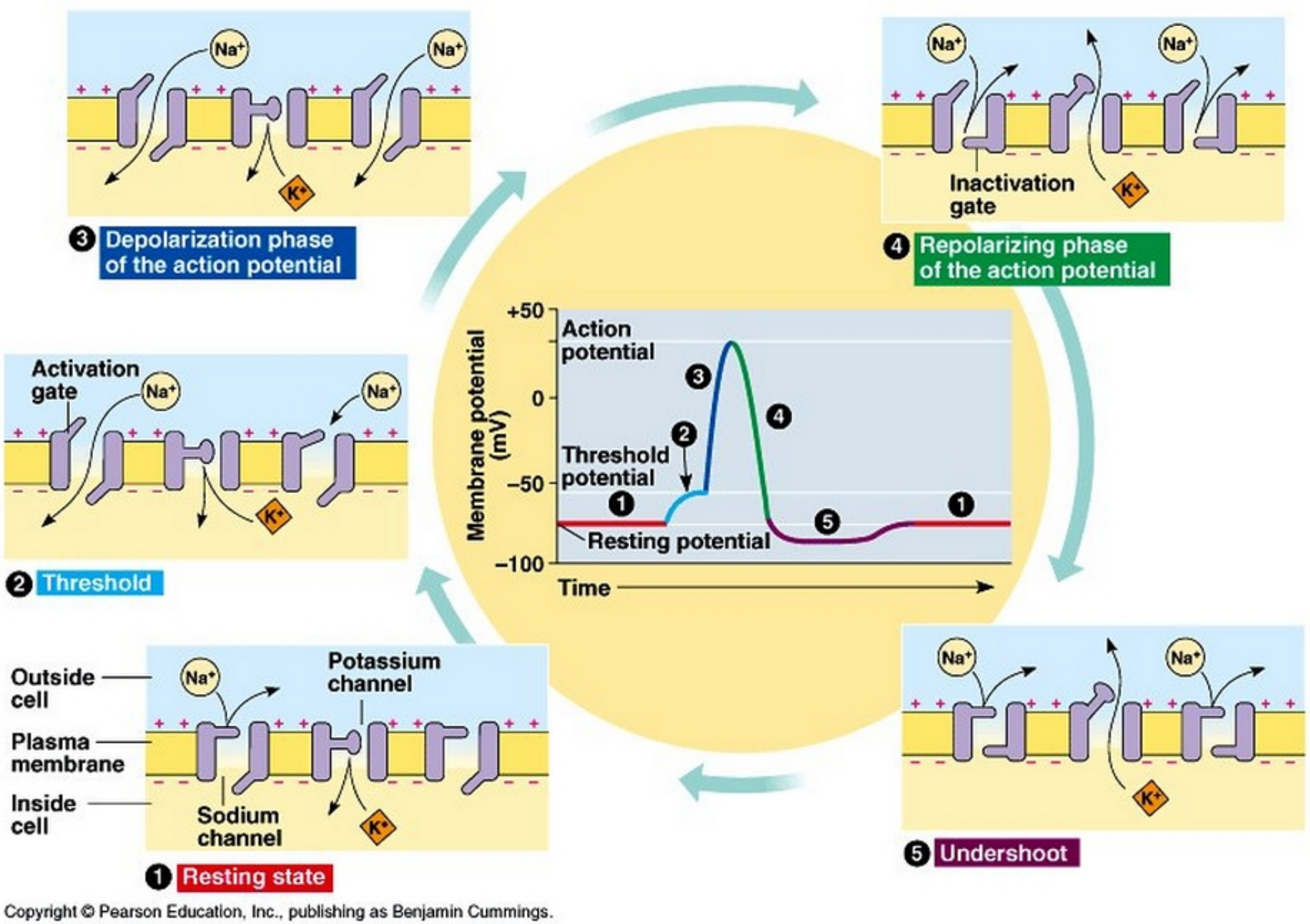
Synaptic connections



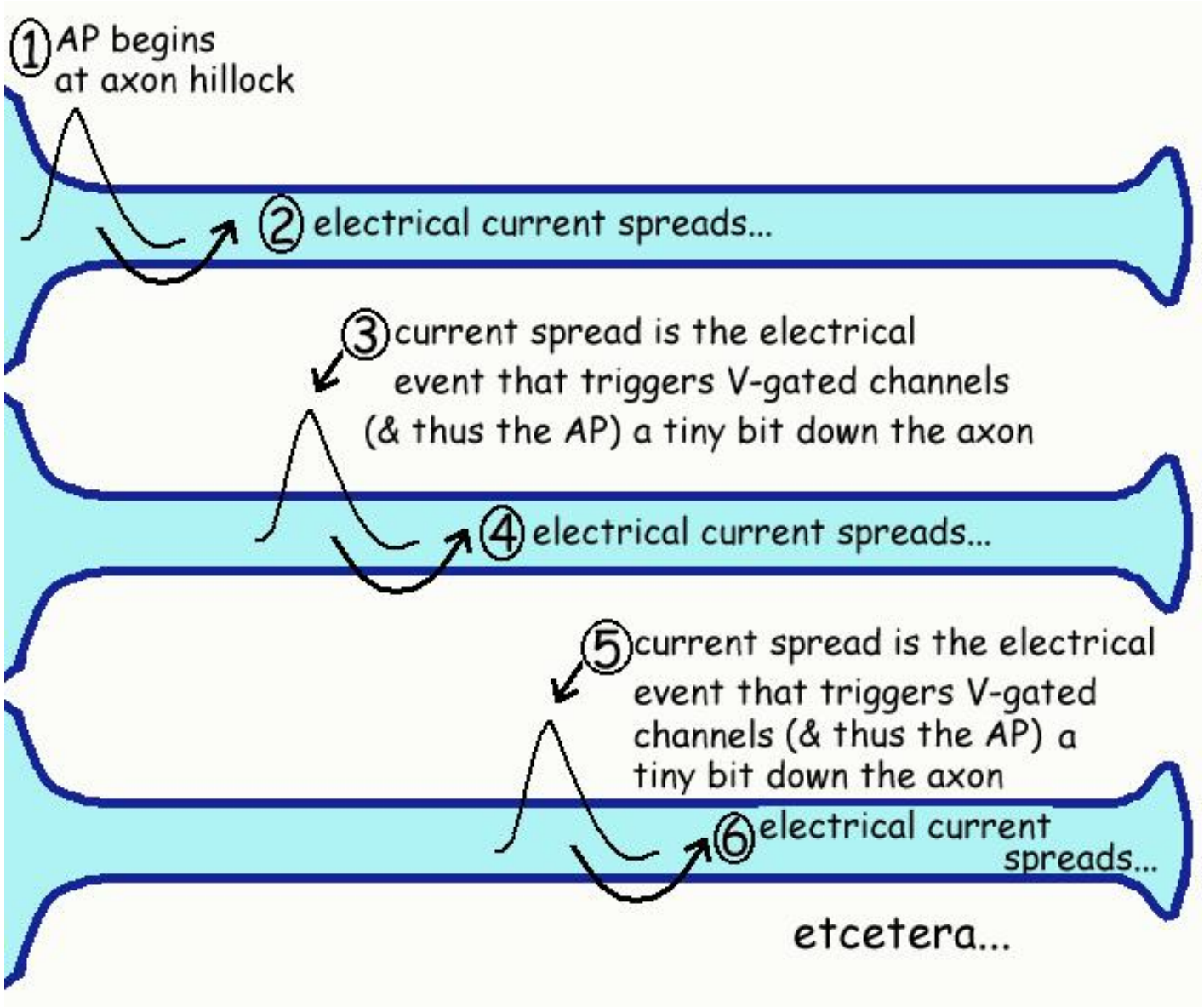
Resting potential



Action potential

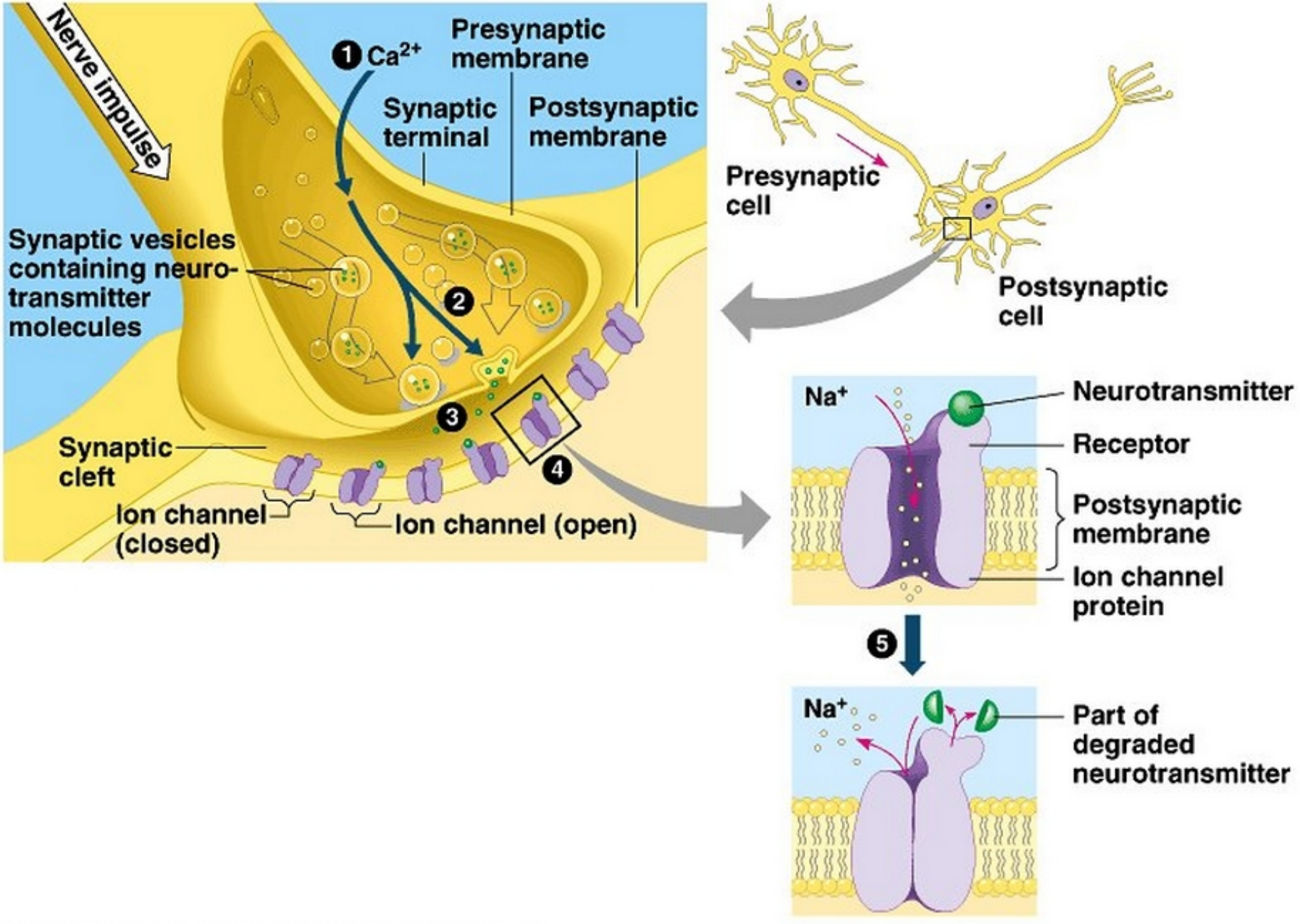


Spreading action in axon



Zdroj: D. A. Tamarkin; STCC Foundation Press

Chemical synapse



Summation

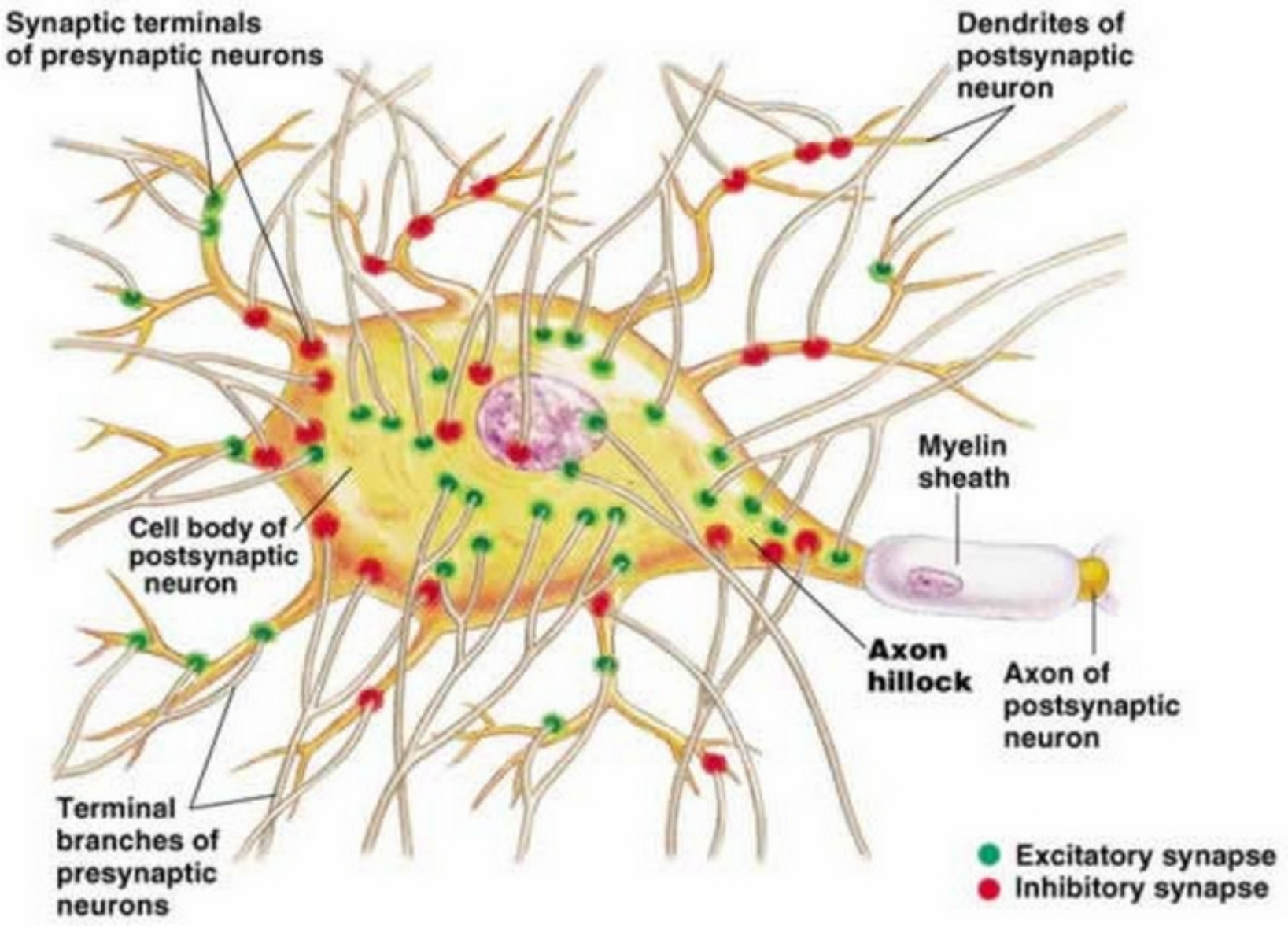
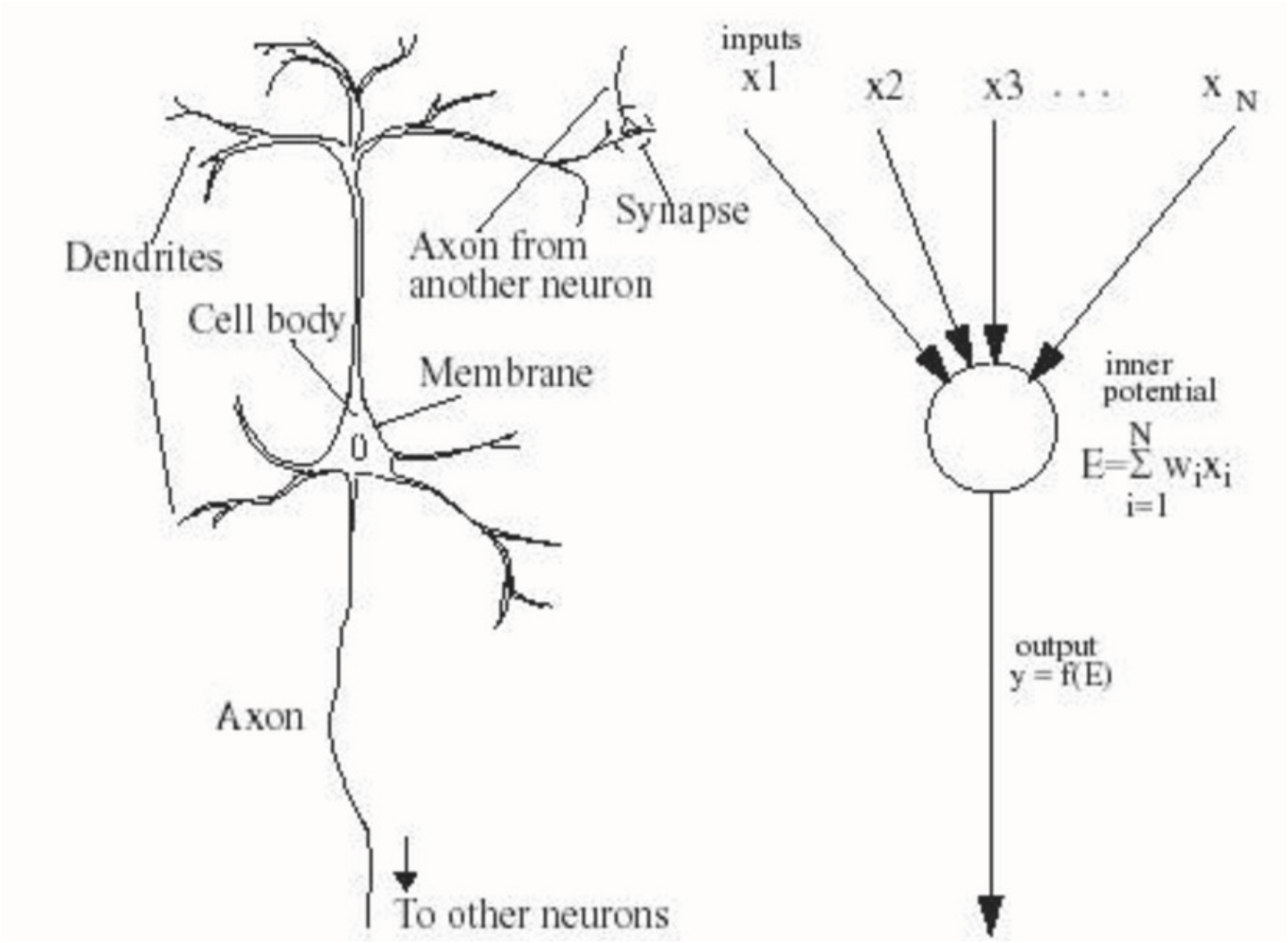
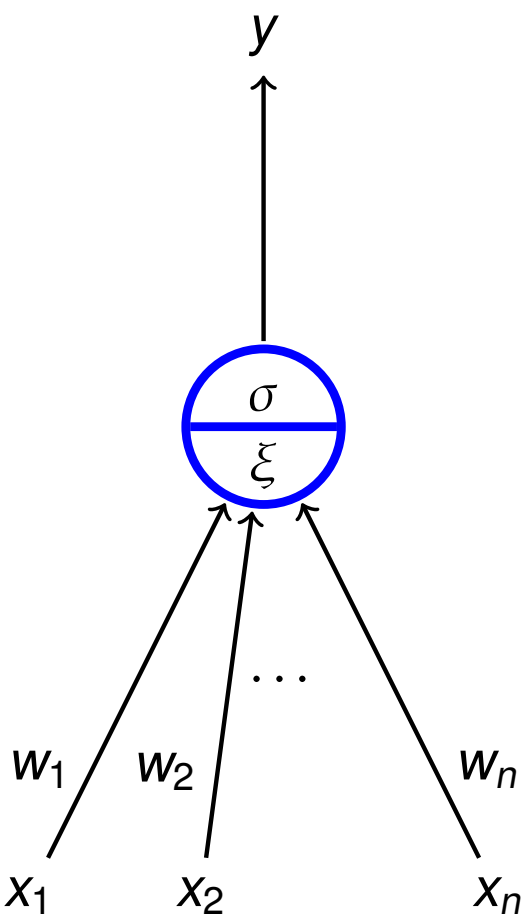


Figure 48.11(a), page 972, Campbell's *Biology, 5th Edition*

Biological and Mathematical neurons



Formal neuron (without bias)

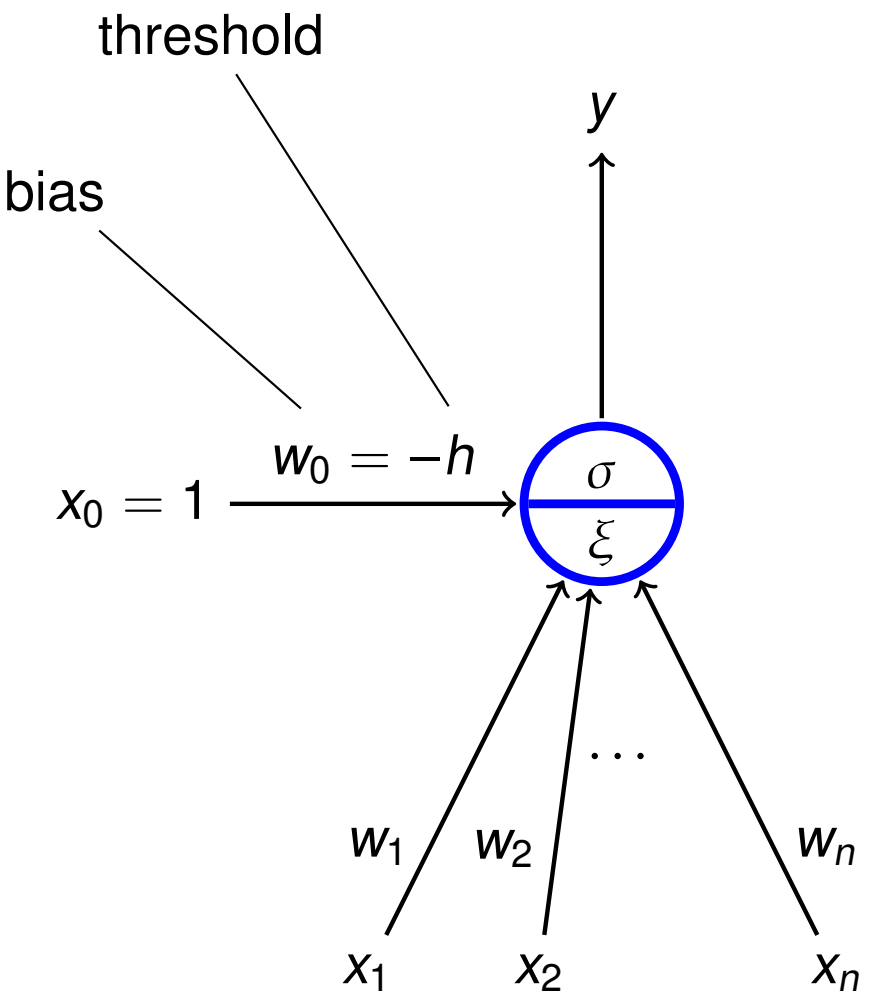


- $x_1, \ldots, x_n \in \mathbb{R}$ are **inputs**
- $w_1, \ldots, w_n \in \mathbb{R}$ are **weights**
- ξ is an **inner potential**; almost always $\xi = \sum_{i=1}^{n} w_i x_i$
- y is an **output** given by $y = \sigma(\xi)$ where σ is an **activation function**; e.g. a *unit step function*

$$\sigma(\xi) = \begin{cases} 1 & \xi \geq h\,; \\ 0 & \xi < h. \end{cases}$$

where $h \in \mathbb{R}$ is a *threshold*.

Formal neuron (with bias)



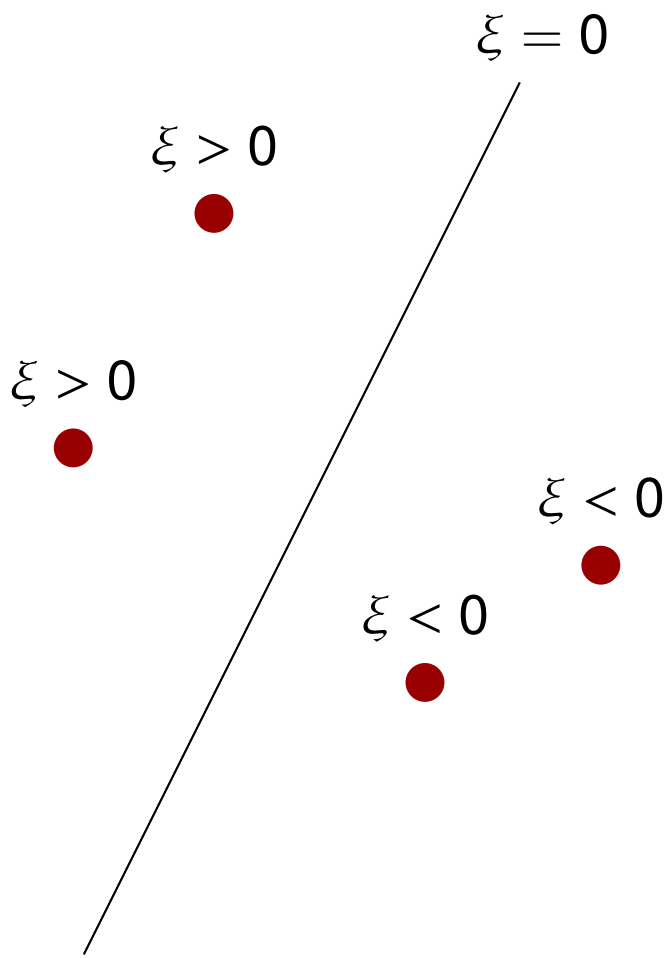
- $x_0 = 1, x_1, \ldots, x_n \in \mathbb{R}$ are **inputs**
- $w_0, w_1, \ldots, w_n \in \mathbb{R}$ are **weights**
- ξ is an **inner potential**; almost always $\xi = w_0 + \sum_{i=1}^{n} w_i x_i$
- y is an **output** given by $y = \sigma(\xi)$ where σ is an **activation function**;

  e.g. a *unit step function*

$$\sigma(\xi) = \begin{cases} 1 & \xi \geq 0\,; \\ 0 & \xi < 0. \end{cases}$$

(The threshold h has been substituted with the new input $x_0 = 1$ and the weight $w_0 = -h$.)

Neuron and linear separation



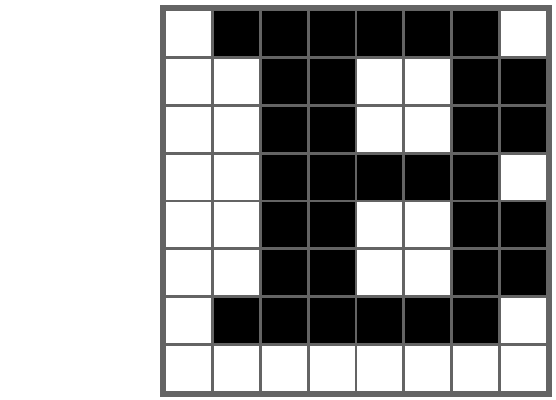
- inner potential

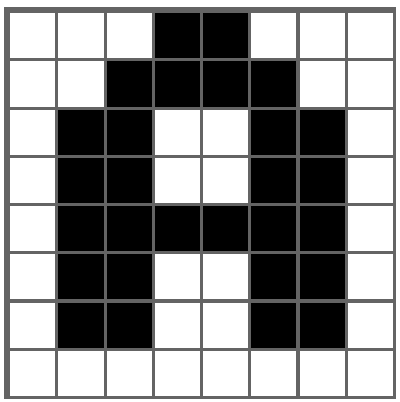
$$\xi = w_0 + \sum_{i=1}^{n} w_i x_i$$

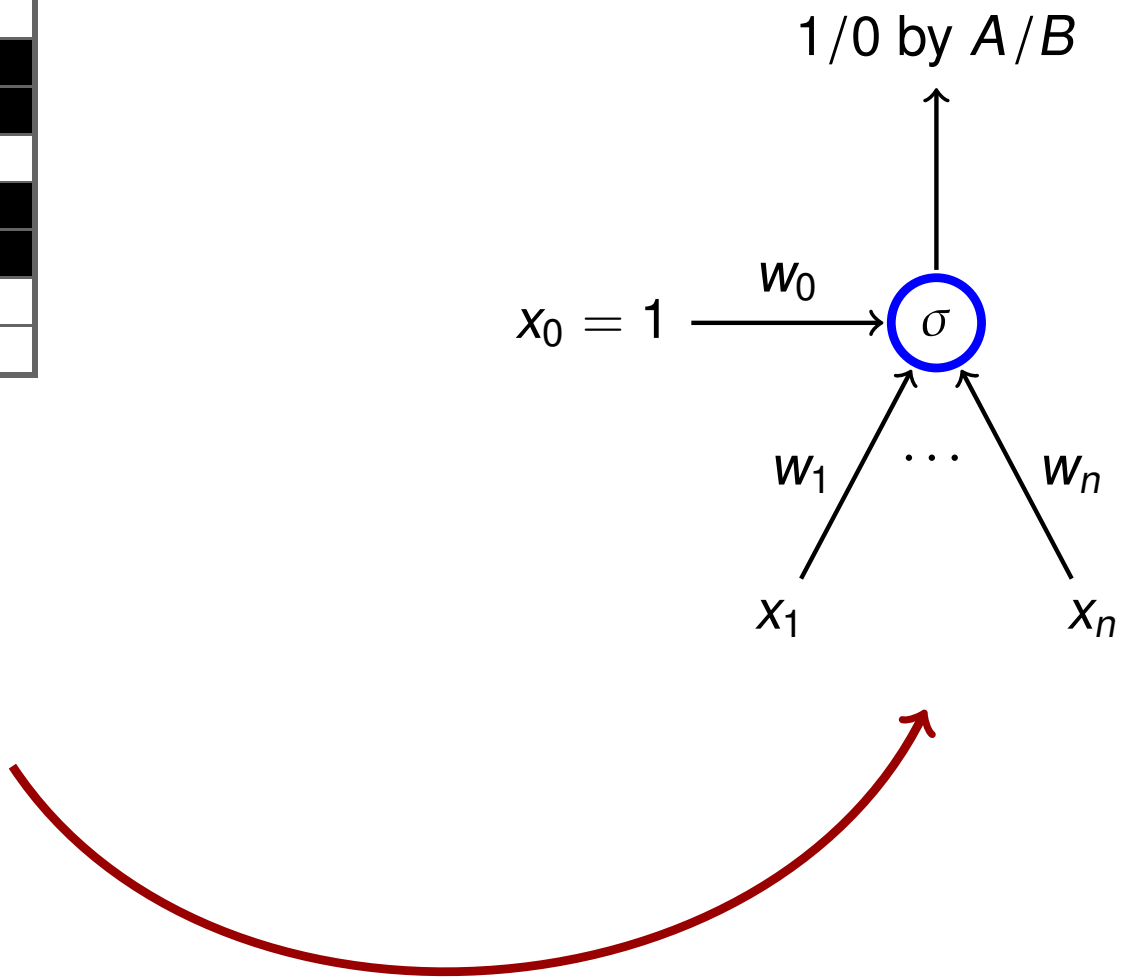
determines a separation hyperplane in the n-dimensional **input space**

- in 2d line
- in 3d plane
- ...

Neuron and linear separation

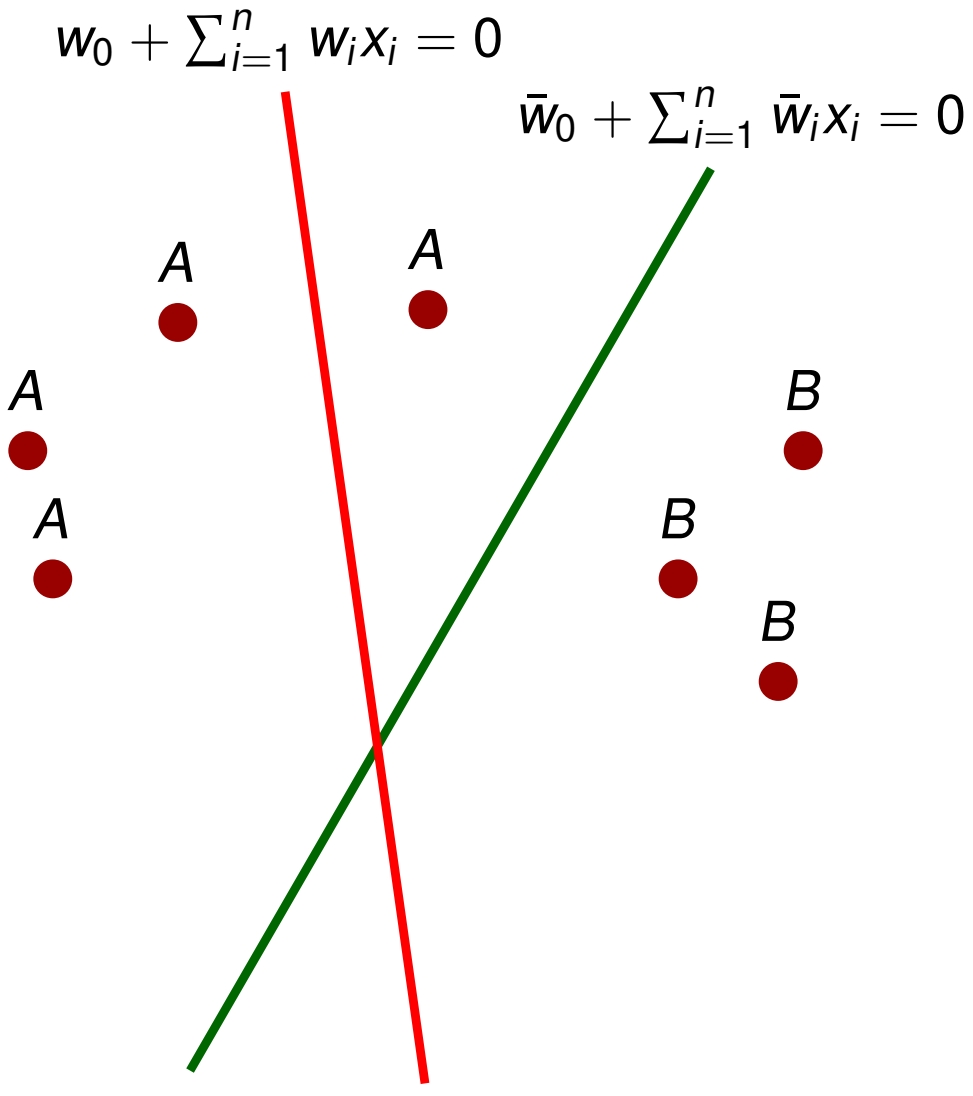






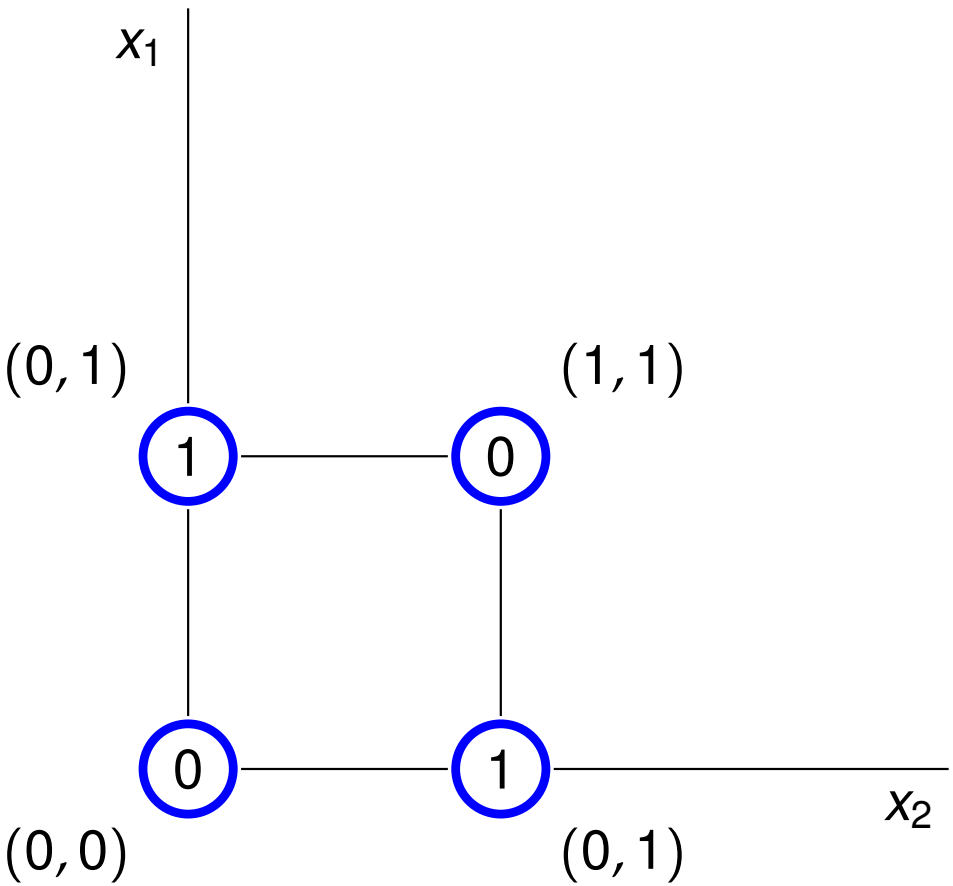
$n = 8 \cdot 8$, i.e. the number of pixels in the images. Inputs are binary vectors of dimension n (black pixel ≈ 1, white pixel ≈ 0).

Neuron and linear separation



- Red line classifies incorrectly
- Green line classifies correctly (may be a result of a correction by a learning algorithm)

Neuron and linear separation (XOR)



- No line separates ones from zeros.

Neural networks

Neural network consists of formal neurons interconnected in such a way that the output of one neuron is an input of several other neurons.

In order to describe a particular type of neural networks we need to specify:

- Architecture

  How the neurons are connected.
- Activity

  How the network transforms inputs to outputs.
- Learning

  How the weights are changed during training.

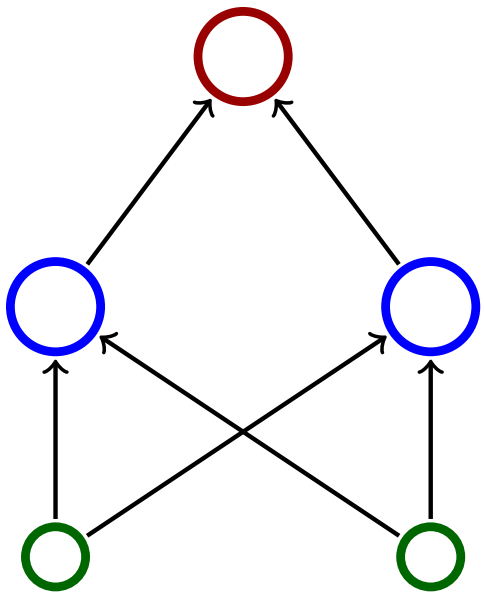
Architecture

Network architecture is given as a digraph whose nodes are neurons and edges are connections.

We distinguish several categories of neurons:

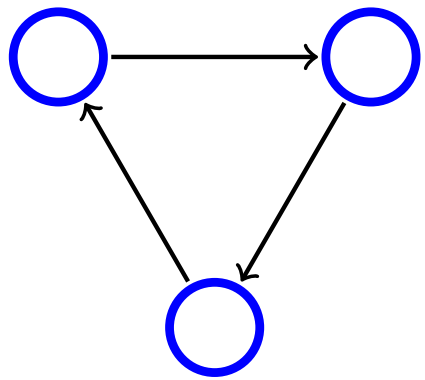
- Output neurons
- Hidden neurons
- Input neurons

(In general, a neuron may be both input and output; a neuron is hidden if it is neither input, nor output.)

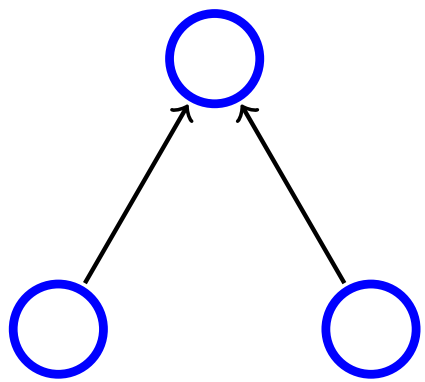


Architecture – Cycles

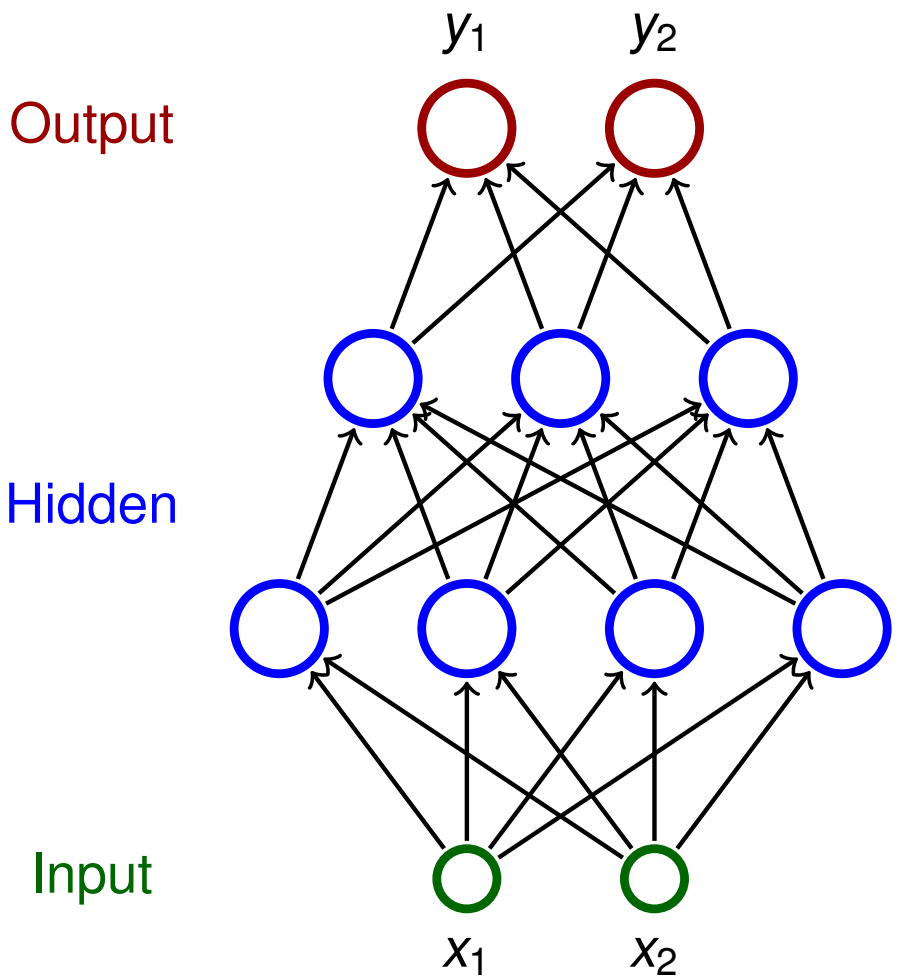
- A network is **cyclic** (recurrent) if its architecture contains a directed cycle.



- Otherwise it is **acyclic** (feed-forward)



Architecture – Multilayer Perceptron (MLP)



- Neurons partitioned into **layers**; one input layer, one output layer, possibly several hidden layers
- layers numbered from 0; the input layer has number 0
  - E.g. three-layer network has two hidden layers and one output layer
- Neurons in the i-th layer are connected with all neurons in the $i + 1$-st layer
- Architecture of a MLP is typically described by numbers of neurons in individual layers (e.g. 2-4-3-2)

Activity

Consider a network with n neurons, k input and ℓ output.

- **State** of a network is a vector of output values of all neurons.

  (States of a network with n neurons are vectors of $\mathbb{R}^n$)

- **State-space** of a network is a set of all states.

- **Network input** is a vector of k real numbers, i.e. an element of $\mathbb{R}^k$.
- **Network input space** is a set of all network inputs.

  (sometimes we restrict ourselves to a proper subset of $\mathbb{R}^k$)

- **Initial state**

  Input neurons set to values from the network input (each component of the network input corresponds to an input neuron)

  Values of the remaining neurons set to 0.

Activity – computation of a network

- **Computation** (typically) proceeds in discrete steps.
  In every step the following happens:
  1. A set of neurons is selected according to some rule.
  2. The selected neurons change their states according to their inputs (they are simply evaluated).
     (If a neuron does not have any inputs, its value remains constant.)

  A computation is **finite** on a network input $\vec{x}$ if the state changes only finitely many times (i.e. there is a moment in time after which the state of the network never changes).
  We also say that the network **stops on** $\vec{x}$.
- **Network output** is a vector of values of all output neurons in the network (i.e. an element of $\mathbb{R}^\ell$).

  Note that the network output keeps changing throughout the computation!

MLP uses the following selection rule:

In the *i*-th step evaluate all neurons in the *i*-th layer.

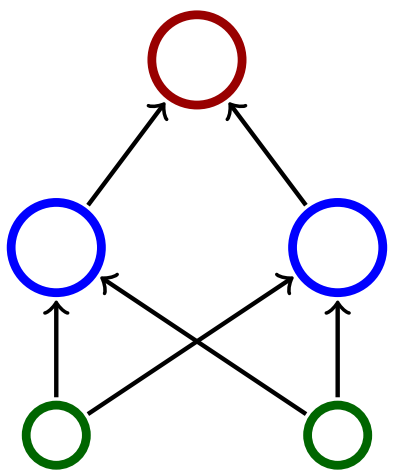
Activity – semantics of a network

Definition

Consider a network with n neurons, k input, ℓ output.
Let $A \subseteq \mathbb{R}^k$ and $B \subseteq \mathbb{R}^\ell$. Suppose that the network stops on every input of A.
Then we say that the network computes a function *$F : A \to B$ if for every network input $\vec{x}$ the vector $F(\vec{x}) \in B$ is the output of the network after the computation on $\vec{x}$ stops.*

Example 1

This network computes a function from $\mathbb{R}^2$ to $\mathbb{R}$.



Activity – inner potential and activation functions

In order to specify activity of the network, we need to specify how the inner potentials ξ are computed and what are the activation functions σ.

We assume (unless otherwise specified) that

$$\xi = w_0 + \sum_{i=1}^{n} w_i \cdot x_i$$

here $\vec{x} = (x_1, \ldots, x_n)$ are inputs of the neuron and $\vec{w} = (w_1, \ldots, w_n)$ are weights.

There are special types of neural network where the inner potential is computed differently, e.g. as a "distance" of an input from the weight vector:

$$\xi = \left\| \vec{x} - \vec{w} \right\|$$

here $\|\cdot\|$ is a vector norm, typically Euclidean.

Activity – inner potential and activation functions

There are many activation functions, typical examples:

- Unit step function

$$\sigma(\xi) = \begin{cases} 1 & \xi \geq 0\,; \\ 0 & \xi < 0. \end{cases}$$

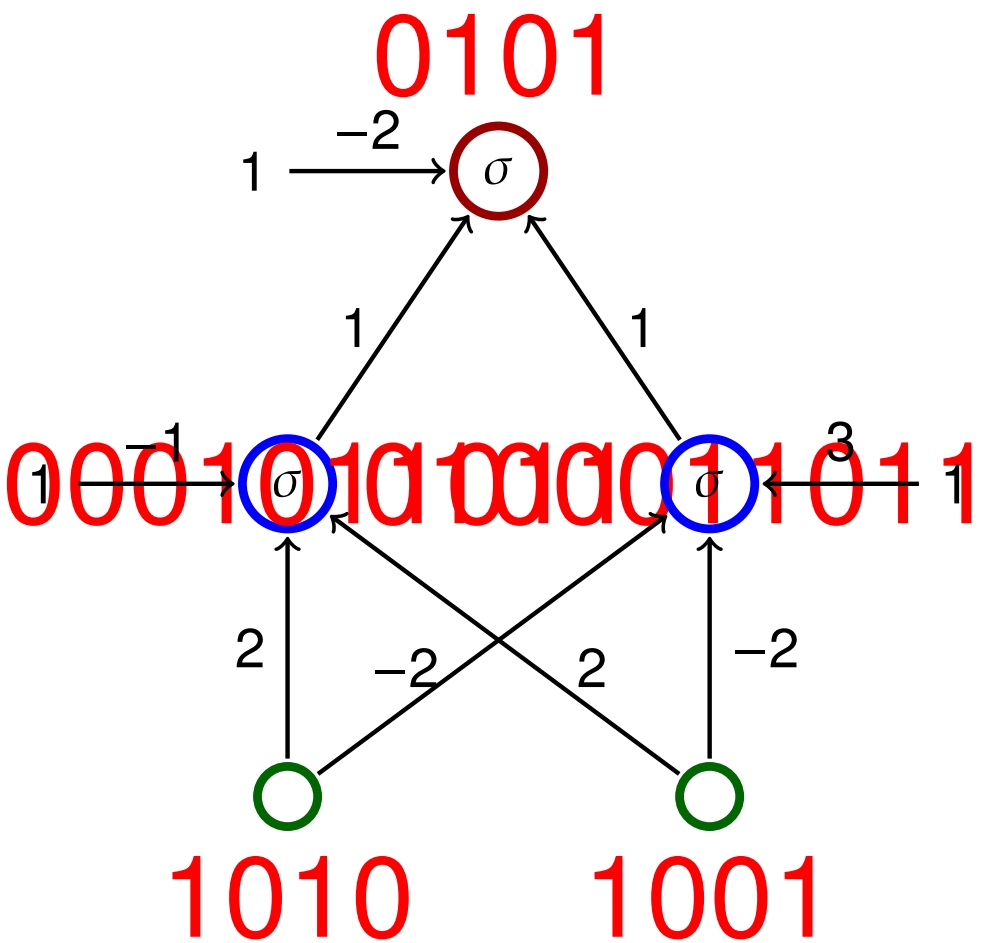
- (Logistic) sigmoid

$$\sigma(\xi) = \frac{1}{1 + e^{-\lambda \cdot \xi}}$$ here $\lambda \in \mathbb{R}$ is a *steepness* parameter.

- Hyperbolic tangens

$$\sigma(\xi) = \frac{1 - e^{-\xi}}{1 + e^{-\xi}}$$

Activity – XOR



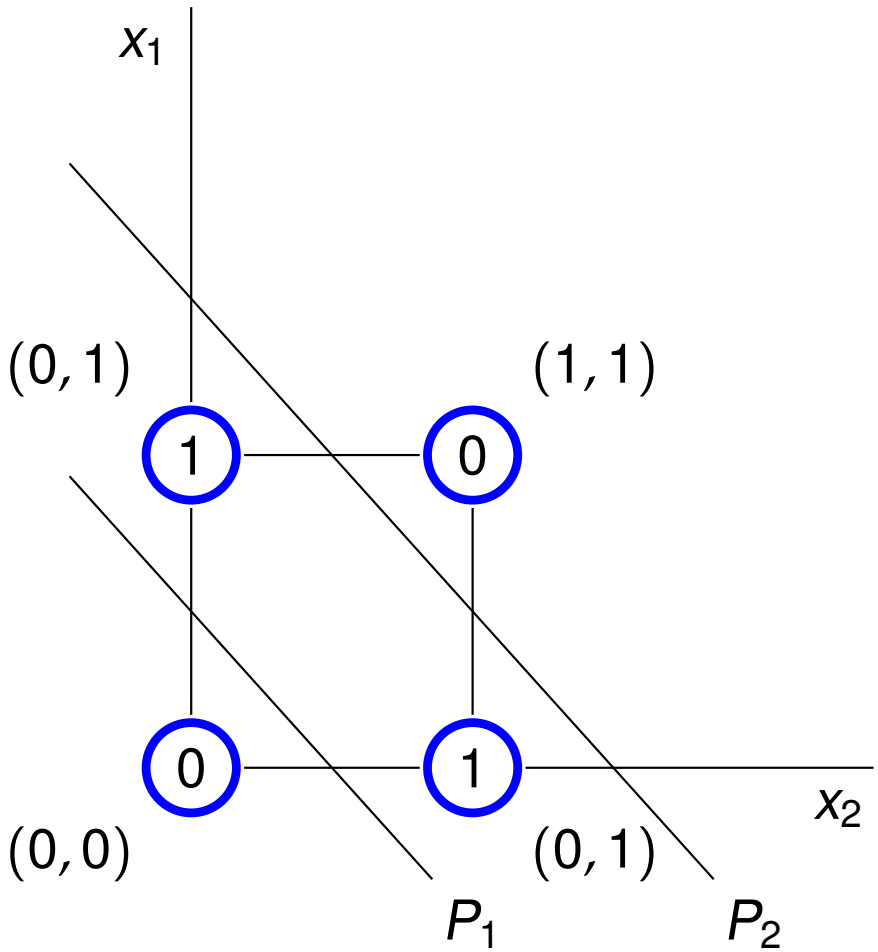
- Activation function is a unit step function

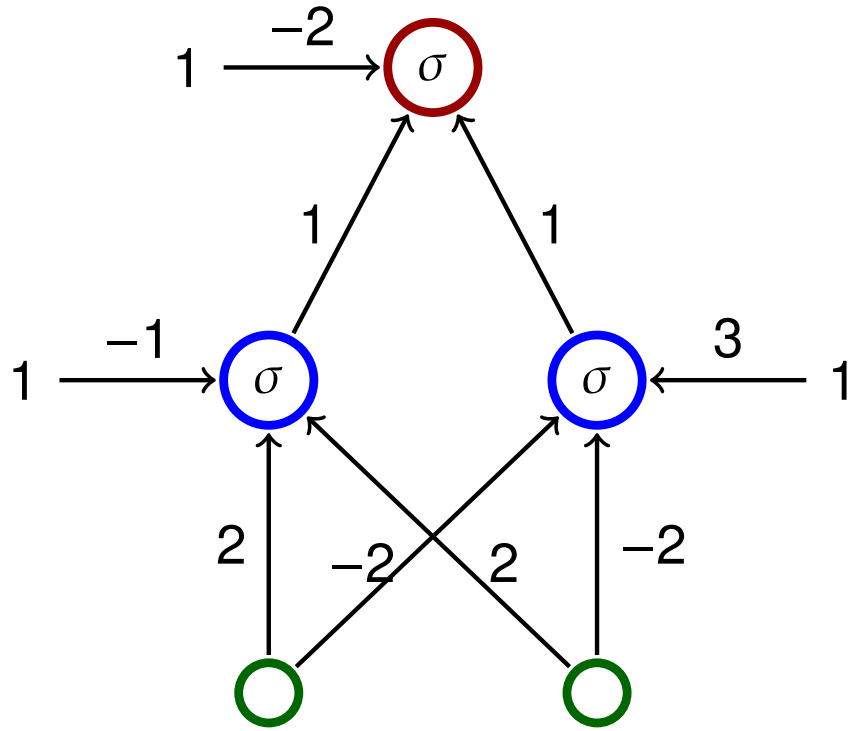
$$\sigma(\xi) = \begin{cases} 1 & \xi \geq 0\,; \\ 0 & \xi < 0. \end{cases}$$

- The network computes $XOR(x_1, x_2)$

x_1	x_2	y
1	1	0
1	0	1
0	1	1
0	0	0

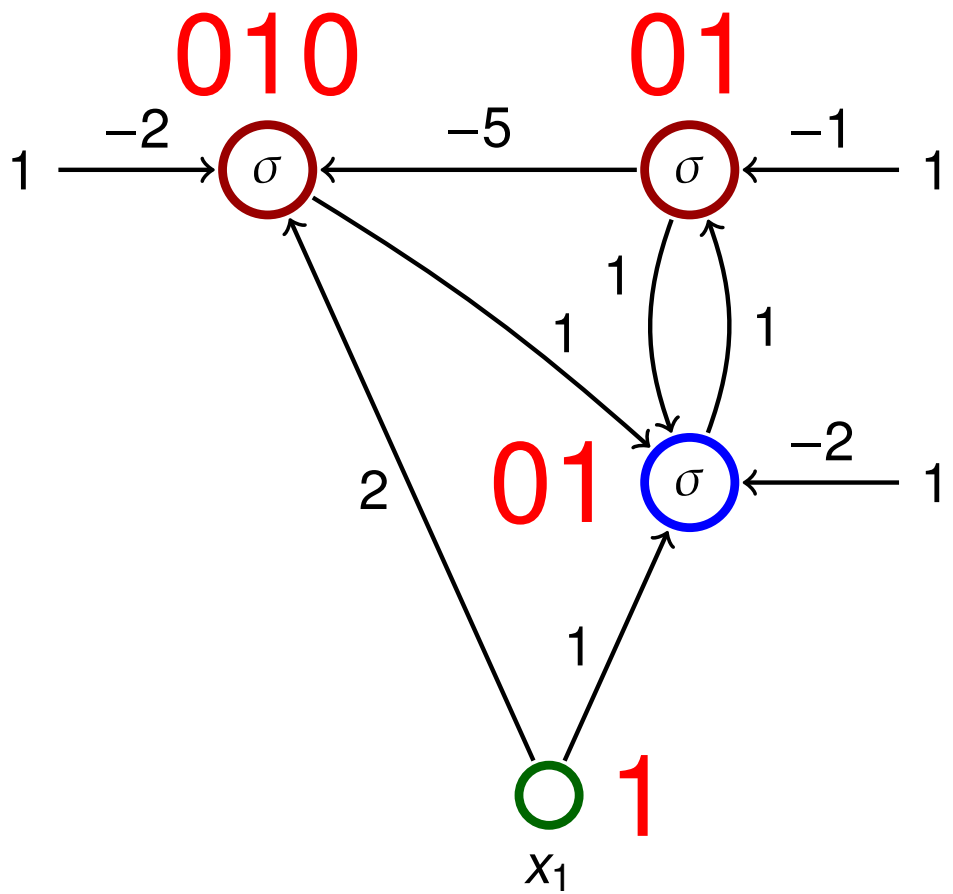
Activity – MLP and linear separation





- The line P_1 is given by $-1 + 2x_1 + 2x_2 = 0$
- The line P_2 is given by $3 - 2x_1 - 2x_2 = 0$

Activity – example



The activation function is the unit step function

$$\sigma(\xi) = \begin{cases} 1 & \xi \geq 0\,; \\ 0 & \xi < 0. \end{cases}$$

The input is equal to 1

Learning

Consider a network with n neurons, k input and ℓ output.

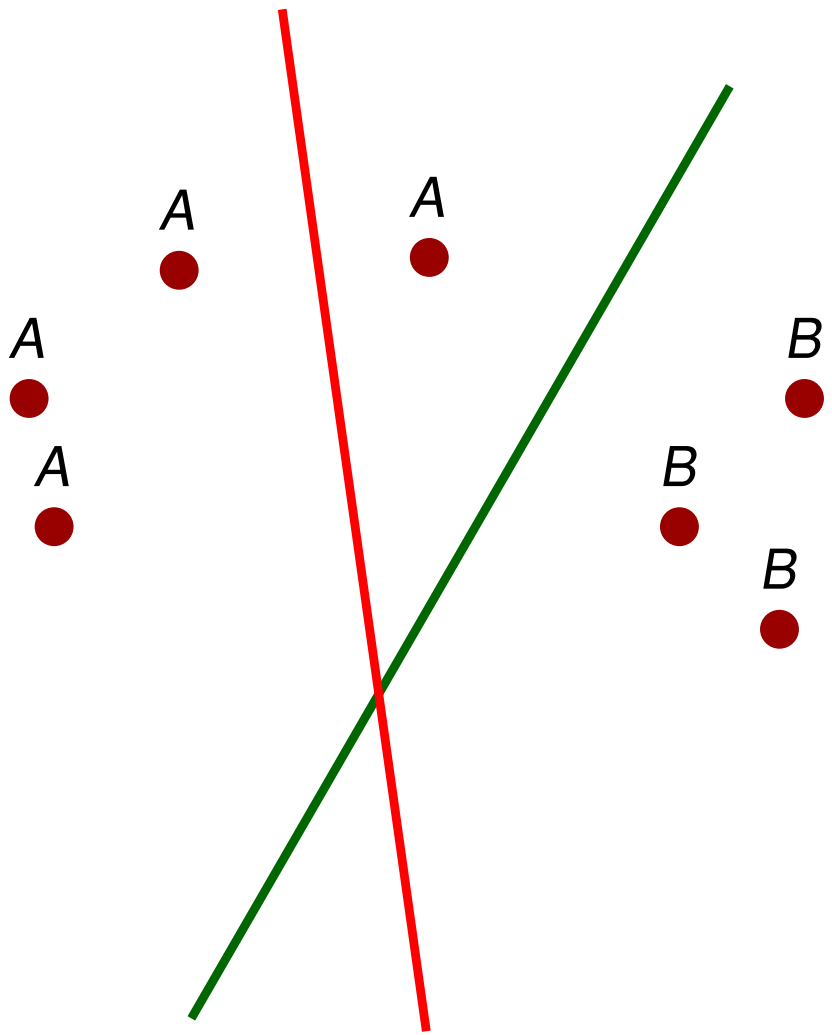
- **Configuration** of a network is a vector of all values of weights.
  (Configurations of a network with m connections are elements of $\mathbb{R}^m$)
- **Weight-space** of a network is a set of all configurations.
- **initial configuration**
  weights can be initialized randomly or using some sophisticated algorithm

Learning algorithms

Learning rule for weight adaptation.
(the goal is to find a configuration in which the network computes a desired function)

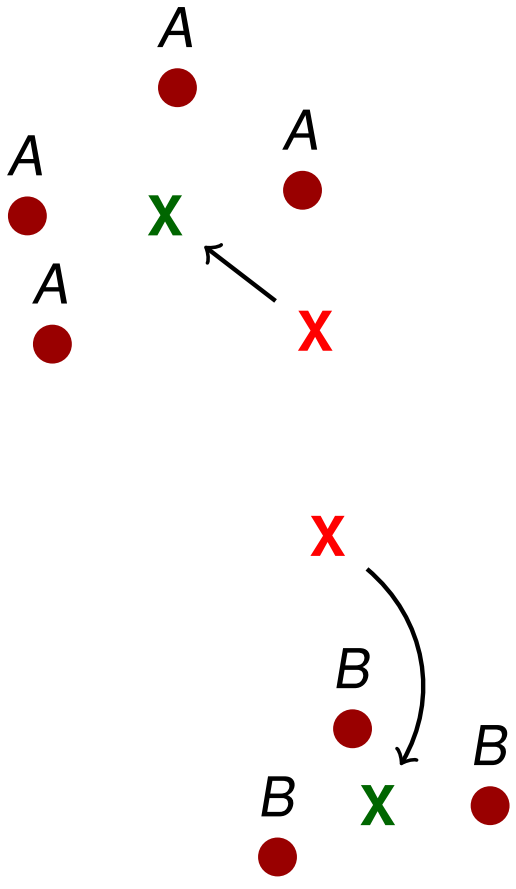
- supervised learning
  - The desired function is described using *training examples* that are pairs of the form (input, output).
  - Learning algorithm searches for a configuration which "corresponds" to the training examples, typically by minimizing an error function.
- Unsupervised learning
  - The training set contains only inputs.
  - The goal is to determine distribution of the inputs (clustering, deep belief networks, etc.)

Supervised learning – illustration



- classification in the plane using a single neuron
- training examples are of the form (point, value) where the value is either 1, or 0 depending on whether the point is either *A*, or *B*
- the algorithm considers examples one after another
- whenever an incorrectly classified point is considered, the learning algorithm turns the line in the direction of the point

Unsupervised learning – illustration



- we search for two *centres* of clusters
- red crosses correspond to potential centres before application of the learning algorithm, green ones after the application

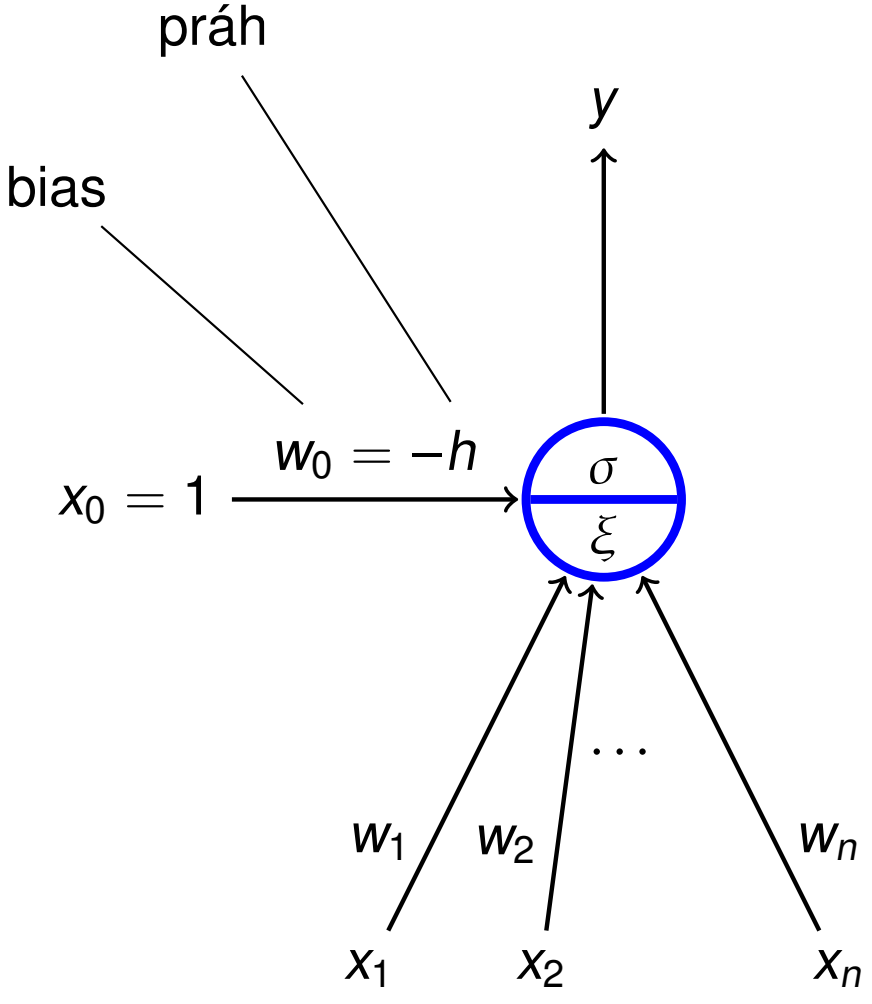
Summary – Advantages of neural networks

- Massive parallelism
  - neurons can be evaluated in parallel
- Learning
  - many sophisticated learning algorithms used to "program" neural networks
- generalization and robustness
  - information is encoded in a distributed manned in weights
  - "close" inputs typicaly get similar values
- Graceful degradation
  - damage typically causes only a decrease in precision of results

Postavení neuronových sítí v informatice

- Vyjadřovací schopnosti neuronových sítí
  - logické funkce
  - nelineární separace
  - spojité funkce
  - algoritmická vyčíslitelnost
- Srovnání s klasickou architekturou počítačů
- Implementace

Formální neuron (s biasem)



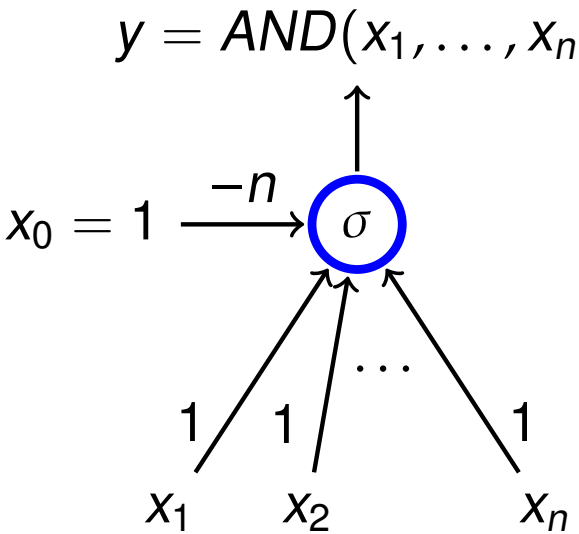
- $x_1, \ldots, x_n$ jsou reálné **vstupy**
- x_0 je speciální vstup, který má vždy hodnotu 1
- $w_0, w_1, \ldots, w_n$ jsou reálné **váhy**
- ξ je **vnitřní potenciál**; většinou $\xi = w_0 + \sum_{i=1}^{n} w_i x_i$
- y je **výstup** daný $y = \sigma(\xi)$ kde σ je **aktivační funkce**; např. *ostrá nelinearita*

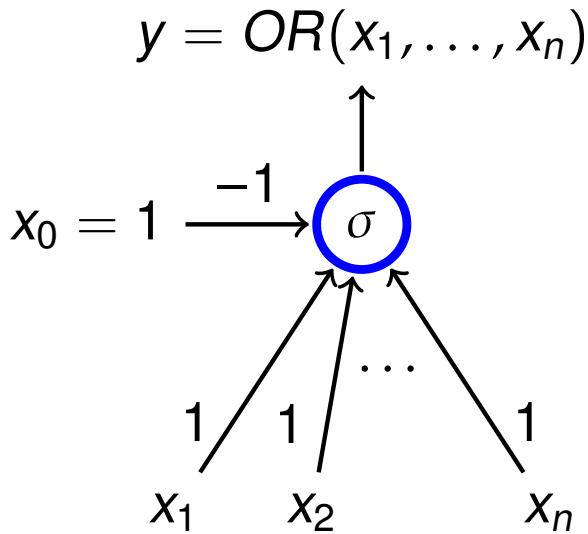
$$\sigma(\xi) = \begin{cases} 1 & \xi \geq 0\,; \\ 0 & \xi < 0. \end{cases}$$

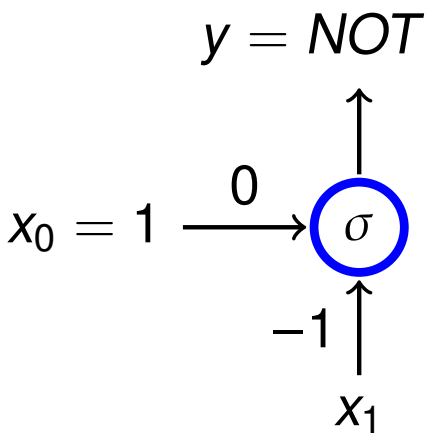
(práh aktivační funkce σ je roven 0; reálný práh byl nahrazen vstupem $x_0 = 1$ a váhou $w_0 = -h$)

Základní logické funkce

Aktivační funkce σ je ostrá nelinearita $\sigma(\xi) = \begin{cases} 1 & \xi \geq 0\,; \\ 0 & \xi < 0. \end{cases}$







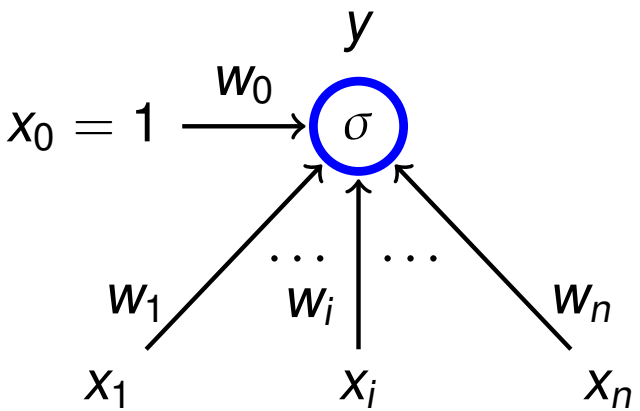
Logické funkce - obecně

Theorem

Nechť σ je ostrá nelinearita. Douvrstvé sítě s aktivační funkcí σ mohou počítat libovolnou funkci $F : \{0,1\}^n \to \{0,1\}$.

Proof.

- Pro každý vektor $\vec{v} = (v_1, \ldots, v_n) \in \{0,1\}^n$ definujeme neuron $N_{\vec{v}}$, jehož výstup je roven 1 právě když vstup je $\vec{v}$:

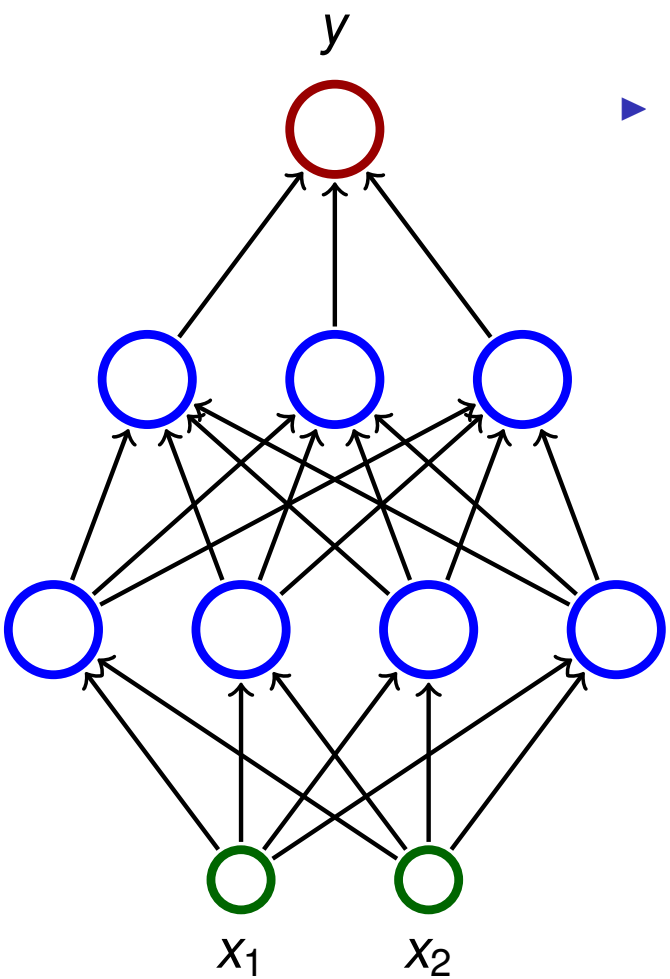


$$w_0 = -\sum_{i=1}^{n} v_i$$

$$w_i = \begin{cases} 1 & v_i = 1 \\ -1 & v_i = 0 \end{cases}$$

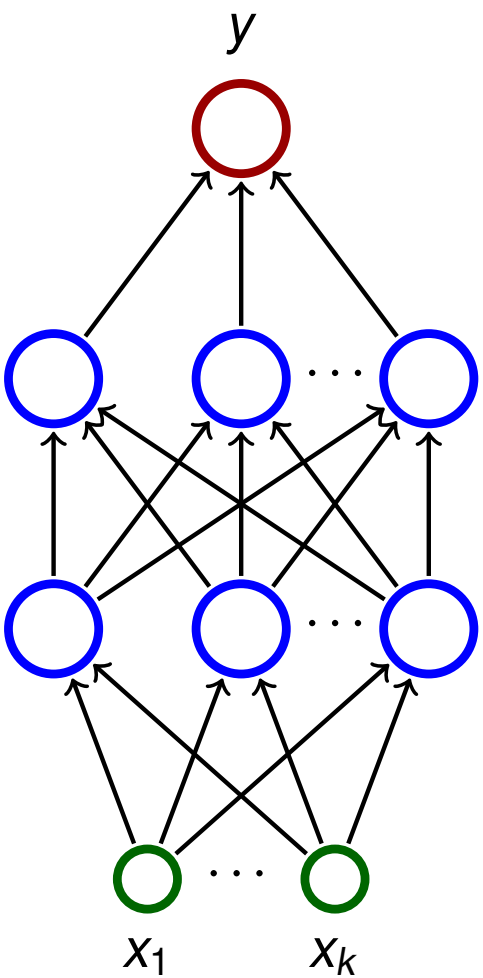
- Spojíme výstupy všech neuronů $N_{\vec{v}}$ pro něž platí $F(\vec{v}) = 1$ pomocí neuronu, který implementuje funkci *OR*. □

Nelineární separace - ostrá nelinearita



- Uvažujme třívrstvou síť; každý neuron má ostrou nelinearitu jako aktivační funkci
- Síť dělí vstupní prostor na dva podprostory podle hodnoty výstupu (0 nebo 1)
  - První (skrytá) vrstva dělí vstupní prostor na poloprostory
  - Druhá může např. dělat průniky poloprostorů - konvexní oblasti
  - Třetí (výstupní) vrstva může např. sjednocovat konvexní oblasti

Nelineární separace - ostrá nelinearita - ilustrace



- Uvažujme třívrstvou síť; každý neuron má ostrou nelinearitu jako aktivační funkci
- Třívrstvá síť může aproximovat libovolnou rozumnou množinu $P \subseteq \mathbb{R}^k$
  - Pokryjeme P hyperkrychlemi (v 2D jsou to ctverečky, v 3D krychle, ...)
  - Každou hyperkrychli K lze separovat pomocí dvouvrstvé sítě N_K
    (Tj. funkce počítaná sítí N_K vrací 1 pro body z K a 0 pro ostatní)
  - Pomocí neuronu, který implementuje funkci OR, spojíme výstupy všech sítí N_K t. ž. $K \cap P \neq \emptyset$.

Nelineární separace - sigmoida

Theorem (Cybenko 1989 - neformální verze)

Nechť σ je spojitá funkce, která je sigmoidální, tedy je rostoucí a splňuje

$$\sigma(x) = \begin{cases} 1 & \text{pro } x \to +\infty \\ 0 & \text{pro } x \to -\infty \end{cases}$$

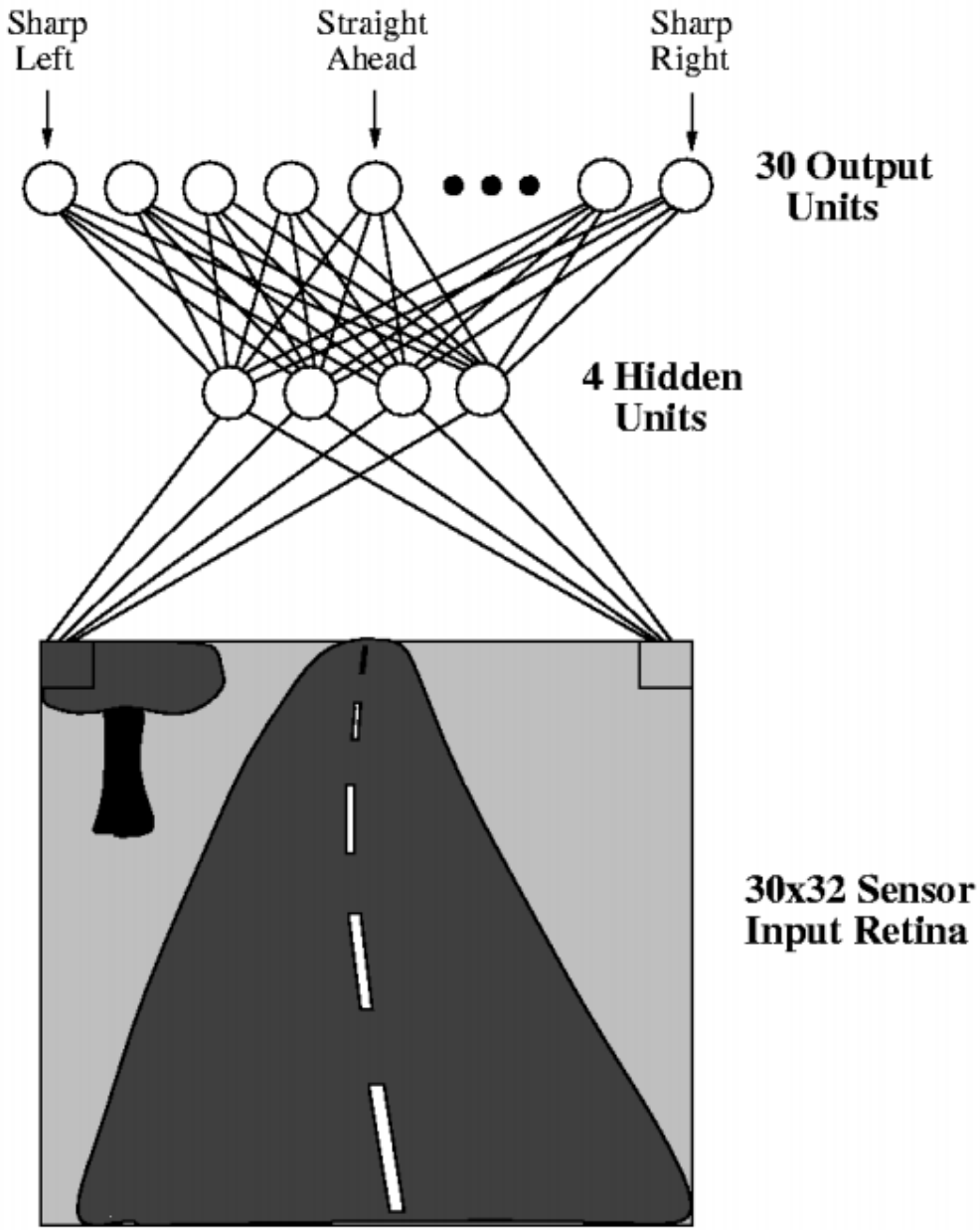
Pro každou rozumnou množinu $P \subseteq [0,1]^n$, existuje **dvouvrstvá síť** *s aktivační funkcí σ ve vnitřních neuronech (výstupní neuron má lineární), která splňuje následující:*

Pro většinu vektorů $\vec{v} \in [0,1]^n$ platí $\vec{v} \in P$ právě když výstup této sítě je > 0 pro vstup $\vec{v}$.

Pro matematicky orientované:

- rozumná množina je Lebesgueovsky měřitelná
- většina znamená, že množina špatně klasifikovaných vektorů má Lebesgueovu míru menší než dané ε (pro každé ε může být nutné konstruovat jinou síť)

Nelineární separace - praktická ilustrace



- ALVINN řídí automobil
- Síť má $30 \times 32 = 960$ vstupních neuronů (vstupní prostor je $\mathbb{R}^{960}$)
- Vstupy berou stupně šedi jednotlivých obrazových bodů
- výstupní neurony klasifikují obrázky silnice podle zakřivení

Zdroj obrázku: `http://jmvidal.cse.sc.edu/talks/ann/alvin.html`

Aproximace spojitých funkcí - třívrstvé sítě

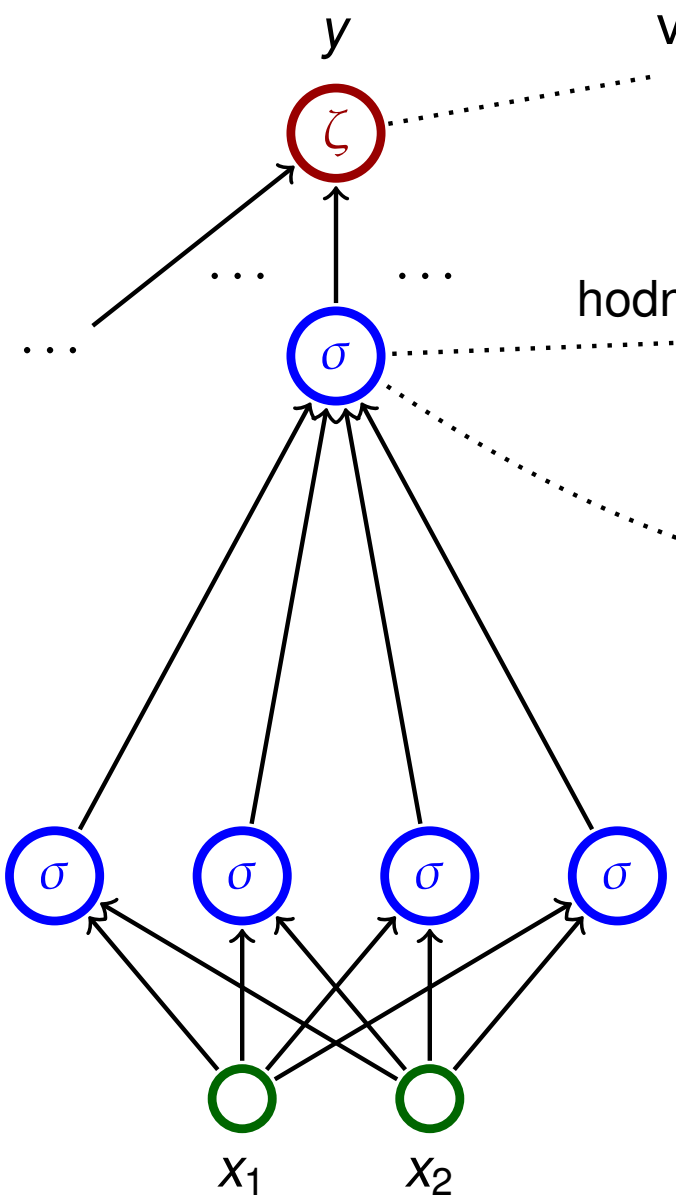
Nechť σ je standardní sigmoida, tj.

$$\sigma(\xi) = \frac{1}{1 + e^{-\xi}}$$

Pro každou spojitou funkci $f : [0,1]^n \to [0,1]$ a $\varepsilon > 0$ lze zkonstruovat třívrstvou síť počítající funkci $F : [0,1]^n \to [0,1]$ takovou, že

- aktivační funkce v nejvyšší vrstvě je lineární, tj. $\zeta(\xi) = \xi$, ve zbylých vrstvách standardní sigmoida σ
- pro každé $\vec{v} \in [0,1]^n$ platí $|F(\vec{v}) - f(\vec{v})| < \varepsilon$.

Aproximace spojitých funkcí - třívrstvé sítě

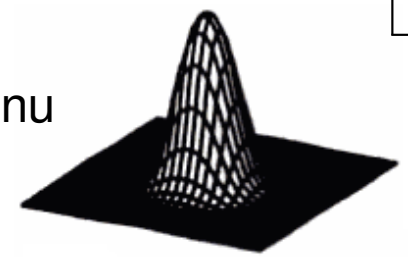


vážený součet špiček

$$\sigma(\xi) = \frac{1}{1+e^{-\xi}}$$

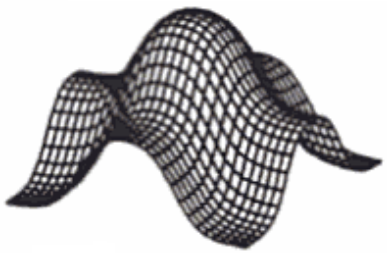
$$\zeta(\xi) = \xi$$

hodnota neuronu



špička

v. potenciál



... a další dvě rotace o 90 stupňů

Zdroj obrázků: C. Bishop; Neural Networks for Pattern Recognition; ISBN 9780198538646

Aproximace spojitých funkcí - dvouvrstvé sítě

Theorem (Cybenko 1989)

Nechť σ je spojitá funkce, která je sigmoidální, tedy je rostoucí a splňuje

$$\sigma(x) = \begin{cases} 1 & \text{pro } x \to +\infty \\ 0 & \text{pro } x \to -\infty \end{cases}$$

Pro každou spojitou funkci $f : [0,1]^n \to [0,1]$ a $\varepsilon > 0$ existuje funkce $F : [0,1]^n \to [0,1]$ počítaná **dvouvrstvou sítí** *jejíž vnitřní neurony mají aktivační funkci σ (výstupní neuron má lineární), která splňuje*

$$|f(\vec{v}) - F(\vec{v})| < \varepsilon \qquad \text{pro každé } \vec{v} \in [0,1]^n.$$

Výpočetní síla neuronových sítí (vyčíslitelnost)

- Uvažujme cyklické sítě
  - s obecně reálnými váhami;
  - jedním vstupním a jedním výstupním neuronem (síť tedy počítá funkci $F : A \to \mathbb{R}$ kde $A \subseteq \mathbb{R}$ obsahuje vstupy nad kterými síť zastaví);
  - plně paralelním pravidlem aktivní dynamiky (v každém kroku se aktualizují všechny neurony);
  - aktivační funkcí

$$\sigma(\xi) = \begin{cases} 1 & \xi \geq 0\,; \\ \xi & 0 \leq \xi \leq 1\,; \\ 0 & \xi < 0. \end{cases}$$

- Slova $\omega \in \{0,1\}^+$ zakódujeme do racionálních čísel pomocí

$$\delta(\omega) = \sum_{i=1}^{|\omega|} \frac{\omega(i)}{2^i} + \frac{1}{2^{|\omega|+1}}$$

Př.: $\omega = 11001$ dá $\delta(\omega) = \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^5} + \frac{1}{2^6}$ $(= 0.110011$ v dvojkové soustavě).

Výpočetní síla neuronových sítí (vyčíslitelnost)

Síť **akceptuje** jazyk $L \subseteq \{0,1\}^+$ pokud počítá funkci $F : A \to \mathbb{R}$ ($A \subseteq \mathbb{R}$) takovou, že

$$\omega \in L \text{ právě když } \delta(\omega) \in A \text{ a } F(\delta(\omega)) > 0.$$

- Cyklické sítě s racionálními váhami jsou ekvivalentní Turingovým strojům
  - Pro každý rekurzivně spočetný jazyk $L \subseteq \{0,1\}^+$ existuje cyklická síť s racionálními váhami a s méně než 1000 neurony, která ho akceptuje.
  - Problém zastavení cyklické sítě s 25 neurony a racionálními váhami je nerozhodnutelný.
  - Existuje univerzální síť (ekvivalent univerzálního Turingova stroje)
- Cyklické sítě s reálnými váhami jsou silnější než Turingovy stroje
  - Pro **každý** jazyk $L \subseteq \{0,1\}^+$ existuje cyklická síť s méně než 1000 neurony, která ho akceptuje.

Shrnutí teoretických výsledků

- Neuronové sítě jsou univerzální výpočetní prostředek
  - dvouvrstvé sítě zvládají Booleovskou logiku
  - dvouvrstvé sítě aproximují libovolné spojité funkce
  - cyklické sítě jsou alespoň tak silné, jako Turingovy stroje
- Tyto výsledky jsou *čistě teoretické*
  - sítě vycházející z obecných argumentů jsou extrémně velké
  - je velmi obtížné je navrhovat
- Sítě mají jiné výhody a účel (učení, generalizace, odolnost, ...)

Srovnání s klasickou architekturou počítačů

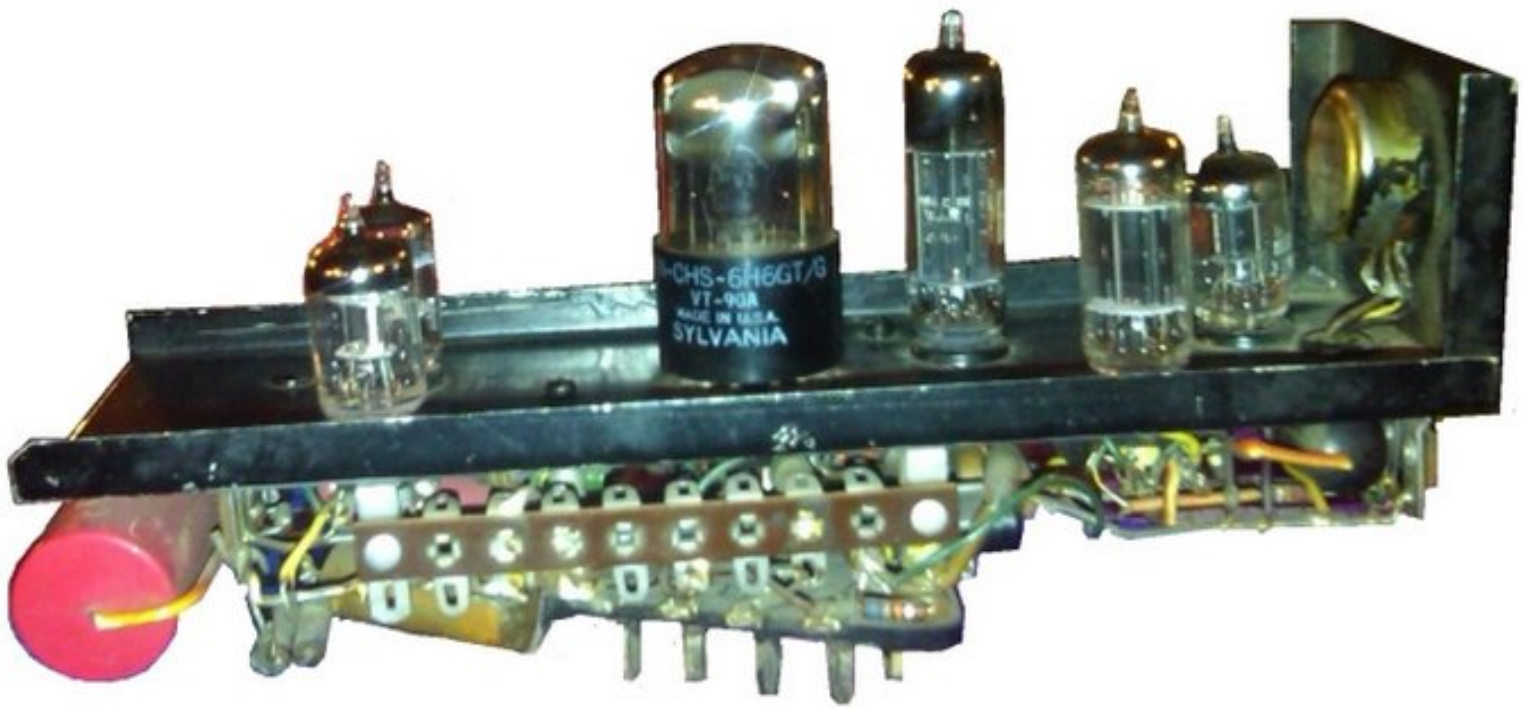
	Neuronové sítě	Klasické počítače
Data	implicitně ve váhách	explicitně
Výpočet	přirozeně paralelní	sekvenční (obvykle), lokalizovaný
Odolnost	odolné vůči nepřesnosti vstupu a poškození	změna jednoho bitu může znamenat krach výpočtu
Přesnost výpočtu	nepřesný, síť si vybaví podobný tréninkový vzor	přesný
Progra-mování	učí se ze vzorového chování	je nutné precizně programovat

Neuropočítače

- Neuropočítač = hardwarová implementace neuronové sítě
- Obvykle jsou to speciální akcelerátory (karty), které dostávají vstupy z obyčejného počítače a vrací výstupy sítě
- Podle typu reprezentace parametrů sítě rozlišujeme neuropočítače na
  - digitální (většina, např. Neuricam TOTEM, IBM TrueNorth a další často pouze výzkumné projekty)
  - analogové (spíše historické, např. Intel ETANN)
  - hybridní (např. AT&T ANNA)
- Lze pozorovat různé stupně hardwarových implementací:
  - hardware pouze provádí výpočet vnitřních potenciálů (lze provádět paralelně)
  - hardware počítá vnitřní potenciály i aktivační funkce (je nutné diskrétně aproximovat spojitou akt. funkci)
  - hardware implementuje učící algoritmus (např. zpětnou propagaci, která se podobá výpočtu sítě, ale od výstupu ke vstupům)

Trocha historie neuropočítačů

- 1951: SNARC (Minski a spol.)
  - první implementace neuronu
  - krysa hledá cestu z ven z bludiště
  - 40 umělých neuronů (300 elektronek, spousta motorů apod.)



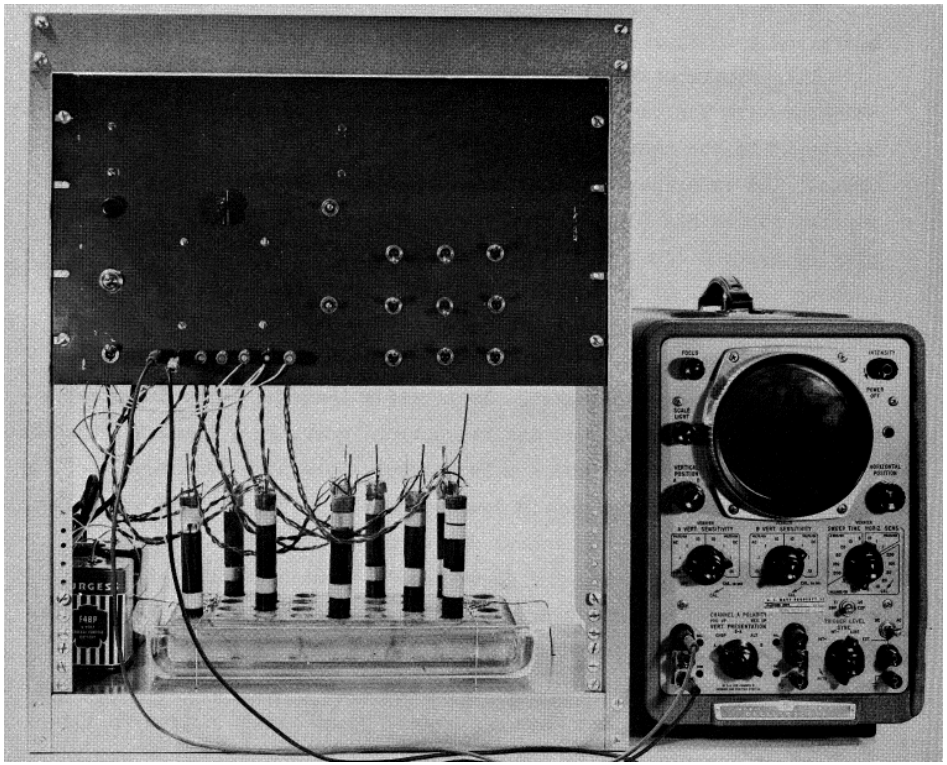
Trocha historie neuropočítačů

- 1957: Mark I Perceptron (Rosenblatt a spol.) - první úspěšná neuronová síť pro rozpoznávání obrazců

  - jednalo se v podstatě o jednovrstvou síť
  - obraz snímán 20×20 fotovodiči
  - intenzita bodů byla vstupem perceptronu (v podstatě formální neuron), který klasifikoval o jaký se jedná znak
  - váhy byly implementovány pomocí potenciometrů (hodnota odporu nastavována motorem pro každý potenciometr zvlášť)
  - umožňoval prohazovat vstupy do neuronů, čímž se demonstrovala schopnost adaptace

Trocha historie neuropočítačů

- 1960: ADALINE (Widrow & Hof)



  - V podstatě jednovrstvá síť
  - Váhy uloženy v pomocí nové součástky **memistor**, která si pamatuje historii proudu ve formě odporu.
  - Widrow založil firmu Memistor Corporation, která prodávala hardwarové implementace neuronových sítí.
- 1960-66: Několik firem zaměřeno na aplikaci neurovýpočtů

Trocha historie neuropočítačů

- 1967-82: Převážně mrtvo po zveřejnění práce Miského a Paperta (oficiálně vydána roku 1969 pod názvem *Perceptrons*)
- 1983-konec devadesátých let: Rozmach neuronových sítí
  - mnoho pokusů o hardwarovou implementaci
    - jednoúčelové čipy (ASIC)
    - programovatelné čipy (FPGA)
  - hardwarové implementace většinou nejsou lepší než softwarové implementace na univerzálních strojích (problémy s pamětí vah, velikostí, rychlostí, nákladností výroby apod.)
- konec devadesátých let-cca 2005: NS zatlačeny do pozadí jinými modely (support vector machines (SVM))
- 2006-nyní: Renesance neuronových sítí
  - hluboké sítě (mnoho vrstev) - většinou lepší než SVM
  - znovuobjeven memistor v HP Labs (nyní se jmenuje memristor, má odlišnou konstrukci)
  - GPU (CUDA) implementace
  - pokusy o velké hardwarové implementace

SyNAPSE (USA)

- Velký výzkumný program financovaný agenturou DARPA
- Hlavní subjekty jsou IBM a HRL, spolupracují s významnými univerzitami, např. Boston, Stanford
- Projekt běží od roku 2008
- Investováno přes 53 milionů dolarů

Cíle

- Vyvinout hardwarovou neuronovou síť odpovídající mozku savce (myši, kočky)
- Výsledný čip by měl simulovat 10 miliard neuronů, 100 bilionů synapsí, spotřeba energie 1 kilowatt ($\sim$ malé topení), velikost 2 dm^3
- Spíše zaměřen na tvorbu počítače s odlišnou architekturou, samotný výzkum mozku není prvořadý

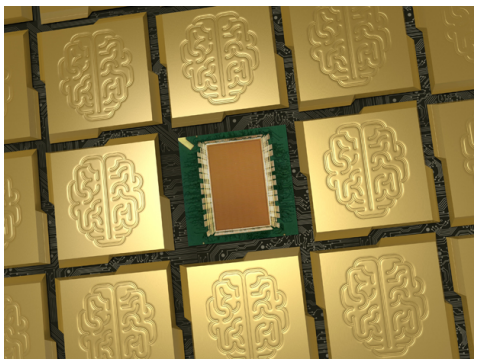
SyNAPSE (USA) – dosavadní výsledky

Simulace mozku “velikosti” kočičího (2009)

- Simulace sítě s 1.6 miliardami neuronů, 8.87 biliony synapsí
- Simulace provedena na superpočítači Dawn (Blue Gene/P), 147,450 CPU, 144 terabytů paměti
- 643 krát pomalejší než reálný běh
- Síť byla modelována dle skutečného mozku (hierarchický model vizuálního kortexu, 4 vrstvy)
- Pozorovány některé děje podobné biologickým (propagace signálu, α, γ vlny)

... simulace podrobena drtivé kritice (viz. později)
... v roce 2012 byl počet neuronů zvýšen na 530 miliard neuronů a 100 bilionů synapsí

SyNAPSE (USA) – TrueNorth



- Chip s 5.4 miliardami tranzistorů
- 4096 neurosynaptických jader propojených sítí, dohromady 1 milion programovatelných "spiking" neuronů, 256 programovatelných synaptických spojů
- globální taktovací frekvence 1-kHz
- nízká spotřeba energie, cca 63mW

- Offline učení, implementováno množství známých algoritmů (konvoluční sítě, RBM apod.)
- Pokusně aplikován na rozpoznávání objektů v obraze.

Human Brain Project, HBP (Evropa)

- Financováno EU, rozpočet 10^9 EUR na 10 let
- Následník Blue Brain Project na EPFL Lausanne (Švýcarsko), další subjekt je ETH Zurich (spolupracují další univerzity) – v roce 2014 přesunuto do Ženevy
- Blue Brain běžel od 2005 do 2012, od 2013 běží Human Brain Project

Hlavní cíl: Poznání funkce lidského mozku

- **léčení onemocnění mozku**
- integrace světového neurovýzkumu
- tvorba myslícího stroje

Postup:

- studium mozkové tkáně pomocí mikroskopů a elektrod
- modelování biologicky věrného modelu
- simulace na superpočítači pomocí programu NEURON (open soft)

HBP (Evropa) – některé výsledky

Blue brain project (2008)

- Model části mozkové kůry krysy (cca 10,000 neuronů), podstatně složitější model neuronu než v případě SyNAPSE
- Simulováno na superpočítači typu Blue Gene/P (IBM dodala se slevou), 16,384 CPU, 56 teraflops, 16 terabytů paměti, 1 PB diskového prostoru (v plánu je použít Deep Project – 10^{18} FLOPS)
- Simulace 300x pomalejší než reálný běh

Human brain project (2015):

- Zjednodušený model nervové soustavy krysy (zhruba 200 000 neuronů)

SyNAPSE vs HBP

IBM Simulates 4.5 percent of the Human Brain, and All of the Cat Brain (Scientific American)

"... performed the first near real-time cortical simulation of the brain that exceeds the scale of a cat cortex" (IBM)

Toto prohlášení bylo podrobeno drtivé kritice v otevřeném dopise Dr. Markrama (šéf HBP)

"This is a mega public relations stunt – a clear case of scientific deception of the public"

"Their so called "neurons" are the tiniest of points you can imagine, a microscopic dot"

"Neurons contain 10's of thousands of proteins that form a network with 10's of millions of interactions. These interactions are incredibly complex and will require solving millions of differential equations. They have none of that."

SyNAPSE vs HBP

"Eugene Izhikevik himself already in 2005 ran a simulation with 100 billion such points interacting just for the fun of it: (over 60 times larger than Modha's simulation)"

Why did they get the Gordon Bell Prize?
"They seem to have been very successful in influencing the committee with their claim, which technically is not peer-reviewed by the respective community and is neuroscientifically outrageous."

But is there any innovation here?
"The only innovation here is that IBM has built a large supercomputer."

But did Mohda not collaborate with neuroscientists?
"I would be very surprised if any neuroscientists that he may have had in his DARPA consortium realized he was going to make such an outrages claim. I can't imagine that the San Fransisco neuroscientists knew he was going to make such a stupid claim. Modha himself is a software engineer with no knowledge of the brain."

... a zatím v Evropě

V roce 2014 dostala Evropská komise otevřený dopis od více než 130 vedoucích významných laboratoří, v němž hrozí bojkotem projektu HBP pokud nedojde k zásadním změnám v řízení.

Peter Dayan, director of the computational neuroscience unit at UCL:

“The main apparent goal of building the capacity to construct a larger-scale simulation of the human brain is radically premature.”

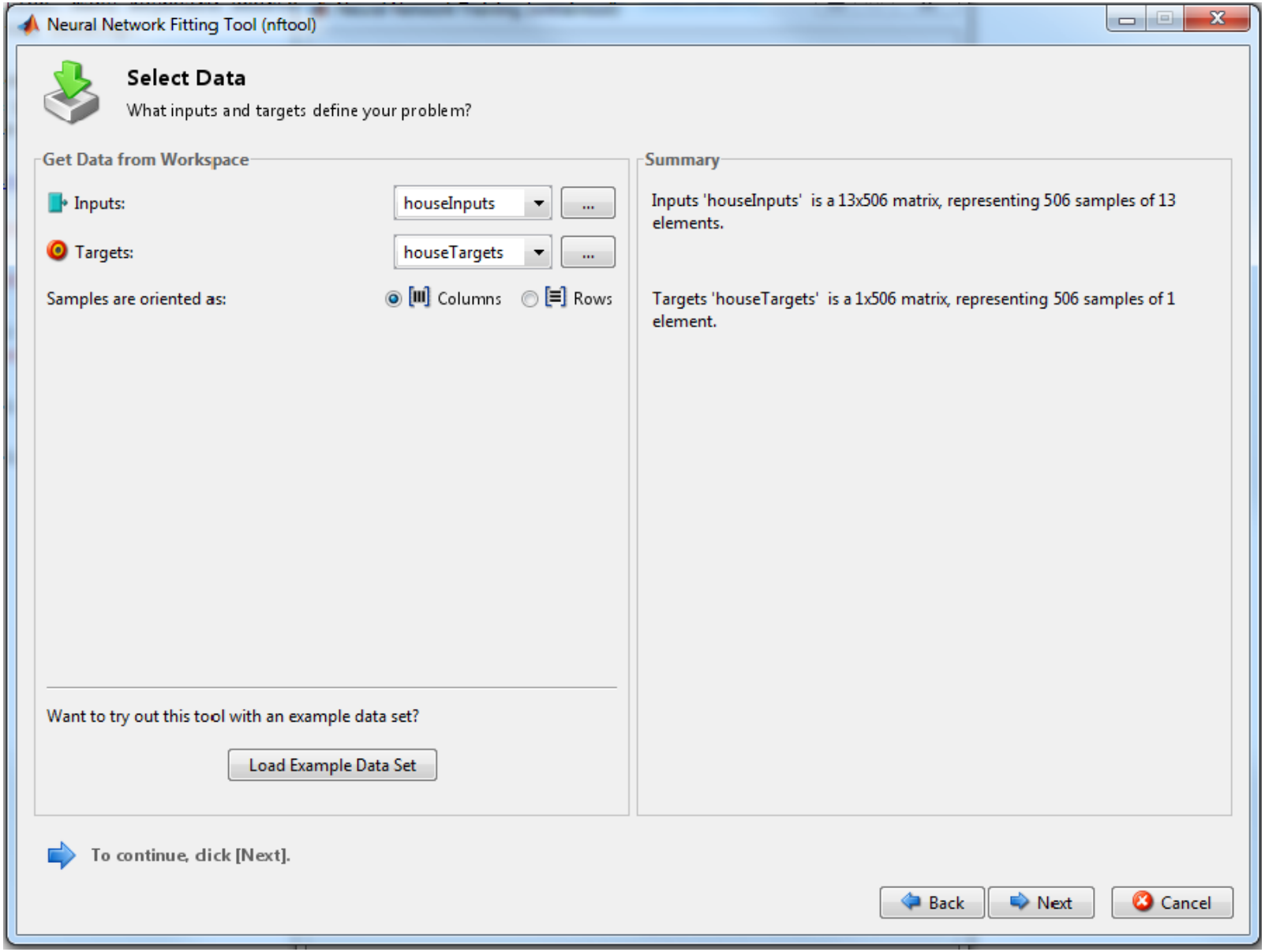
“We are left with a project that can’t but fail from a scientific perspective. It is a waste of money, it will suck out funds from valuable neuroscience research, and would leave the public, who fund this work, justifiably upset.”

Implementace neuronových sítí - software

- NS jsou součástí několika komerčně dostupných balíků pro analýzu dat, např.
  - Alyuda Neurosolutions (software pro tvorbu a aplikaci NS v mnoha oblastech, funguje v Excelu, skvělý marketing :-))
    od finančnictví po biotechnologie, zejména predikce a analýza dat
  - Knowledge Miner (NS + genetické algoritmy pro konstrukci NS, funguje v Excelu)
    použití: např. předvídání změn klimatu, makro a mikro ekonomický vývoj, plánování zdrojů, predikce spotřeby zdrojů, ...
- NS jsou součástí mnoha známých systémů pro vědecké výpočty a analýzu dat
  - např. MATLAB, Mathematica, Statistica, Weka ...
  - tyto systémy obvykle obsahují balíky s příkazy pro tvorbu, učení a aplikaci NS
  - často nabízejí grafické prostředí pro tvorbu sítí

Ilustrace - MatLab

Zdroj: MATLAB - Neural Network Toolbox



Zdroj: MATLAB - Neural Network Toolbox

Ilustrace - MatLab

```
% Solve an Input-Output Fitting problem with a Neural Network
% Script generated by NFTOOL
%
% This script assumes these variables are defined:
%
%   houseInputs - input data.
%   houseTargets - target data.

inputs = houseInputs;
targets = houseTargets;

% Create a Fitting Network
hiddenLayerSize = 10;
net = fitnet(hiddenLayerSize);

% Set up Division of Data for Training, Validation, Testing
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;

% Train the Network
[net,tr] = train(net,inputs,targets);

% Test the Network
outputs = net(inputs);
errors = gsubtract(outputs,targets);
performance = perform(net,targets,outputs)

% View the Network
view(net)
```

Knihovny

Existuje několik knihoven, které lze využít při programování strojového učení, např. FANN a openNN (obě open source)

FANN

- vícevrstvé sítě
- čtyři učící algoritmy (zpětná propagace a další)
- multi-platformní (unix, Linux (Debian), windows) a lze ji použít z různých jazyků (C, C++, PHP, Python, Mathematica, ...)
- existují grafické nástroje postavené na této knihovně (např. FANN Tool)

Další informace o knihovnách lze nalézt např. zde:
http://openann.github.io/OpenANN-apidoc/OtherLibs.html

FANN – ukázka

```
#include "fann.h"

int main()
{
    struct fann *ann = fann_create(1, 0.7, 3, 26, 13, 3);
    fann_train_on_file(ann, "frequencies.data", 200, 10,
        0.0001);
    fann_save(ann, "language_classify.net");
    fann_destroy(ann);
    return 0;
}
```

Python

Python libraries:

- Neurolab: vícevrstvé sítě (učení pomocí variant gradientního sestupu, sdružených gradientů apod.), Hopfieldova síť
- PyBrain (Python-Based Reinforcement Learning, Artificial Intelligence and Neural Network Library):
  - sdružuje algoritmy pro neuronové sítě a reinforcement learning
  - většina známých druhů sítí (včetně hlubokých sítí, RBM, rekurentních sítí, Kohonenových map apod.)
  - ... další učící a optimalizační algoritmy
- Theano:
  - knihovna pro definici, optimalizaci a řešení matematických výrazů s multi-dimenzionálními poli (na GPU)
  - vhodná pro implementaci hlubokých sítí (viz. http://deeplearning.net/tutorial/)
  - Existuje několik knihoven založených na Theano: Pylearn 2, Lasagne, atd.

R

(R is a language and environment for statistical computing and graphics.)

- neuralnet: vícevrstvé sítě (libovolný počet vrstev), varianty zpětné propagace
- nnet: vícevrstvé sítě s jednou skrytou vrstvou, zpětná propagace

Hluboké sítě:

- deepnet a darch: hluboké sítě, restricted Boltzmann machines, zpětná propagace (téměř přesně to, co budeme probírat)
- H2O:
  - určen pro "velká data"
  - H2O is a Java Virtual Machine that is optimized for doing processing of distributed, parallel machine learning algorithms on clusters
  - data nejsou uložena přímo v R, které pouze komunikuje s JVM
  - algoritmy: stochastický gradientní sestup & zpětná propagace

Implementace neuronových sítí - PMML

- značkovací jazyk založený na XML (vyvinut Data Mining Group)
- umožňuje popis modelů pro predikci a analýzu dat (mezi nimi Neuronové sítě)
- podporován mnoha systmémy (Statistica, TERADATA Warehouse Miner, Pentaho Weka, atd.)

```
<NeuralLayer numberOfNeurons="2">
    <Neuron id="3" bias="-3.1808306946637">
        <Con from="0" weight="0.1194776869635O4" />
        <Con from="1" weight="-1.97301278112877" />
        <Con from="2" weight="3.043812517609O6" />
    </Neuron>
    <Neuron id="4" bias="0.7431613537293 23">
        <Con from="0" weight="-0.49411146396721" />
        <Con from="1" weight="2.18588757615864" />
        <Con from="2" weight="-2.01213331163562" />
    </Neuron>
</NeuralLayer>
```

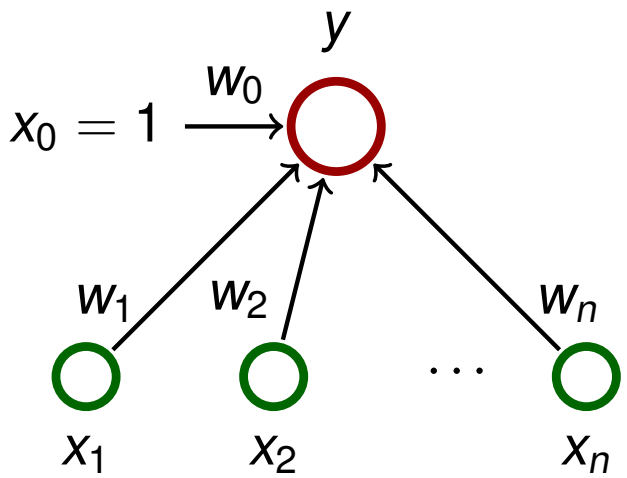
Zdroj: http://www.ibm.com/developerworks/industry/library/ind-PMML1/index.html

Perceptron a ADALINE

- Perceptron
- Perceptronové učící pravidlo
- Konvergence učení perceptronu
- ADALINE
- Učení ADALINE

Perceptron

Organizační dynamika:



$\vec{w} = (w_0, w_1, \ldots, w_n)$ a $\vec{x} = (x_0, x_1, \ldots, x_n)$ kde $x_0 = 1$.

Aktivní dynamika:

- vnitřní potenciál: $\xi = w_0 + \sum_{i=1}^{n} w_i x_i = \sum_{i=0}^{n} w_i x_i = \vec{w} \cdot \vec{x}$
- aktivační funkce: $\sigma(\xi) = \begin{cases} 1 & \xi \geq 0\,; \\ 0 & \xi < 0. \end{cases}$
- funkce sítě: $y[\vec{w}](\vec{x}) = \sigma(\xi) = \sigma(\vec{w} \cdot \vec{x})$

Perceptron

Adaptivní dynamika:

- Dána množina **tréninkových vzorů**

$$\mathcal{T} = \left\{\left(\vec{x}_1, d_1\right), \left(\vec{x}_2, d_2\right), \ldots, \left(\vec{x}_p, d_p\right)\right\}$$

Zde $\vec{x}_k = (x_{k0}, x_{k1} \ldots, x_{kn}) \in \mathbb{R}^{n+1}$, $x_{k0} = 1$, je vstup k-tého vzoru a $d_k \in \{0, 1\}$ je očekávaný výstup.

(d_k určuje, do které ze dvou kategorií dané $\vec{x}_k = (x_{k0}, x_{k1} \ldots, x_{kn})$ patří).

- Vektor vah $\vec{w} \in \mathbb{R}^{n+1}$ je **konzistentní s** $\mathcal{T}$ pokud $y[\vec{w}](\vec{x}_k) = \sigma(\vec{w} \cdot \vec{x}_k) = d_k$ pro každé $k = 1, \ldots, p$.

  Množina $\mathcal{T}$ je **vnitřně konzistentní** pokud existuje vektor $\vec{w}$, který je s ní konzistentní.

- Cílem je nalézt vektor $\vec{w}$, který je konzistentní s $\mathcal{T}$ za předpokladu, že $\mathcal{T}$ je vnitřně konzistentní.

Perceptron - adaptivní dynamika

Online učící algoritmus:

Idea: Cyklicky prochází vzory a adaptuje podle nich váhy, tj. otáčí dělící nadrovinu tak, aby se zmenšila vzdálenost špatně klasifikovaného vzoru od jeho příslušného poloprostoru.

Prakticky počítá posloupnost vektorů vah $\vec{w}^{(0)}, \vec{w}^{(1)}, \vec{w}^{(2)}, \ldots$.

- váhy v $\vec{w}^{(0)}$ jsou inicializovány náhodně blízko 0
- v kroku $t+1$ je $\vec{w}^{(t+1)}$ vypočteno takto:

$$\begin{aligned} \vec{w}^{(t+1)} &= \vec{w}^{(t)} - \varepsilon \cdot \left(y[\vec{w}^{(t)}](\vec{x}_k) - d_k \right) \cdot \vec{x}_k \\ &= \vec{w}^{(t)} - \varepsilon \cdot \left(\sigma\left(\vec{w}^{(t)} \cdot \vec{x}_k\right) - d_k \right) \cdot \vec{x}_k \end{aligned}$$

Zde $k = (t \mod p) + 1$ (tj. cyklické procházení vzorů) a $0 < \varepsilon \leq 1$ je **rychlost učení**.

Theorem (Rosenblatt)

Jestliže je $\mathcal{T}$ vnitřně konzistentní, pak existuje t^* takové, že $\vec{w}^{(t^*)}$ je konzistentní s $\mathcal{T}$.

Důkaz Rosenblattovy věty

Pro zjednodušení budeme dále předpokládat, že $\varepsilon = 1$.

Nejprve si algoritmus přepíšeme do méně kompaktní formy:

- váhy v $\vec{w}^{(0)}$ jsou inicializovány náhodně blízko 0
- v kroku $t+1$ je $\vec{w}^{(t+1)}$ vypočteno takto:
  - **Jestliže** $\sigma(\vec{w}^{(t)} \cdot \vec{x}_k) = d_k$, **pak** $\vec{w}^{(t+1)} = \vec{w}^{(t)}$
  - **Jestliže** $\sigma(\vec{w}^{(t)} \cdot \vec{x}_k) \neq d_k$, **pak**
    - $\vec{w}^{(t+1)} = \vec{w}^{(t)} + \vec{x}_k$ pro $d_k = 1$
    - $\vec{w}^{(t+1)} = \vec{w}^{(t)} - \vec{x}_k$ pro $d_k = 0$

    (Řekneme, že nastala korekce.)

  kde $k = (t \mod p) + 1$.

Důkaz Rosenblattovy věty

(Pro daný vektor $\vec{a} = (a_0, \ldots, a_n)$ označme $\left\|\vec{a}\right\|$ jeho eukleidovskou normu $\sqrt{\vec{a} \cdot \vec{a}} = \sqrt{\sum_{i=0}^{n} a_i^2}$)

Idea:

- Uvážíme *hodně dlouhý vektor* (spočítáme jak dlouhý) $\vec{w}^*$, který je konzistentní s $\mathcal{T}$.
- Ukážeme, že pokud došlo v kroku $t+1$ ke korekci vah (tedy buď $\vec{w}^{(t+1)} = \vec{w}^{(t)} + \vec{x}_k$ nebo $\vec{w}^{(t+1)} = \vec{w}^{(t)} - \vec{x}_k$), pak

$$\left\|\vec{w}^{(t+1)} - \vec{w}^*\right\|^2 \leq \left\|\vec{w}^{(t)} - \vec{w}^*\right\|^2 - \max_i \left\|\vec{x}_i\right\|$$

  Všimněte si, že $\max_i \left\|\vec{x}_i\right\| > 0$ *nezávisí* na t.
- Z toho plyne, že algoritmus nemůže udělat nekonečně mnoho korekcí.

Důkaz Rosenblattovy věty

Uvažme vektor $\vec{w}^*$ konzistentní s $\mathcal{T}$.

Búno předpokládejme, že $\vec{w}^* \cdot \vec{x}_k \neq 0$ pro $k = 1, \ldots, p$.

Předp., že v kroku $t+1$ došlo ke korekci, a že $k = (t \mod p) + 1$.

Ukážeme, že

$$\left\|\vec{w}^{(t+1)} - \vec{w}^*\right\|^2 \quad \leq \quad \left\|\vec{w}^{(t)} - \vec{w}^*\right\|^2 \quad + \quad \left\|\vec{x}_k\right\|^2 - 2|\vec{w}^* \cdot \vec{x}_k|$$

(Potom bude k důkazu věty stačit nahlédnout, že pro "dlouhý" vektor $\vec{w}^*$ je $|\vec{w}^* \cdot \vec{x}_k|$ "velké" kladné číslo.)

Rozlišíme dva případy: $\vec{d}_k = 1$ a $\vec{d}_k = 0$.

Důkaz Rosenblattovy věty

Předpokládejme $\vec{d}_k = 1$:

Došlo ke korekci, tedy $\vec{w}^{(t+1)} = \vec{w}^{(t)} + \vec{x}_k$ a tedy $\vec{w}^{(t+1)} - \vec{w}^* = (\vec{w}^{(t)} - \vec{w}^*) + \vec{x}_k$. Pak

$$\begin{aligned}
\left\|\vec{w}^{(t+1)} - \vec{w}^*\right\|^2 &= \left\|(\vec{w}^{(t)} - \vec{w}^*) + \vec{x}_k\right\|^2 \\
&= \left[(\vec{w}^{(t)} - \vec{w}^*) + \vec{x}_k\right]\left[(\vec{w}^{(t)} - \vec{w}^*) + \vec{x}_k\right] \\
&= \left\|(\vec{w}^{(t)} - \vec{w}^*)\right\|^2 + \left\|\vec{x}_k\right\|^2 + 2(\vec{w}^{(t)} - \vec{w}^*) \cdot \vec{x}_k \\
&= \left\|(\vec{w}^{(t)} - \vec{w}^*)\right\|^2 + \left\|\vec{x}_k\right\|^2 + 2\vec{w}^{(t)} \cdot \vec{x}_k - 2\vec{w}^* \cdot \vec{x}_k \\
&\leq \left\|(\vec{w}^{(t)} - \vec{w}^*)\right\|^2 + \left\|\vec{x}_k\right\|^2 - 2|\vec{w}^* \cdot \vec{x}_k|
\end{aligned}$$

Poslední nerovnost plyne z toho, že:

- došlo ke korekci při $\vec{d}_k = 1$, tedy muselo platit $\vec{w}^{(t)} \cdot \vec{x}_k < 0$,
- $\vec{w}^*$ je konzistentní s $\mathcal{T}$ a tedy $\vec{w}^* \cdot \vec{x}_k > 0$.

Důkaz Rosenblattovy věty

Předpokládejme $\vec{d}_k = 0$:

Došlo ke korekci, tedy $\vec{w}^{(t+1)} = \vec{w}^{(t)} - \vec{x}_k$ a tedy $\vec{w}^{(t+1)} - \vec{w}^* = (\vec{w}^{(t)} - \vec{w}^*) - \vec{x}_k$. Pak

$$\begin{aligned}
\left\|\vec{w}^{(t+1)} - \vec{w}^*\right\|^2 &= \left\|(\vec{w}^{(t)} - \vec{w}^*) - \vec{x}_k\right\|^2 \\
&= \left[(\vec{w}^{(t)} - \vec{w}^*) - \vec{x}_k\right]\left[(\vec{w}^{(t)} - \vec{w}^*) - \vec{x}_k\right] \\
&= \left\|(\vec{w}^{(t)} - \vec{w}^*)\right\|^2 + \left\|\vec{x}_k\right\|^2 - 2(\vec{w}^{(t)} - \vec{w}^*) \cdot \vec{x}_k \\
&= \left\|(\vec{w}^{(t)} - \vec{w}^*)\right\|^2 + \left\|\vec{x}_k\right\|^2 - 2\vec{w}^{(t)} \cdot \vec{x}_k + 2\vec{w}^* \cdot \vec{x}_k \\
&\leq \left\|(\vec{w}^{(t)} - \vec{w}^*)\right\|^2 + \left\|\vec{x}_k\right\|^2 - 2|\vec{w}^* \cdot \vec{x}_k|
\end{aligned}$$

Poslední nerovnost plyne z toho, že:

- došlo ke korekci při $\vec{d}_k = 0$, tedy muselo platit $\vec{w}^{(t)} \cdot \vec{x}_k \geq 0$,
- $\vec{w}^*$ je konzistentní s $\mathcal{T}$ a tedy $\vec{w}^* \cdot \vec{x}_k < 0$.

Důkaz Rosenblattovy věty

Máme dokázáno:

$$\left\|\vec{w}^{(t+1)} - \vec{w}^*\right\|^2 \leq \left\|\vec{w}^{(t)} - \vec{w}^*\right\|^2 + \left\|\vec{x}_k\right\|^2 - 2|\vec{w}^* \cdot \vec{x}_k|$$

Nechť nyní $\vec{w}^* = \alpha \cdot \vec{w}^+$ kde $\alpha > 0$. Pak

$$\left\|\vec{w}^{(t+1)} - \vec{w}^*\right\|^2 \leq \left\|\vec{w}^{(t)} - \vec{w}^*\right\|^2 + \left\|\vec{x}_k\right\|^2 - 2\alpha|\vec{w}^+ \cdot \vec{x}_k|$$

Nyní stačí uvážit $\alpha = \frac{\max_k \left\|\vec{x}_k\right\|^2}{\min_k |\vec{w}^+ \cdot \vec{x}_k|}$ a dostaneme

$$\left\|\vec{x}_k\right\|^2 - 2\alpha|\vec{w}^+ \cdot \vec{x}_k| \leq \left\|\vec{x}_k\right\|^2 - 2\max_k \left\|\vec{x}_k\right\|^2 \frac{|\vec{w}^+ \cdot \vec{x}_k|}{\min_k |\vec{w}^+ \cdot \vec{x}_k|} \leq -\max_k \left\|\vec{x}_k\right\|^2$$

Což dá

$$\left\|\vec{w}^{(t+1)} - \vec{w}^*\right\| \leq \left\|\vec{w}^{(t)} - \vec{w}^*\right\|^2 - \max_k \left\|\vec{x}_k\right\|^2$$

kdykoliv došlo ke korekci.

Z toho plyne, že nemůže dojít k nekonečně mnoha korekcím. □

Perceptron - adaptivní dynamika

Dávkový učící algoritmus:

Vypočte posloupnost $\vec{w}^{(0)}, \vec{w}^{(1)}, \vec{w}^{(2)}, \ldots$ váhových vektorů.

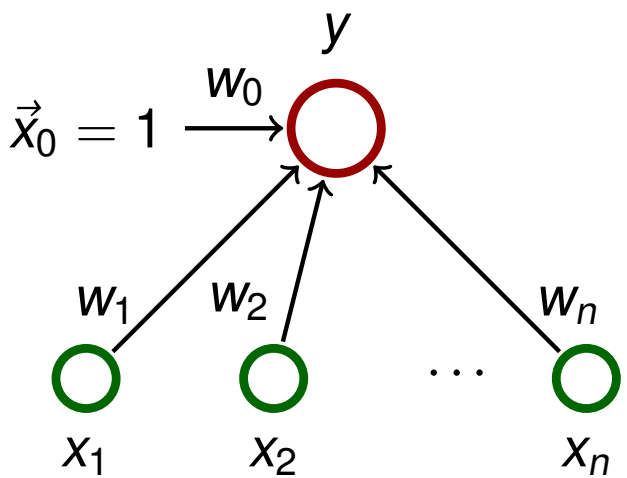
- váhy v $\vec{w}^{(0)}$ jsou inicializovány náhodně blízko 0
- v kroku $t+1$ je $\vec{w}^{(t+1)}$ vypočteno takto:

$$\vec{w}^{(t+1)} = \vec{w}^{(t)} - \varepsilon \cdot \sum_{k=1}^{p} (\sigma(\vec{w}^{(t)} \cdot \vec{x}_k) - d_k) \cdot \vec{x}_k$$

Zde $k = (t \mod p) + 1$
a $0 < \varepsilon \leq 1$ je **rychlost učení**.

ADALINE

Organizační dynamika:



$\vec{w} = (w_0, w_1, \ldots, w_n)$ a $\vec{x} = (x_0, x_1, \ldots, x_n)$ kde $x_0 = 1$.

Aktivní dynamika:

- vnitřní potenciál: $\xi = w_0 + \sum_{i=1}^{n} w_i x_i = \sum_{i=0}^{n} w_i x_i = \vec{w} \cdot \vec{x}$
- aktivační funkce: $\sigma(\xi) = \xi$
- funkce sítě: $y[\vec{w}](\vec{x}) = \sigma(\xi) = \vec{w} \cdot \vec{x}$

ADALINE

Adaptivní dynamika:

- Dána množina **tréninkových vzorů**

$$\mathcal{T} = \left\{\left(\vec{x}_1, d_1\right), \left(\vec{x}_2, d_2\right), \ldots, \left(\vec{x}_p, d_p\right)\right\}$$

Zde $\vec{x}_k = (x_{k0}, x_{k1} \ldots, x_{kn}) \in \mathbb{R}^{n+1}$, $x_{k0} = 1$, je vstup k-tého vzoru a $d_k \in \mathbb{R}$ je očekávaný výstup.

Intuice: chceme, aby síť počítala afinní aproximaci funkce, jejíž (některé) hodnoty nám předepisuje tréninková množina.

- **Chybová funkce:**

$$E(\vec{w}) = \frac{1}{2}\sum_{k=1}^{p}\left(\vec{w}\cdot\vec{x}_k - d_k\right)^2 = \frac{1}{2}\sum_{k=1}^{p}\left(\sum_{i=0}^{n} w_i x_{ki} - d_k\right)^2$$

- Cílem je nalézt $\vec{w}$, které minimalizuje $E(\vec{w})$.

Gradient chybové funkce

Uvažme **gradient** chybové funkce:

$$\nabla E(\vec{w}) = \left(\frac{\partial E}{\partial w_0}(\vec{w}), \ldots, \frac{\partial E}{\partial w_n}(\vec{w}) \right)$$

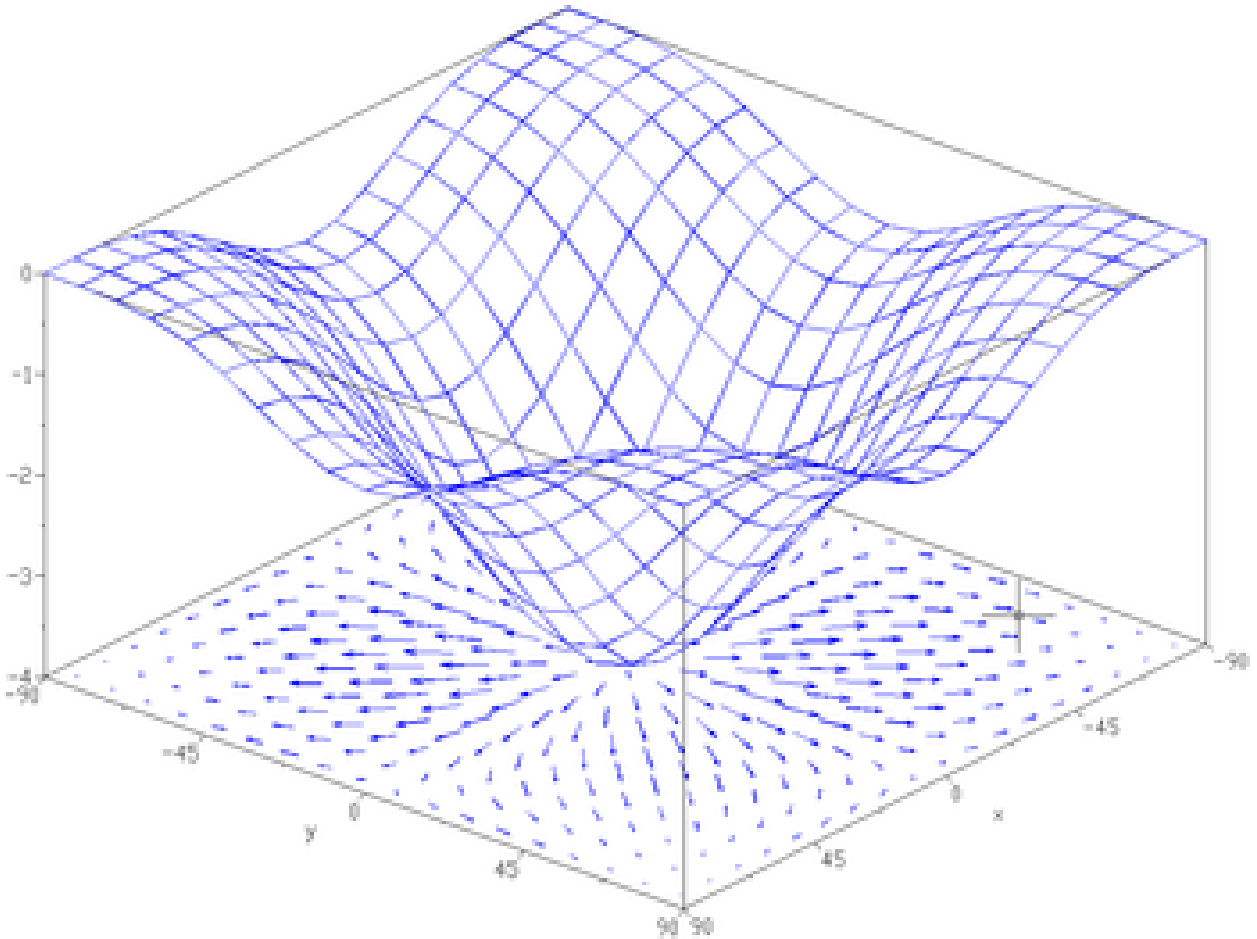
Intuice: $\nabla E(\vec{w})$ je vektor ve **váhovém prostoru**, který ukazuje směrem nejstrmějšího růstu funkce $E(\vec{w})$. Vektory $\vec{x}_k$ zde slouží pouze jako parametry funkce $E(\vec{w})$ a jsou tedy fixní!

Fact

Pokud $\nabla E(\vec{w}) = \vec{0} = (0, \ldots, 0)$, *pak* $\vec{w}$ *je globální minimum funkce* E.

Námi uvažovaná chybová funkce $E(\vec{w})$ má globální minimum, protože je konvexním paraboloidem.

Gradient - ilustrace



Pozor! Tento obrázek pouze ilustruje pojem gradientu, nezobrazuje chybovou funkci $E(\vec{w})$

Gradient chybové funkce ADALINE

$$
\begin{aligned}
\frac{\partial E}{\partial w_\ell}(\vec{w}) &= \frac{1}{2}\sum_{k=1}^{p}\frac{\delta E}{\delta w_\ell}\left(\sum_{i=0}^{n} w_i x_{ki} - d_k\right)^2 \\
&= \frac{1}{2}\sum_{k=1}^{p} 2\left(\sum_{i=0}^{n} w_i x_{ki} - d_k\right)\frac{\delta E}{\delta w_\ell}\left(\sum_{i=0}^{n} w_i x_{ki} - d_k\right) \\
&= \frac{1}{2}\sum_{k=1}^{p} 2\left(\sum_{i=0}^{n} w_i x_{ki} - d_k\right)\left(\sum_{i=0}^{n}\left(\frac{\delta E}{\delta w_\ell} w_i x_{ki}\right) - \frac{\delta E}{\delta w_\ell} d_k\right) \\
&= \sum_{k=1}^{p}\left(\vec{w}\cdot\vec{x}_k - d_k\right) x_{k\ell}
\end{aligned}
$$

Tedy

$$
\nabla E(\vec{w}) = \left(\frac{\partial E}{\partial w_0}(\vec{w}), \ldots, \frac{\partial E}{\partial w_n}(\vec{w})\right) = \sum_{k=1}^{p}\left(\vec{w}\cdot\vec{x}_k - d_k\right)\vec{x}_k
$$

ADALINE - adaptivní dynamika

Dávkový algoritmus (gradientní sestup):

- váhy v $\vec{w}^{(0)}$ jsou inicializovány náhodně blízko 0
- v kroku $t+1$ je $\vec{w}^{(t+1)}$ vypočteno takto:

$$\begin{aligned}\vec{w}^{(t+1)} &= \vec{w}^{(t)} - \varepsilon \cdot \nabla E(\vec{w}^{(t)}) \\ &= \vec{w}^{(t)} - \varepsilon \cdot \sum_{k=1}^{p} \left(\vec{w}^{(t)} \cdot \vec{x}_k - d_k\right) \cdot \vec{x}_k\end{aligned}$$

Zde $k = (t \mod p) + 1$
a $0 < \varepsilon \leq 1$ je rychlost učení.

(Všimněte si, že tento algoritmus je téměř stejný jako pro perceptron, protože $\vec{w}^{(t)} \cdot \vec{x}_k$ je hodnota funkce sítě (tedy $\sigma(\vec{w}^{(t)} \cdot \vec{x}_k)$ kde $\sigma(\xi) = \xi$).)

Proposition

Pro dostatečně malé $\varepsilon > 0$ posloupnost $\vec{w}^{(0)}, \vec{w}^{(1)}, \vec{w}^{(2)}, \ldots$ konverguje (po složkách) ke globálnímu minimu funkce E (tedy k vektoru $\vec{w}$, který splňuje $\nabla E(\vec{w}) = \vec{0}$).

ADALINE - adaptivní dynamika

Online algoritmus (Delta-rule, Widrow-Hoff rule):

- váhy v $\vec{w}^{(0)}$ jsou inicializovány náhodně blízko 0
- v kroku $t+1$ je $\vec{w}^{(t+1)}$ vypočteno takto:

$$\vec{w}^{(t+1)} = \vec{w}^{(t)} - \varepsilon(t) \cdot \left(\vec{w}^{(t)} \cdot \vec{x}_k - d_k\right) \cdot \vec{x}_k$$

kde $k = t \mod p + 1$
a $0 < \varepsilon(t) \leq 1$ je rychlost učení v kroku $t+1$.

Všimněte si, že tento algoritmus nepracuje s celým gradientem, ale jenom s jeho částí, která přísluší právě zpracovávanému vzoru!

Theorem (Widrow & Hoff)

Pokud $\varepsilon(t) = \frac{1}{t}$ pak $\vec{w}^{(0)}, \vec{w}^{(1)}, \vec{w}^{(2)}, \ldots$ konverguje ke globálnímu minimu chybové funkce E.

ADALINE - klasifikace

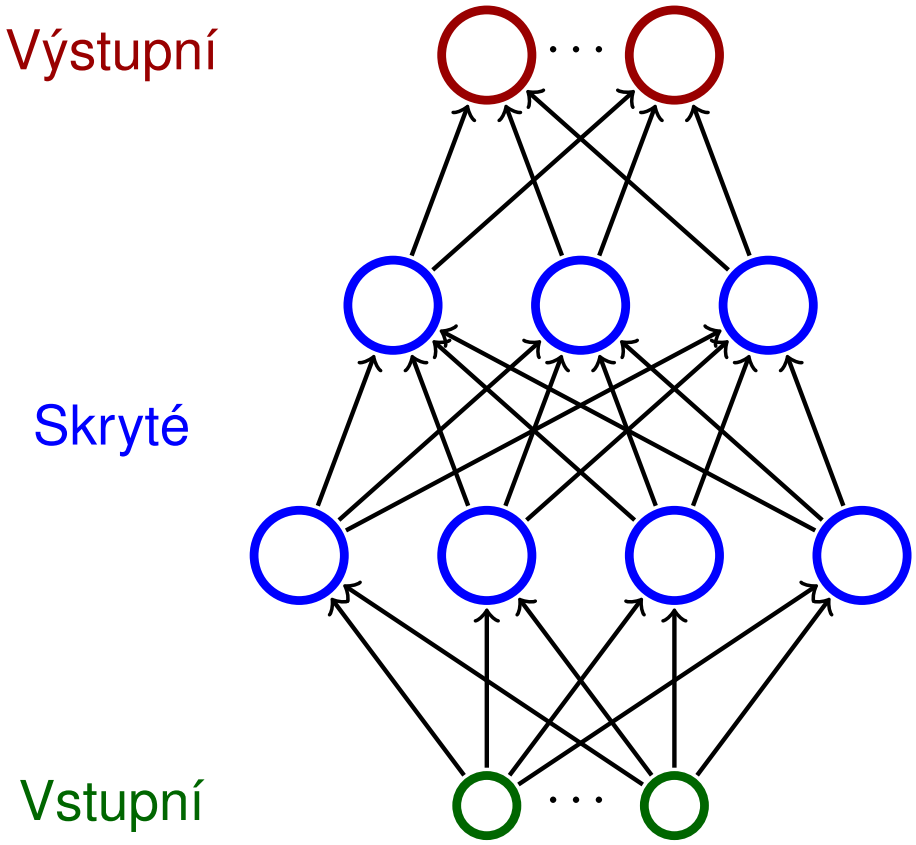
- Množina **tréninkových vzorů** je

$$\mathcal{T} = \left\{\left(\vec{x}_1, d_1\right), \left(\vec{x}_2, d_2\right), \ldots, \left(\vec{x}_p, d_p\right)\right\}$$

  kde $\vec{x}_k = (x_{k0}, x_{k1}, \ldots, x_{kn}) \in \mathbb{R}^{n+1}$ a $d_k \in \{1, -1\}$.
- Síť se natrénuje ADALINE algoritmem.
- Očekáváme, že bude platit následující:
  - jestliže $d_k = 1$, pak $\vec{w} \cdot \vec{x}_k \geq 0$
  - jestliže $d_k = -1$, pak $\vec{w} \cdot \vec{x}_k < 0$
- To nemusí vždy platit, ale často platí. Výhoda je, že se ADALINE algoritmus postupně stabilizuje i v neseparabilním případě (na rozdíl od perceptronového algoritmu).

Vícevrstvá síť

Organizační dynamika:



- Neurony jsou rozděleny do **vrstev** (vstupní a výstupní vrstva, obecně několik skrytých vrstev)
- Vrstvy číslujeme od 0; vstupní vrstva je nultá
  - Např. třívrstvá síť se skládá z jedné vstupní, dvou skrytých a jedné výstupní vrstvy.
- Neurony v ℓ-té vrstvě jsou spojeny se všemi neurony ve vrstvě $\ell + 1$.
- Vícevrstvou síť lze zadat počty neuronů v jednotlivých vrstvách (např. 2-4-3-2)

Vícevrstvá síť

Značení:

- Označme
  - X množinu vstupních neuronů
  - Y množinu výstupních neuronů
  - Z množinu všech neuronů (tedy $X, Y \subseteq Z$)
- jednotlivé neurony budeme značit indexy i, j apod.
- ξ_j je vnitřní potenciál neuronu j po skončení výpočtu
- y_j je stav (výstup) neuronu j po skončení výpočtu

  (zde definujeme $y_0 = 1$ jako hodnotu formálního jednotkového vstupu)
- w_{ji} je váha spoje **od** neuronu i **do** neuronu j

  (zejména w_{j0} je váha speciálního jednotkového vstupu, tj. $w_{j0} = -b_j$ kde b_j je bias neuronu j)
- $j_\leftarrow$ je množina všech neuronů, **z nichž** vede spoj do j (zejména $0 \in j_\leftarrow$)
- $j^\rightarrow$ je množina všech neuronů, **do nichž** vede spoj z j

Vícevrstvá síť

Aktivní dynamika:

- vnitřní potenciál neuronu j:

$$\xi_j = \sum_{i \in j_{\leftarrow}} w_{ji} y_i$$

- aktivační funkce σ_j pro neuron j je libovolná diferencovatelná funkce
  [např. logistická sigmoida $\sigma_j(\xi) = \frac{1}{1+e^{-\lambda_j \xi}}$]
- Stav nevstupního neuronu $j \in Z \setminus X$ po skončení výpočtu je

$$y_j = \sigma_j(\xi_j)$$

  (y_j závisí na konfiguraci $\vec{w}$ a vstupu $\vec{x}$, proto budu občas psát $y_j(\vec{w}, \vec{x})$)

- Síť počítá funkci z $\mathbb{R}^{|X|}$ do $\mathbb{R}^{|Y|}$. Výpočet probíhá po vrstvách. Na začátku jsou hodnoty vstupních neuronů nastaveny na vstup sítě. V kroku ℓ jsou vyhodnoceny neurony z ℓ-té vrstvy.

Vícevrstvá síť

Adaptivní dynamika:

- Dána množina $\mathcal{T}$ **tréninkových vzorů** tvaru

$$\left\{ \left(\vec{x}_k, \vec{d}_k \right) \quad \middle| \quad k = 1, \ldots, p \right\}$$

kde každé $\vec{x}_k \in \mathbb{R}^{|X|}$ je vstupní vektor a každé $\vec{d}_k \in \mathbb{R}^{|Y|}$ je očekávaný výstup sítě. Pro každé $j \in Y$ označme d_{kj} očekávaný výstup neuronu j pro vstup $\vec{x}_k$
(vektor $\vec{d}_k$ lze tedy psát jako $\left(d_{kj} \right)_{j \in Y}$).

- **Chybová funkce:**

$$E(\vec{w}) = \sum_{k=1}^{p} E_k(\vec{w})$$

kde

$$E_k(\vec{w}) = \frac{1}{2} \sum_{j \in Y} \left(y_j(\vec{w}, \vec{x}_k) - d_{kj} \right)^2$$

Vícevrstvá síť - učící algoritmus

Dávkový algoritmus (gradientní sestup):

Algoritmus počítá posloupnost vektorů vah $\vec{w}^{(0)}, \vec{w}^{(1)}, \ldots$.

- váhy v $\vec{w}^{(0)}$ jsou inicializovány náhodně blízko 0
- v kroku $t+1$ (zde $t = 0, 1, 2 \ldots$) je $\vec{w}^{(t+1)}$ vypočteno takto:

$$w_{ji}^{(t+1)} = w_{ji}^{(t)} + \Delta w_{ji}^{(t)}$$

kde

$$\Delta w_{ji}^{(t)} = -\varepsilon(t) \cdot \frac{\partial E}{\partial w_{ji}}(\vec{w}^{(t)})$$

je změna váhy w_{ji} v kroku $t+1$ a $0 < \varepsilon(t) \leq 1$ je rychlost učení v kroku $t+1$.

Všimněte si, že $\frac{\partial E}{\partial w_{ji}}(\vec{w}^{(t)})$ je komponenta gradientu ∇E, tedy změnu vah v kroku $t+1$ lze zapsat také takto: $\vec{w}^{(t+1)} = \vec{w}^{(t)} - \varepsilon(t) \cdot \nabla E(\vec{w}^{(t)})$.

Vícevrstvá síť - gradient chybové funkce

Pro každé w_{ji} máme

$$\frac{\partial E}{\partial w_{ji}} = \sum_{k=1}^{p} \frac{\partial E_k}{\partial w_{ji}}$$

kde pro každé $k = 1, \ldots, p$ platí

$$\frac{\partial E_k}{\partial w_{ji}} = \frac{\partial E_k}{\partial y_j} \cdot \sigma_j'(\xi_j) \cdot y_i$$

a pro každé $j \in Z \setminus X$ dostaneme

$$\frac{\partial E_k}{\partial y_j} = y_j - d_{kj} \qquad \text{pro } j \in Y$$

$$\frac{\partial E_k}{\partial y_j} = \sum_{r \in j^{\rightarrow}} \frac{\partial E_k}{\partial y_r} \cdot \sigma_r'(\xi_r) \cdot w_{rj} \qquad \text{pro } j \in Z \setminus (Y \cup X)$$

(Zde všechna y_j jsou ve skutečnosti $y_j(\vec{w}, \vec{x}_k)$).

Vícevrstvá síť - gradient chybové funkce

- Pokud $\sigma_j(\xi) = \frac{1}{1+e^{-\lambda_j \xi}}$ pro každé $j \in Z$ pak

$$\sigma_j'(\xi_j) = \lambda_j y_j (1 - y_j)$$

a dostaneme pro každé $j \in Z \setminus X$:

$$\frac{\partial E_k}{\partial y_j} = y_j - d_{kj} \qquad \text{pro } j \in Y$$

$$\frac{\partial E_k}{\partial y_j} = \sum_{r \in j^{\rightarrow}} \frac{\partial E_k}{\partial y_r} \cdot \lambda_r y_r (1 - y_r) \cdot w_{rj} \qquad \text{pro } j \in Z \setminus (Y \cup X)$$

- Pokud $\sigma_j(\xi) = a \cdot \tanh(b \cdot \xi_j)$ pro každé $j \in Z$ pak

$$\sigma_j'(\xi_j) = \frac{b}{a}(a - y_j)(a + y_j)$$

Vícevrstvá síť - výpočet gradientu

Algoritmicky lze $\frac{\partial E}{\partial w_{ji}} = \sum_{k=1}^{p} \frac{\partial E_k}{\partial w_{ji}}$ spočítat takto:

Polož $\mathcal{E}_{ji} := 0$ (na konci výpočtu bude $\mathcal{E}_{ji} = \frac{\partial E}{\partial w_{ji}}$)

Pro každé $k = 1, \ldots, p$ udělej následující

1. spočítej y_j pro každé $j \in Z$ a k-tý vzor, tedy $y_j = y_j(\vec{w}, \vec{x}_k)$
   (tj. vyhodnoť síť ve standardním aktivním režimu)
2. pro všechna $j \in Z$ spočítej $\frac{\partial E_k}{\partial y_j}$ pomocí zpětného šíření
   (viz. následující slajd!)
3. spočítej $\frac{\partial E_k}{\partial w_{ji}}$ pro všechna w_{ji} pomocí vzorce
   $$\frac{\partial E_k}{\partial w_{ji}} := \frac{\partial E_k}{\partial y_j} \cdot \sigma_j'(\xi_j) \cdot y_i$$
4. $\mathcal{E}_{ji} := \mathcal{E}_{ji} + \frac{\partial E_k}{\partial w_{ji}}$

Výsledné E_{ji} se rovná $\frac{\partial E}{\partial w_{ji}}$.

Vícevrstvá síť - zpětné šíření

$\frac{\partial E_k}{\partial y_j}$ lze spočítat pomocí zpětného šíření:

- spočítáme $\frac{\partial E_k}{\partial y_j}$ pro $j \in Y$ pomocí vzorce $\frac{\partial E_k}{\partial y_j} = y_j - d_{kj}$
- rekurzivně spočítáme zbylé $\frac{\partial E_k}{\partial y_j}$:

  Nechť j je v ℓ-té vrstvě a předpokládejme, že $\frac{\partial E_k}{\partial y_r}$ už máme spočítáno pro všechny neurony z vyšších vrstev (tedy vrstev $\ell + 1, \ell + 2, \ldots$).
  Pak lze $\frac{\partial E_k}{\partial y_j}$ spočítat pomocí vzorce

$$\frac{\partial E_k}{\partial y_j} = \sum_{r \in j^{\rightarrow}} \frac{\partial E_k}{\partial y_r} \cdot \sigma_r'(\xi_r) \cdot w_{rj}$$

  protože všechny neurony $r \in j^{\rightarrow}$ patří do vrstvy $\ell + 1$.

Složitost dávkového algoritmu

Výpočet hodnoty $\frac{\partial E}{\partial w_{ji}}(\vec{w}^{(t-1)})$ probíhá v lineárním čase vzhledem k velikosti sítě a počtu tréninkových vzorů

(za předpokladu jednotkové ceny operací včetně vyhodnocení $\sigma_r'(\xi_r)$ pro dané ξ_r)

Zdůvodnění: Algoritmus provede p krát následující

1. vyhodnocení sítě (tj. výpočet $y_j(\vec{w}, \vec{x}_k)$)
2. výpočet $\frac{\partial E_k}{\partial y_j}$ zpětným šířením
3. výpočet $\frac{\partial E_k}{\partial w_{ji}}$
4. přičtení $\frac{\partial E_k}{\partial w_{ji}}$ k $\mathcal{E}_{ji}$

Kroky 1. - 3. proběhnou v lineárním čase a krok 4. v konstantním vzhledem k velikosti sítě.

Počet iterací gradientního sestupu pro přiblížení se (lokálnímu) minimu může být velký ...

Vícevrstvá síť - učící algoritmus

Online algoritmus:

Algoritmus počítá posloupnost vektorů vah $\vec{w}^{(0)}, \vec{w}^{(1)}, \ldots$.

- váhy v $\vec{w}^{(0)}$ jsou inicializovány náhodně blízko 0
- v kroku $t+1$ (zde $t = 0, 1, 2, \ldots$) je $\vec{w}^{(t+1)}$ vypočteno takto:

$$w_{ji}^{(t+1)} = \vec{w}^{(t)} + \Delta w_{ji}^{(t)}$$

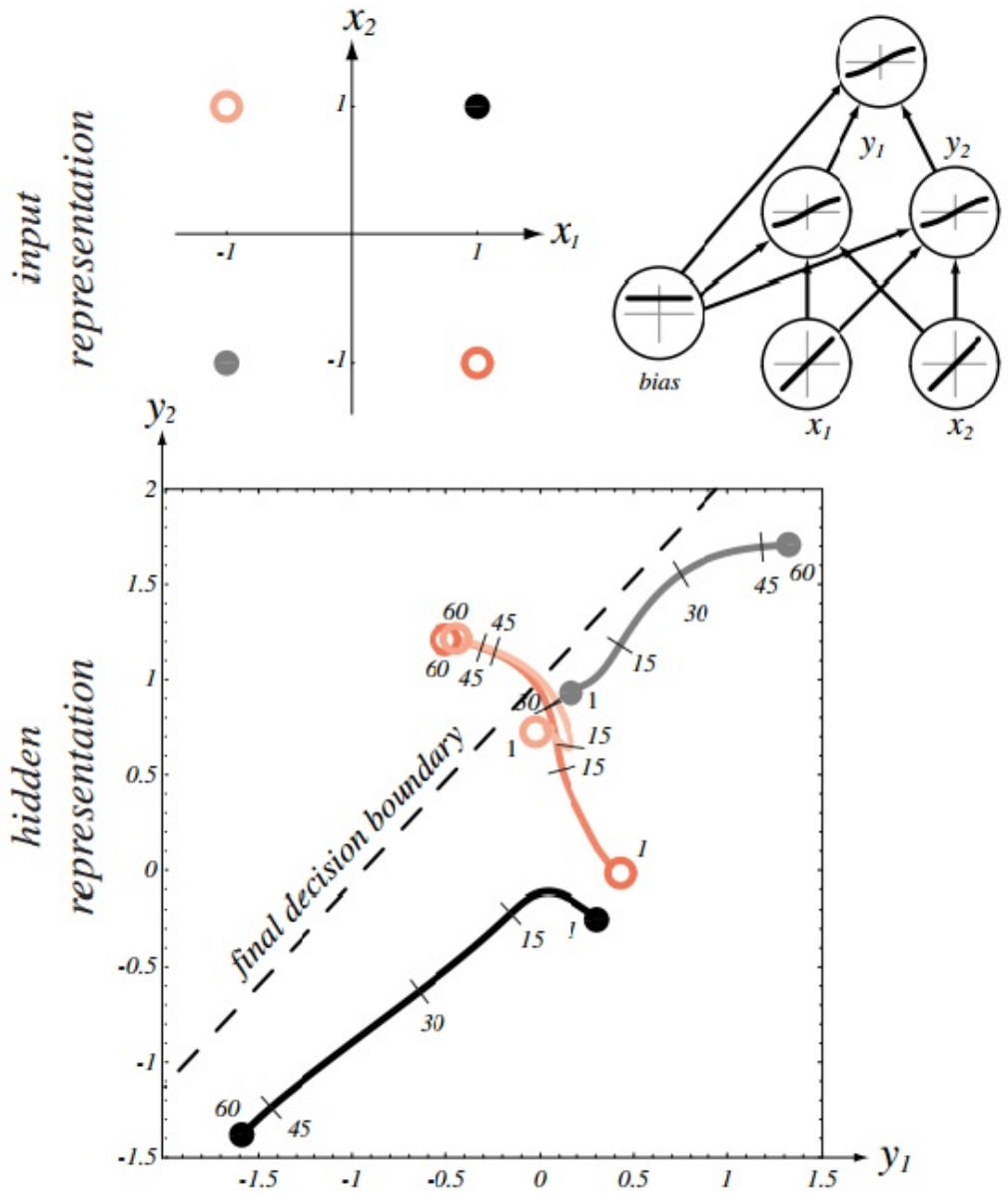
kde

$$\Delta w_{ji}^{(t)} = -\varepsilon(t) \cdot \frac{\partial E_k}{\partial w_{ji}}(w_{ji}^{(t)})$$

kde $k = (t \mod p) + 1$ a $0 < \varepsilon(t) \leq 1$ je rychlost učení v kroku $t+1$.

Lze použít i **stochastickou verzi** tohoto algoritmu, v níž je k voleno náhodně z $\{1, \ldots, p\}$.

Ilustrace gradientního sestupu - XOR



Zdroj: Pattern Classification (2nd Edition); Richard O. Duda, Peter E. Hart, David G. Stork

Simulace

Simulace

Praktické otázky gradientního sestupu

- Otázky týkající se efektivity učení:
  - Dávkový nebo online algoritmus?
  - Je nutné data předzpracovat?
  - Jak volit iniciální váhy?
  - Je nutné vhodně volit požadované hodnoty sítě?
  - Jak volit rychlost učení $\varepsilon(t)$?
  - Dává gradient nejlepší směr změny vah?
- Otázky týkající se výsledného modelu:
  - Kdy zastavit učení?
  - Kolik vrstev sítě a jak velkých?
  - Jak velkou tréninkovou množinu použít?
  - Jak zlepšit vlastnosti (přesnost a schopnost generalizace) výsledného modelu?

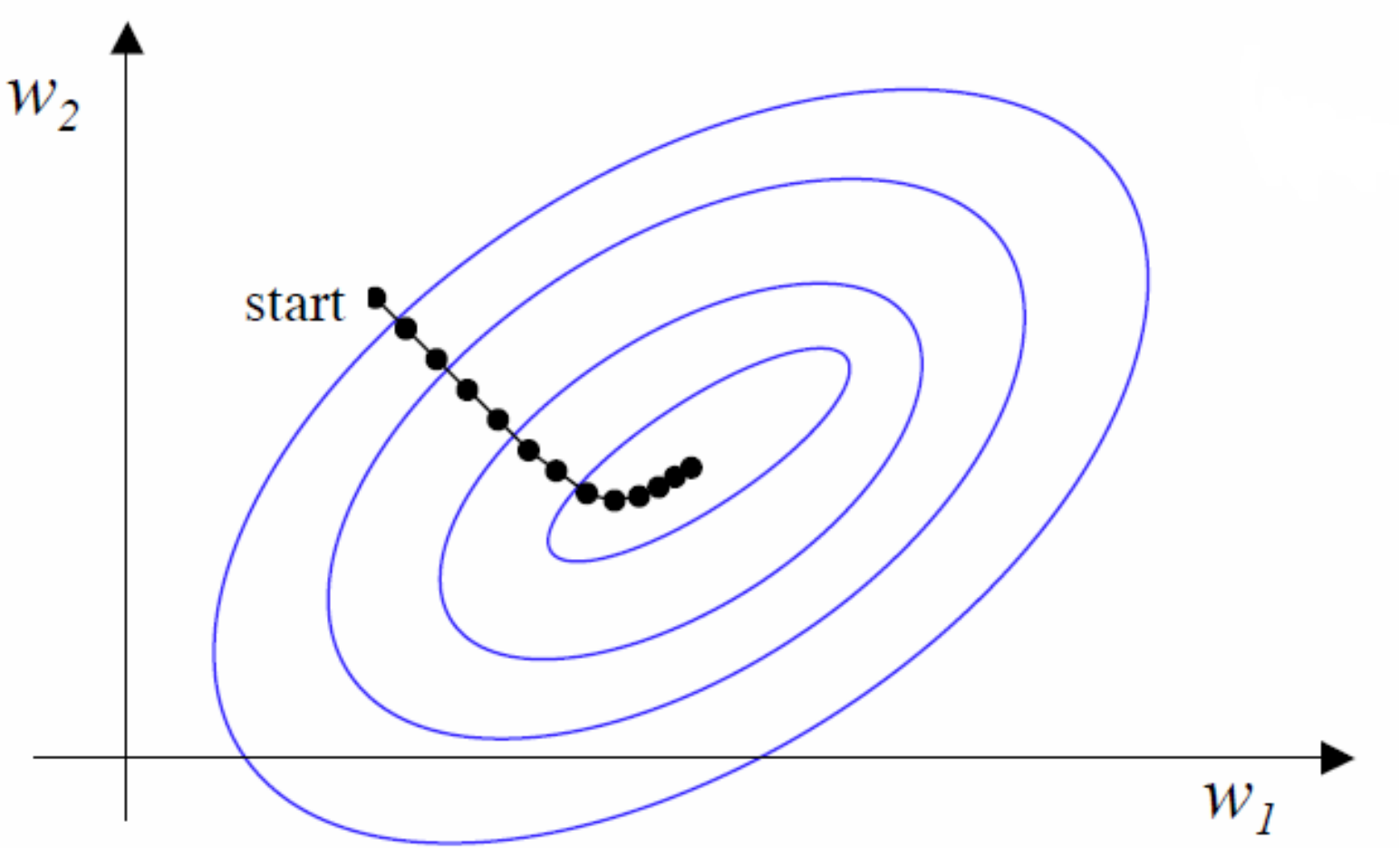
Poznámky o konvergenci gradientního sestupu

- Chybová funkce může být velmi komplikovaná:
  - může obsahovat mnoho lokálních minim, učící algoritmus v nich může skončit
  - může obsahovat mnoho plochých míst s velmi malým gradientem, zejména pokud se aktivační funkce tzv. saturují (tedy jejich hodnoty jsou blízko extrémům; v případě sigmoidálních funkcí to znamená, že mají velké argumenty)
  - může obsahovat velmi strmá místa, což vede k přeskočení minim gradientním sestupem
- pro velmi malou rychlost učení máme větší šanci dokonvergovat do lokálního minima, ale konvergence je hodně pomalá

  Teorie: pokud $\varepsilon(t) = \frac{1}{t}$ pak dávkový i stochastický gradientní sestup konvergují k lokálnímu minimu (nicméně extrémně pomalu).
- pro příliš velkou rychlost učení může vektor vah střídavě přeskakovat minimum nebo dokonce divergovat

Gradientní sestup - ilustrace

Gradientní sestup s malou rychlostí učení:

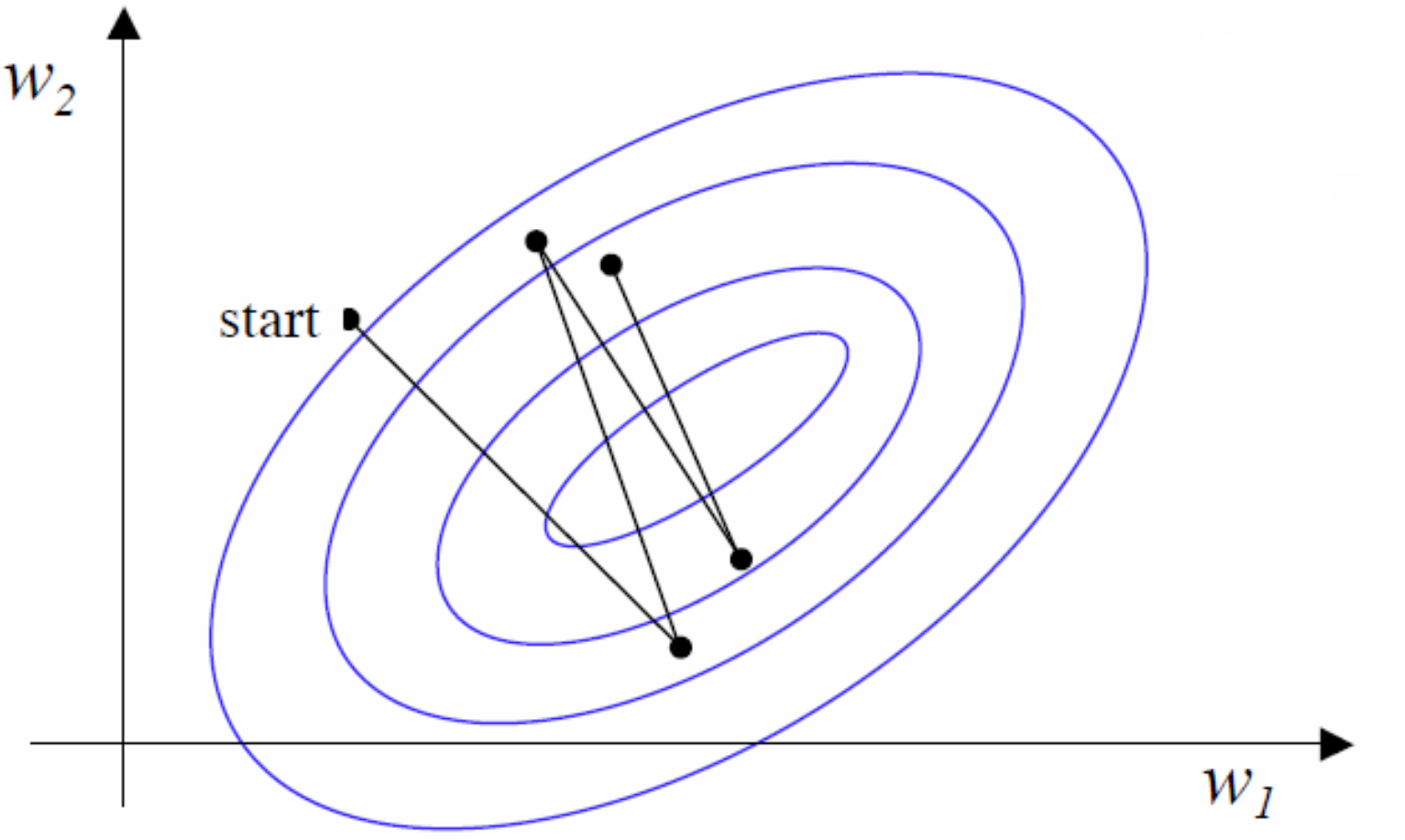


Více méně hladká křivka, každý krok je kolmý na vrstevnici.

Obrázek: Neural Computation, Dr John A. Bullinaria

Gradientní sestup - ilustrace

Gradientní sestup s příliš velkou rychlostí učení:

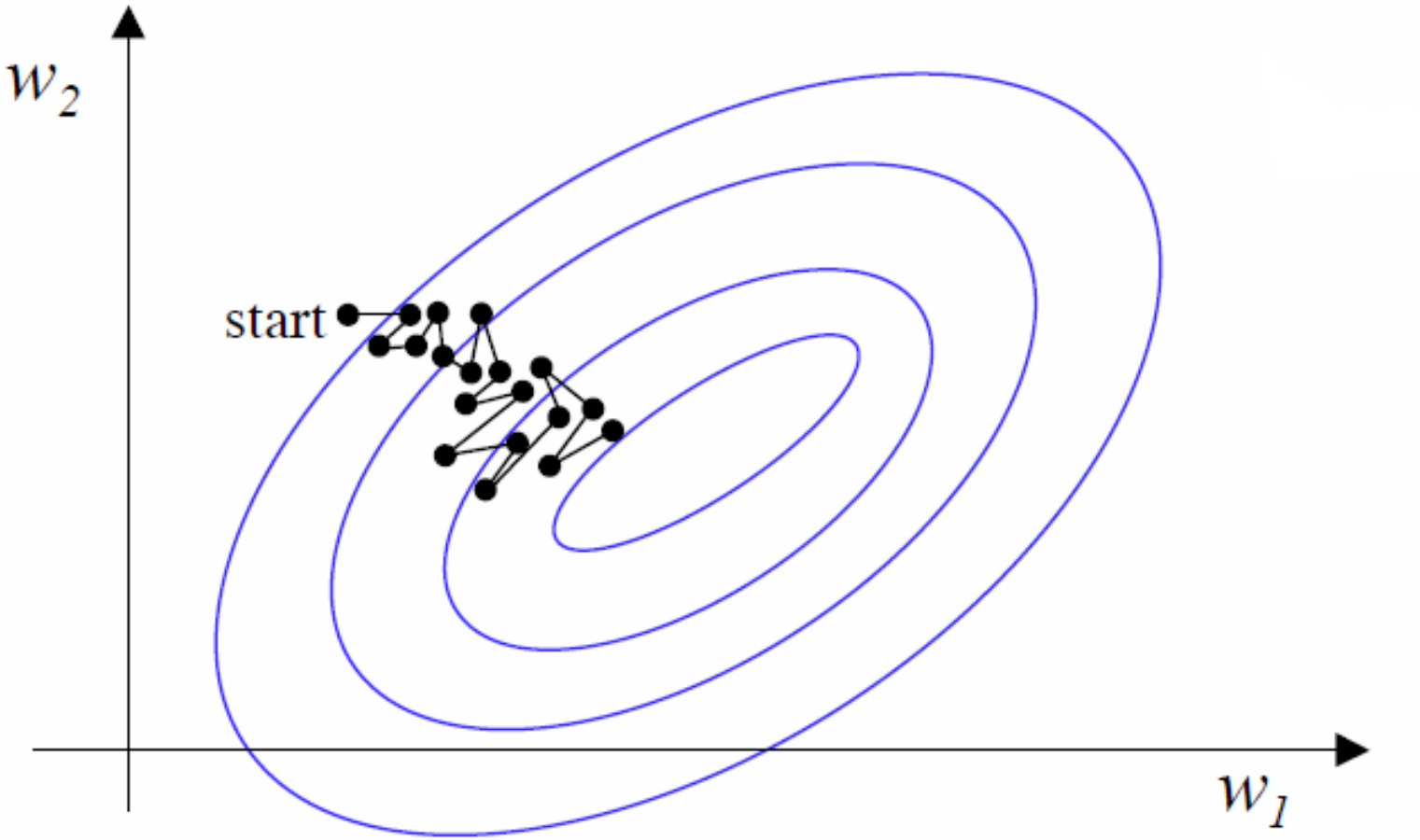


Jednotlivé kroky přeskakují minimum, učení je pomalé.

Obrázek: Neural Computation, Dr John A. Bullinaria

Gradientní sestup - ilustrace

Online učení:



Kroky nemusí být kolmé na vrstevnice, může hodně kličkovat.

Obrázek: Neural Computation, Dr John A. Bullinaria

Dávkový vs online učící algoritmus

Výhody dávkového:

- realizuje přesně gradientní sestup pro chybovou fci (většinou konverguje k lokálnímu minimu)
- snadno paralelizovatelný (vzory je možné zpracovávat odděleně)

Nevýhody dávkového:

- paměťová náročnost (musí si pamatovat chyby všech vzorů)
- redundantní data nepřidávají informaci ke gradientu.

Výhody online (stochastického)

- stochastický má šanci uniknout z okolí mělkého minima (protože nerealizuje přesný grad. sestup)
- má menší paměťovou náročnost a snadno se implementuje
- je rychlejší, zejména na hodně redundantních datech

Nevýhody online (stochastického)

- není vhodný pro paralelizaci
- může hodně kličkovat (více než gradientní sestup)

Moment

Problémy s gradientním sestupem:

- Může se stát, že gradient $\nabla E(\vec{w}^{(t)})$ neustále mění směr, ale chyba se postupně zmenšuje.

  Toto často nastává, pokud jsme v mělkém údolí, které se mírně svažuje jedním směrem (něco jako U-rampa pro snowboarding, učící algoritmus potom opisuje dráhu snowboardisty)
- Online algoritmus také může zbytečně kličkovat.

  Tento algoritmus se vůbec nemusí pohybovat ve směru největšího spádu!

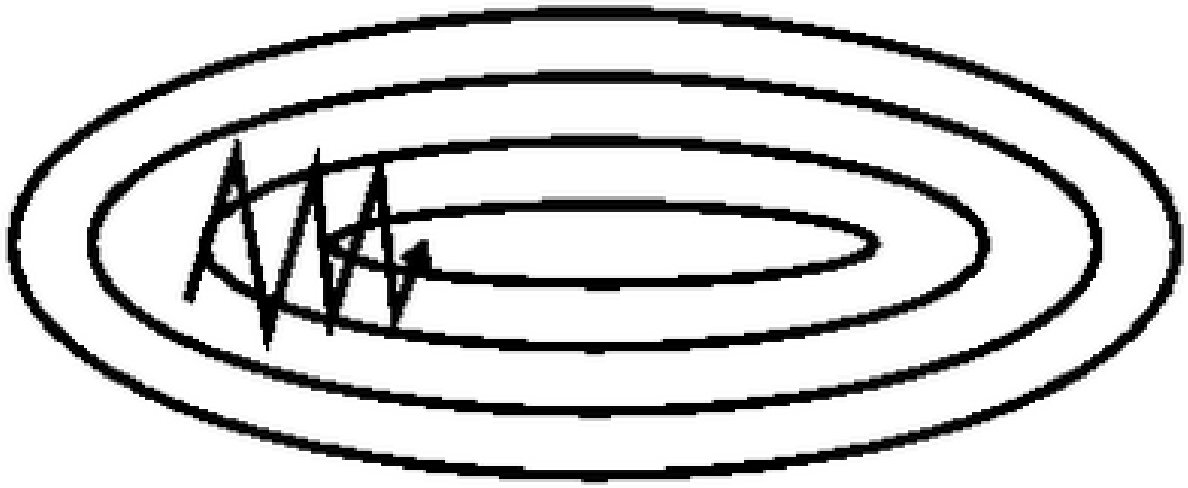
Řešení: Ke změně vah dané gradientním sestupem přičteme část změny v předchozím kroku, tedy definujeme

$$\Delta w_{ji}^{(t)} = \Delta \vec{w}^{(t)} + \alpha \cdot \Delta w_{ji}^{(t-1)}$$

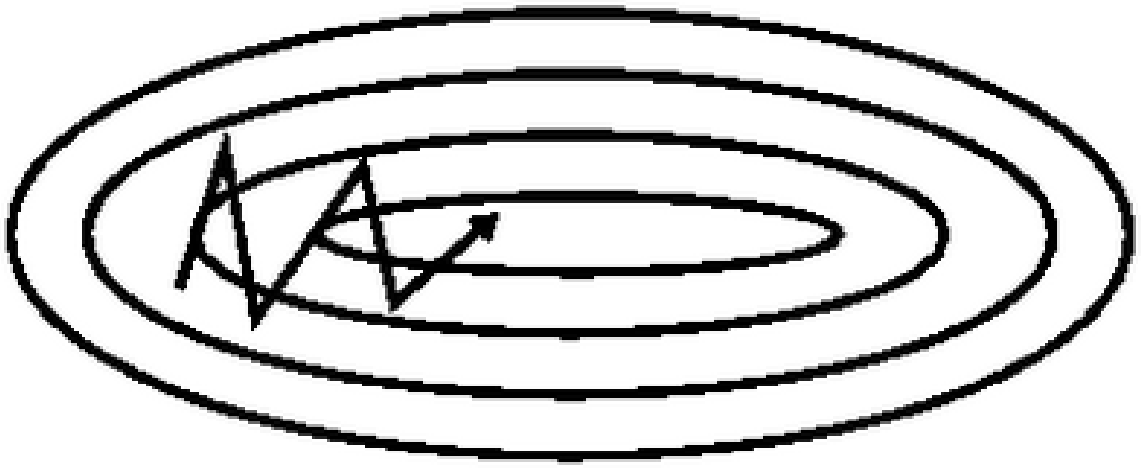
kde $0 < \alpha < 1$.

Moment - ilustrace

Bez momentu:



S momentem:



Volba aktivační funkce

Požadavky na vhodnou aktivační funkci $\sigma(\xi)$:

1. diferencovatelnost (jinak neuděláme gradientní sestup)
2. nelinearita (jinak by vícevrstvé sítě byly ekvivalentní jednovrstvým)
3. omezenost (váhy a potenciály budou také omezené $\Rightarrow$ konečnost učení)
4. monotónnost (lokální extrémy fce σ zanáší nové lokální extrémy do chybové funkce)
5. linearita pro malé ξ (umožní lineární model, pokud stačí k řešení úlohy)

Sigmoidální funkce splňují všechny požadavky.

Síť se sigmoidálními funkcemi obvykle reprezentuje data distribuovaně (tj. více neuronů vrací větší hodnotu pro daný vstup).

Později se seznámíme se sítěmi, jejichž aktivační funkce porušují některé požadavky (Gaussovy funkce) - používá se jiný typ učení.

Volba aktivační funkce

- Z hlediska rychlosti konvergence je výhodné volit lichou aktivační funkci.
- Standardní sigmoida není lichá, vhodnější je použít hyperbolický tangens:

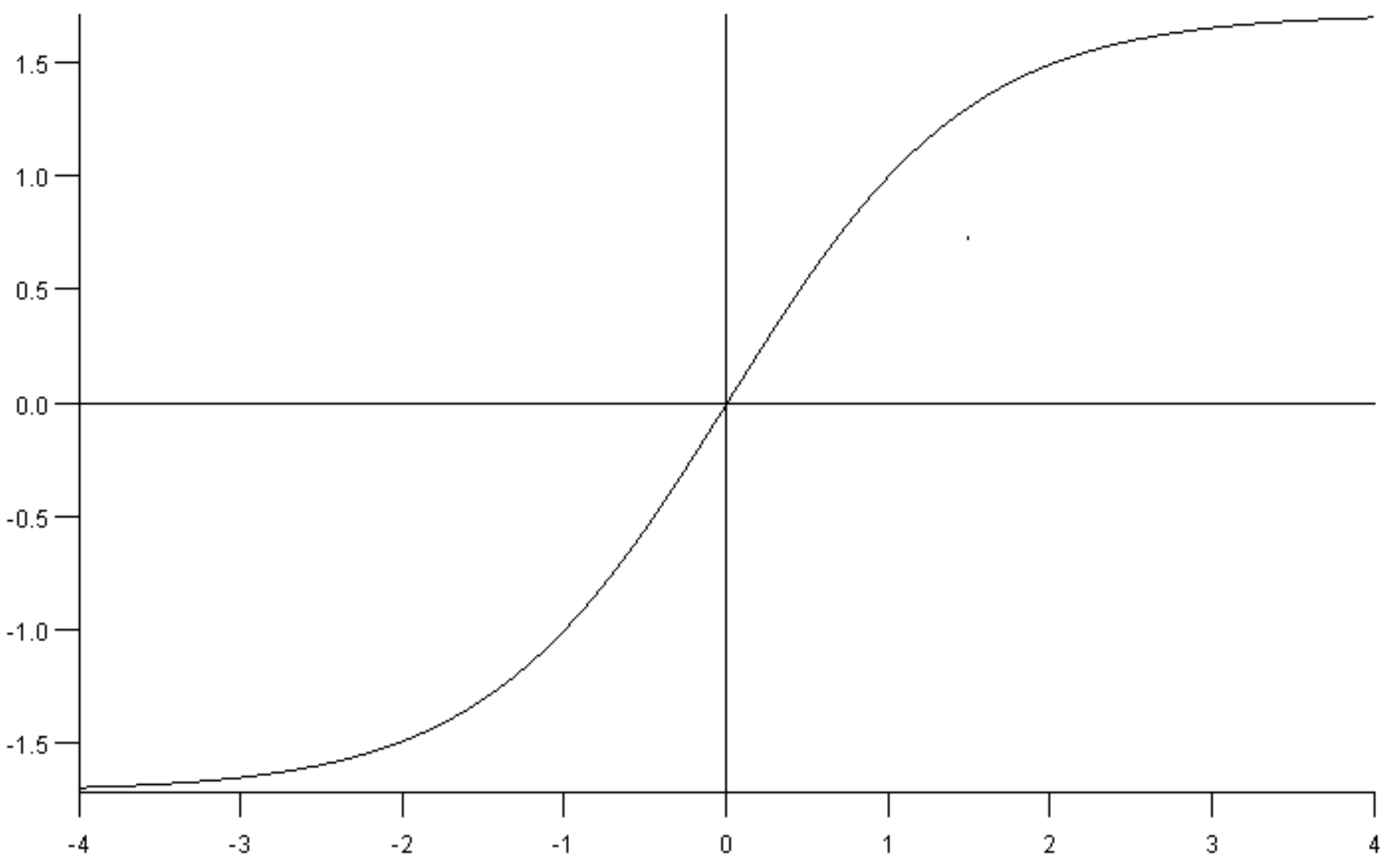
$$\sigma(\xi) = a \cdot \tanh(b \cdot \xi)$$

- Při optimalizaci lze předpokládat, že sigmoidální funkce jsou fixní a hýbat pouze dalšími paramtery.
- Z technických důvodů *budeme dále předpokládat, že všechny aktivační funkce jsou tvaru*

$$\sigma_j(\xi_j) = a \cdot \tanh(b \cdot \xi_j)$$

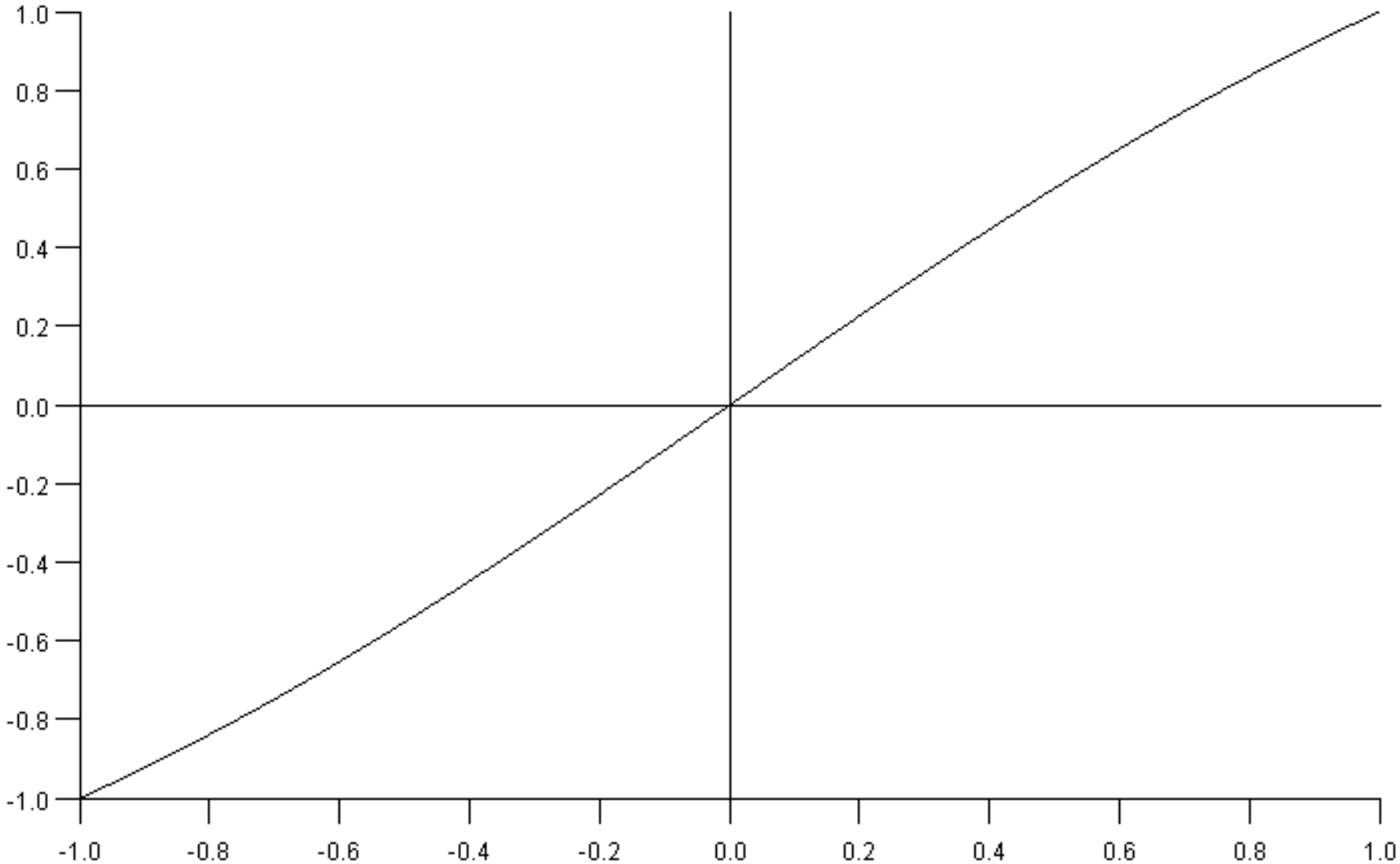
kde $a = 1.7159$ *a* $b = \frac{2}{3}$.

Volba aktivační funkce



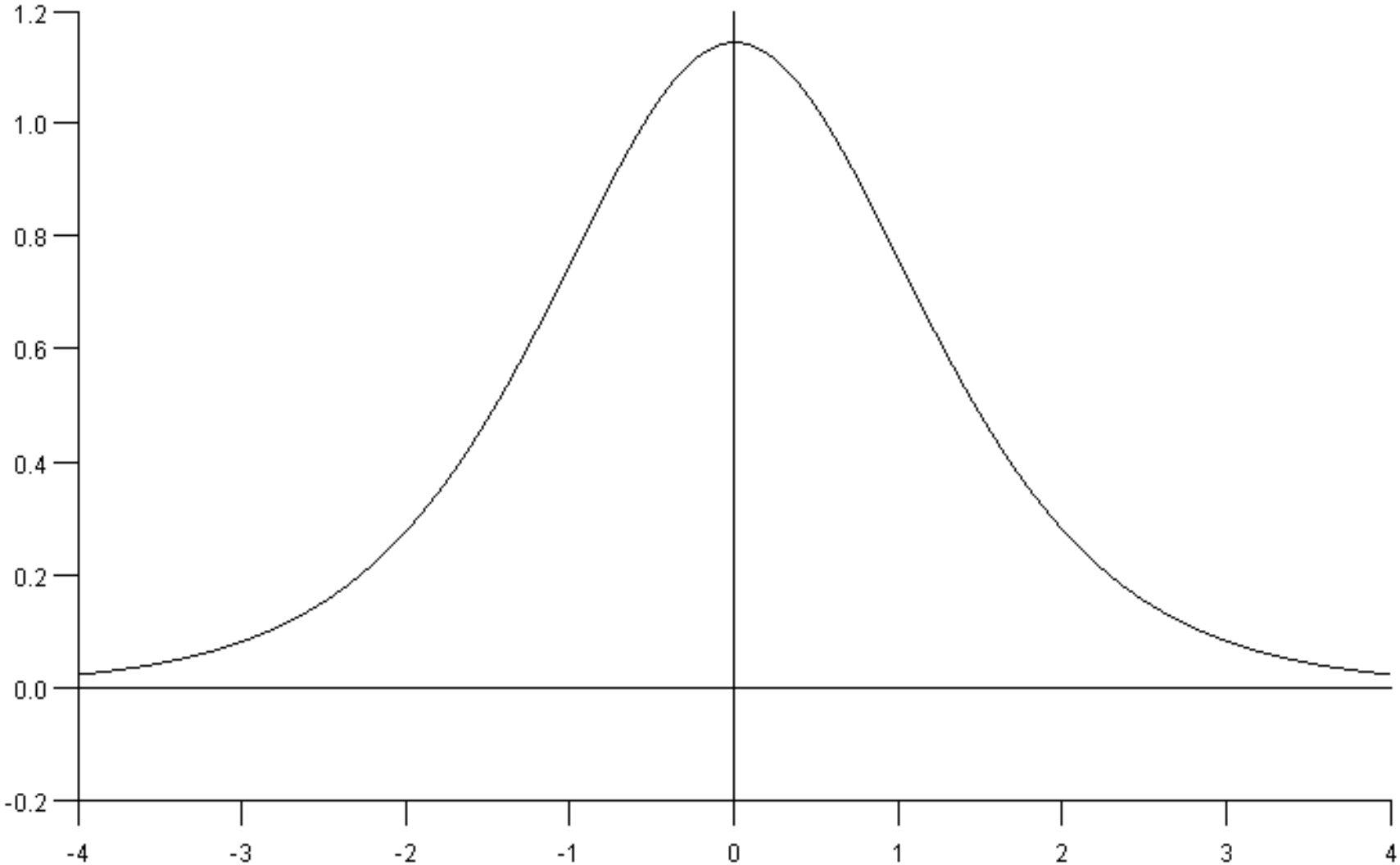
$\sigma(\xi) = 1.7159 \cdot \tanh(\frac{2}{3} \cdot \xi)$, platí $\lim_{\xi \to \infty} \sigma(\xi) = 1.7159$ a $\lim_{\xi \to -\infty} \sigma(\xi) = -1.7159$

Volba aktivační funkce



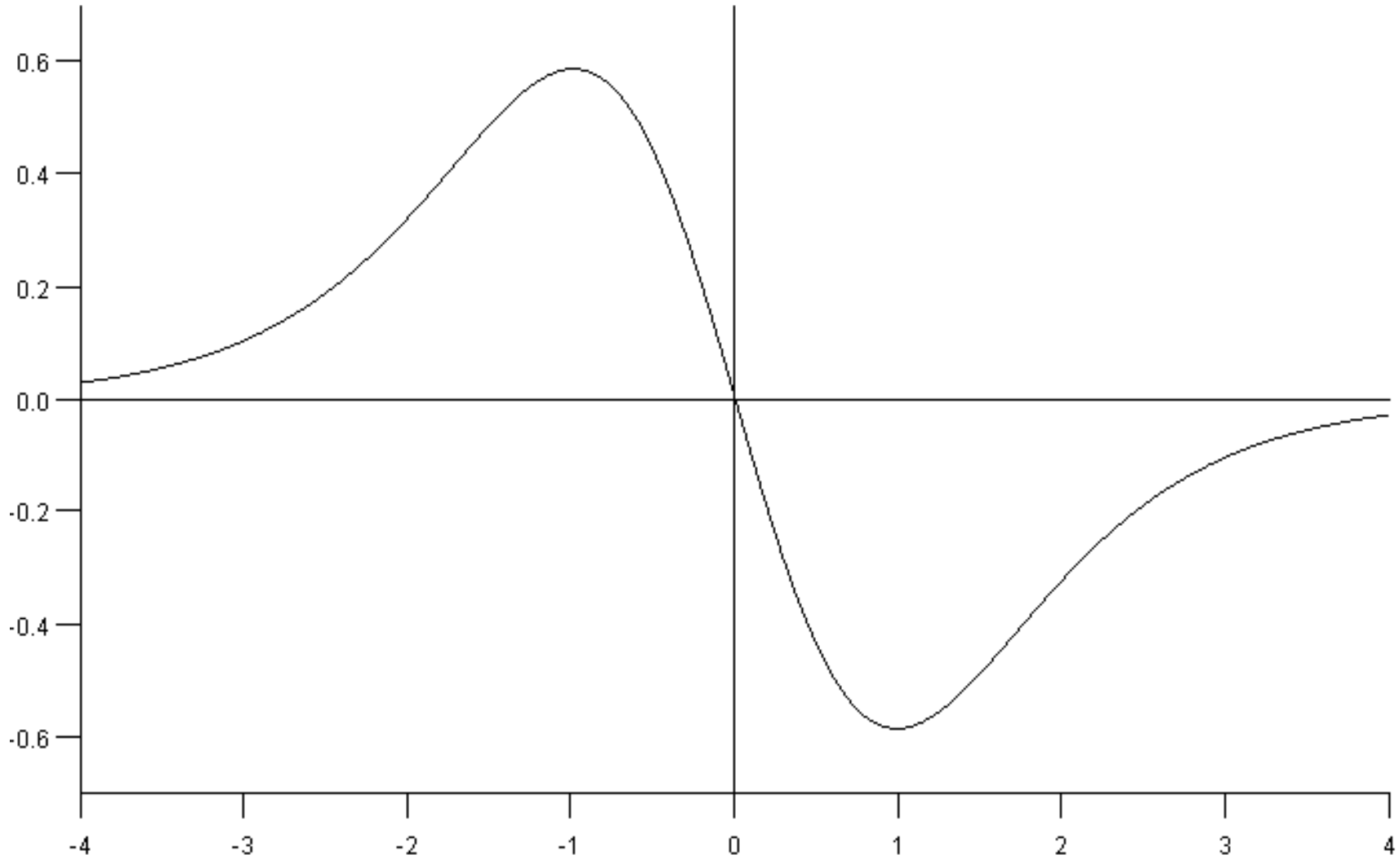
$\sigma(\xi) = 1.7159 \cdot \tanh(\frac{2}{3} \cdot \xi)$ je téměř lineární na $[-1, 1]$

Volba aktivační funkce



první derivace funkce $\sigma(\xi) = 1.7159 \cdot \tanh(\frac{2}{3} \cdot \xi)$

Volba aktivační funkce



druhá derivace funkce $\sigma(\xi) = 1.7159 \cdot \tanh(\frac{2}{3} \cdot \xi)$

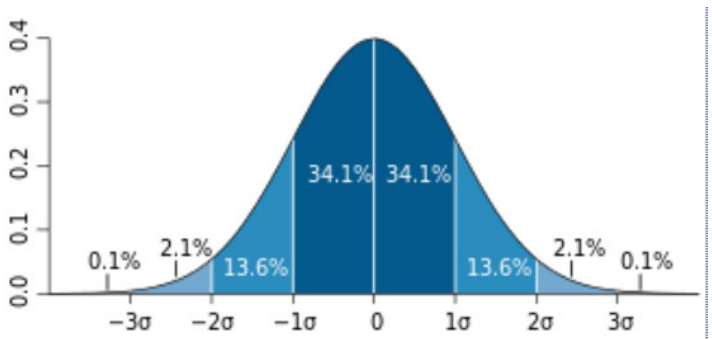
Předzpracování vstupů

- Některé vstupy mohou být mnohem větší než jiné

  Př.: Výška člověka vs délka chodidla (oboje v cm), maximální rychlost auta (v km/h) vs cena (v Kč), apod.

- Velké komponenty vstupů ovlivňují učení více, než ty malé. Navíc příliš velké vstupy mohou zpomalit učení.
- Numerická data se obvykle normalizují tak, že mají
  - průměrnou hodnotu $= 0$ (normalizace odečtením průměru)
  - rozptyl $= 1$ (normalizace podělením směrodatnou odchylkou)

  Zde průměr a směrodatná odchylka mohou být odhadnuty z náhodného vzorku (např. z tréninkové množiny).

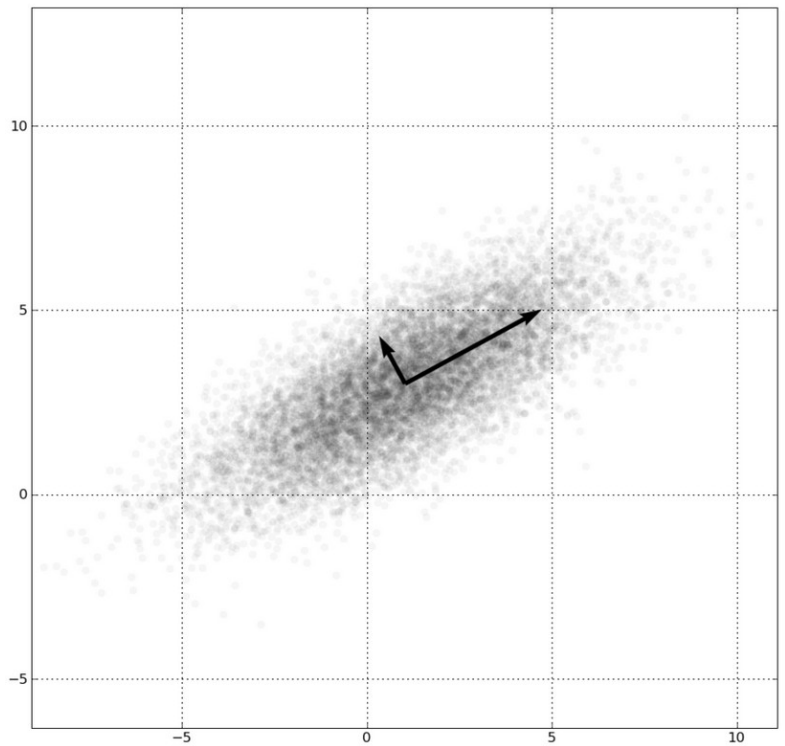


(Ilustrace směrodatné odchylky)

Předzpracování vstupů

- Jednotlivé komponenty vstupu by měly mít co nejmenší korelaci (tj. vzájemnou závislost).

  (Metody pro odstranění korelace dat jsou např. analýza hlavních komponent (Principal Component Analysis, PCA). Lze ji implementovat i pomocí neuronových sítí (probereme později)).



Iniciální volba vah

- Iniciálně jsou váhy nastaveny náhodně z daného intervalu $[-w, w]$ kde w závisí na počtu vstupů daného neuronu. Jaké je vhodné w?
- Uvažujeme aktivační funkci $1.7159 \cdot \tanh(\frac{2}{3} \cdot \xi)$ pro všechny neurony.
  - na intervalu $[-1, 1]$ se σ chová téměř lineárně,
  - extrémy σ'' jsou přibližně v bodech -1 a 1,
  - mimo interval $[-4, 4]$ je σ blízko extrémních hodnot.

  Tedy
  - Pro velmi malé w hrozí, že obdržíme téměř lineární model (to bychom mohli dostat s použitím jednovrstvé sítě). Navíc chybová funkce je velmi plochá v blízkosti $\vec{0}$ (malý gradient).
  - Pro w mnohem větší než 1 hrozí, že vnitřní potenciály budou vždy příliš velké a síť se nebude učit (gradient chyby E bude velmi malý, protože hodnota sítě se téměř nezmění se změnou vah).

  Chceme tedy zvolit w tak, aby vnitřní potenciály byly zhruba z intervalu $[-1, 1]$.

Iniciální volba vah

- Data mají po normalizaci (zhruba) nulovou střední hodnotu, rozptyl (zhruba) 1 a předpokládejme, že jednotlivé komponenty vstupů jsou (téměř) nekorelované.
- Uvažme neuron j z první vrstvy s d vstupy, předpokládejme, že jeho váhy jsou voleny uniformně z intervalu $[-w, w]$.
- **Pravidlo:** w je dobré volit tak, že *směrodatná odchylka* vnitřního potenciálu ξ_j (označme ji o_j) leží na hranici intervalu, na němž je aktivační funkce σ_j téměř lineární tj. v našem případě chceme $o_j \approx 1$.
- Z našich předpokladů plyne $o_j = \sqrt{\frac{d}{3}} \cdot w$.

  tj. v našem případě položíme $w = \frac{\sqrt{3}}{\sqrt{d}}$.
- Totéž funguje pro vyšší vrstvy, d potom odpovídá počtu neuronů ve vrstvě o jedna nižší.

  (Zde je důležité, že je aktivační funkce lichá)

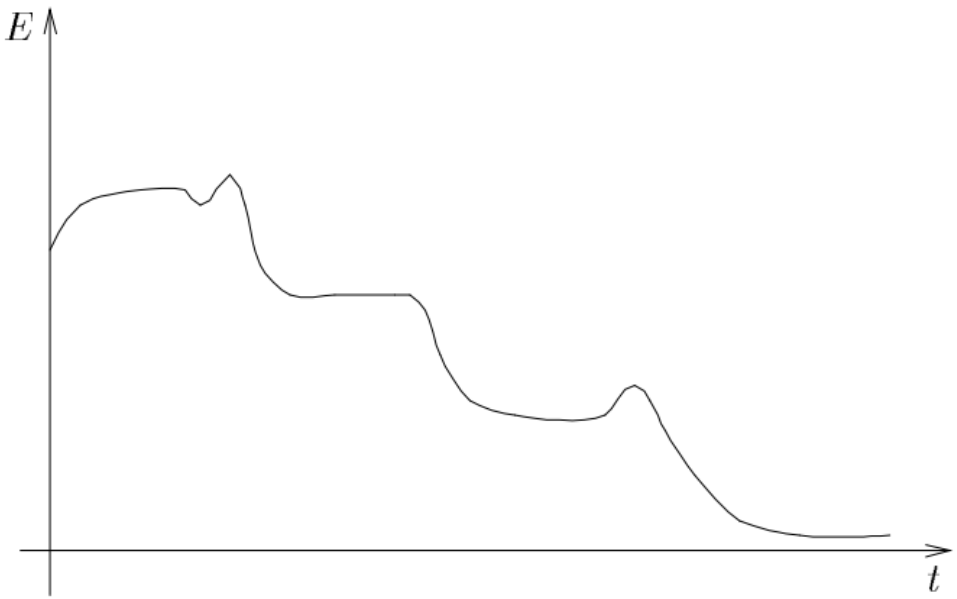
Požadované hodnoty

- Požadované hodnoty d_{kj} by měly být voleny v oboru hodnot aktivačních funkcí, což je v našem případě $[-1.716, 1.716]$.
- Požadované hodnoty příliš blízko extrémům ± 1.716 způsobí, že váhy neomezeně porostou, vnitřní neurony budou mít velké potenciály, gradient chybové funkce bude malý a učení se zpomalí.
- Proto je vhodné volit požadované hodnoty z intervalu $[-1.716 + \delta, 1.716 - \delta]$. Optimální je, když tento interval odpovídá maximálnímu intervalu na němž jsou aktivační fce lineární. Tedy v našem případě $\delta \approx 0.716$, tj. hodnoty d_{kj} je dobré brát z intervalu $[-1, 1]$.

Rychlost učení

Obecné zásady pro volbu a změny rychlosti učení ε

- Je dobré začít s malou rychlostí (např. $\varepsilon = 0.01$).
  (existují i metody, které pracují s velkou počáteční rychlostí a poté ji postupně snižují)
- Pokud se chyba znatelně zmenšuje (učení konverguje), můžeme rychlost nepatrně zvýšit.
- Pokud se chyba zjevně zvětšuje (učení diverguje), můžeme rychlost snížit.
- Krátkodobé zvýšení chyby nemusí nutně znamenat divergenci.



Obr. 2.3: Typický vývoj chyby v čase při učení pomocí backpropagation.

Rychlost učení

Chceme, aby se neurony učily pokud možno stejně rychle. Více vstupů obvykle znamená rychlejší učení.

Pro každou váhu w_{ji} můžeme mít zvláštní rychlost učení ε_{ji}

- ε_{ji} lze iniciovat např. $1/|j_{\leftarrow}|$, kde $|j_{\leftarrow}|$ je počet vstupů neuronu j.
- Po zahájení učení rychlost zvolna zvyšujeme

  (třeba $\varepsilon_{ji}(t) = K^+ \cdot \varepsilon_{ji}(t-1)$ kde $K^+ > 1$)
- Jakmile se změní znaménko $\Delta w_{ji}^{(t)}$ (tedy $\Delta w_{ji}^{(t-1)} \cdot \Delta w_{ji}^{(t)} < 0$), rychlost snížíme

  (třeba takto $\varepsilon_{ji}(t) = K^- \cdot \varepsilon_{ji}(t-1)$ kde $K^- < 1$)

Algoritmus **Rprop** bere v potaz pouze směr gradientu, velikost kroku se mění výše uvedeným násobením fixními konstantami K^+ a K^-.

(Více na `https://en.wikipedia.org/wiki/Rprop`)

Rychlost a směr - přesněji

Na gradientní sestup se můžeme dívat obecněji takto:

$$\Delta\vec{w}^{(t)} = r(t) \cdot s(t)$$

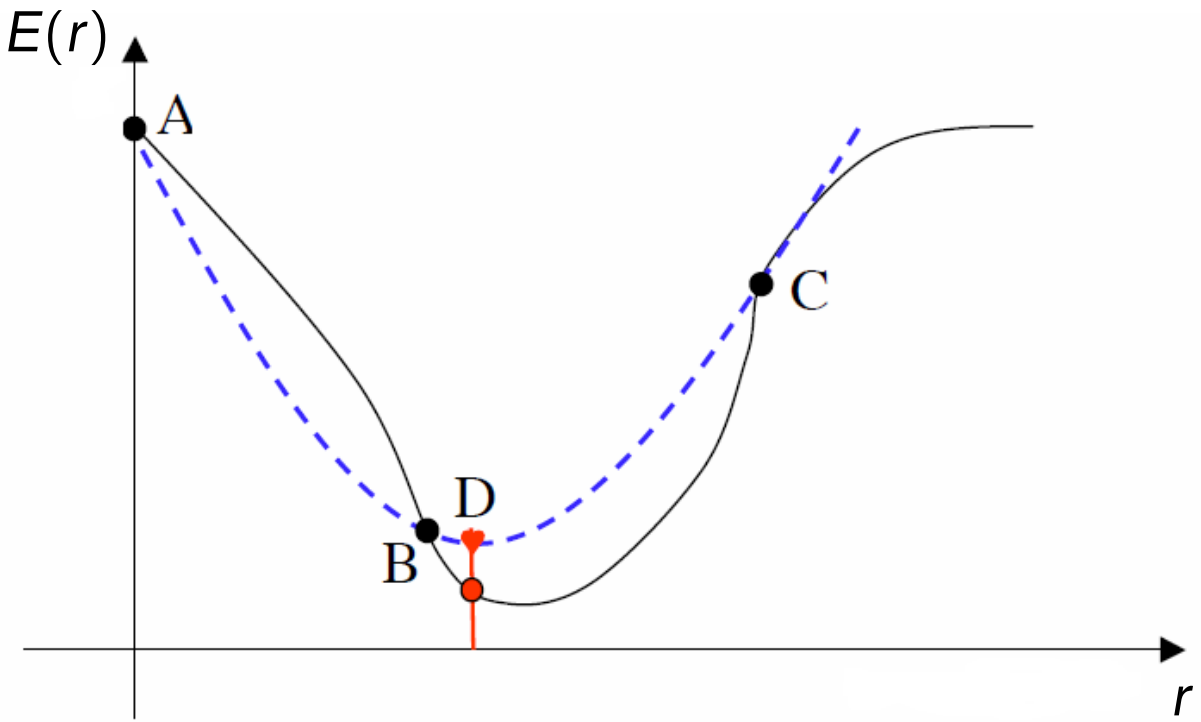
kde $r(t)$ je rychlost změny vah a $s(t)$ je směr změny vah.
V našem případě: $r(t) = \varepsilon(t)$ a $s(t) = -\nabla E(\vec{w}^{(t)})$
(nebo $s(t) = -\nabla E_k(\vec{w}^{(t)})$ pro online učení).

- Ideální by bylo volit $r(t)$ tak, abychom minimalizovali $E(\vec{w}^{(t)} + r(t) \cdot s(t))$.

  Nebo se aspoň chceme přesunout (ve směru $s(t)$) do místa, v němž začne gradient opět růst.
- Toho lze (přibližně) dosáhnout malými přesuny bez změny směru - nedostaneme ovšem o mnoho lepší algoritmus než standardní grad. sestup (který stále mění směr).
- Existují lepší metody, např. **parabolická interpolace** chybové funkce.

Rychlost a směr - parabolická interpolace

Označme $E(r) = E(\vec{w}^{(t)} + r \cdot s(t))$; minimalizujeme $E(r)$.

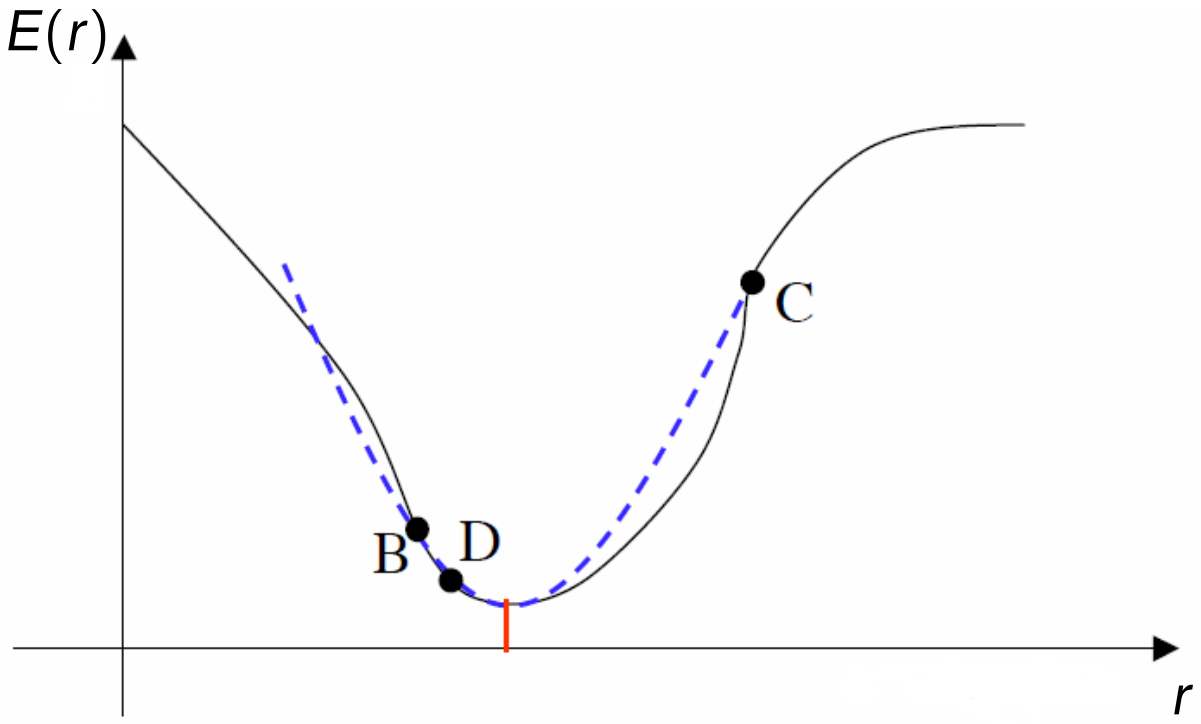


Předpokládejme, že jsme schopni nalézt body A, B, C takové, že $E(A) > E(B)$ a $E(C) > E(B)$. Pak lze tyto body proložit parabolou a nalézt její minimum D. Toto D je dobrým odhadem minima $E(r)$.

Obrázek: Neural Computation, Dr John A. Bullinaria

Rychlost a směr - parabolická interpolace

Parabolickou interpolaci lze dále iterovat, čímž dosáhneme ještě lepšího odhadu:



Je jasné, že $E(B) \geq E(D)$ a $E(C) > E(D)$. Pokud $E(B) > E(D)$, lze použít stejný postup jako předtím
(jinak je nutné nalézt nový bod B' t. ž. $E(B') > E(D)$).

Optimální směr

Zbývá otázka, jestli je záporný gradient správným směrem.

Rychlost $r(t)$ jsme volili tak, abychom minimalizovali

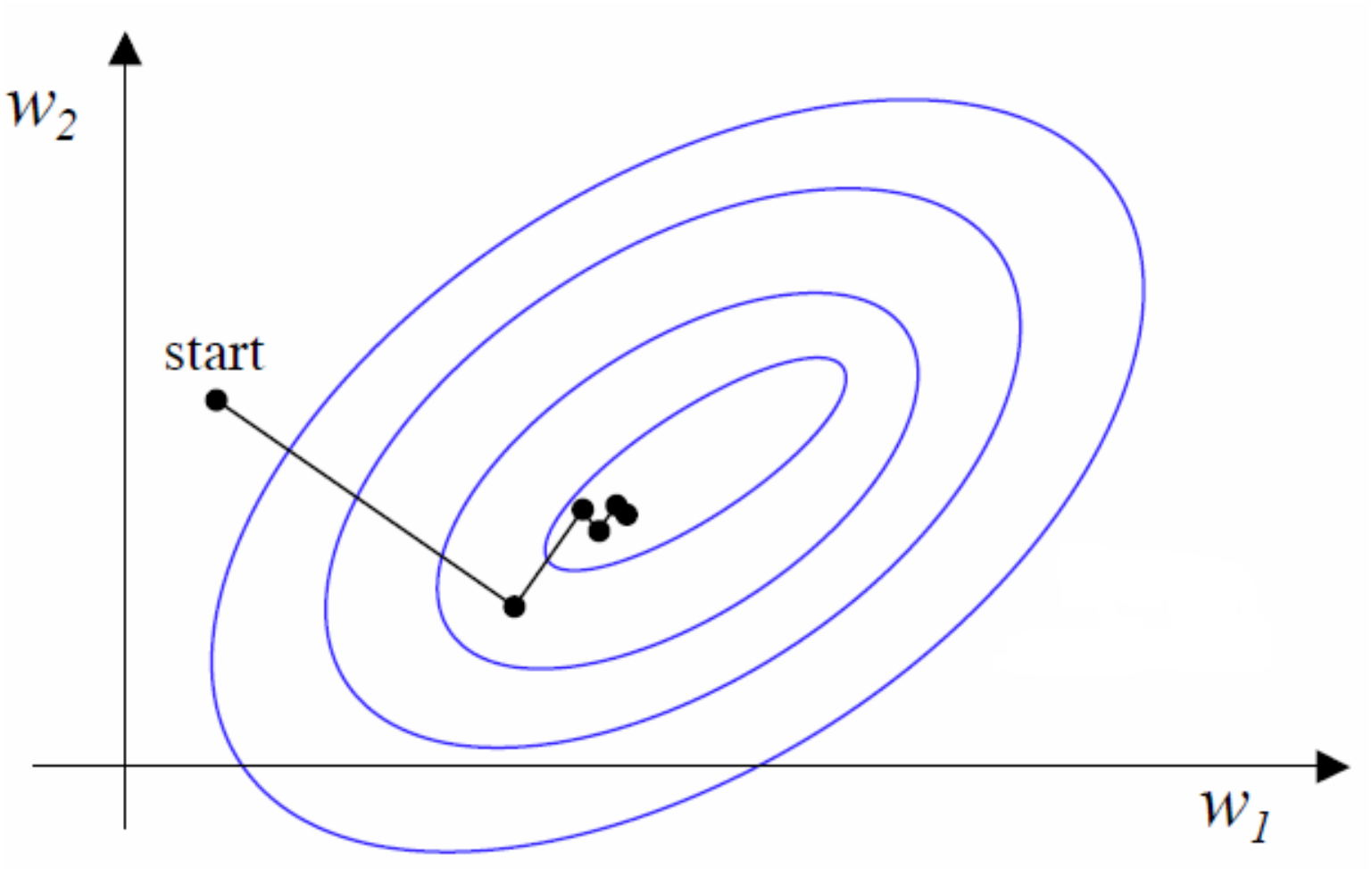
$$\begin{aligned} E(\vec{w}^{(t+1)}) &= E\left(\vec{w}^{(t)} + r(t) \cdot s(t)\right) \\ &= E\left(\vec{w}^{(t)} + r(t) \cdot (-\nabla E(\vec{w}^{(t)})\right) \end{aligned}$$

To ovšem znamená, že derivace $E(\vec{w}^{(t+1)})$ podle $r(t)$ (zde bereme $r(t)$ jako nezávislou proměnnou) bude 0, tedy

$$\begin{aligned} \frac{\delta E}{\delta r}(\vec{w}^{(t+1)}) &= \sum_{j,i} \frac{\delta E}{\delta w_{ji}}(\vec{w}^{(t+1)}) \cdot \left(-\frac{\delta E}{\delta w_{ji}}(\vec{w}^{(t)})\right) \\ &= \nabla E(\vec{w}^{(t+1)}) \cdot (-\nabla E(\vec{w}^{(t)})) \\ &= 0 \end{aligned}$$

Tj. nový a starý směr jsou vzájemně kolmé, výpočet tedy kličkuje.

Gradientní sestup s optimální rychlostí



Obrázek: Neural Computation, Dr John A. Bullinaria

Rychlost a směr - přesněji

Řešení: Do nového směru zahrneme částečně i předchozí směr a tím zmenšíme kličkování.

$$s(t) = -\nabla E(\vec{w}^{(t)}) + \beta \cdot s(t-1)$$

Jak určit β ? Metoda **sdružených gradientů** je založena na tom, že nový směr by měl co nejméně kazit minimalizaci dosaženou v předchozím směru. Chceme nalézt nový směr $s(t)$ takový, že gradient funkce funkce E v novém bodě $\vec{w}^{(t+1)} = \vec{w}^{(t)} + r(t) \cdot s(t)$ ve starém směru $s(t-1)$ je nulový:

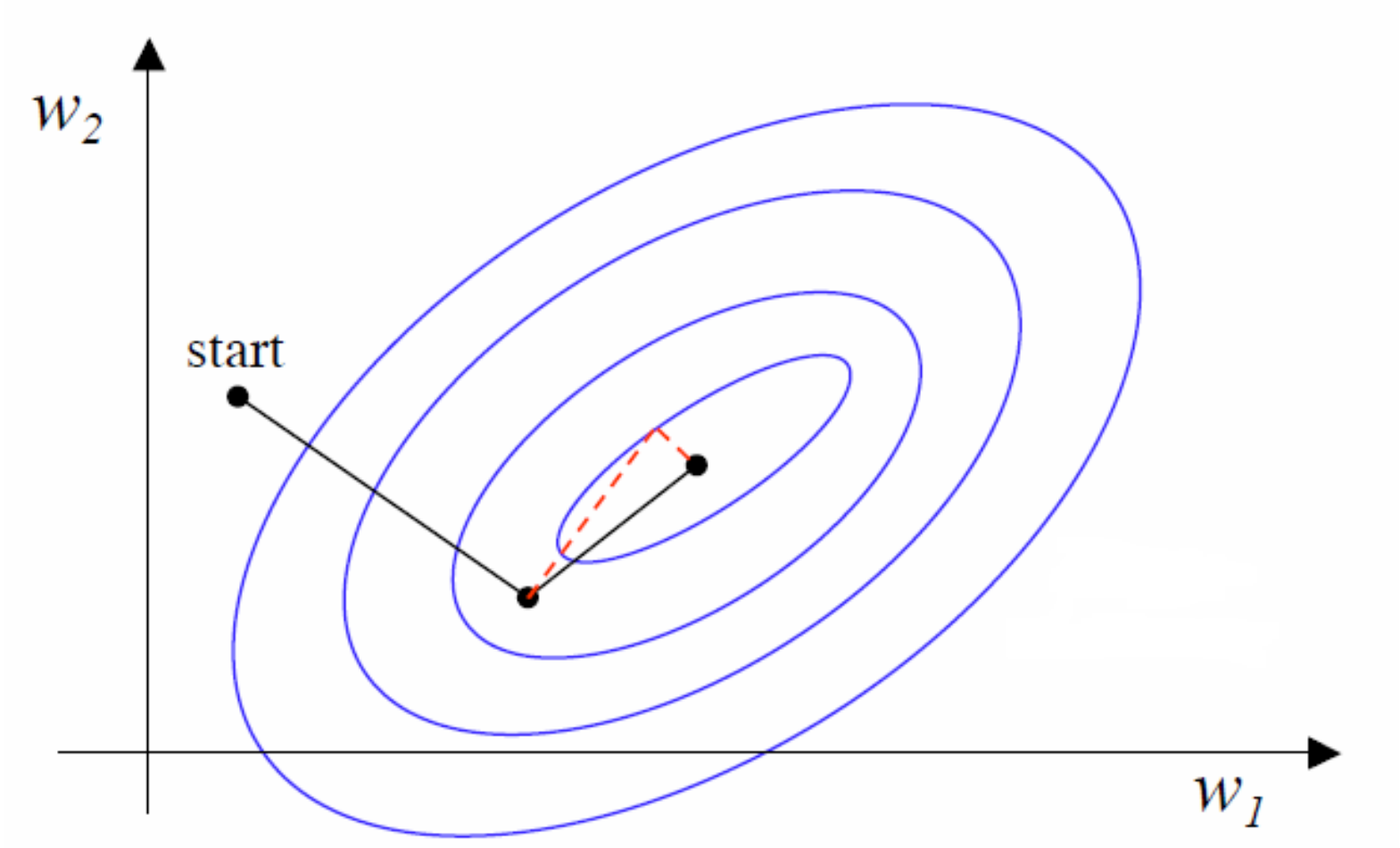
$$s(t-1) \cdot \nabla E(\vec{w}^{(t+1)}) = 0$$

Vhodné β, které to splňuje je dáno následujícím pravidlem (Polak-Ribiere):

$$\beta = \frac{(\nabla E(\vec{w}^{(t+1)}) - \nabla E(\vec{w}^{(t)})) \cdot \nabla E(\vec{w}^{(t+1)})}{\nabla E(\vec{w}^{(t)}) \cdot \nabla E(\vec{w}^{(t)})}$$

(Pokud by E byla kvadratická funkce, pak to konverguje v nejvýše n krocích)

Gradientní sestup s optimální rychlostí



Obrázek: Neural Computation, Dr John A. Bullinaria

Další metody

Existuje mnoho metod druhého řádu, které jsou přesnější, ale obvykle výpočetně mnohem náročnější (příliš se nepoužívají). Např. Newtonova metoda, Levenberg-Marquardt, atd.

Většina těchto metod vyžaduje výpočet (nebo aspoň aproximaci) druhé derivace chybové funkce nebo funkce sítě (tj. Hessián).

Lze nalézt v literatuře, např.
Haykin, Neural Neworks and Learning Machines

Schopnost generalizace

V klasifikačních problémech se dá generalizace popsat jako schopnost vyrovnat se s novými vzory.

Pokud síť trénujeme na náhodně vybraných datech, není ideální přesně klasifikovat tréninkové vzory.

Pokud aproximujeme funkční závislost vstupů a očekávaných výstupů pak obvykle nechceme aby funkce sítě vracela přesné hodnoty pro tréninkové vzory.

Exaktněji: Obvykle se předpokládá, že tréninková množina byla generována takto:

$$d_{kj} = g_j(\vec{x}_k) + \Theta_{kj}$$

kde g_j je správná funkce výstupního neuronu $j \in Y$ a Θ_{kj} je náhodný šum. Chceme, aby síť pokud možno realizovala funkce g_j.

Kdy zastavit učení?

Standardní kritérium: Chyba E je dostatečně malá.

Další možnost: po několik iterací je gradient chyby malý.

(Výhodou tohoto kritéria je, že se nemusí počítat chyba E)

Problém: Příliš dlouhé učení způsobí, že síť opisuje tréninkové vzory (je přetrénovaná). Důsledkem je špatná generalizace.

Řešení: Množinu vzorů rozdělíme do následujících množin

- **tréninková** (např. 60%) - podle těchto vzorů se síť učí
- **validační** (např. 20%) - používá se k zastavení učení.
- **testovací** (např. 20%) - používá se po skončení učení k testování přesnosti sítě, tedy srovnání několika natrénovaných sítí.

Trénink, testování, validace

Obvykle se realizuje několik iterací (cca 5) tréninku na tréninkové množině. Poté se vyhodnotí chyba E na validační množině.

Ideálně chceme zastavit v minimu chyby na validační množině. V praxi se sleduje, zda chyba na validační množině klesá. Jakmile začne růst (nebo roste po několik iterací za sebou), učení zastavíme.

Problém: Co když máme příliš málo vzorů?

Můžeme tréninkovou množinu rozdělit na K skupin $S_1, \ldots, S_K$. Trénujeme v K fázích, v ℓ-té fázi provedeme následující:

- trénujeme na $S_1 \cup \ldots \cup S_{\ell-1} \cup S_{\ell+1} \cup \ldots \cup S_K$
- spočteme chybu e_ℓ funkce E na skupině S_ℓ

Celková chyba je potom průměr $e = \frac{1}{K} \sum_{\ell=1}^{K} e_\ell$.

Extrémní verze: $K =$ počet vzorů (používá se při extrémně málo vzorech)

Typicky se volí $K = 10$.

Velikost sítě

Podobný problém jako v případě délky učení:

- Příliš malá síť není schopna dostatečně zachytit tréninkovou množinu a bude mít stále velkou chybu
- Příliš velká síť má tendenci přesně opsat tréninkové vzory - špatná generalizace

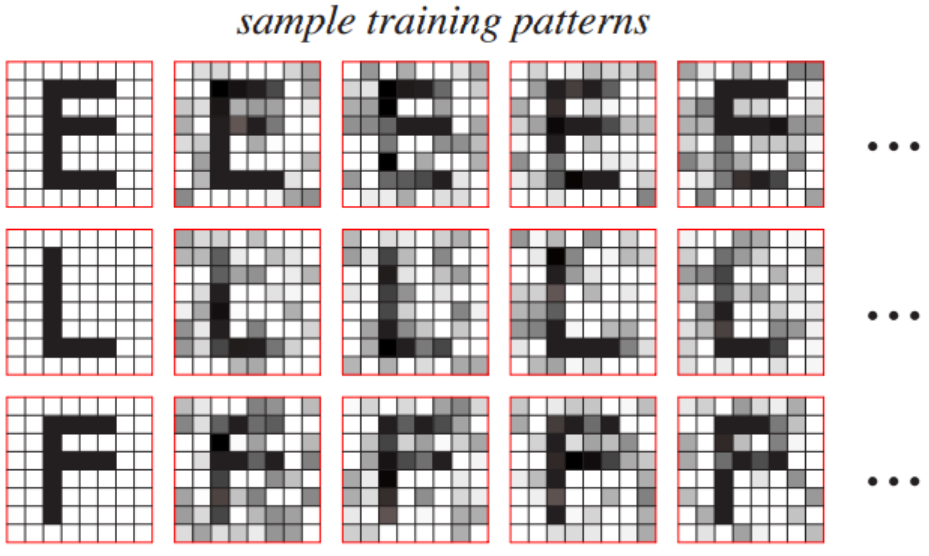
Řešení: Optimální počet neuronů :-)

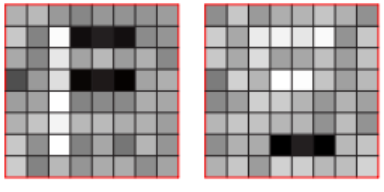
- teoretické výsledky existují, ale jsou většinou nepoužitelné
- existují učící algoritmy, které postupně přidávají neurony (konstruktivní algoritmy) nebo odstraňují spoje a neurony (prořezávání)
- V praxi se počet vrstev a neuronů stanovuje experimentálně (prostě se to zkusí, uvidí, opraví) a/nebo na základě zkušeností.

Velikost sítě

Uvažme dvouvrstvou síť. Neurony ve vnitřní vrtvě často reprezentují "vzory" ve vstupní množině.

Př.: Síť 64-2-3 pro klasifikaci písmen:





learned input-to-hidden weights

Vynechávání neuronů během učení (dropout)

Přetrénování může souviset s přílišnou "závislostí" jednotlivých neuronů na chování ostatních neuronů.

Pokud má síť mnoho neuronů, je schopna zachytit složité závislosti v datech mnoha různými způsoby, přičemž téměř všechny budou špatné na validační množině.

Tomu lze předcházet použitím následující metody: V každém kroku gradientního sestupu měníme váhy každého jednotlivého neuronu pouze s pravděpodobností $1/2$ (v praxi lze i $< 1/2$).

Jinými slovy: Každý neuron je pro daný krok vyhozen ze sítě (neučí se) s pravděpodobností $1/2$.

Tato metoda by měla předcházet složitým "společným" adaptacím více neuronů.

Lze ji také vidět jako metodu pro trénink skupiny mnoha neuronových sítí vytvořených vyhazováním neuronů.

Velikost tréninkové množiny

Příliš malá nebo nereprezentativní tréninková množina vede ke špatné generalizaci

Příliš velká zvyšuje složitost učení. V případě dávkového algoritmu způsobuje velkou chybu (chyby na vzorech se sčítají), což může vést k přetrénování.

Pravidlo pro klasifikační úlohy: počet vzorů by měl zhruba odpovídat W/δ kde

- W je počet vah v síti
- $0 < \delta < 1$ je tolerovaná chyba na testovací množině
  (tj. tolerujeme δ chybně klasifikovaných vzorů z testovací množiny)

Regularizace - upadání vah (weight decay)

Generalizaci lze zlepšit tak, že v síti necháme jen nutné neurony a spoje. Možná heuristika spočívá v odstranění malých vah. Penalizací velkých vah dostaneme silnější indikaci důležitosti vah.

V každém kroku učení zmenšíme uměle všechny váhy

$$w_{ji}^{(t+1)} = (1 - \zeta)(w_{ji}^{(t)} + \Delta w_{ji}^{(t)})$$

Idea: Nedůležité váhy budou velmi rychle klesat k 0 (potom je můžeme vyhodit). Důležité váhy dokážou přetlačit klesání a zůstanou dostatečně velké.

Toto je ekvivalentní gradientnímu sestupu s konstantní rychlostí učení ε, pokud chybovou funkci modifikujeme takto:

$$E'(\vec{w}) = E(\vec{w}) + \frac{2\zeta}{\varepsilon}(\vec{w} \cdot \vec{w})$$

Zde **regularizační člen** $\frac{2\zeta}{\varepsilon}(\vec{w} \cdot \vec{w})$ penalizuje velké váhy.

Praxe - Matice zmatenosti

Confusion matrix

Síť má za úkol klasifikovat objekty do K tříd $T_1, \ldots, T_K$. Confusion matrix je tabulka, jejíž pole v i-tém řádku a j-tém sloupci obsahuje počet objektů z třídy T_i, které byly klasifikovány jako objekty z třídy T_j.

Example confusion matrix

		Predicted		
		Cat	**Dog**	**Rabbit**
Actual	**Cat**	5	3	0
	Dog	2	3	1
	Rabbit	0	2	11

Zdroj: http://en.wikipedia.org/wiki/Confusion_matrix

Asociativní sítě

Cílem je uchovat množinu vzorů $\{(\vec{x}_k, \vec{d}_k) \mid k = 1, \ldots, p\}$ tak, aby platilo následující:

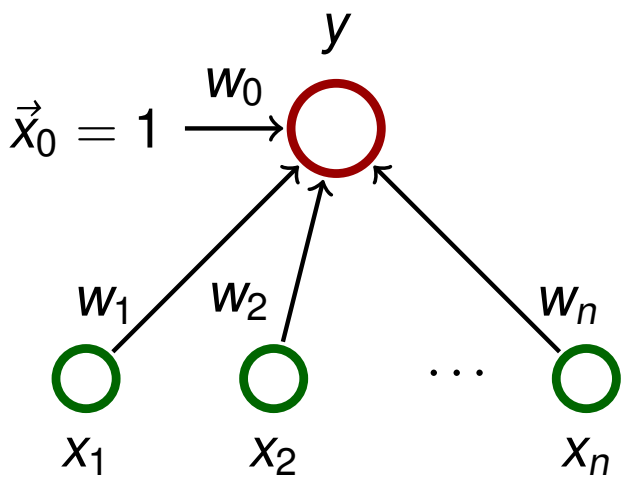
Po předložení nového vstupu $\vec{x}$, který je blízko některému $\vec{x}_k$ bude výstup sítě roven nebo alespoň blízko $\vec{d}_k$
(tato schopnost se nazývá *asociace*).

Zejména by síť měla mít schopnost *reprodukce*: Pro vstup $\vec{x}_k$ by měla dát výstup $\vec{d}_k$.

Lineární asociativní síť (alias ADALINE s Hebbovým učením)

(Pro jednoduchost a srovnání s ADALINE uvážíme pouze jeden výstup)

Organizační dynamika LAS:



$\vec{w} = (w_0, w_1, \ldots, w_n)$ a $\vec{x} = (x_0, x_1, \ldots, x_n)$ kde $x_0 = 1$.

Aktivní dynamika:

funkce sítě: $y[\vec{w}](\vec{x}) = \vec{w} \cdot \vec{x} = \sum_{i=0}^{n} w_i x_i$

Hebbovo učení

Adaptivní dynamika:

Dána množina **tréninkových vzorů**

$$\mathcal{T} = \left\{\left(\vec{x}_1, d_1\right), \left(\vec{x}_2, d_2\right), \ldots, \left(\vec{x}_p, d_p\right)\right\}$$

Zde $\vec{x}_k = (x_{k0}, x_{k1} \ldots, x_{kn})^\top \in \mathbb{R}^{n+1}$, $x_{k0} = 1$, je vstup k-tého vzoru a $d_k \in \mathbb{R}$ je očekávaný výstup.

Intuice: chceme, aby síť počítala afinní aproximaci funkce, jejíž (některé) hodnoty nám předepisuje tréninková množina.

Hebbovo učení

Hebbův zákon: *When an axon of a 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 in one or both cells such that A's efficiency, as one of the cells firing B, is increased.*

Zákon formuloval neuropsycholog Donald Hebb v knize the organization of Behavior z roku 1949.

Jinými slovy: *Cells that fire together, wire together.*

Formulace používaná v umělých NS:
Změna váhy spoje mezi dvěma neurony je úměrná jejich souhlasné aktivitě.

Hebb se snažil vysvětlit podmíněné reflexy: Současná aktivita/pasivita presynaptického neuronu (příčina) a postsynaptického neuronu (reakce) posiluje/zeslabuje synaptickou vazbu.

Hebbovo učení LAS

Algoritmus počítá posloupnost vektorů vah $\vec{w}^{(0)}, \vec{w}^{(1)}, \ldots, \vec{w}^{(p)}$:

- $\vec{w}_i^{(0)} = 0$ pro $0 \leq i \leq n$,
- v kroku k (zde $k = 1, 2, \ldots$) je síti předložen vzor $(\vec{x}_k, d_k)$ a váhy se adaptují podle Hebbova zákona:

$$\vec{w}^{(k)} = \vec{w}^{(k-1)} + \vec{x}_k d_k$$

Výsledný vektor:

$$\vec{w} = \vec{w}^{(p)} = \sum_{k=1}^{p} \vec{x}_k d_k = X^\top \vec{d}$$

kde X je matice, která má v *i-tém řádku* vektor $\vec{x}_i^\top$ a $\vec{d} = (d_1, \ldots, d_p)^\top$

LAS a ortonormální vstupy

Pokud jsou $\vec{x}_1, \ldots, \vec{x}_p$ ortonormální, tedy

$$\vec{x}_i^\top \vec{x}_j = \begin{cases} 0 & i \neq j \\ 1 & i = j \end{cases}$$

pak LAS má schopnost **reprodukce**:

$$\vec{w}^\top \vec{x}_i = \sum_{k=1}^{p} (\vec{x}_k d_k)^\top \vec{x}_i = \sum_{k=1}^{p} d_k (\vec{x}_k^\top \vec{x}_i) = d_i$$

LAS a ortonormální vstupy

... a **asociace**:

Uvažme vstup: $\vec{x}_r + \vec{u}$ kde norma $\left\|\vec{u}\right\|$ je malá.

Chyba sítě pro r-tý vzor perturbovaný vektorem $\vec{u}$:

$$E_r := |\vec{w}^\top(\vec{x}_r + \vec{u}) - d_r| = |\vec{w}^\top \vec{x}_r + \vec{w}^\top \vec{u} - d_r| = |\vec{w}^\top \vec{u}|$$

Pokud $\vec{d}_r \in \{-1, 1\}$, pak

$$E_r = |\vec{w}^\top \vec{u}| = \left|\left(\sum_{k=1}^{p} \vec{x}_k d_k\right)^\top \vec{u}\right| = \left|\sum_{k=1}^{p} d_k(\vec{x}_k^\top \vec{u})\right|$$

$$\leq \sum_{k=1}^{p} |d_k(\vec{x}_k^\top \vec{u})| \leq \sum_{k=1}^{p} |d_k| \left\|\vec{x}_k\right\| \left\|\vec{u}\right\| \leq n \left\|\vec{u}\right\|$$

(Zde první nerovnost plyne z trojúhelníkové nerovnosti, druhá z Cauchyho-Schwarzovy nerovnosti a poslední z $p \leq n$, protože mohutnost množiny ortonormálních vektorů v $\mathbb{R}^n$ nemůže být větší než n.)

Tedy pro vstupy blízké vzorům odpovídá přibližně požadovaným výstupem

Hopfieldova síť

- Definice
- Energetická funkce
- Reprodukce
- Asociace

Hopfieldova síť

Autoasociativní síť.

Organizační dynamika:

- úplná topologie, tj. každý neuron je spojen s každým
- všechny neurony jsou současně vstupní i výstupní
- označme $\xi_1, \ldots, \xi_n$ vnitřní potenciály a $y_1, \ldots, y_n$ výstupy (stavy) jednotlivých neuronů
- označme w_{ji} *celočíselnou* váhu spoje od neuronu $i \in \{1, \ldots, n\}$ k neuronu $j \in \{1, \ldots, n\}$
- **Zatím:** žádný neuron nemá bias a přepokládáme $w_{jj} = 0$ pro každé $j = 1, \ldots, n$

Hopfieldova síť

Adaptivní dynamika: Dána tréninková množina

$$\mathcal{T} = \{\vec{x}_k \mid \vec{x}_k = (x_{k1}, \ldots, x_{kn}) \in \{-1, 1\}^n, k = 1, \ldots, p\}$$

Adaptace probíhá podle Hebbova zákona (podobně jako u LAS). Výsledná konfigurace je

$$w_{ji} = \sum_{k=1}^{p} x_{kj} x_{ki} \qquad 1 \leq j \neq i \leq n$$

Všimněte si, že $w_{ji} = w_{ij}$, tedy matice vah je symetrická.

Adaptaci lze vidět jako hlasování vzorů o vazbách neuronů: $w_{ji} = w_{ij}$ se rovná rozdílu mezi počtem souhlasných stavů $x_{kj} = x_{ki}$ neuronů i a j a počtem rozdílných stavů $x_{kj} \neq x_{ki}$.

Hopfieldova síť

Aktivní dynamika: Iniciálně jsou neurony nastaveny na vstup $\vec{x} = (x_1, \ldots, x_n)$ sítě, tedy $y_j^{(0)} = x_j$ pro každé $j = 1, \ldots, n$.

Cyklicky aktualizujeme stavy neuronů, tedy v kroku $t + 1$ aktualizujeme neuron j, t. ž. $j = (t \mod p) + 1$, takto:

nejprve vypočteme vnitřní potenciál

$$\xi_j^{(t)} = \sum_{i=1}^{n} w_{ji} y_i^{(t)}$$

a poté

$$y_j^{(t+1)} = \begin{cases} 1 & \xi_j^{(t)} > 0 \\ y_j^{(t)} & \xi_j^{(t)} = 0 \\ -1 & \xi_j^{(t)} < 0 \end{cases}$$

Hopfieldova síť - aktivní dynamika

Výpočet končí v kroku t^* pokud se síť nachází (poprvé) ve *stabilním* stavu, tj.

$$y_j^{(t^*+n)} = y_j^{(t^*)} \qquad (j = 1, \ldots, n)$$

Theorem

Za předpokladu symetrie vah, výpočet Hopfieldovy sítě skončí pro každý vstup.

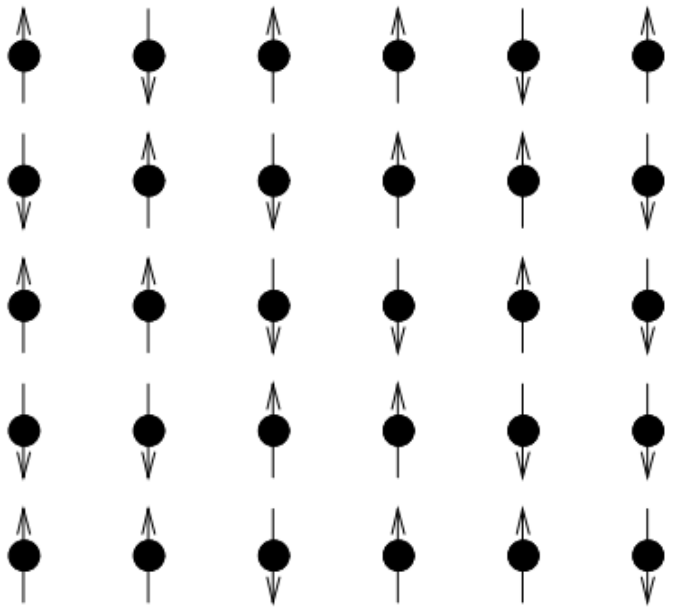
Z toho plyne, že Hopfieldova síť počítá funkci z $\{-1, 1\}^n$ do $\{-1, 1\}^n$ (která závisí na hodnotách vah neuronů).

Označme $\vec{y}(W, \vec{x}) = \left(y_1^{(t^*)}, \ldots, y_n^{(t^*)}\right)$ hodnotu funkce sítě pro vstup $\vec{x}$ a matici vah W. Dále označme $y_j(W, \vec{x}) = y_j^{(t^*)}$ složku hodnoty funkce sítě, která odpovídá neuronu j.

Pokud bude W jasné z kontextu, budu psát jen $y(\vec{x})$ a $y_j(\vec{x})$

Fyzikální analogie (Isingův model)

Jednoduché modely magnetických materiálů připomínají Hopfieldovu síť.



- atomické magnety poskládané do mřížky
- každý magnet může mít pouze jednu ze dvou orientací (v Hopfieldově síti $+1$ a -1)
- orientaci každého magnetu ovlivňuje jednak vnější magnetické pole (vstup sítě), jednak magnetické pole ostatních magnetů (závisí na jejich orientaci)
- synaptické váhy modelují vzájemnou interakci magnetů

Energetická funkce

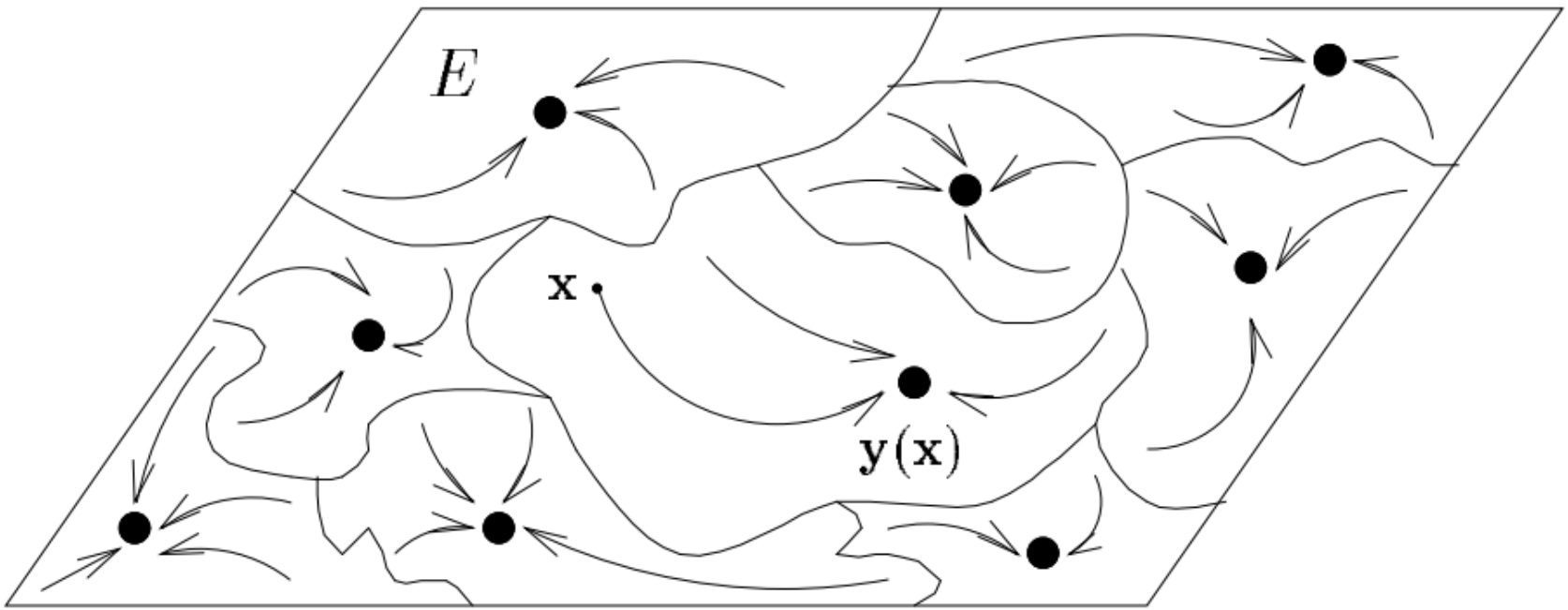
Energetická funkce E přiřazuje každému stavu sítě $\vec{y} \in \{-1, 1\}^n$ potenciální energii danou

$$E(\vec{y}) = -\frac{1}{2} \sum_{j=1}^{n} \sum_{i=1}^{n} w_{ji} y_j y_i$$

- stavy s nízkou energií jsou stabilní (málo neuronů chce změnit svůj stav), stavy s vysokou energií jsou nestabilní
- tj. velké (kladné) $w_{ji} y_j y_i$ je stabilní a malé (záporné) $w_{ji} y_j y_i$ nestabilní

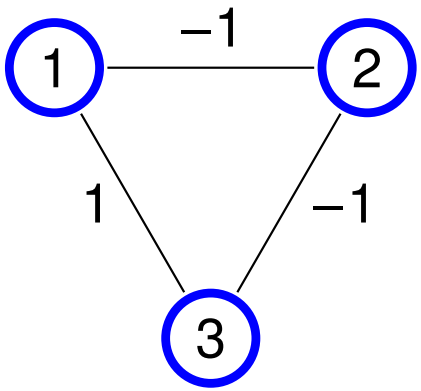
V průběhu výpočtu se energie nezvyšuje: $E(\vec{y}^{(t)}) \geq E(\vec{y}^{(t+1)})$, stav $\vec{y}^{(t^*)}$ odpovídá lokálnímu minimu funkce E.

Energetická plocha



Obr. 3.4: Energetická plocha.

Hopfield - příklad



y_1	y_2	y_3	E
1	1	1	1
1	1	−1	1
1	−1	1	−3
1	−1	−1	1
−1	1	1	1
−1	1	−1	−3
−1	−1	1	1
−1	−1	−1	1

- Hopfieldova síť se třemi neurony
- naučili jsme ji jeden vzor $(1, -1, 1)$ pomocí Hebbova učení (síť se automaticky naučila i vzor $(-1, 1, -1)$)

Energetická funkce - konvergence výpočtu sítě

Pomocí pojmu energie lze snadno dokázat, že výpočet Hopfieldovy sítě vždy zastaví:

- v průběhu výpočtu se energie nezvyšuje: $E(\vec{y}^{(t)}) \geq E(\vec{y}^{(t+1)})$
- pokud dojde v kroku $t+1$ ke změně stavu, pak $E(\vec{y}^{(t)}) > E(\vec{y}^{(t+1)})$
- existuje pouze konečně mnoho stavů sítě: výpočet dosáhne lokálního minima funkce E, ze kterého už se nedostane

Hopfieldova síť - odučování

Při učení podle Hebbova zákona mohou vznikat lokální minima funkce E, tzv. nepravé vzory (*fantomy*), které neodpovídají tréninkovým vzorům.

Fantomy je možné odučovat např. pomocí následujícího pravidla: Mějme fantom $(x_1, \ldots, x_n) \in \{-1, 1\}^n$ a váhy w_{ji}, pak nové váhy w'_{ji} spočítáme pomocí

$$w'_{ji} = w_{ji} - x_i x_j$$

(tj. podobně jako při adaptaci podle Hebbova zákona, ale s opačným znaménkem)

Reprodukce - statistická analýza

Kapacita Hopfieldovy paměti je dána poměrem p/n.

Zde n je počet neuronů a p je počet vzorů.

Předpokládejme, že tréninkové vzory jsou voleny náhodně takto: při volbě $\vec{x}_k$ volím postupně (nezávisle) jednotlivé složky (1 s pravd. $1/2$ a -1 s pravd. $1/2$).

Uvažme konfiguraci W, kterou obdržíme Hebbovským učením na zvolených vzorech.

Označme

$$\beta = \mathbf{P}\left[\vec{x}_k = \vec{y}(W, \vec{x}_k) \text{ pro } k = 1, \ldots, p\right]$$

Pak pro $n \to \infty$ a $p \leq n/(4 \log n)$ dostaneme $\beta \to 1$.

Tj. maximální počet vzorů, které lze věrně uložit do Hopfieldovy paměti je úměrný $n/(4 \log n)$.

Hopfieldova síť - asociace

Problém:

- příliš mnoho vzorů implikuje existenci lokálních minim funkce E, která neodpovídají vzorům (tzv. *fantomy*)
- lokální minima pro vzory mohou dokonce zanikat

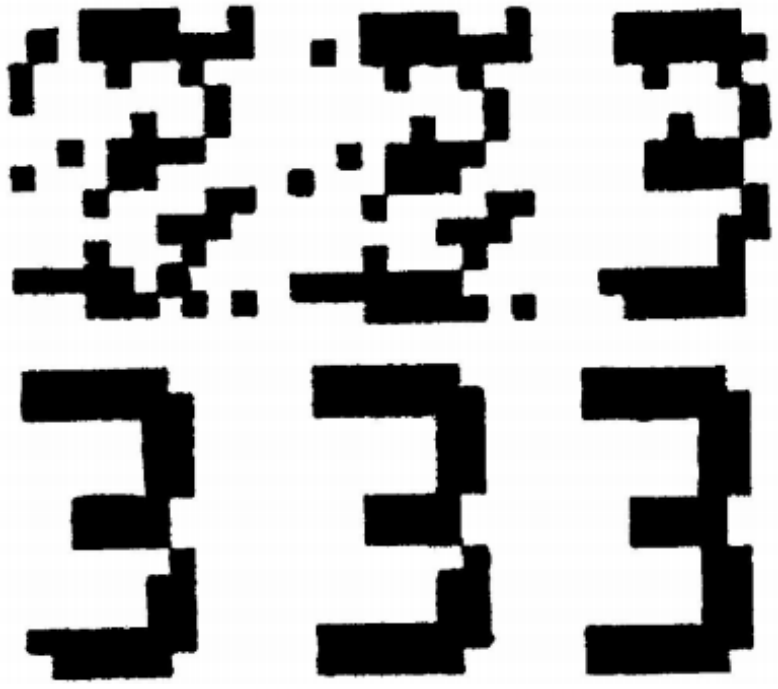
Podrobná analýza ukazuje následující

- Pro $p \leq 0.138n$ tréninkové vzory *zhruba* odpovídají lokálním minimům funkce E
- Pro $p > 0.138n$ lokální minima podobající se vzorům zanikají (ostrá nespojitost v 0.138)
- Pro $p < 0.05n$ energie stavů podobajících se tréninkovým vzorům odpovídají globálním minimům E a fantomy mají ostře větší energii

Tj. pro dobré zapamatování 10 vzorů je potřeba 200 neuronů a tedy 40000 spojů ohodnocených celočíselnými váhami

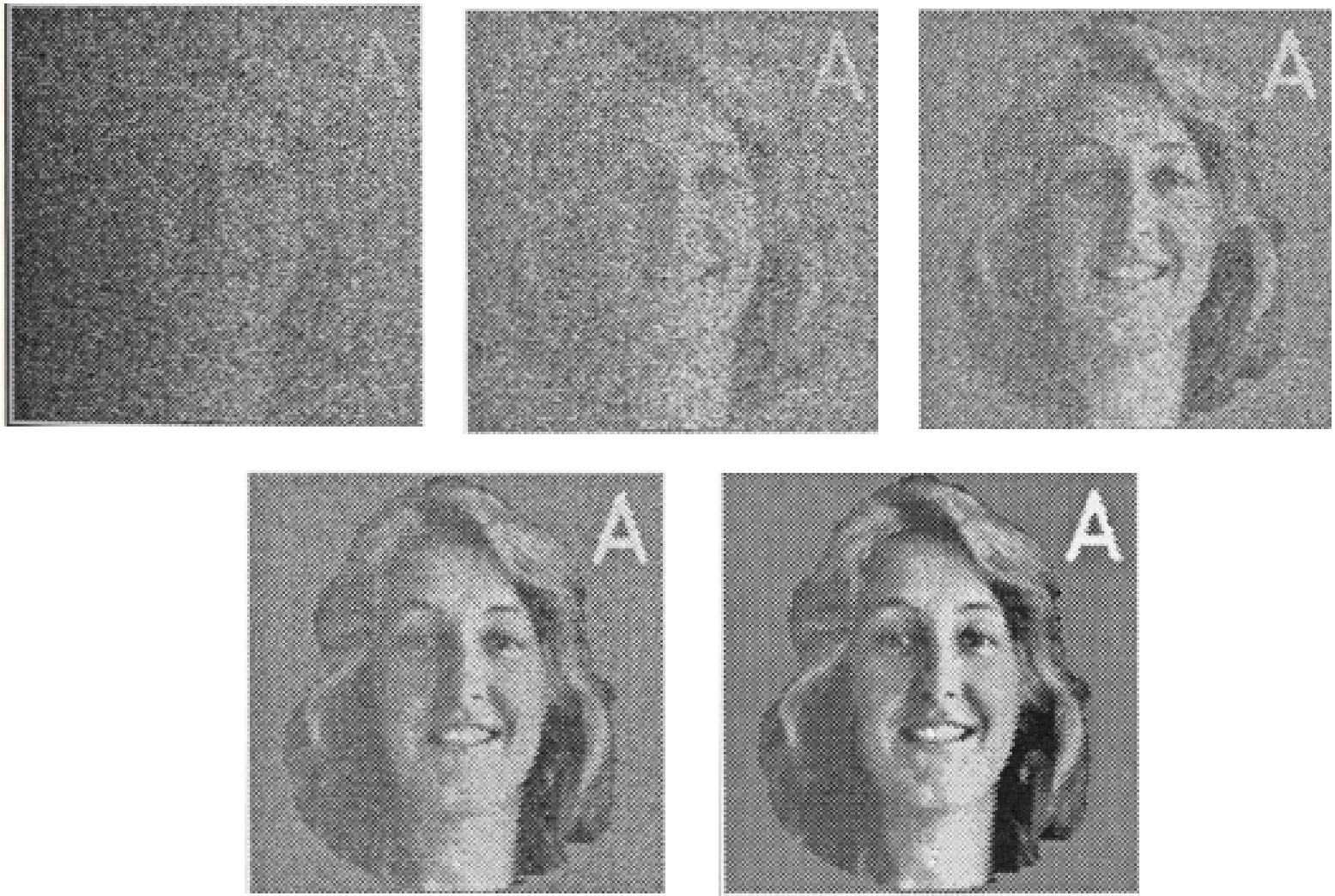
Pozn. Nevýhodou Hopfieldovy sítě je deterministický výpočet, který může skončit v mělkém lokálním minimu E bez možnosti uniknout. Tento problém částečně vyřeší stochastická verze aktivní dynamiky.

Hopfieldova síť - příklad kódování

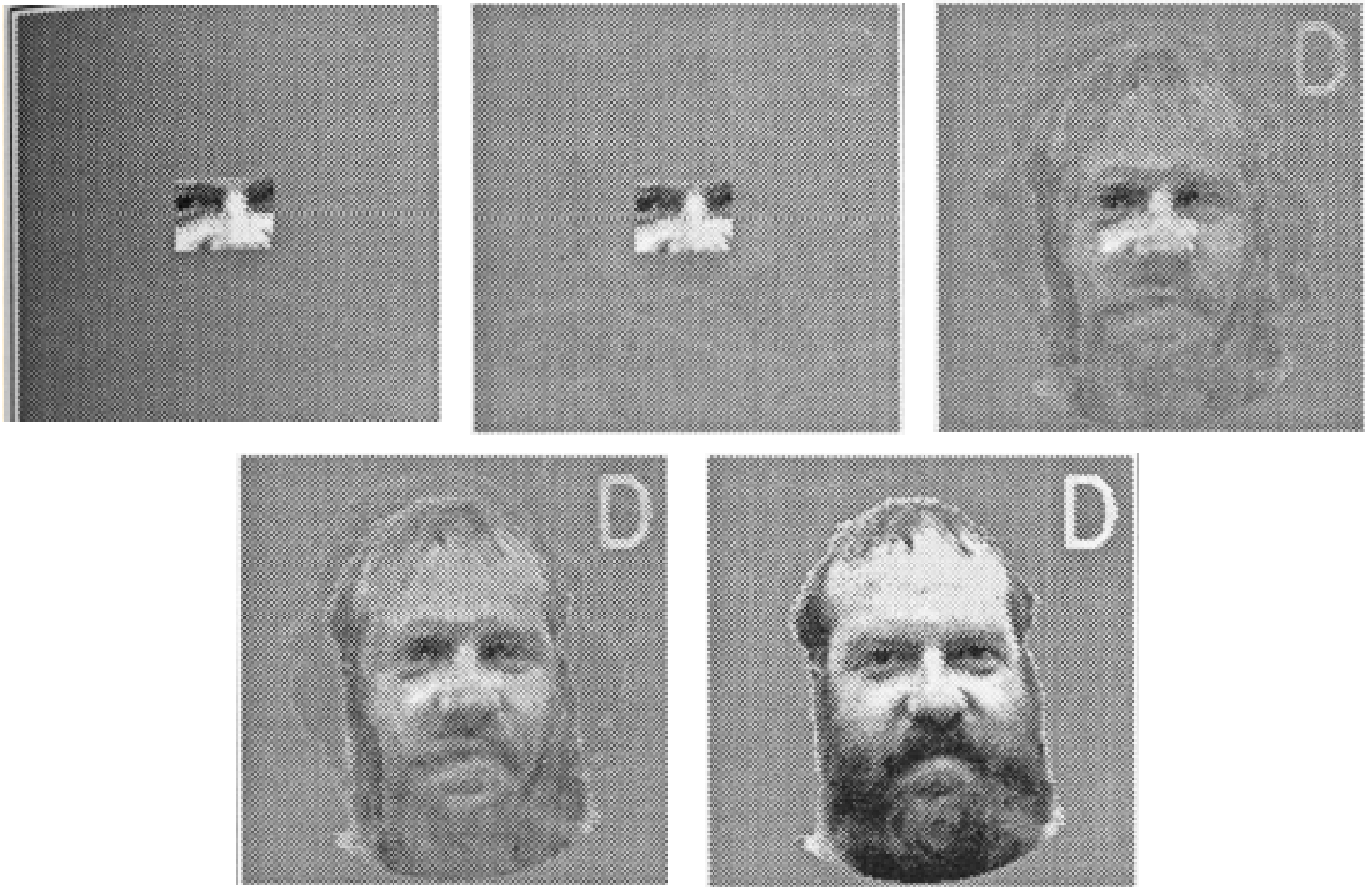


- číslice 12×10 bodů
  (120 neuronů, -1 je bílá a 1 je černá)
- naučeno 8 číslic
- vstup vygenerován ze vzoru 25% šumem
- obrázek ukazuje postup výpočtu Hopfieldovy sítě

Hopfieldova síť - příklad obnovení vzoru



Hopfieldova síť - příklad rekonstrukce vzoru



Hopfieldova síť (pro optimalizační úlohy)

Autoasociativní síť.

Organizační dynamika:

- úplná topologie, tj. každý neuron je spojen s každým
- všechny neurony jsou současně vstupní i výstupní
- označme $\xi_1, \ldots, \xi_n$ vnitřní potenciály a $y_1, \ldots, y_n$ výstupy (stavy) jednotlivých neuronů
- označme w_{ji} *celočíselnou* váhu spoje od neuronu $i \in \{1, \ldots, n\}$ k neuronu $j \in \{1, \ldots, n\}$.
- přepokládáme $w_{jj} = 0$ pro každé $j = 1, \ldots, n$.
- **Nyní:**
  - každý neuron má bias θ_i
  - stavy neuronů jsou z $\{0, 1\}$

Hopfieldova síť (s biasy)

Aktivní dynamika: Iniciálně jsou neurony nastaveny na vstup $\vec{x} = (x_1, \ldots, x_n) \in \{0,1\}^n$ sítě, tedy $y_j^{(0)} = x_j$ pro $j = 1, \ldots, n$.

V kroku $t+1$ aktualizujeme neuron j, t. ž. $j = (t \mod p) + 1$, takto:

nejprve vypočteme vnitřní potenciál

$$\xi_j^{(t)} = \sum_{i=1}^{n} w_{ji} y_i^{(t)} - \theta_j$$

a poté

$$y_j^{(t+1)} = \begin{cases} 1 & \xi_j^{(t)} > 0 \\ y_j^{(t)} & \xi_j^{(t)} = 0 \\ 0 & \xi_j^{(t)} < 0 \end{cases}$$

Hopfieldova síť - aktivní dynamika

Výpočet končí v kroku t^* pokud se síť nachází (poprvé) ve *stabilním* stavu, tj.

$$y_j^{(t^*+n)} = y_j^{(t^*)} \qquad (j = 1, \ldots, n)$$

Energetická funkce E přiřazuje každému stavu sítě $\vec{y} \in \{0,1\}^n$ potenciální energii danou

$$E(\vec{y}) = -\frac{1}{2} \sum_{j=1}^{n} \sum_{i=1}^{n} w_{ji} y_j y_i + \sum_{i=1}^{n} \theta_i y_i$$

V průběhu výpočtu se energie nezvyšuje: $E(\vec{y}^{(t)}) \geq E(\vec{y}^{(t+1)})$, stav $\vec{y}^{(t^*)}$ odpovídá lokálnímu minimu funkce E.

Theorem

Za předpokladu symetrie vah, výpočet Hopfieldovy sítě skončí pro každý vstup.

(Důkaz stejný jako předtím)

Hopfieldova síť a optimalizační úlohy

Optimalizační úloha je zadána množinou přípustných řešení a účelovou funkcí. Cílem je nalézt přípustné řešení, které minimalizuje účelovou funkci U.

Pro mnoho optimalizačních úloh lze nalézt Hopfieldovu síť takovou, že

- minima $E \approx$ přípustná řešení vzhledem k U
- globální minima $E \approx$ řešení minimalizující U

Cílem je nalézt globální minimum funkce E (a tedy i U).

Příklad: multiflop

Cílem je nalézt vektor z $\{0,1\}^n$, který má všechny složky nulové kromě právě jedné.

Definujeme účelovou funkci $U : \{0,1\}^n \to \mathbb{R}$ takto:

$$U(\vec{u}) = \left(\left(\sum_{i=1}^{n} u_i\right) - 1\right)^2$$

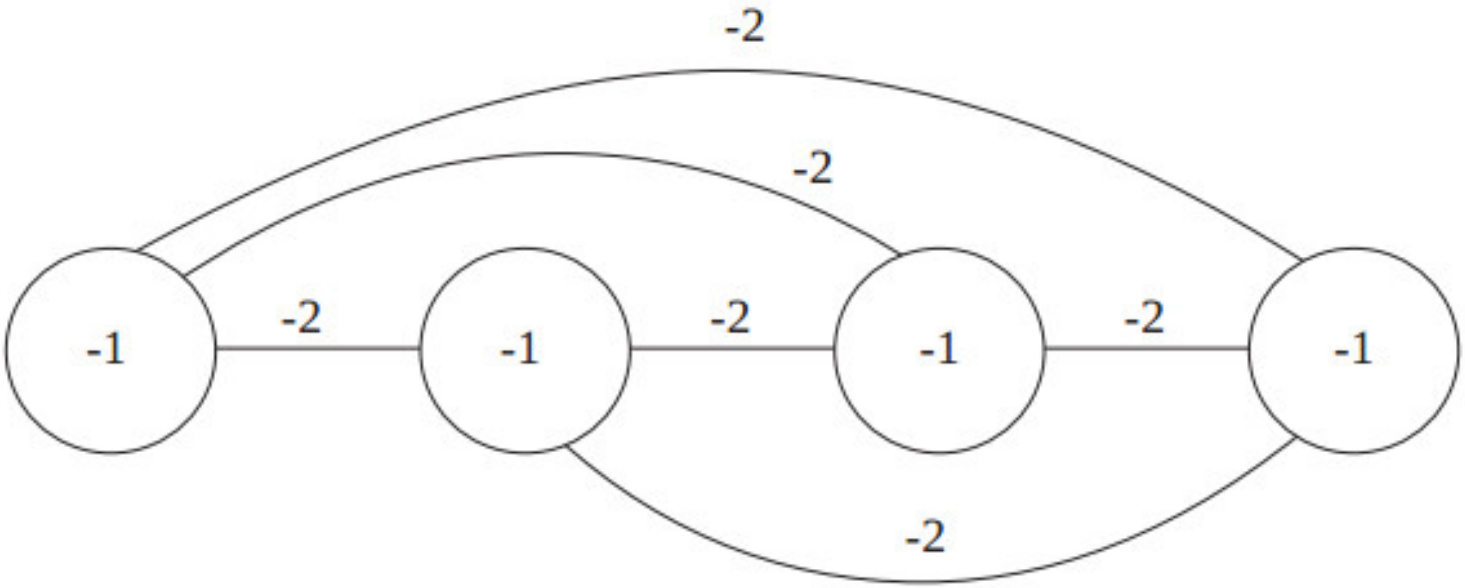
Požadované vektory jsou právě minima této funkce.

Ale

$$U(\vec{u}) = -\frac{1}{2}\sum_{i\neq j}^{n}(-2)u_i u_j + \sum_{i=1}^{n}(-1)u_i + 1$$

a tedy $U(\vec{u}) - 1$ je energetickou funkcí sítě (viz. následující slajd).

Příklad: multiflop (síť)



$$E(\vec{u}) = -\frac{1}{2}\sum_{i\neq j}^{n}(-2)u_i u_j + \sum_{i=1}^{n}(-1)u_i$$

Příklad: n věží

Cílem je rozmístit n věží na šachovnici $n \times n$ tak, aby se vzájemně neohrožovaly.

Definujeme účelovou funkci $U_1 : \{0,1\}^{n \cdot n} \to \mathbb{R}$:

$$U_1(\vec{u}) = \sum_{j=1}^{n} \left(\left(\sum_{i=1}^{n} u_{ji} \right) - 1 \right)^2$$

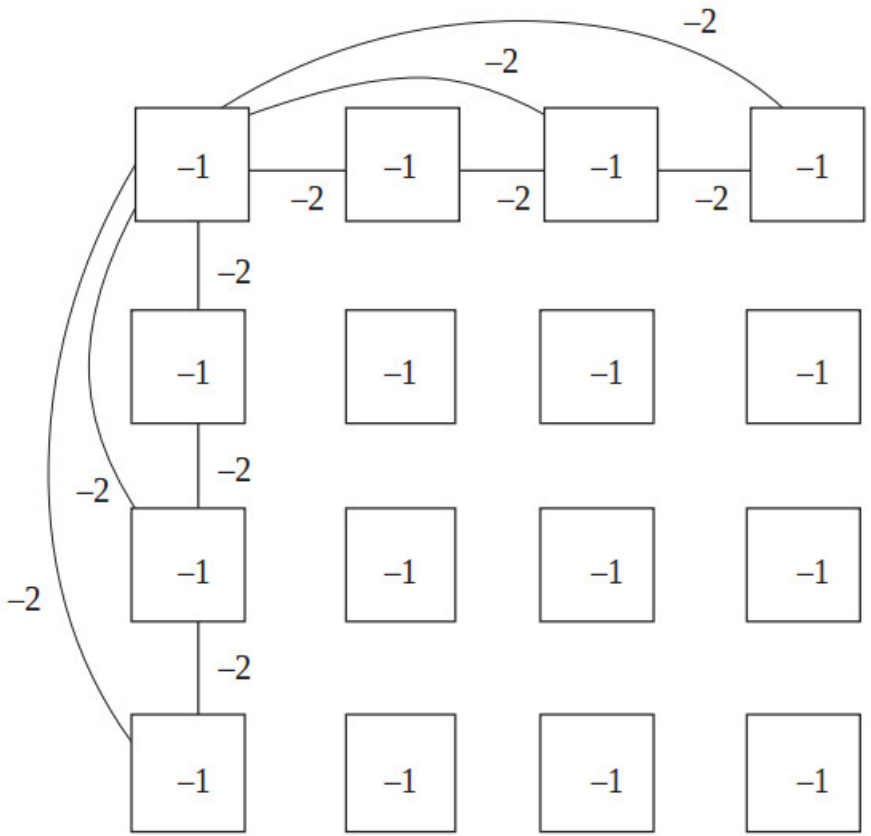
a $U_2 : \{0,1\}^{n \cdot n} \to \mathbb{R}$:

$$U_2(\vec{u}) = \sum_{i=1}^{n} \left(\left(\sum_{j=1}^{n} u_{ji} \right) - 1 \right)^2$$

Požadované vektory jsou právě minima funkce $U = U_1 + U_2$.

Minima U odpovídají stavům s minimální energií v následující síti.

Příklad: n věží (síť)



$$E(\vec{u}) = U(\vec{u}) - 2n$$

(Tento příklad se dá zobecnit na problém obchodního cestujícího)

Hopfieldova síť a lokální minima

Hledáme (globální) minima energie E ....

Problém: není jasné, v jakém stavu začít, abychom dosáhli globálního minima. Síť může skončit v mělkém minimu.

Řešení: V každém stavu umožníme s malou pravděpodobností přechod do stavů s vyšší energií. Tuto pravděpodobnost budeme postupně snižovat. Využijeme dynamiku Boltzmannova stroje ...

Boltzmannovská aktivní dynamika

Aktivní dynamika: Stavy neuronů jsou iniciálně nastaveny na hodnoty z množiny $\{0,1\}$, tj. $y_j^{(0)} \in \{0,1\}$ pro $j \in \{1,\ldots,n\}$.

V kroku $t+1$ aktualizujeme náhodně vybraný neuron $j \in \{1,\ldots,n\}$ takto: nejprve vypočteme vnitřní potenciál

$$\xi_j^{(t)} = \sum_{i=1}^{n} w_{ji} y_i^{(t)} - \theta_j$$

a poté náhodně zvolíme hodnotu $y_j^{(t+1)} \in \{0,1\}$ tak, že

$$\mathbf{P}\left[y_j^{(t+1)} = 1\right] = \sigma\left(\xi_j^{(t)}\right)$$

kde

$$\sigma(\xi) = \frac{1}{1 + e^{-2\xi/T(t)}}$$

Parametr $T(t)$ se nazývá **teplota** v čase t.

Teplota a energie

- Velmi vysoká teplota $T(t)$ znamená, že $\mathbf{P}\left[y_j^{(t+1)} = 1\right] \approx \frac{1}{2}$ a síť se chová téměř (uniformně) náhodně.
- Velmi nízká teplota $T(t)$ znamená, že buď $\mathbf{P}\left[y_j^{(t+1)} = 1\right] \approx 1$ nebo $\mathbf{P}\left[y_j^{(t+1)} = 1\right] \approx 0$ v závislosti na tom, jestli $\xi_j^{(t)} > 0$ nebo $\xi_j^{(t)} < 0$. Potom se sít chová téměř deterministicky (tj. jako v původní aktivní dynamice).

Poznámky:

- Boltzmannovská aktivní dynamika funguje jako deterministická dynamika s náhodným šumem,
- energie $E(\vec{y}) = -\frac{1}{2}\sum_{j=1}^{n}\sum_{i=1}^{n} w_{ji}y_jy_i + \sum_{i=1}^{n}\theta_i y_i$ se může (s pravděpodobností závislou na teplotě) zvýšit,
- pravděpodobnost přechodů do vyšší energetické hladiny se exponenciálně zmenšuje s velikostí energetického skoku.

Simulované žíhání

Následujícím postupem lze dosáhnout globálního minima funkce E:

- Na začátku výpočtu nastavíme vysokou teplotu $T(t)$
- Teplotu postupně snižujeme, např takto:
  - $T(t) = \eta^t \cdot T(0)$ kde $\eta < 1$ je blízko 1
  - nebo $T(t) = T(0)/\log(1+t)$

Lze dokázat, že při vhodném postupu chlazení dosáhneme globálního minima.

Pozn:

- Tento proces je analogií žíhání používané při výrobě tvrdých kovových materiálů s krystalickou strukturou: materiál se nejprve zahřeje, čímž se poruší vazby mezi atomy, v průběhu následného pomalého chlazení se materiál usadí do stavu s minimální vnitřní energií a s pravidelnou vnitřní strukturou.
- Jedná se také o rozšíření fyzikální motivace Hopfieldovy sítě: orientace magnetů jsou ovlivněny nejen vnitřním a vnějším magnetickým polem, ale také termálními fluktuacemi.

Boltzmannův stroj

Organizační dynamika:

- Cyklická síť se symetrickými spoji (tj. libovolný neorientovaný graf)
- Množinu všech neuronů značíme N
- označme ξ_j vnitřní potenciál a y_j výstup (stav) neuronu j
- stav stroje: $\vec{y} \in \{-1, 1\}^{|N|}$.
- označme w_{ji} *reálnou* váhu spoje od neuronu i k neuronu j.
- žádný neuron nemá bias a přepokládáme $w_{jj} = 0$ pro $j \in N$.

Botzmannův stroj

Aktivní dynamika: Stavy neuronů jsou iniciálně nastaveny na hodnoty z množiny $\{-1, 1\}$, tj. $y_j^{(0)} \in \{-1, 1\}$ pro $j \in N$.

V t-tém kroku aktualizujeme náhodně vybraný neuron $j \in N$ takto: nejprve vypočteme vnitřní potenciál

$$\xi_j^{(t-1)} = \sum_{i \in j_{\leftarrow}}^{n} w_{ji} y_i^{(t-1)}$$

a poté náhodně zvolíme hodnotu $y_j^{(t)} \in \{-1, 1\}$ tak, že $\mathbf{P}\left[y_j^{(t)} = 1\right] = \sigma(\xi_j^{(t-1)})$ kde

$$\sigma(\xi) = \frac{1}{1 + e^{-2\xi/T(t)}}$$

Parametr $T(t)$ se nazývá **teplota** v čase t.

Boltzmannův stroj

- Velmi vysoká teplota $T(t)$ znamená, že $\mathbf{P}\left[y_j^{(t)} = 1\right] \approx \frac{1}{2}$ a stroj se chová téměř náhodně.
- Velmi nízká teplota $T(t)$ znamená, že buď $\mathbf{P}\left[y_j^{(t)} = 1\right] \approx 1$ nebo $\mathbf{P}\left[y_j^{(t)} = 1\right] \approx 0$ v závislosti na tom, jestli $\xi_j^{(t)} > 0$ nebo $\xi_j^{(t)} < 0$. Potom se stroj chová téměř deterministicky (tj. jako Hopfieldova síť).

Boltzmannův stroj - reprezentace rozložení

Cíl: Chceme sestrojit síť, která bude reprezentovat dané pravděpodobnostní rozložení na množině vektorů $\{-1,1\}^{|N|}$.

Velmi hrubá a nepřesná idea: Boltzmannův stroj mění náhodně svůj stav z množiny $\{-1,1\}^{|N|}$.

Když necháme B. stroj běžet dost dlouho s fixní teplotou, potom budou frekvence návštěv jednotlivých stavů nezávislé na iniciálním stavu.

Tyto frekvence budeme považovat za pravděpodobnostní rozložení na $\{-1,1\}^{|N|}$ reprezentované B. strojem.

V adaptivním režimu bude zadáno nějaké rozložení na stavech a cílem bude nalézt konfiguraci takovou, že rozložení reprezentované strojem bude odpovídat zadanému rozložení.

Rovnovážný stav

Fixujme teplotu T (tj. $T(t) = T$ pro $t = 1, 2, \ldots$).

Boltzmannův stroj se po jisté době dostane do tzv. *termální rovnováhy*. Tj. existuje čas t' takový, že pro libovolný stav stroje $\gamma^* \in \{-1, 1\}^{|N|}$ a libovolné $t^* \geq t'$ platí, že

$$p_N(\gamma^*) := \mathbf{P}\left[\vec{y}^{(t^*)} = \gamma^*\right]$$

splňuje $p_N(\gamma^*) \approx \frac{1}{Z} e^{-E(\gamma^*)/T}$ kde

$$Z = \sum_{\gamma \in \{-1,1\}^{|N|}} e^{-E(\gamma)/T} \qquad E(\gamma) = -\frac{1}{2} \sum_{i,j} w_{ij} y_i^\gamma y_j^\gamma$$

tj. Boltzmannovo rozložení

Pozn.: Teorie Markovových řetězců říká, že $\mathbf{P}\left[\vec{y}^{(t^*)} = \gamma^*\right]$ je také dlouhodobá frekvence návštěv stavu γ^*.

Toto platí *bez ohledu na iniciální nastavení neuronů!* Síť tedy reprezentuje rozložení p_N.

Boltzmannův stroj - učení

Problém: Tak jak jsme si jej definovali má Boltzmannův stroj omezenou schopnost reprezentovat daná rozložení.

Proto množinu neuronů N disjunktně rozdělíme na

- množinu **viditelných** neuronů V
- množinu **skrytých** neuronů S.

Pro daný stav viditelných neuronů $\alpha \in \{-1, 1\}^{|V|}$ označme

$$p_V(\alpha) = \sum_{\beta \in \{-1,1\}^{|S|}} p_N(\alpha, \beta)$$

pravděpodobnost stavu viditelných neuronů α v termálním ekvilibriu bez ohledu na stav skrytých neuronů.

Cílem bude adaptovat síť podle daného rozložení na $\{-1, 1\}^{|V|}$.

Boltzmannův stroj - učení

Adaptivní dynamika:
Nechť p_d je pravděpodobnostní rozložení na množině stavů viditelných neuronů, tj. na $\{-1, 1\}^{|V|}$.

Cílem je nalézt konfiguraci sítě W takovou, že p_V odpovídá p_d.

Vhodnou mírou rozdílu mezi rozděleními p_V a p_d je relativní entropie zvážená pravděpodobnostmi vzorů
(tzv. Kullback-Leibler divergence)

$$\mathcal{E}(W) = \sum_{\alpha \in \{-1,1\}^{|V|}} p_d(\alpha) \ln \frac{p_d(\alpha)}{p_V(\alpha)}$$

Boltzmannův stroj - učení

$\mathcal{E}(\vec{w})$ budeme minimalizovat pomocí gradientního sestupu, tj. budeme počítat poslounost matic vah $W^{(0)}, W^{(1)}, \ldots$

- váhy v $W^{(0)}$ jsou inicializovány náhodně blízko 0
- v ℓ-tém kroku (zde $\ell = 1, 2, \ldots$) je $W^{(\ell)}$ vypočteno takto:

$$W_{ji}^{(\ell)} = W_{ji}^{(\ell-1)} + \Delta W_{ji}^{(\ell)}$$

kde

$$\Delta W_{ji}^{(\ell)} = -\varepsilon(\ell) \cdot \frac{\partial \mathcal{E}}{\partial w_{ji}}(W^{(\ell-1)})$$

je změna váhy w_{ji} v ℓ-tém kroku a $0 < \varepsilon(\ell) \leq 1$ je rychlost učení v ℓ-tém kroku.

Zbývá spočítat (odhadnout) $\frac{\partial \mathcal{E}}{\partial w_{ji}}(W)$.

Boltzmannův stroj - učení

Formálním derivováním funkce $\mathcal{E}$ lze ukázat, že

$$\frac{\partial \mathcal{E}}{\partial w_{ji}} = -\frac{1}{T}\left(\left\langle y_j^{(t^*)} y_i^{(t^*)}\right\rangle_{fixed} - \left\langle y_j^{(t^*)} y_i^{(t^*)}\right\rangle_{free}\right)$$

- $\left\langle y_j^{(t^*)} y_i^{(t^*)}\right\rangle_{fixed}$ je průměrná hodnota $y_j^{(t^*)} y_i^{(t^*)}$ v termální rovnováze za předpokladu, že hodnoty viditelných neuronů jsou **fixovány** na počátku výpočtu dle rozložení p_d.
- $\left\langle y_j^{(t^*)} y_i^{(t^*)}\right\rangle_{free}$ je průměrná hodnota $y_j^{(t^*)} y_i^{(t^*)}$ v termální rovnováze bez fixace viditelných neuronů.

Celkově

$$\begin{aligned}\Delta w_{ji}^{(\ell)} &= -\varepsilon(\ell) \cdot \frac{\partial \mathcal{E}}{\partial w_{ji}}(W^{(\ell-1)}) \\ &= \frac{\varepsilon(\ell)}{T}\left(\left\langle y_j^{(t^*)} y_i^{(t^*)}\right\rangle_{fixed} - \left\langle y_j^{(t^*)} y_i^{(t^*)}\right\rangle_{free}\right)\end{aligned}$$

Boltzmannův stroj - učení

Pro výpočet $\left\langle y_j^{(t^*)} y_i^{(t^*)} \right\rangle_{fixed}$ proveď následující:

- Polož $\mathcal{Y} := 0$ a proveď následující akce q krát:
  1. fixuj náhodně hodnoty viditelných neuronů dle rozložení p_d (tj. v průběhu následujících kroků je neaktualizuj)
  2. simuluj stroj po t^* kroků
  3. přičti aktuální hodnotu $y_j^{(t^*)} y_i^{(t^*)}$ k proměnné $\mathcal{Y}$.
- $\mathcal{Y}/q$ bude dobrým odhadem $\left\langle y_j^{(t^*)} y_i^{(t^*)} \right\rangle_{fixed}$ za předpokladu, že q je dostatečně velké číslo.

$\left\langle y_j^{(t^*)} y_i^{(t^*)} \right\rangle_{free}$ se odhadne podobně, pouze se nefixují viditelné neurony (tj. v kroku 1. se zvolí libovolný startovní stav a v následném výpočtu se mohou aktualizovat všechny neurony).

Boltzmannův stroj - učení

Pro upřesnění analytická verze:

$$\left\langle y_i^{(t^*)} y_j^{(t^*)} \right\rangle_{fixed} =$$

$$= \sum_{\alpha \in \{-1,1\}^{|V|}} p_d(\alpha) \sum_{\beta \in \{-1,1\}^{|S|}} \frac{p_N(\alpha,\beta)}{p_V(\alpha)} y_j^{\alpha\beta} y_i^{\alpha\beta}$$

kde $y_j^{\alpha\beta}$ je výstup neuronu j ve stavu (α,β).

$$\left\langle y_i^{(t^*)} y_j^{(t^*)} \right\rangle_{free} = \sum_{\gamma \in \{-1,1\}^{|N|}} p_N(\gamma) y_j^{\gamma} y_i^{\gamma}$$

Omezený Boltzmannův stroj (OBS)

Organizační dynamika:

- Cyklická síť se symetrickými spoji, neurony jsou rozděleny do dvou skupin:
  - V - viditelné
  - S - skryté

  Množina spojů je $V \times S$ (tj. úplný bipartitní graf)
- Množinu všech neuronů značíme N
- označme ξ_j vnitřní potenciál a y_j výstup (stav) neuronu j
- stav stroje: $\vec{y} \in \{0,1\}^{|N|}$.
- označme w_{ji} váhu spoje mezi neuronem i a neuronem j.
- Uvažme bias: w_{j0} je váha mezi neuronem j a "fiktivním" neuronem 0 jehož hodnota y_0 je stále 1.

Omezený Botzmannův stroj

Aktivní dynamika: Hodnoty viditelných neuronů jsou iniciálně nastaveny na hodnoty z množiny $\{0, 1\}$.

V t-tém kroku aktualizujeme neurony takto:

- t liché: náhodně zvolíme nové hodnoty skrytých neuronů, pro $j \in S$

$$\mathbf{P}\left[y_j^{(t)} = 1\right] = 1 \Big/ \left(1 + \exp\left(-w_{j0} - \sum_{i \in V} w_{ji} y_i^{(t-1)}\right)\right)$$

- t sudé: náhodně zvolíme nové hodnoty viditelných neuronů, pro $j \in V$

$$\mathbf{P}\left[y_j^{(t)} = 1\right] = 1 \Big/ \left(1 + \exp\left(-w_{j0} - \sum_{i \in S} w_{ji} y_i^{(t-1)}\right)\right)$$

Rovnovážný stav

Definujeme energetickou funkci E

$$E(\vec{y}) = -\sum_{i \in V, j \in S} w_{ji} y_j y_i - \sum_{i \in V} w_{i0} y_i - \sum_{j \in S} w_{j0} y_j$$

Omezený Boltzmannův stroj se po jisté době dostane do *termální rovnováhy*. Tj. existuje čas t^* takový, že pro libovolný stav stroje $\gamma^* \in \{0,1\}^{|N|}$ platí

$$\mathbf{P}\left[\vec{y}^{(t^*)} = \gamma^*\right] \approx p_N(\gamma^*)$$

Zde $p_N(\gamma^*) = \frac{1}{Z} e^{-E(\gamma^*)}$ kde

$$Z = \sum_{\gamma \in \{0,1\}^{|N|}} e^{-E(\gamma)}$$

tj. Boltzmannovo rozložení

Síť tedy reprezentuje rozložení p_N.

Omezený Boltzmannův stroj - učení

Pro daný stav viditelných neuronů $\alpha \in \{0,1\}^{|V|}$ označme

$$p_V(\alpha) = \sum_{\beta \in \{0,1\}^{|S|}} p_N(\alpha, \beta)$$

pravděpodobnost stavu viditelných neuronů α v termální rovnováze bez ohledu na stav skrytých neuronů.

Adaptivní dynamika:
Nechť p_d je pravděpodobnostní rozložení na množině stavů viditelných neuronů, tj. na $\{0,1\}^{|V|}$.

Rozložení p_d může být dáno tréninkovou posloupností

$$\mathcal{T} = \vec{x}_1, \vec{x}_2, \ldots, \vec{x}_p$$

tak, že

$$p_d(\alpha) = \#(\alpha, \mathcal{T})/p$$

kde $\#(\alpha, \mathcal{T})$ je počet výskytů α v posloupnosti $\mathcal{T}$

Cílem je nalézt konfiguraci sítě W takovou, že p_V odpovídá p_d.

Omezený Boltzmannův stroj - učení

Vhodnou mírou rozdílu mezi rozděleními p_V a p_d je relativní entropie zvážená pravděpodobnostmi vzorů (tzv. Kullback Leibler distance)

$$\mathcal{E}(\vec{w}) = \sum_{\alpha \in \{0,1\}^{|V|}} p_d(\alpha) \ln \frac{p_d(\alpha)}{p_V(\alpha)}$$

(Odpovídá maximální věrohodnosti vůči posloupnosti $\mathcal{T}$ v případě, že p_d je definováno pomocí $\mathcal{T}$)

Omezený Boltzmannův stroj - učení

$\mathcal{E}(\vec{w})$ budeme minimalizovat pomocí gradientního sestupu, tj. budeme počítat posloupnost vektorů vah $\vec{w}^{(0)}, \vec{w}^{(1)}, \ldots$

- váhy v $\vec{w}^{(0)}$ jsou inicializovány náhodně blízko 0
- v t-tém kroku (zde $t = 1, 2, \ldots$) je $\vec{w}^{(t)}$ vypočteno takto:

$$w_{ji}^{(t)} = w_{ji}^{(t-1)} + \Delta w_{ji}^{(t)}$$

kde

$$\Delta w_{ji}^{(t)} = -\varepsilon(t) \cdot \frac{\partial \mathcal{E}}{\partial w_{ji}}(\vec{w}^{(t-1)})$$

je změna váhy w_{ji} v t-tém kroku a $0 < \varepsilon(t) \leq 1$ je rychlost učení v t-tém kroku.

Zbývá spočítat (odhadnout) $\frac{\partial \mathcal{E}}{\partial w_{ji}}(\vec{w})$.

Omezený Boltzmannův stroj - učení

Lze ukázat, že

$$\frac{\partial \mathcal{E}}{\partial w_{ji}} = -\left(\left\langle y_j y_i \right\rangle_{fixed} - \left\langle y_j^{(t^*)} y_i^{(t^*)} \right\rangle_{free}\right)$$

- $\left\langle y_j y_i \right\rangle_{fixed}$ je průměrná hodnota $y_j y_i$ po jednom kroku výpočtu za předpokladu, že hodnoty viditelných neuronů jsou fixovány dle rozložení p_d.
- $\left\langle y_j^{(t^*)} y_i^{(t^*)} \right\rangle_{free}$ je průměrná hodnota $y_j^{(t^*)} y_i^{(t^*)}$ v termální rovnováze bez fixace viditelných neuronů.

Problém: výpočet $\left\langle y_j^{(t^*)} y_i^{(t^*)} \right\rangle_{free}$ trvá dlouho (musíme opakovaně přivést stroj do termální rovnováhy).

$\left\langle y_j^{(t^*)} y_i^{(t^*)} \right\rangle_{free}$ se proto nahrazuje $\left\langle y_j y_i \right\rangle_{recon}$ což je průměrná hodnota $y_j^{(3)} y_i^{(3)}$ za předpokladu, že iniciální hodnoty viditelných neuronů jsou voleny dle p_d.

Omezený Boltzmannův stroj - učení

Tedy

$$\Delta w_{ji}^{(t)} = \varepsilon(t) \cdot \left(\left\langle y_j y_i \right\rangle_{fixed} - \left\langle y_j y_i \right\rangle_{recon} \right)$$

- $\left\langle y_j y_i \right\rangle_{fixed}$ se vypočte takto: Polož $\mathcal{Y} := 0$ a opakuj q krát:
  - fixuj náhodně hodnoty viditelných neuronů dle p_d
  - simuluj jeden krok výpočtu a přičti aktuální hodnotu $y_j y_i$ k $\mathcal{Y}$

  Pro vhodné q bude $\mathcal{Y}/q$ dobrým odhadem $\left\langle y_j y_i \right\rangle_{fixed}$
- $\left\langle y_j y_i \right\rangle_{recon}$ se vypočte takto: Polož $\mathcal{Y} := 0$ a opakuj q krát:
  - nastav náhodně hodnoty viditelných neuronů dle p_d
  - simuluj tři kroky výpočtu a přičti aktuální hodnotu $y_j y_i$ k $\mathcal{Y}$
    (tj. vypočti hodnoty skrytých neuronů, potom hodnoty viditelných (tzv. rekonstrukci vstupu) a potom hodnoty skrytých neuronů)

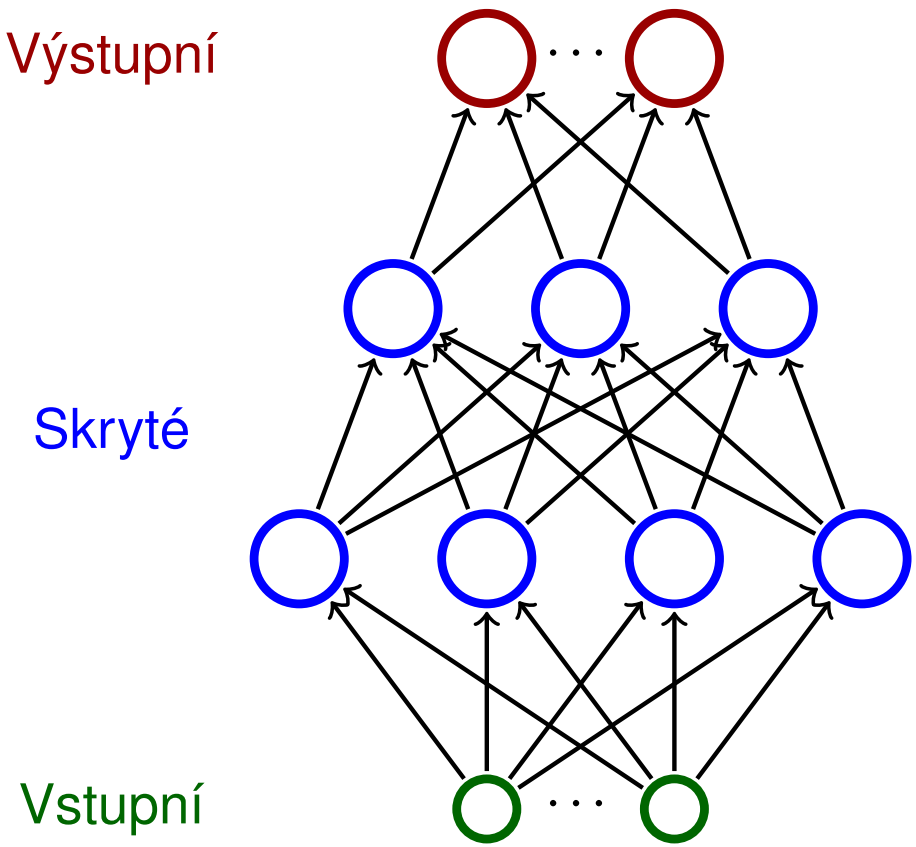
  Pro vhodné q bude $\mathcal{Y}/q$ dobrým odhadem $\left\langle y_j y_i \right\rangle_{recon}$

$\left\langle y_j y_i \right\rangle_{fixed}$ je možné počítat analyticky pro vhodně zadané p_d.

Hluboké sítě

Standardní vícevrstvá síť

Organizační dynamika:



- Neurony jsou rozděleny do **vrstev** (vstupní a výstupní vrstva, obecně několik skrytých vrstev)
- Vrstvy číslujeme od 0; vstupní vrstva je nultá
  - Např. třívrstvá síť se skládá z jedné vstupní, dvou skrytých a jedné výstupní vrstvy.
- Neurony v ℓ-té vrstvě jsou spojeny se všemi neurony ve vrstvě $\ell + 1$.

Hluboké sítě

Značení:

- Označme
  - X množinu vstupních neuronů
  - Y množinu výstupních neuronů
  - Z množinu všech neuronů (tedy $X, Y \subseteq Z$)
- jednotlivé neurony budeme značit indexy i, j apod.
- ξ_j je vnitřní potenciál neuronu j po skončení výpočtu
- y_j je stav (výstup) neuronu j po skončení výpočtu
- w_{ji} je váha spoje **od** neuronu i **do** neuronu j
- $j_{\leftarrow}$ je množina všech neuronů, **z nichž** vede spoj do j (zejména $0 \in j_{\leftarrow}$)
- $j^{\rightarrow}$ je množina všech neuronů, **do nichž** vede spoj z j

Hluboké sítě

Aktivní dynamika:

- vnitřní potenciál neuronu j:

$$\xi_j = \sum_{i \in j_{\leftarrow}} w_{ji} y_i$$

- aktivační funkce

$$\sigma = \frac{1}{1 + e^{-\xi}}$$

pro všechny neurony stejná!

- Stav nevstupního neuronu $j \in Z \setminus X$ po skončení výpočtu je

$$y_j = \sigma(\xi_j)$$

(y_j závisí na konfiguraci $\vec{w}$ a vstupu $\vec{x}$, proto budu občas psát $y_j(\vec{w}, \vec{x})$)

- Síť počítá funkci z $\mathbb{R}^{|X|}$ do $\mathbb{R}^{|Y|}$. Výpočet probíhá po vrstvách. Na začátku jsou hodnoty vstupních neuronů nastaveny na vstup sítě. V kroku ℓ jsou vyhodnoceny neurony z ℓ-té vrstvy.

Hluboké sítě – adaptivní dynamika

Tréninková posloupnost $\mathcal{T}$ vzorů tvaru

$$\left(\vec{x}_1, \vec{d}_1\right), \left(\vec{x}_2, \vec{d}_2\right), \ldots, \left(\vec{x}_p, \vec{d}_p\right)$$

kde každé $\vec{x}_k \in \{0,1\}^{|X|}$ je vstupní vektor a každé $\vec{d}_k \in \{0,1\}^{|Y|}$ je očekávaný výstup sítě. Pro každé $j \in Y$ označme d_{kj} očekávaný výstup neuronu j pro vstup $\vec{x}_k$ (vektor $\vec{d}_k$ lze tedy psát jako $\left(d_{kj}\right)_{j \in Y}$).

Chybová funkce

$$E(\vec{w}) = \sum_{k=1}^{p} E_k(\vec{w})$$

kde

$$E_k(\vec{w}) = \frac{1}{2} \sum_{j \in Y} \left(y_j(\vec{w}, \vec{x}_k) - d_{kj}\right)^2$$

Používají se i jiné funkce.

Proč hluboké sítě

... když jedna vrstva stačí k aproximaci libovolné rozumné funkce?

- Jedna vrstva je často značně neefektivní, tj. může vyžadovat obrovský počet skrytých neuronů pro reprezentaci dané funkce

  Výsledky z teorie Booleovských obvodů ukazují, že nutný počet neuronů může být exponenciální vzhledem k dimenzi vstupu

... ok, proč neučit hluboké sítě pomocí obyčejné zpětné propagace?

- Rychlost učení rapidně klesá s počtem vrstev
- Hluboké sítě mají tendenci se snadno přetrénovat

Vícevrstvá síť - mizející gradient

Pro každé w_{ji} máme

$$\frac{\partial E}{\partial w_{ji}} = \sum_{k=1}^{p} \frac{\partial E_k}{\partial w_{ji}}$$

kde pro každé $k = 1, \ldots, p$ platí

$$\frac{\partial E_k}{\partial w_{ji}} = \frac{\partial E_k}{\partial y_j} \cdot \sigma_j'(\xi_j) \cdot y_i$$

a pro každé $j \in Z \setminus X$ dostaneme

$$\frac{\partial E_k}{\partial y_j} = y_j - d_{kj} \qquad \text{pro } j \in Y$$

$$\frac{\partial E_k}{\partial y_j} = \sum_{r \in j^{\rightarrow}} \frac{\partial E_k}{\partial y_r} \cdot \sigma_r'(\xi_r) \cdot w_{rj} \qquad \text{pro } j \in Z \setminus (Y \cup X)$$

Pro standardní logistickou sigmoidu a váhy inicializované blízko 0 je $\sigma_r'(\xi_r) \cdot w_{rj}$ menší než 1 (pro velké váhy zase větší než 1).

Hluboké sítě – adaptivní dynamika

Předpokládejme k vrstvou síť. Označme

- W_i matici vah mezi vrstvami $i-1$ a i
- F_i funkci počítanou částí sítě s vrstvami $0, 1, \ldots, i$

  tedy F_1 je funkce počítaná jednovrstvou sítí skládající se ze vstupní a první vrstvy sítě, F_k je funkce celé sítě

Všimněte si, že pro každé i lze vrstvy $i-1$ a i společně s maticí W_i chápat jako omezený Boltzmannův stroj B_i (předpokládáme $T = 1$)

Učení ve dvou fázích:

- předtrénování bez učitele: Postupně pro každé $i = 1, \ldots, k$ trénuj OBS B_i na náhodně volených vstupech z posloupnosti

  $$F_{i-1}(\vec{x}_1), \ldots, F_{i-1}(\vec{x}_p)$$

  pomocí algoritmu pro učení OBS (zde $F_0(\vec{x}_i) = \vec{x}_i$)

  (tedy B_i se trénuje na tréninkových vzorech transformovaných vrstvami $0, \ldots, i-1$)
- doladění sítě s učitelem např. pomocí zpětné propagace

Hluboké sítě

Po první fázi dostaneme k vrstvou síť, která reprezentuje rozložení na datech. Z tohoto rozložení lze samplovat takto:

- přiveď nejvyšší OBS do termální rovnováhy (to dá hodnoty neuronů v nejvyšších dvou vrstvách)
- propaguj hodnoty do nižších vrstev (tj. proveď jeden krok aktualizace stavů mezilehlých OBS)
- stav neuronů v nejspodnější vrstvě potom bude představovat vzorek dat; pravděpodobnost s jakou se tam objeví konkrétní stav je pravděpodností onoho stavu v rozložení reprezentovaném sítí

Hluboké sítě - klasifikace

Předpokládejme, že každý vstup patří do jedné ze dvou tříd. Chceme vstupy klasifikovat pomocí vícevrstvé sítě.

Vícevrstvou síť lze trénovat pomocí zpětné propagace. Ta je silně závislá na vhodné inicializaci, často hrozí dosažení mělkého lokálního minima apod. Dobře fungující síť si vyvine systém extraktorů vlastností (tj. každý neuron reaguje na nějakou vlastnost vstupu). Jak toho dosáhnout?

- Natrénuj hlubokou síť na datech (v této fázi ignoruj příslušnost do tříd)
- Uvažuj výslednou síť jako obyčejnou vícevrstvou síť, tj. zaměň dynamiku Boltzmannova stroje za sigmoidální aktivační funkce a obvyklé vyhodnocení zdola nahoru
- přidej výstupní vrstvu s jedním neuronem
- doladď síť pomocí zpětné propagace (malá rychlost učení pro skryté vrstvy, velká pro výstupní vrstvu): Pro vstupy z jedné třídy uvažujeme očekávaný výstup 1 pro ostatní 0.

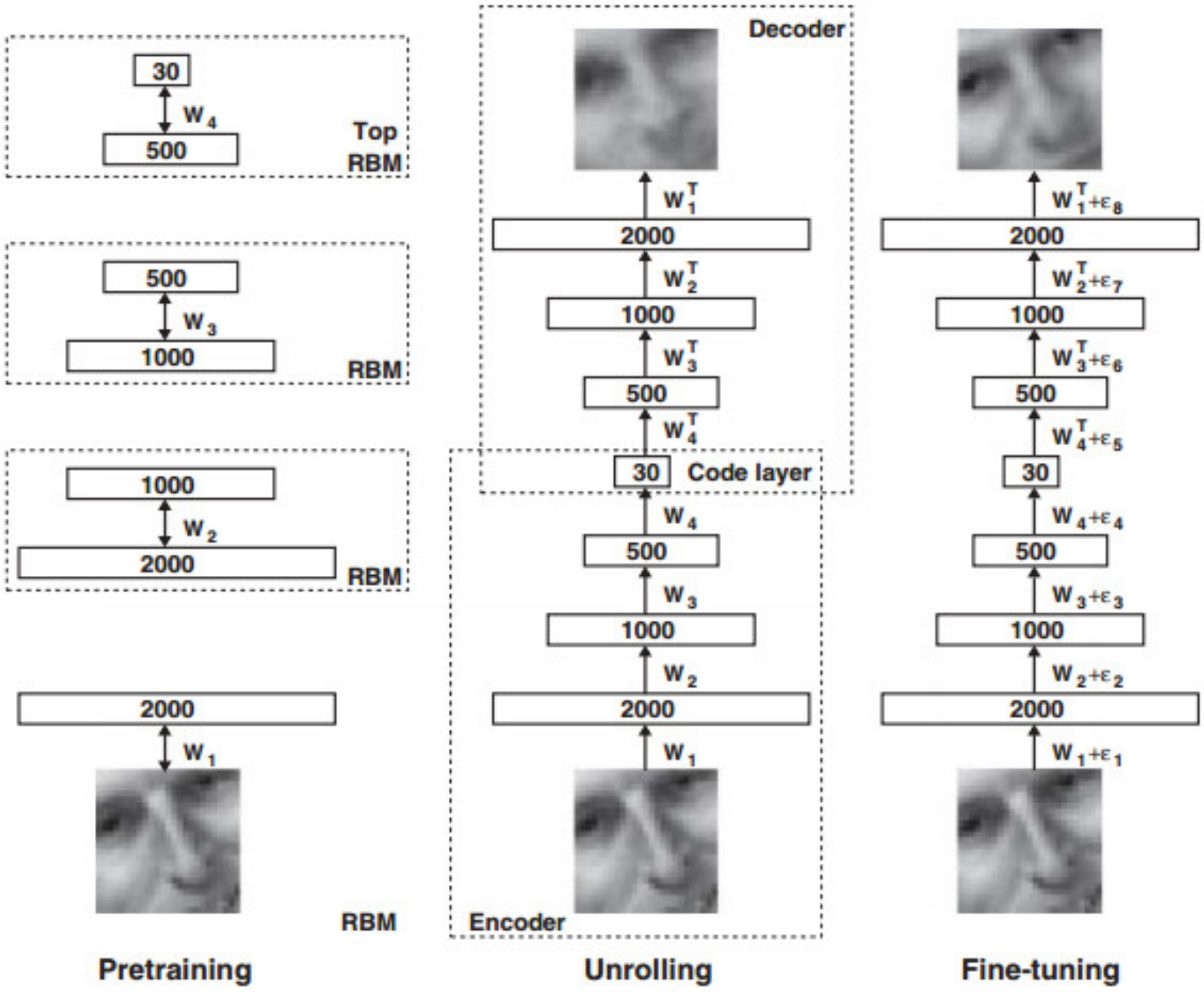
Aplikace – redukce dimenze

- Redukce dimenze dat: Tedy zobrazení R z $\mathbb{R}^n$ do $\mathbb{R}^m$ takové, že
  - $m < n$,
  - pro každý vzor $\vec{x}$ platí, že $\vec{x}$ je možné "rekonstruovat" z $R(\vec{x})$.
- Standardní metoda PCA (exituje mnoho lineárních i nelineárních variant)

Rekonstrukce – PCA

1024 pixelů komprimováno do 100 dimenzí (tj. 100 čísel).

Redukce dimenze pomocí hlubokých sítí



Obrázky – Předtrénování

- **Data:** 165 600 černobílých obrázků o rozměrech 25×25, střední intenzita bodu 0, rozptyl 1.

  Obdrženo z Olivetti Faces databáze obrázků 64×64 standardními úpravami.
- 103 500 tréninková sada, 20 700 validační, 41 400 testovací
- **Síť:** Struktura sítě 2000-100-500-30, trénink pomocí postupného vrstvení RBM.

Poznámky:

Trénink nejnižší skryté vrsty (2000 neuronů): Hodnoty pixelů "rozmlženy" Gaussovským šumem, rychlost učení nízká: 0.001, počítáno 200 iterací

Ve všech vrstvách kromě nejvyšší jsou hodnoty skrytých neuronů při tréninku binární (v nejvyšší jsou hodnoty vypočteny ze sigmoidální pravděpodobnosti přidáním šumu)

Hodnoty viditelných neuronů jsou při tréninku reálné z intervalu $[0, 1]$ (zde je mírná odchylka od našeho algoritmu)

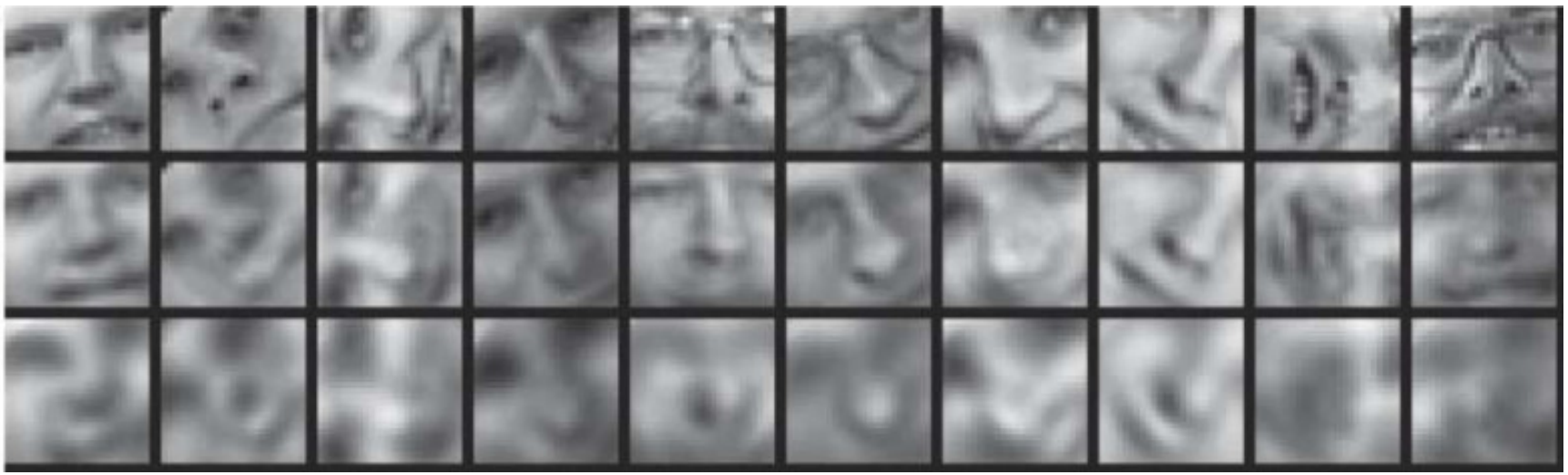
Obrázky – Doladění

- Stochastické aktivace nahrazeny deterministickými
  Tj. hodnota skrytých neuronů není výsledkem náhodného pokusu, ale přímo výpočtu sigmoid udávajících pravděpodobnost.
- Zpětná propagace
- Chybová funkce cross-entropy

$$-\sum_i p_i \ln \hat{p}_i - \sum_i (1 - p_i) \ln(1 - \hat{p}_i)$$

kde p_i je intenzita i-tého pixelu ve vstupu a $\hat{p}_i$ v rekonstrukci.

Výsledky



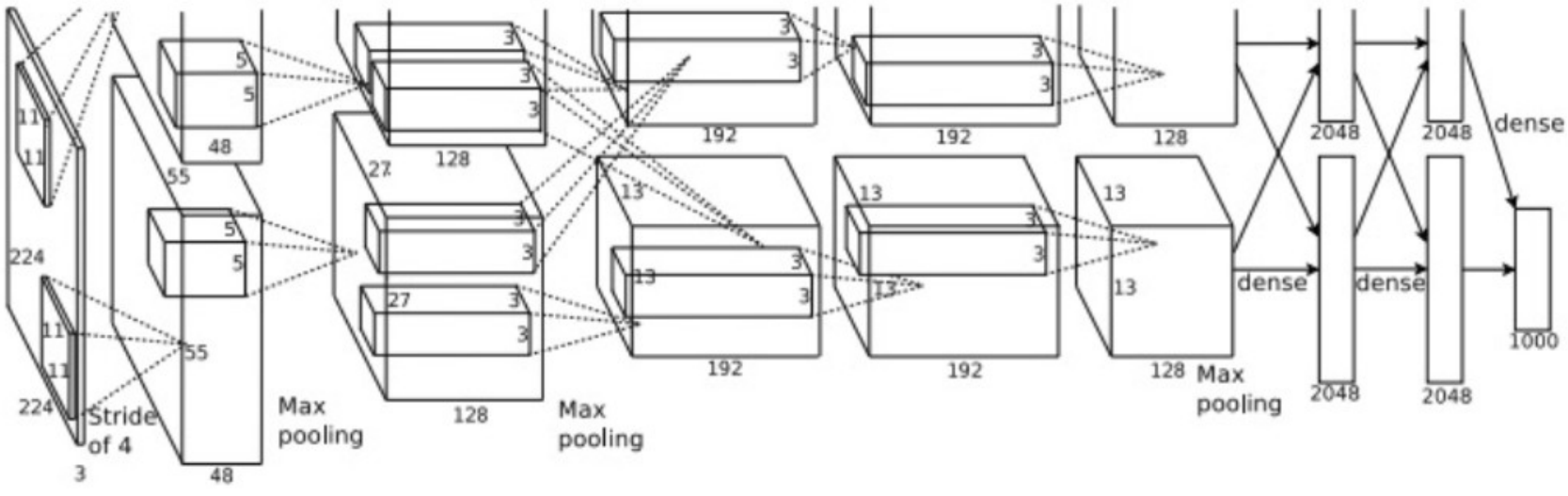
1. Originál
2. Rekonstrukce hlubokou sítí (redukce na 30-dim)
3. Rekonstrukce PCA (redukce na 30-dim)

Konvoluční sítě

Zbytek přednášky je založen na nové online knize Neural Networks and Deep Learning, autor Michael Nielsen.
`http://neuralnetworksanddeeplearning.com/index.html`

- Konvoluční sítě jsou v současné době nejlepší metodou pro klasifikaci obrázků z databáze ImageNet.
- Jejich počátky sahají do 90. let – síť LeNet-5

  Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 1998



Konvoluční sítě vs MNIST (ilustrace)

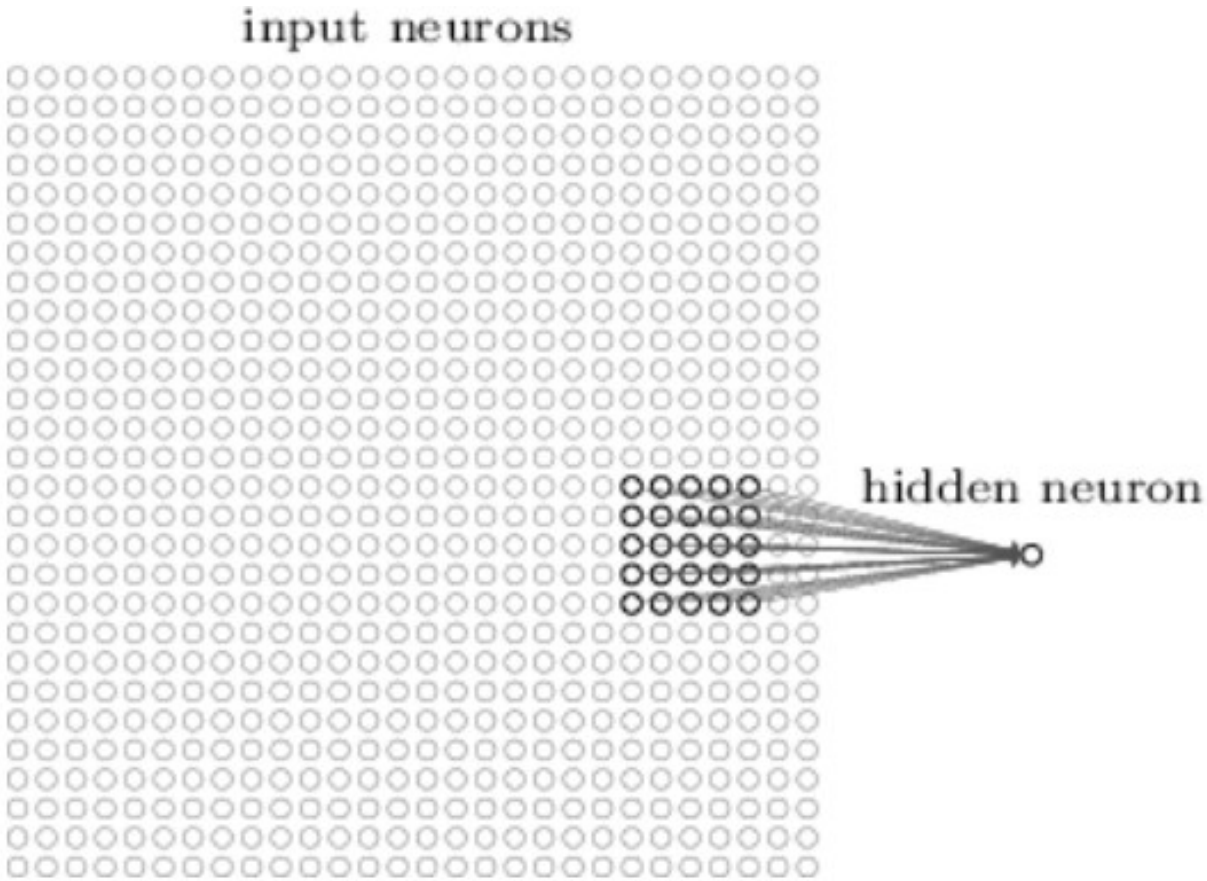
MNIST:

- Databáze (označkovaných) obrázků rukou psaných číslic: 60 000 tréninkových, 10 000 testovacích.
- Dimenze obrázků je 28 x 28 pixelů, jsou vycentrované do "těžiště" jednotlivých pixelů a normalizované na fixní velikost
- Více na `http://yann.lecun.com/exdb/mnist/`

Databáze se používá jako standardní benchmark v mnoha publikacích o rozpoznávání vzorů (nejen) pomocí neuronových sítí.

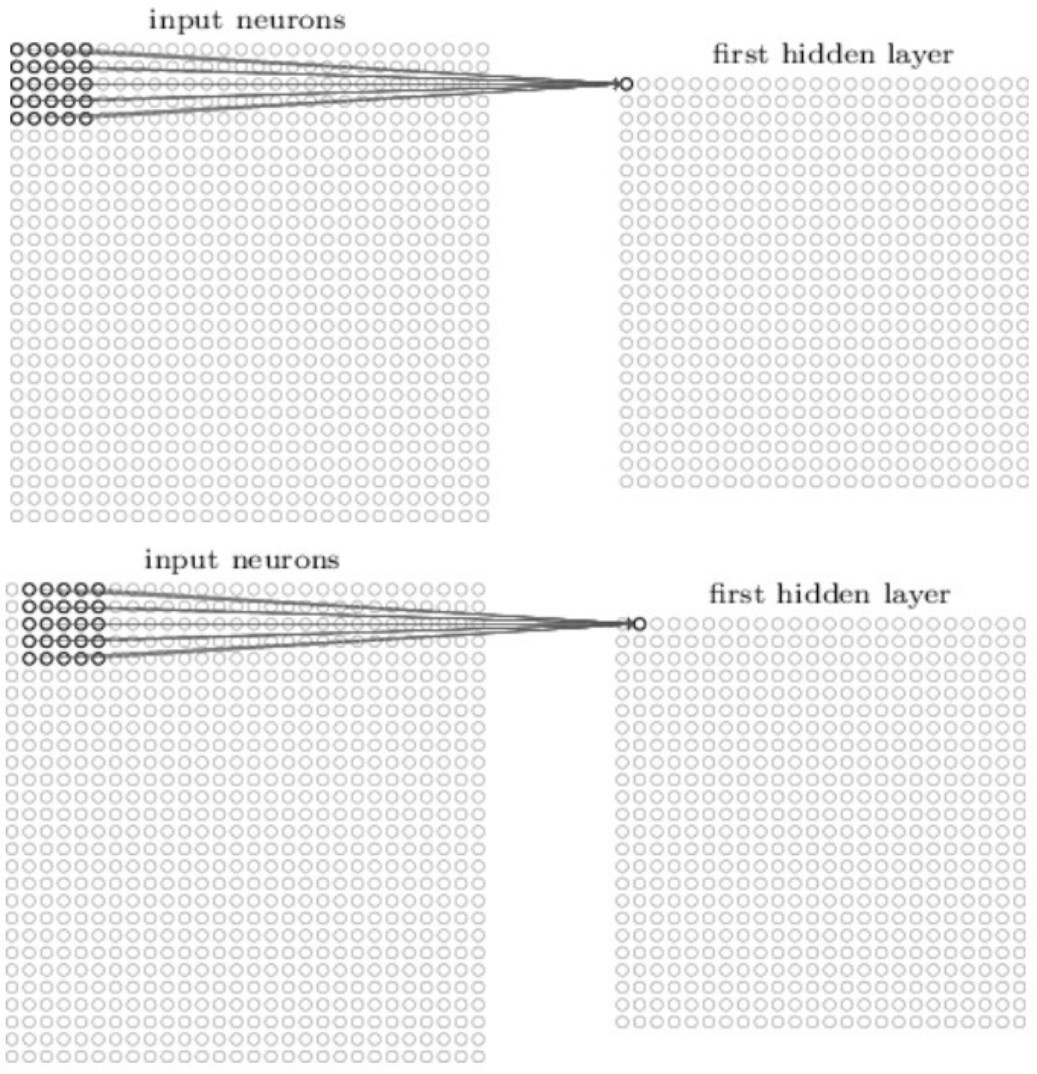
Lze porovnávat přesnost klasifikace různých metod.

Konvoluční sítě - local receptive fields



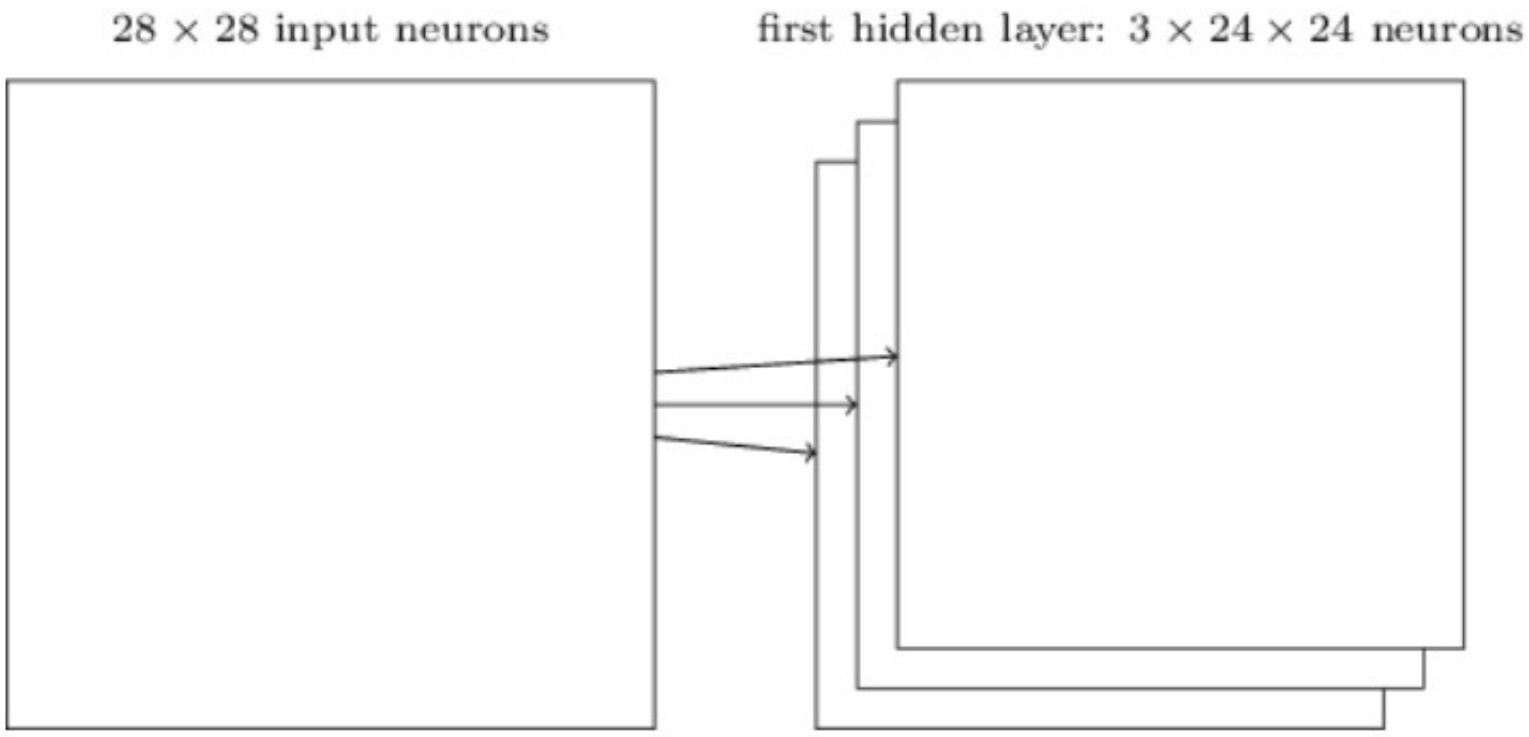
Každý neuron je spojen s polem 5×5 neuronů v nižší vrstvě (toto pole se nazývá *local receptive field*)
Neuron je "standardní": Počítá vážený součet vstupů, na výsledek aplikuje aktivační funkci.

Konvoluční sítě - stride length



Velikost posun vedle sebe ležících polí je *stride length*.
Celá skupina neuronů (tj. všechny polohy) je *feature map*.
Všechny neurony z dané feature map sdílí váhy a bias!

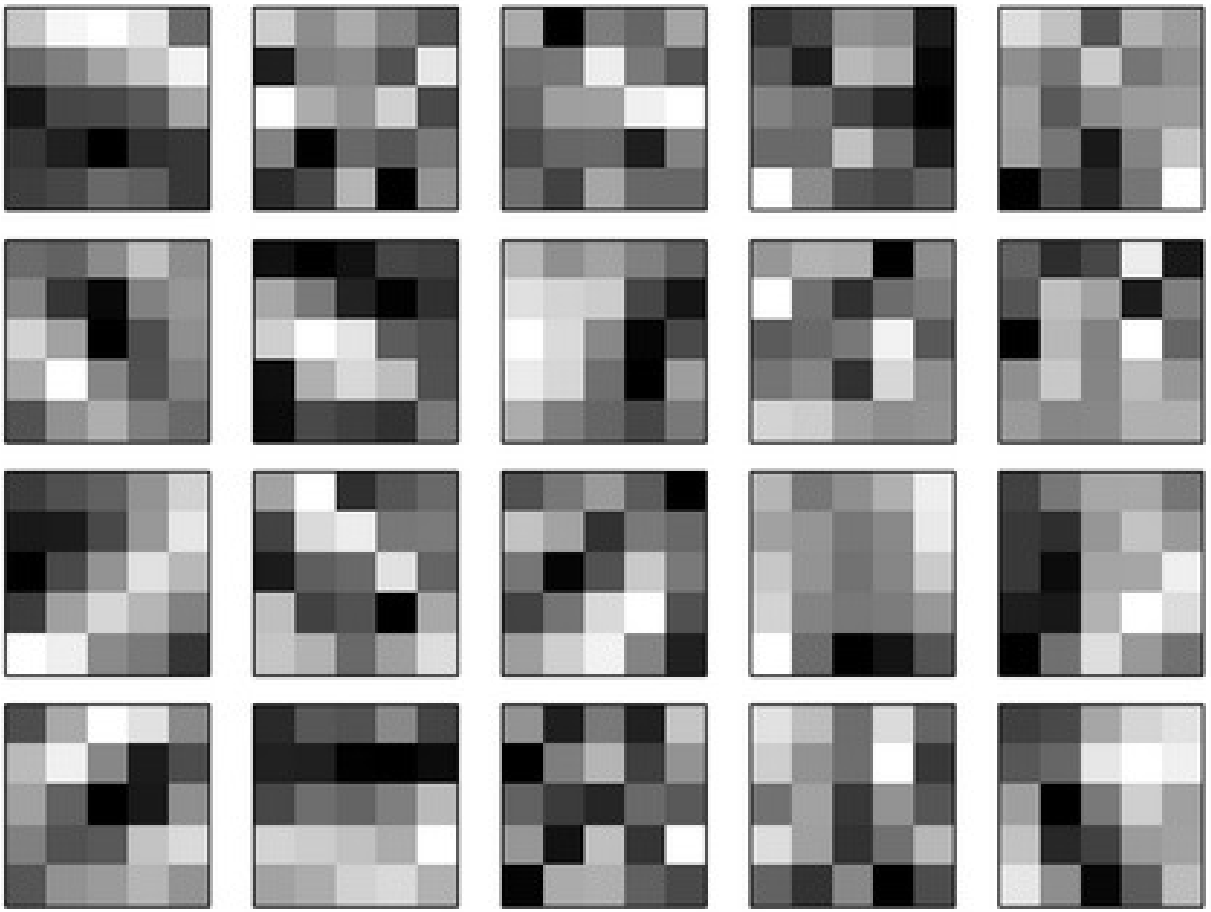
Feature maps



Každá feature map "reprezentuje" nějakou vlastnost vstupu.
(v libovolné poloze ve vstupu)

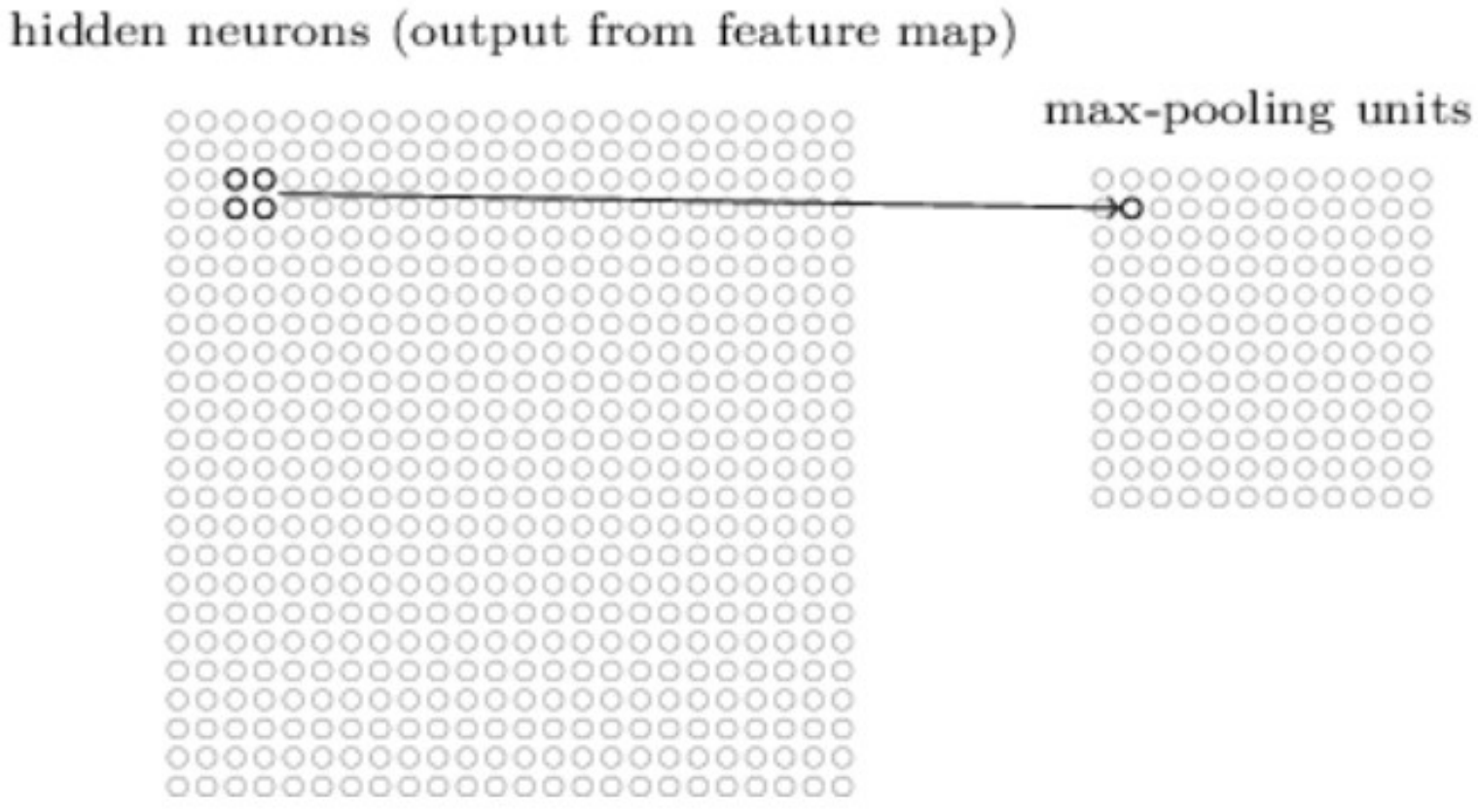
Typicky se uvažuje několik feature maps v jedné vrstvě.

Natrénované feature maps



(zde 20 feature maps, receptive fields 5×5)

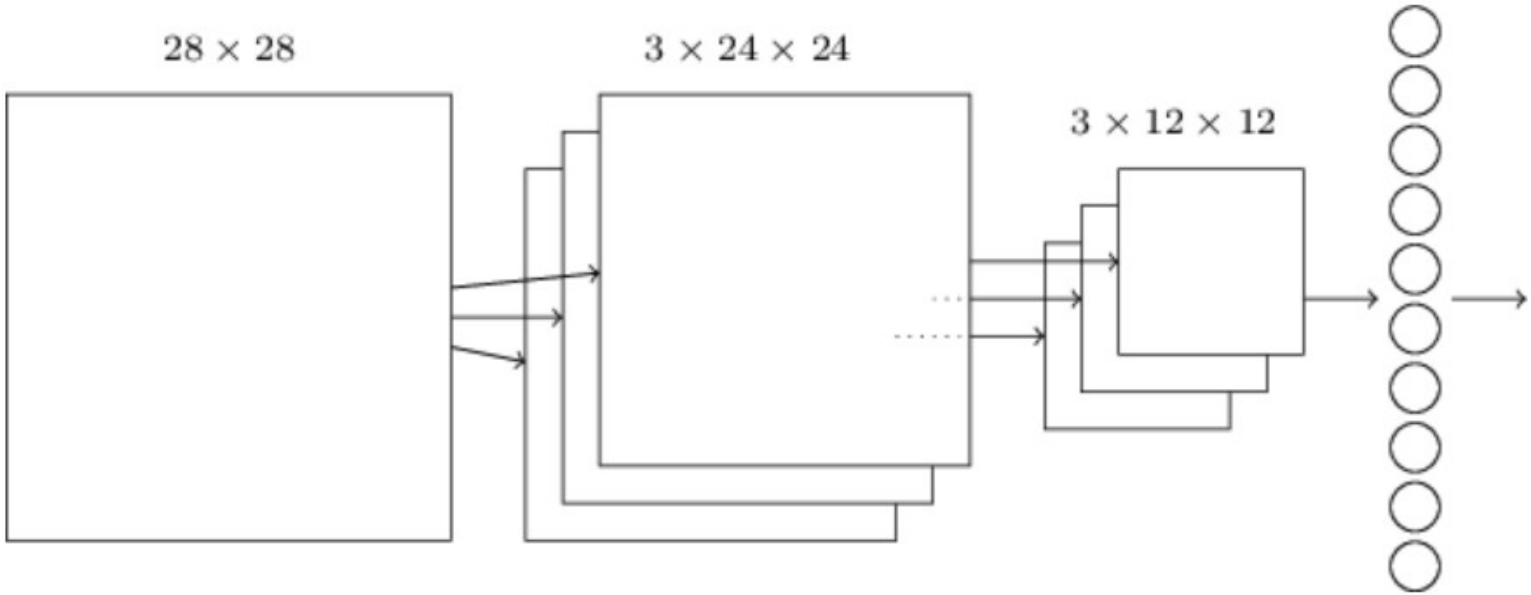
Pooling



Neuron v pooling vrstvě počítá funkci svého pole:

- **Max-pooling** : maximum hodnot neuronů
- **L2-pooling** : odmocnina ze součtu druhých mocnin hodnot neuronů
- **Average-pooling** : aritmetický průměr
- ...

Jednoduchá konvoluční síť

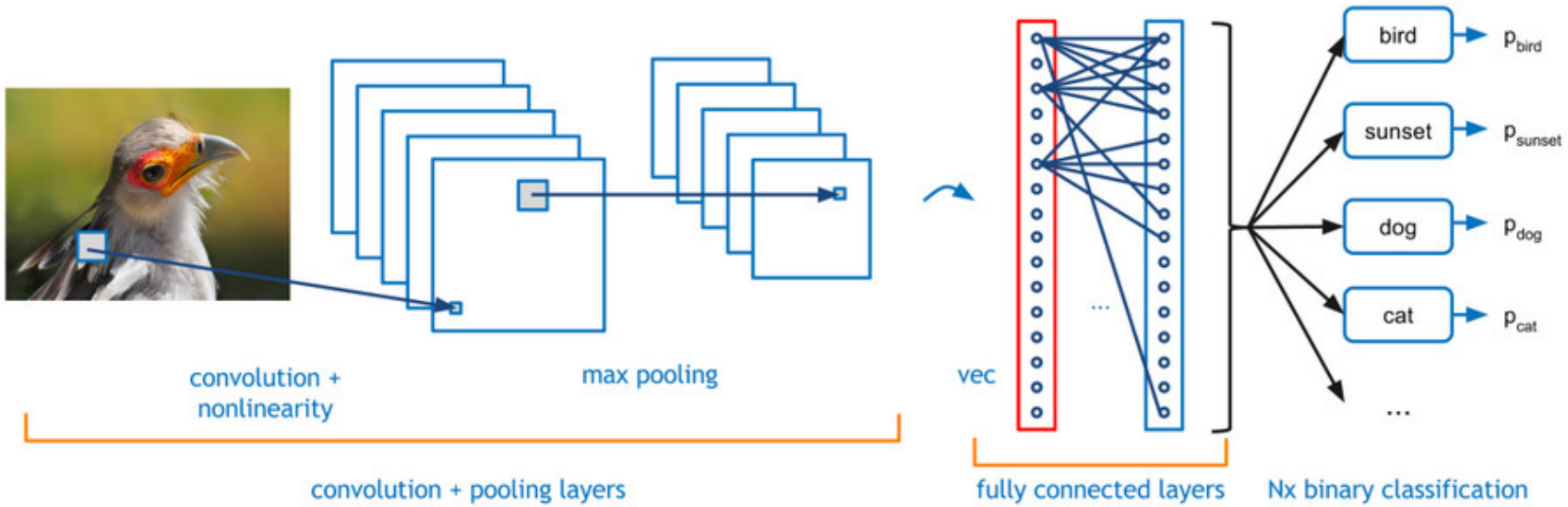


28×28 obraz na vstupu, 3 feature maps, každá má svůj max-pooling (pole 5×5, stride = 1), 10 výstupních neuronů.

Každý neuron ve výstupní vrstvě bere vstup z každého neuronu v pooling layer.

Trénuje se typicky zpětnou propagací, kterou lze snadno upravit pro konvoluční síť (přepočítají se derivace).

Konvoluční síť



Jednoduchá konvoluční síť vs MNIST

dvě konvoluční-pooling vrstvy, jedna 20, druhá 40 feature maps, dvě úplně propojené vrstvy (1000-1000), výtupní vrstva (10)

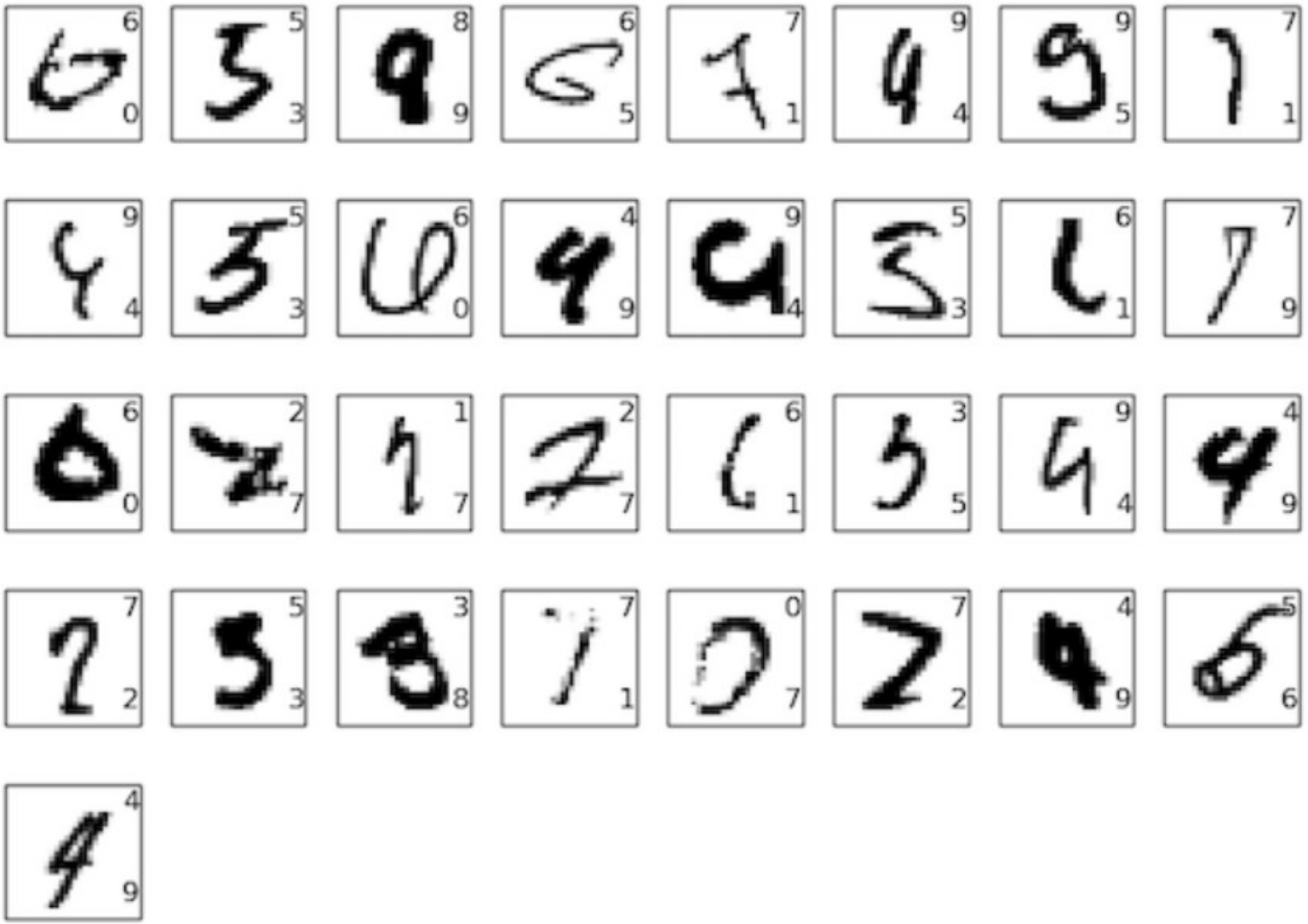
- Aktivační funkce feature maps a plně propojených: ReLU
- max-pooling
- výstupní vrstva: soft-max
- Chybová fce: negative log-likelihood
- V podstatě stochastický gradientní sestup (ve skutečnosti mini-batch velikosti 10)
- rychlost učení 0.03
- L2 regularizace s "váhou" $\lambda = 0.1$ + dropout s pravd. $1/2$
- trénink 40 epoch (tj. vše se projde 40 krát)
- Datová sada upravena posouváním o jeden pixel do libovolného směru.
- Nakonec použito hlasování pěti sítí.

Konvoluční síť - Theano

```
>>> net = Network([
        ConvPoolLayer(image_shape=(mini_batch_size, 1, 28, 28),
                      filter_shape=(20, 1, 5, 5),
                      poolsize=(2, 2),
                      activation_fn=ReLU),
        ConvPoolLayer(image_shape=(mini_batch_size, 20, 12, 12),
                      filter_shape=(40, 20, 5, 5),
                      poolsize=(2, 2),
                      activation_fn=ReLU),
        FullyConnectedLayer(
            n_in=40*4*4, n_out=1000, activation_fn=ReLU, p_dropout=0.5),
        FullyConnectedLayer(
            n_in=1000, n_out=1000, activation_fn=ReLU, p_dropout=0.5),
        SoftmaxLayer(n_in=1000, n_out=10, p_dropout=0.5)],
        mini_batch_size)
>>> net.SGD(expanded_training_data, 40, mini_batch_size, 0.03,
            validation_data, test_data)
```

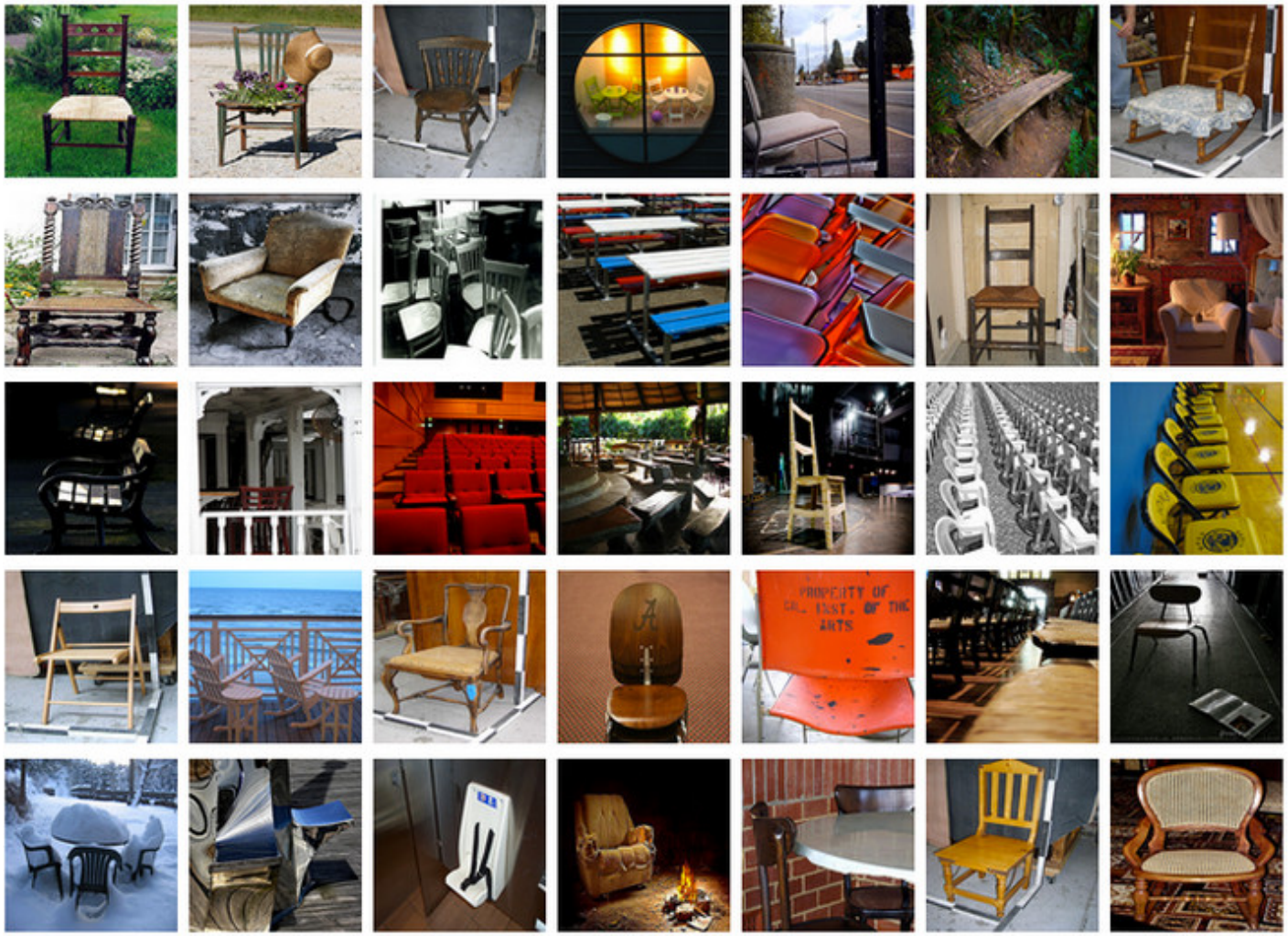
MNIST

Z 10 000 obrázků databáze MNIST, pouze těchto 33 bylo špatně klasifikováno:



Složitější konvoluční síť

Konvoluční sítě byly použity pro klasifikaci obrázků z databáze ImageNet (16 milionů barevných obrázků, 20 tisíc kategorií)



ImageNet Large-Scale Visual Recognition Challenge (ILSVRC)

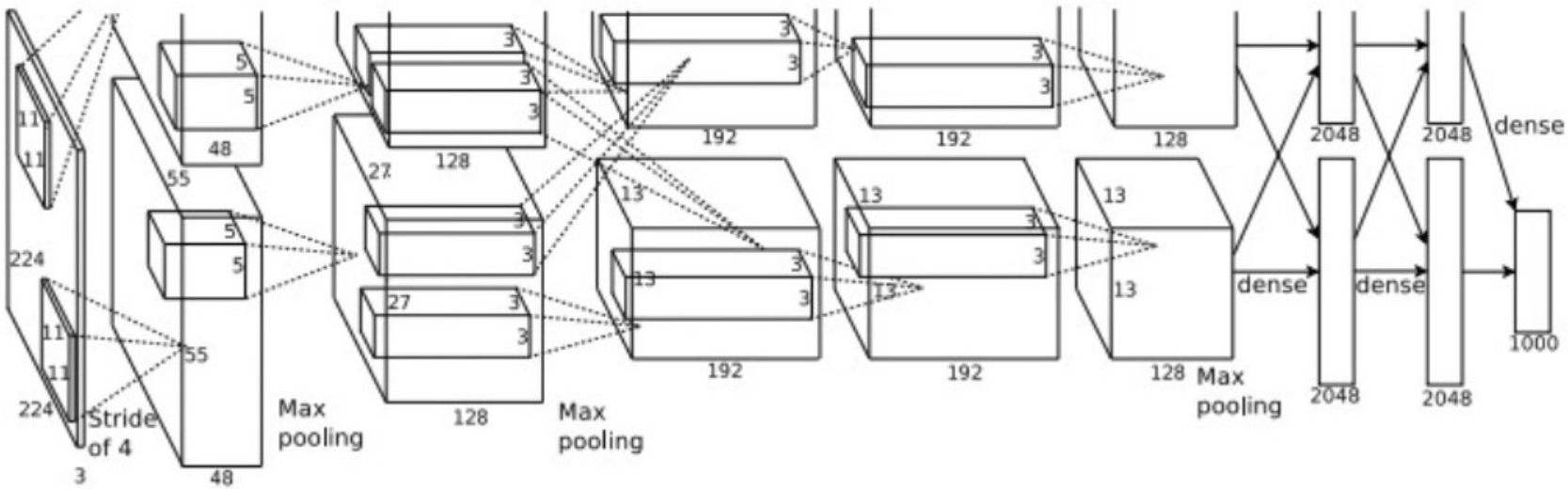
Soutěž v klasifikaci nad podmnožinou obrázků z ImageNet.

V roce 2012: tréninková množina 1.2 milionů obrázků z 1000 kategorií. Validační množina 50 000, testovací 150 000.

Mnoho obrázků obsahuje více objektů → za správnou odpověď se typicky považuje, když je správná kategorie mezi pěti, jimž síť přiřadí nejvyšší pravděpodobnost (top-5 kritérium).

KSH síť

ImageNet classification with deep convolutional neural networks, by Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton (2012).



Trénováno na dvou GPUs (NVIDIA GeForce GTX 580)

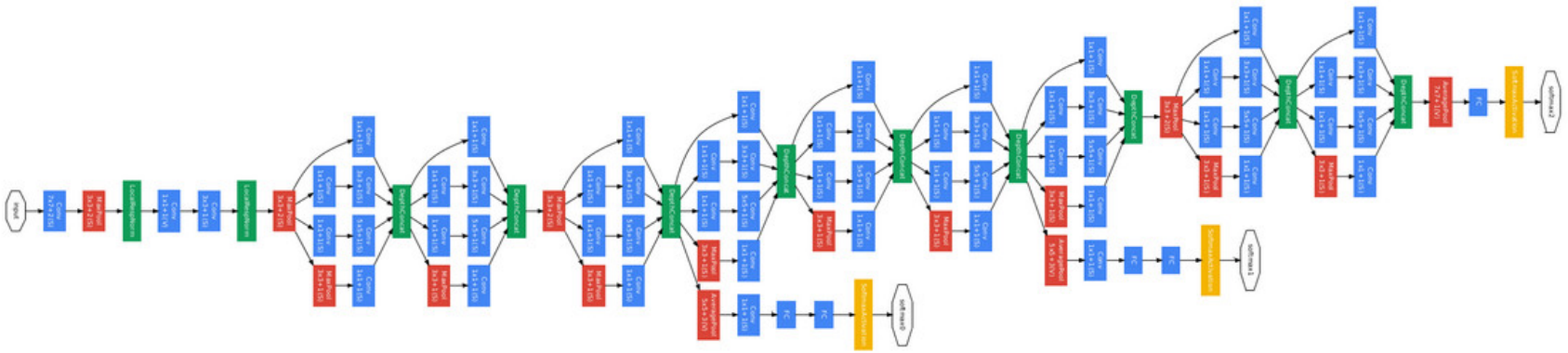
Výsledky:

- úspěšnost 84.7 procenta v top-5 (druhý nejlepší alg. 73.8 procent)
- 63.3 procenta v "absolutní" klasifikaci

ILSVRC 2014

Stejná sada jako v roce 2012, top-5 kritérium.

Síť GoogLeNet: hluboká konvoluční síť, 22 vrstev



Výsledky:

- 93.33 procent top-5

Jsou lidé stále schopni obstát proti tomuhle?

Superhuman GoogLeNet?!

Andrej Karpathy: ...the task of labeling images with 5 out of 1000 categories quickly turned out to be extremely challenging, even for some friends in the lab who have been working on ILSVRC and its classes for a while. First we thought we would put it up on [Amazon Mechanical Turk]. Then we thought we could recruit paid undergrads. Then I organized a labeling party of intense labeling effort only among the (expert labelers) in our lab. Then I developed a modified interface that used GoogLeNet predictions to prune the number of categories from 1000 to only about 100. It was still too hard - people kept missing categories and getting up to ranges of 13-15% error rates. In the end I realized that to get anywhere competitively close to GoogLeNet, it was most efficient if I sat down and went through the painfully long training process and the subsequent careful annotation process myself... The labeling happened at a rate of about 1 per minute, but this decreased over time... Some images are easily recognized, while some images (such as those of fine-grained breeds of dogs, birds, or monkeys) can require multiple minutes of concentrated effort. I became very good at identifying breeds of dogs... Based on the sample of images I worked on, the GoogLeNet classification error turned out to be 6.8%... My own error in the end turned out to be 5.1%, approximately 1.7% better.