ITMO

Neural Networks and Deep Learning

Computer Vision

Outline

- Biological neural network
- Artificial neuron
- Artificial neural network
- Neural network training

Neural Networks and Deep Learning

Biological neural network

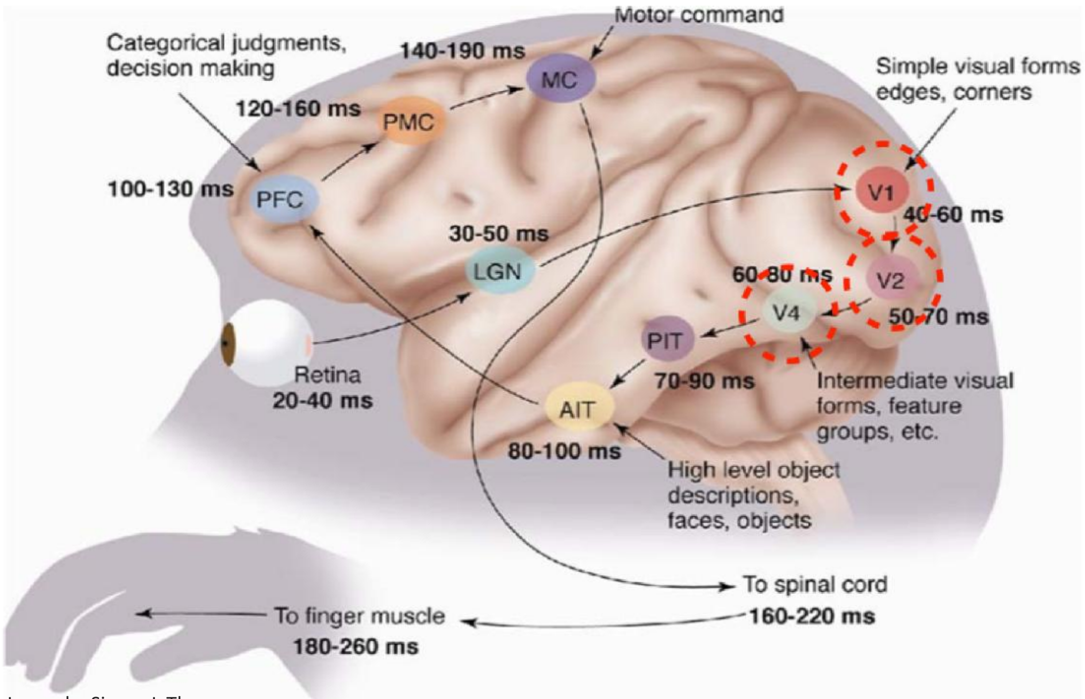
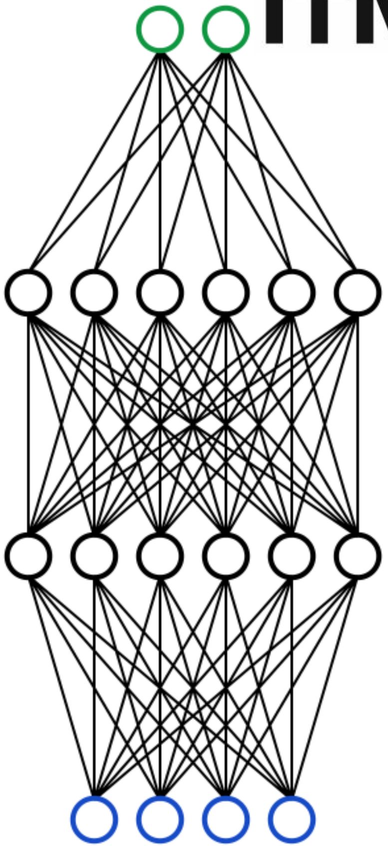


Image by Simon J. Thorpe



The number of neurons in the human brain is estimated to be around $10^{10} - 10^{11}$

Biological neural network

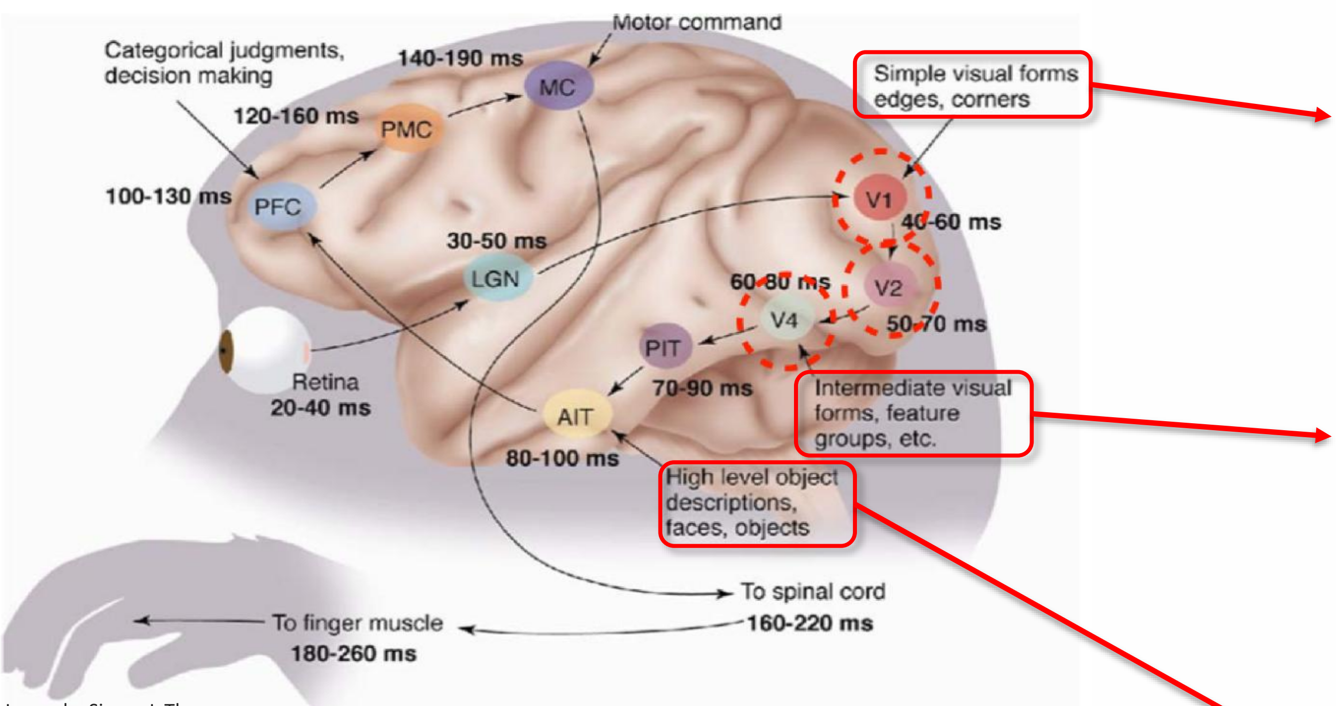
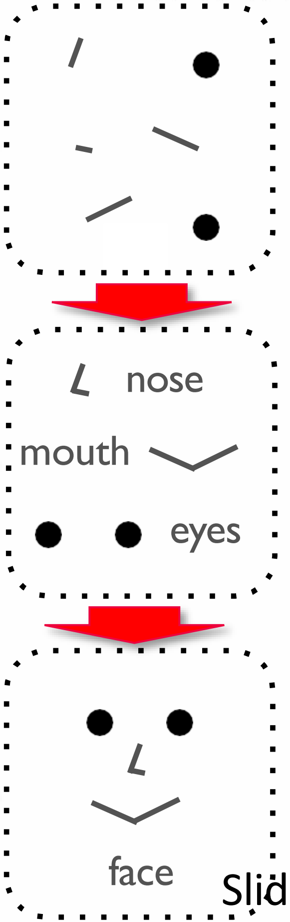


Image by Simon J. Thorpe



The number of neurons in the human brain is estimated to be around $10^{10} - 10^{11}$

Biological neural network

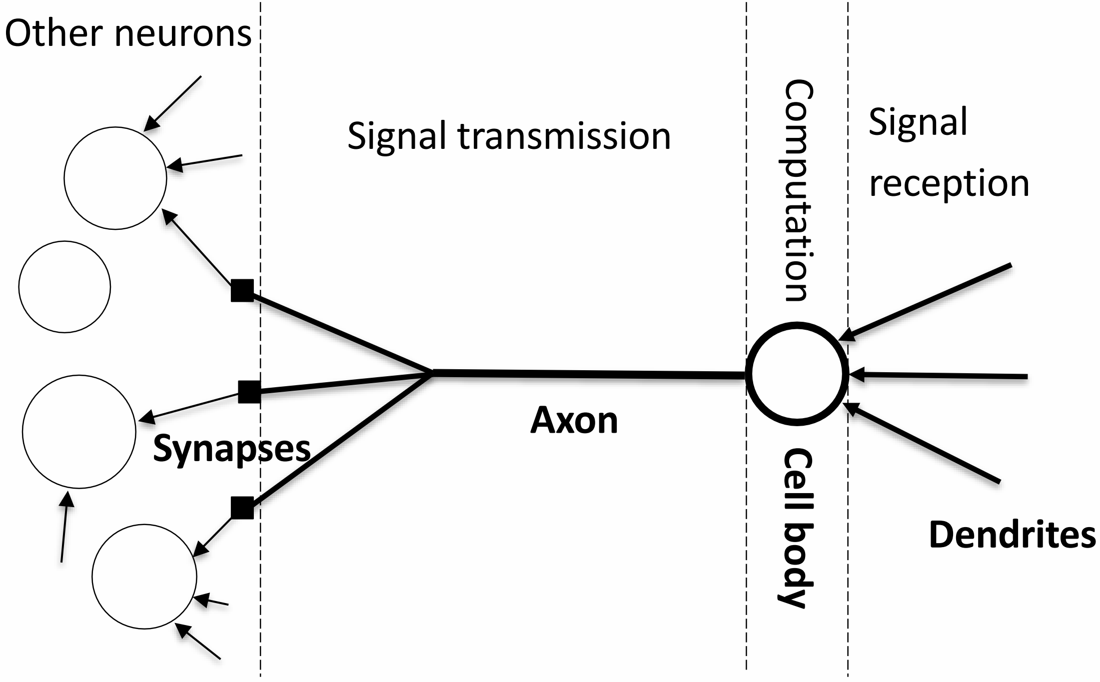


Image by Aapo Hyvärinen, Jarmo Hurri, Patrick O. Hoyer

- Neurons receive information from other neurons through their dendrites
- Neurons process the information in their cell body (soma)
- Neurons send information through an axon
- The connection between the axon branches and other neurons' dendrites are called synapses

Biological neuron

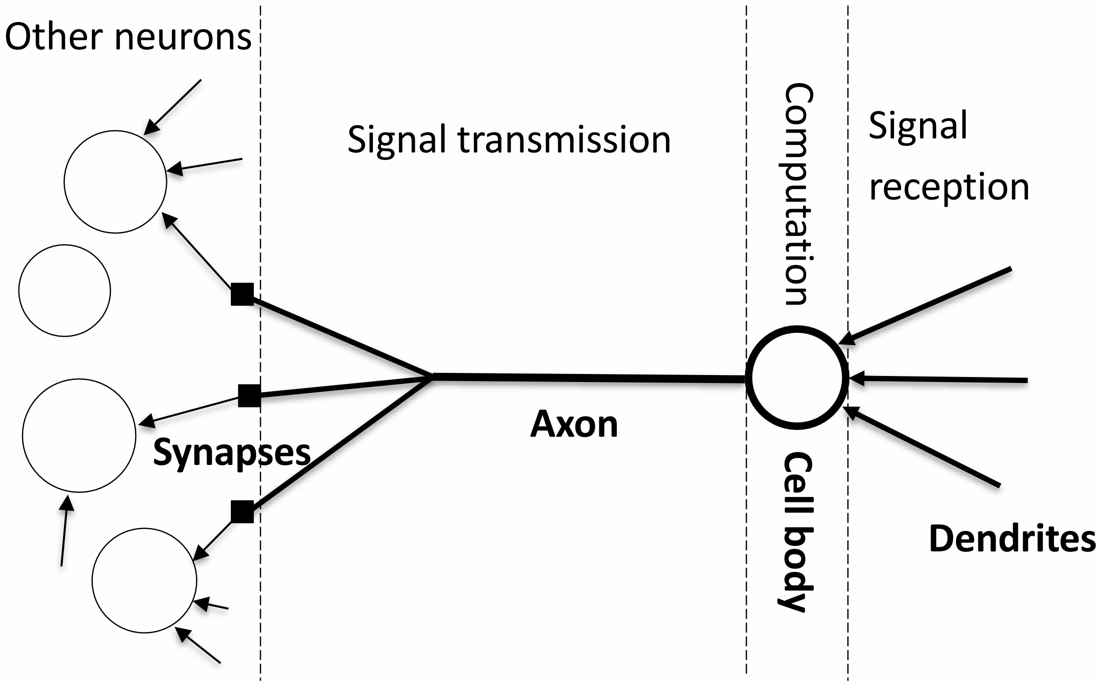


Image by Aapo Hyvärinen, Jarmo Hurri, Patrick O. Hoyer

An action potential is an electrical impulse that travels through the axon

- this is how neurons communicate
- neuron generates a spike in the electric potential of the axon
- an action potential is generated at neuron only if the pattern of spikes it receives from other neurons is above some threshold

Biological neuron

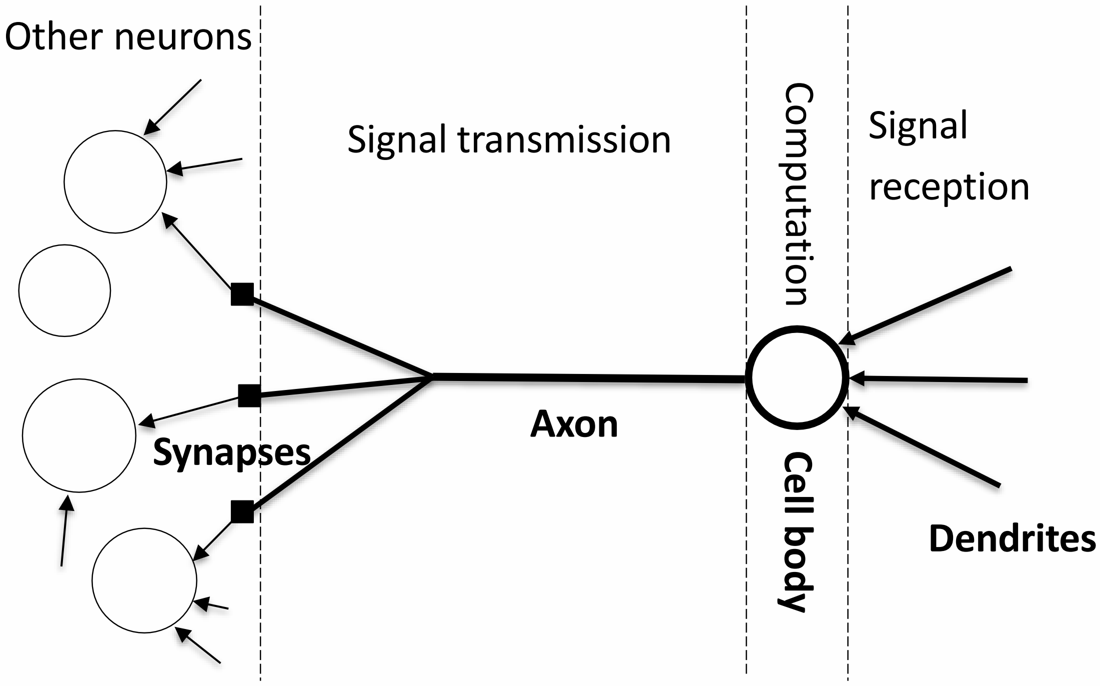


Image by Aapo Hyvärinen, Jarmo Hurri, Patrick O. Hoyer

Neurons generate several spikes every seconds

- the neuron activity is characterized by its firing rate which is the frequency of the spikes
- neurons have a spontaneous firing rate, so they always fire a little bit, but they will increase the rate if receive the right stimulus

Firing rates of different input neurons are combined and influence the firing rate of other neurons

- depending on the dendrite and axon, a neuron can either work to increase (excite) or descrease (inhibit) the firing rate of another neuron

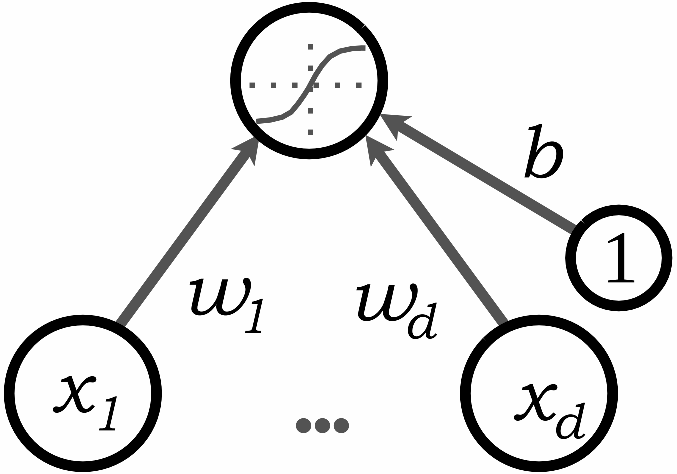
Artificial neuron

Neuron pre-activation (or input activation):

- $a(\boldsymbol{x}) = b + \sum_i w_i x_i = b + \boldsymbol{w}^T \boldsymbol{x}$

Neuron (output) activation:

- $\mathrm{h}(\boldsymbol{x}) = g\big(a(\boldsymbol{x})\big) = g(b + \sum_i w_i x_i)$



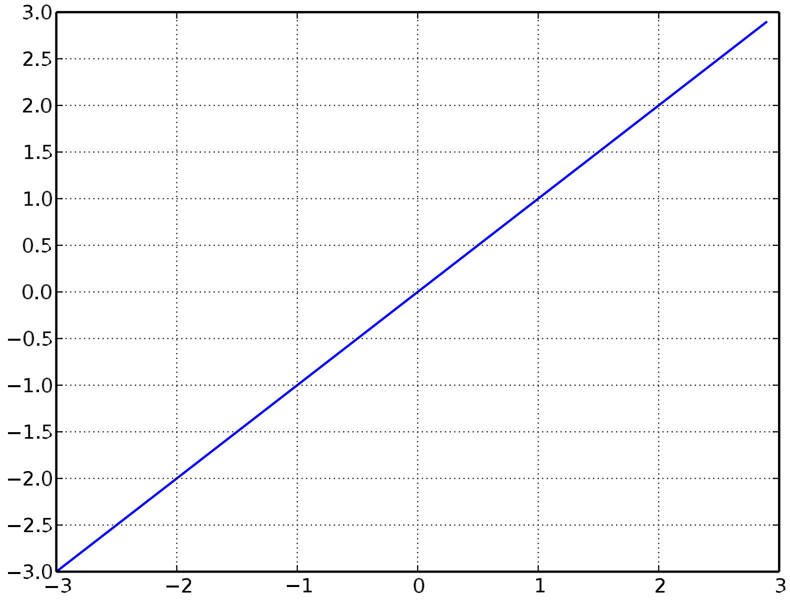
where

- $\boldsymbol{w}$ are connection weights
- b is neuron bias
- $g\big(a(\boldsymbol{x})\big)$ is the activation function

Activation function

Linear activation function

- Performs no input squashing
- Has infinite values

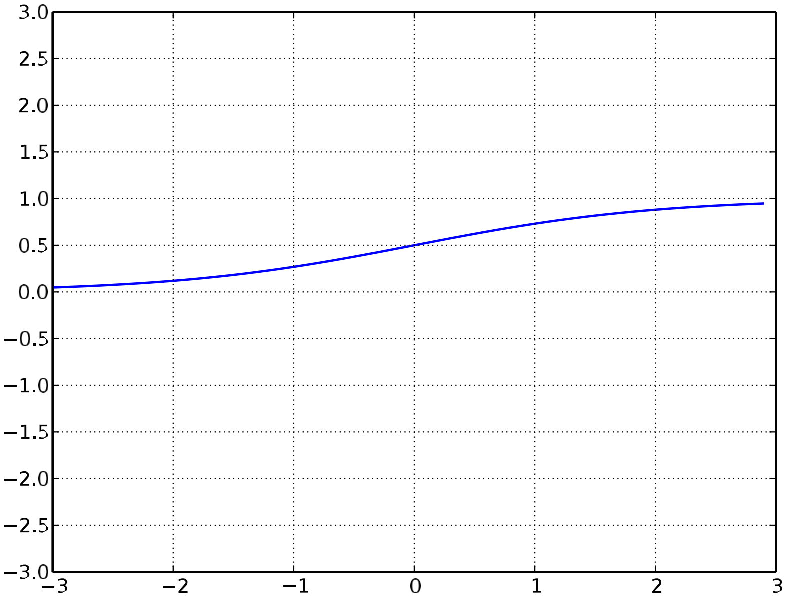


$$g(a) = a$$

Activation function

Sigmoid activation function

- Squashes the neuron pre-activation value to [0, 1] range
- Has finite values
- Always positive
- Bounded
- Strictly increasing

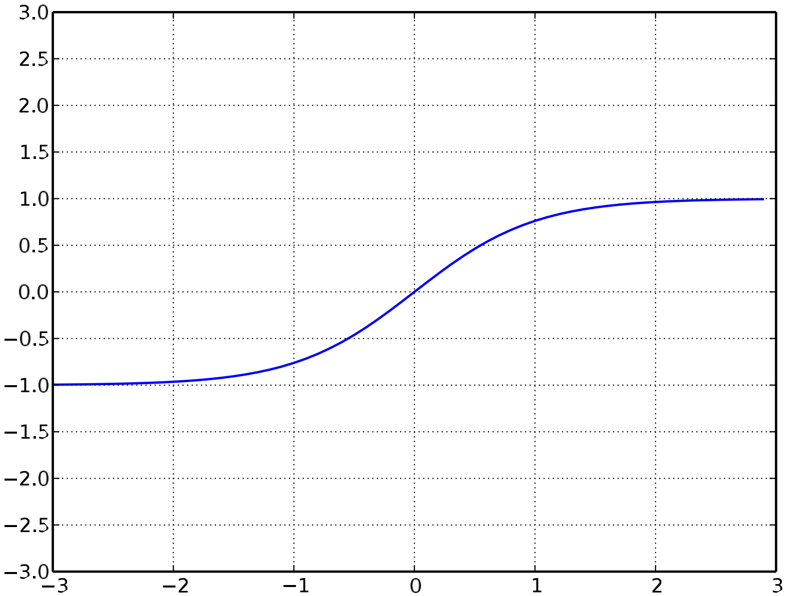


$$g(a) = sigm(a) = \frac{1}{1 + e^{-a}}$$

Activation function

Hyperbolic tangent activation function

- Squashes the neuron pre-activation value to [-1, 1] range
- Has finite values
- Can be positive or negative
- Bounded
- Strictly increasing

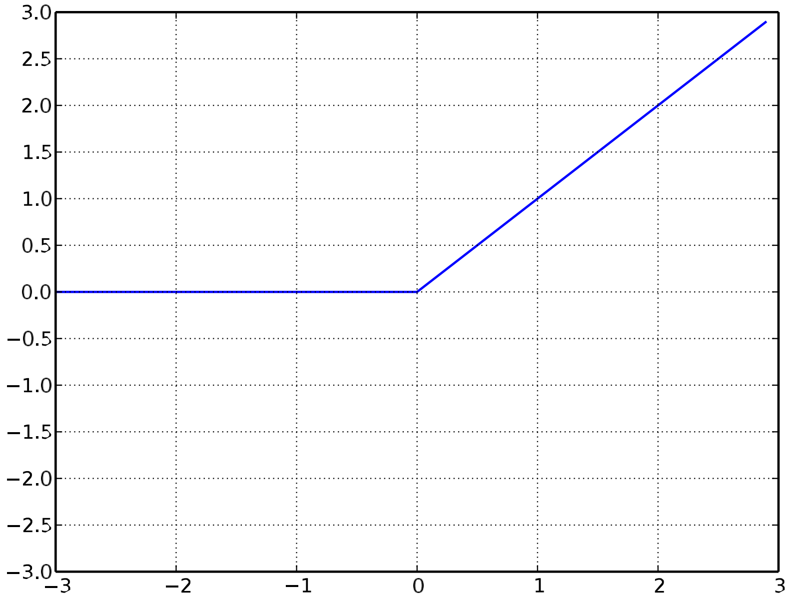


$$g(a) = tanh(a) = \frac{e^{a} - e^{-a}}{e^{a} + e^{-a}} = \frac{e^{2a} - 1}{e^{2a} + 1}$$

Activation function

Rectified linear activation function

- Bounded below by 0
- Has infinite values (not upper bounded)
- Always non-negative
- Tends to give neurons with sparse activities



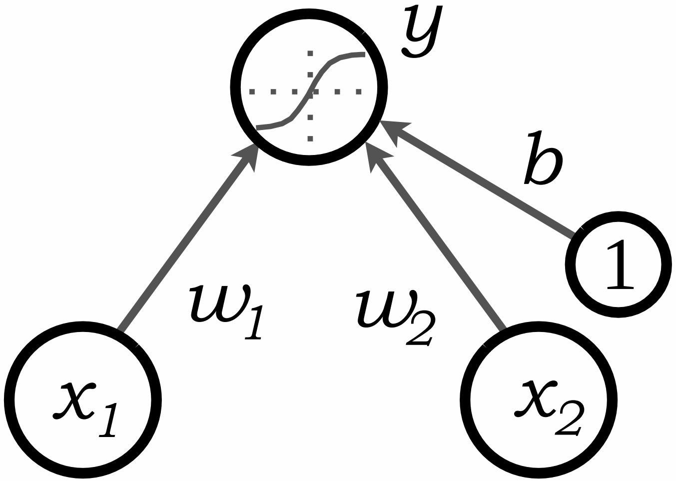
$$g(a) = reclin(a) = max(0, a)$$

Capacity of a single neuron

Range is determined by the activation function $g(a)$

- $\mathrm{h}(x) = g\big(a(x)\big) = g(b + \sum_i w_i x_i)$

Bias b only changes the position of the riff



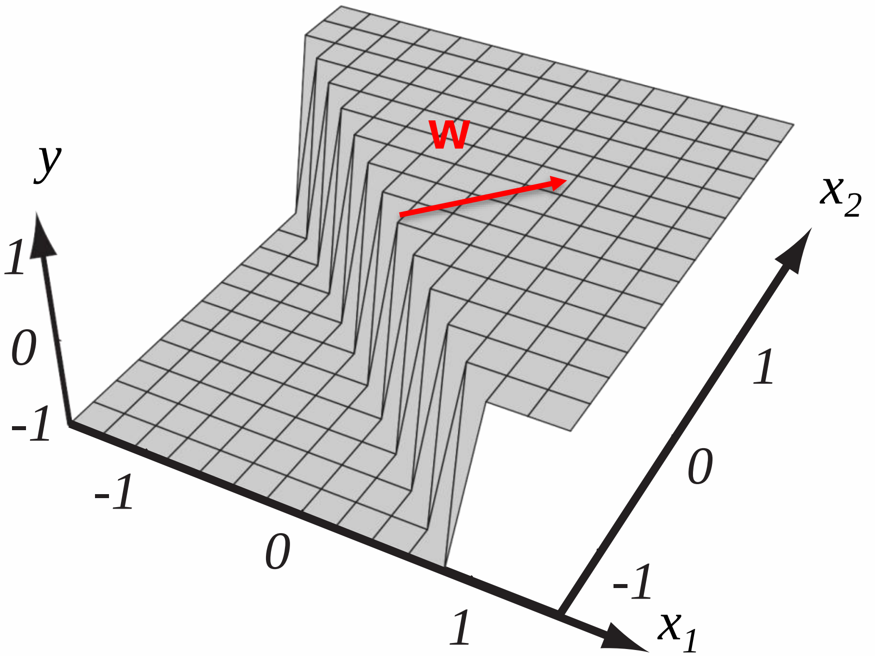


Image by Pascal Vincent

Capacity of a single neuron

Single neuron can do a binary classification

With sigmoid activation function it can be an estimation of probability

- Logical regression classifier
- $f(\boldsymbol{x}) = p(y = 1|\boldsymbol{x})$
- If greater then 0.5 predicts class 1,
- Otherwise, predicts class 2

Decision boundary is linear

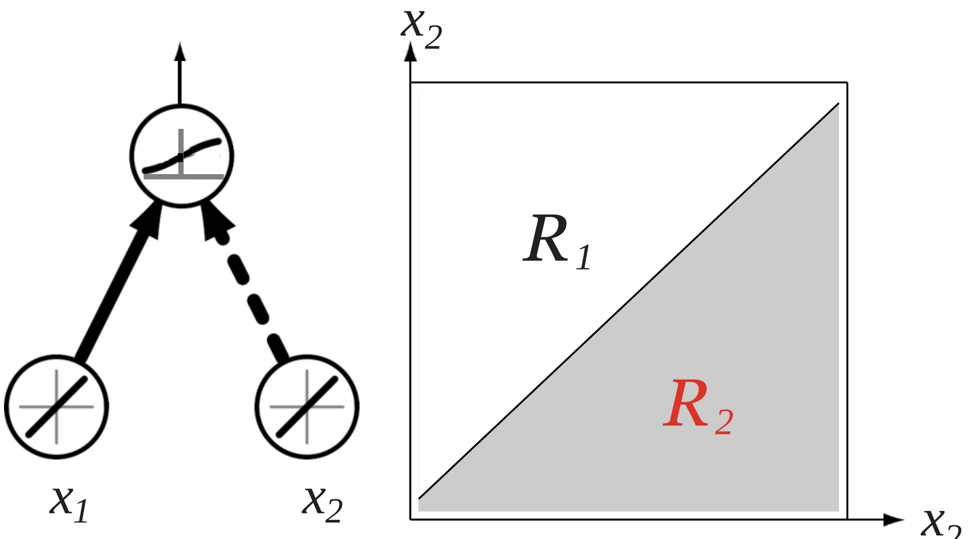
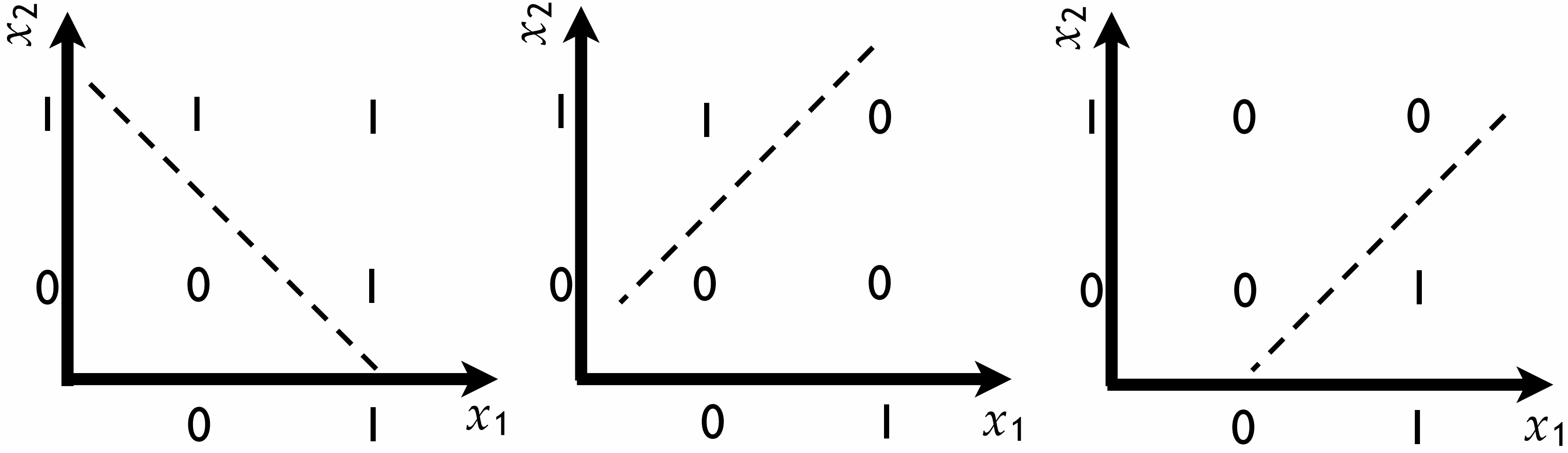


Image by Pascal Vincent

Capacity of a single neuron

Single neuron can solve linearly separable problems

$OR\,(x_1, x_2)$ $AND\,(\overline{x_1}, x_2)$ $AND\,(x_1, \overline{x_2})$

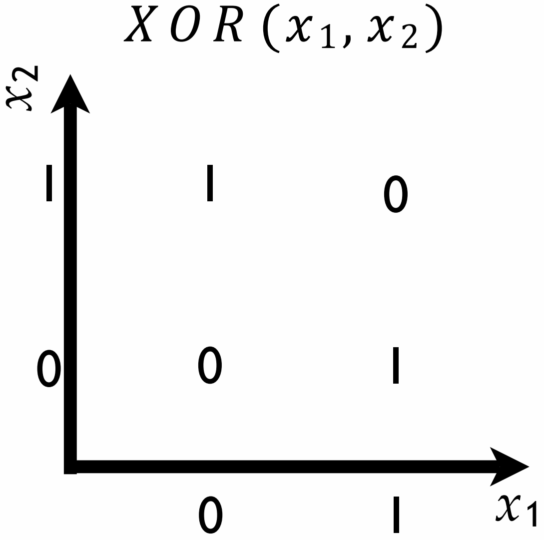


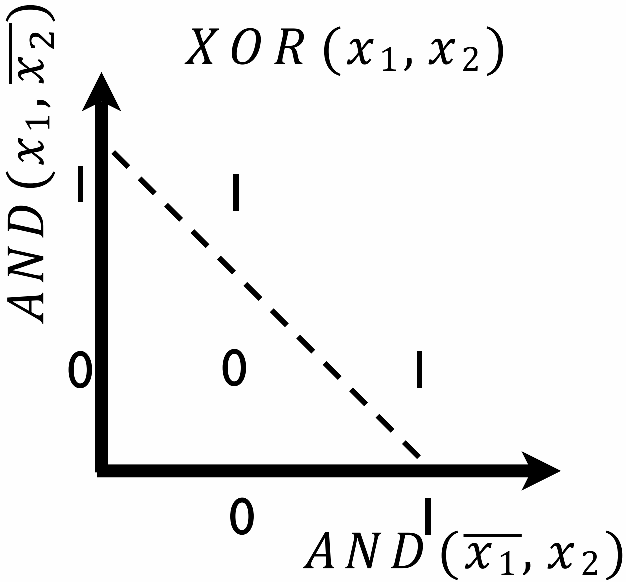
Capacity of a single neuron

Single neuron can't solve not linearly separable problems

Unless we transform an input into a better representation

- $XOR\,(x_1, x_2) = OR(AND\,(x_1, \overline{x_2})\,, AND\,(\overline{x_1}, x_2)\,)$





Single hidden layer

Hidden layer pre-activation:

- $a(\boldsymbol{x})_i = b_i^{(1)} + \sum_j w_{i,j}^{(1)} x_j = b_i^{(1)} + \boldsymbol{w}_{\boldsymbol{i}}^{(\mathbf{1})^T} \boldsymbol{x}$
- $\boldsymbol{a}(\boldsymbol{x}) = \boldsymbol{b}^{(1)} + \boldsymbol{W}^{(1)} \boldsymbol{x}$

Hidden layer activation:

- $\boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{g}(\boldsymbol{a}(\boldsymbol{x}))$

Output layer activation:

- $f(\boldsymbol{x}) = o\left(b^{(2)} + \boldsymbol{w}^{(1)^T} \boldsymbol{h}(\boldsymbol{x})\right)$

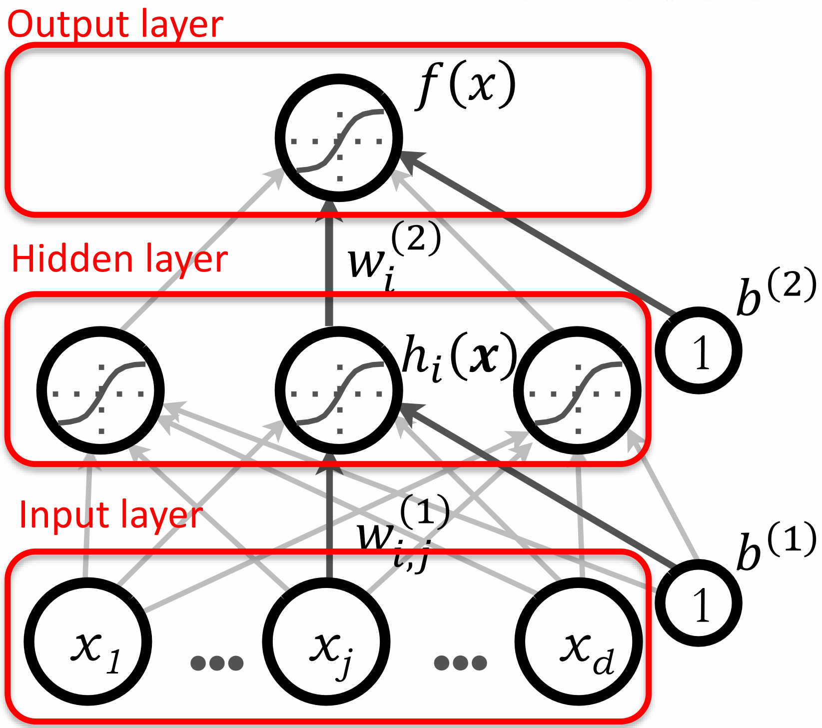


Image by Hugo Larochelle

Multilayer neural network

For a network with L hidden layers:

k-th hidden layer pre-activation:

- $\boldsymbol{a}^{(k)}(\boldsymbol{x}) = \boldsymbol{b}^{(k)} + \boldsymbol{W}^{(k)}\boldsymbol{h}^{(k-1)}(\boldsymbol{x})$

k-th hidden layer activation:

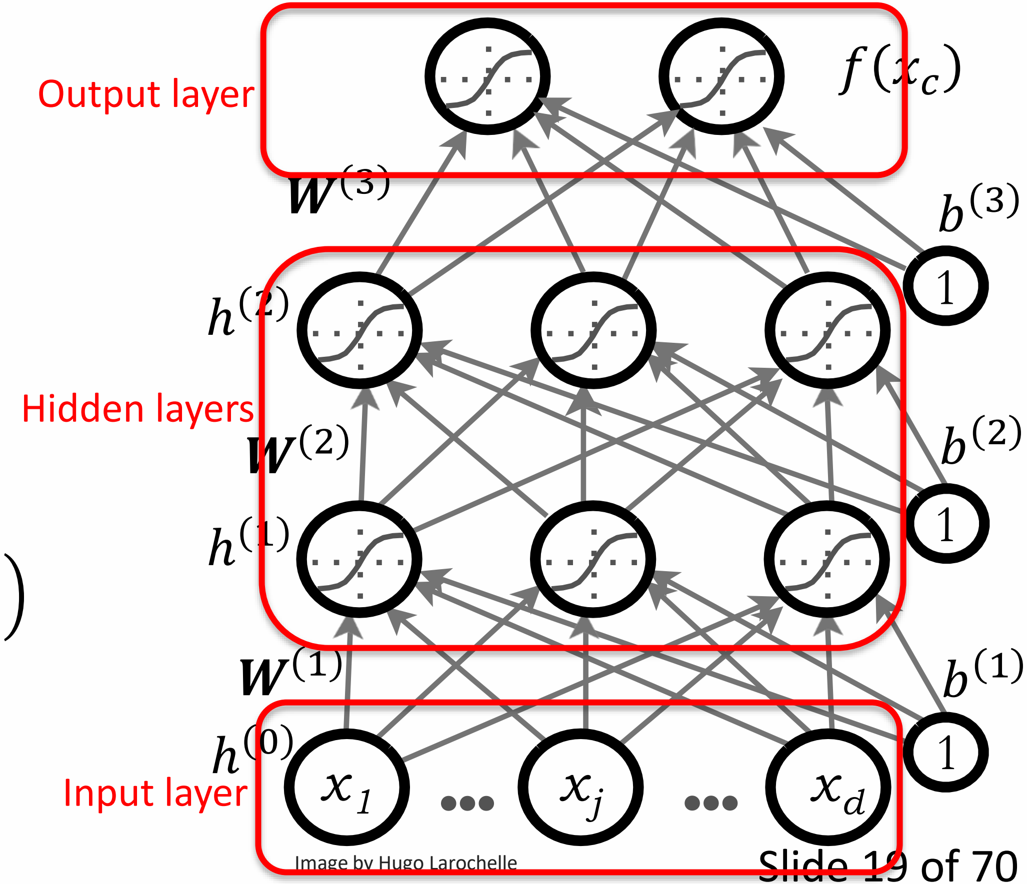
- $\boldsymbol{h}^{(k)}(\boldsymbol{x}) = \boldsymbol{g}\left(\boldsymbol{a}^{(k)}(\boldsymbol{x})\right)$

Output layer activation:

- $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{h}^{(L+1)}(\boldsymbol{x}) = o\left(\boldsymbol{a}^{(L+1)}(\boldsymbol{x})\right)$

Input

- $\boldsymbol{h}^{(0)}(\boldsymbol{x}) = \boldsymbol{x}$



Multiple outputs

For multiple class classification we need multiple outputs:

- 1 output per class
- Each output estimates the probability of belonging to a class:
  - $f(\boldsymbol{x})_c = p(y = c|\boldsymbol{x})$
- Predicted class is the one with the highest probability

Softmax activation function:

Exponent, $exp(a) = e^a$

- $\boldsymbol{o}(\boldsymbol{a}) = softmax(\boldsymbol{a}) = \left[\frac{e^{a_1}}{\sum_c e^{a_c}} \quad \cdots \quad \frac{e^{a_N}}{\sum_c e^{a_c}}\right]^T$
  - where N is the number of classes
- Strictly positive
- Sums to 1

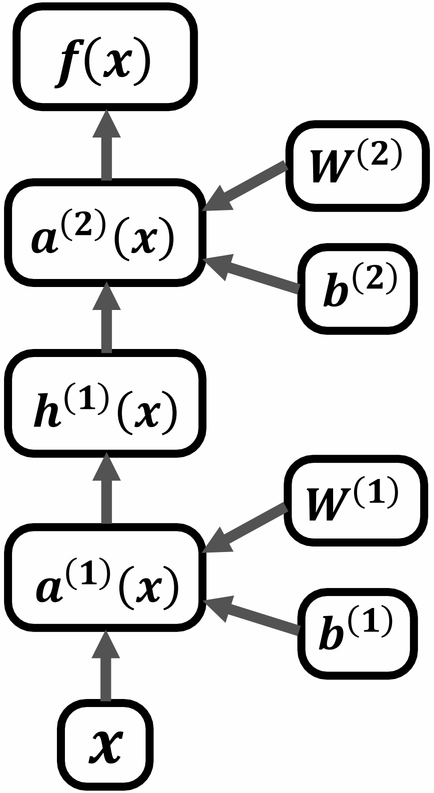
Forward propagation

Calculation of the neural network is called the **forward propagation**

It can be represented as an acyclic flow graph

Can be easily implemented in modular way:

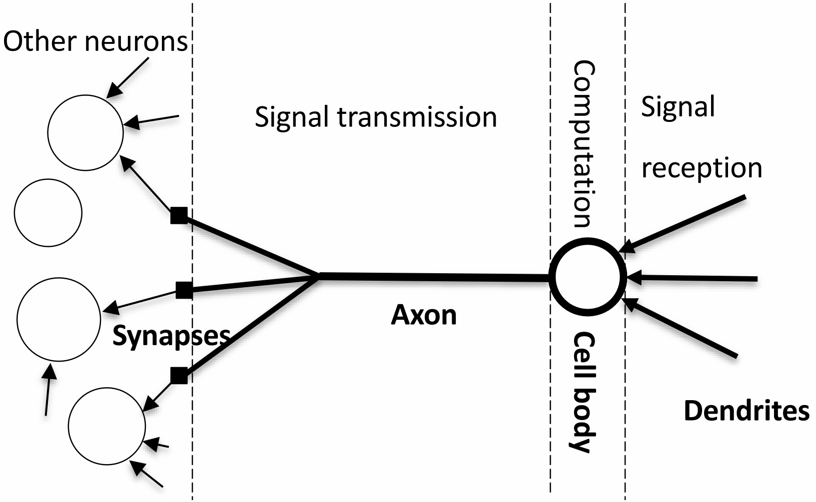
- Each block is a function with arguments of its children
- Calling functions in a right order implements a forward propagation

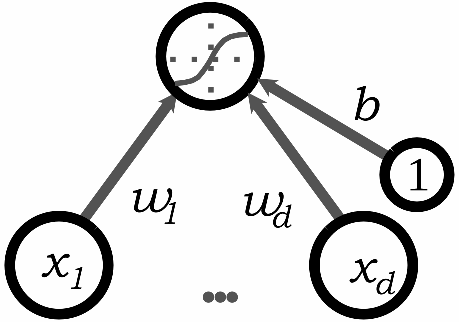


Biological vs artificial neuron

The artificial neuron approximates the biological one:

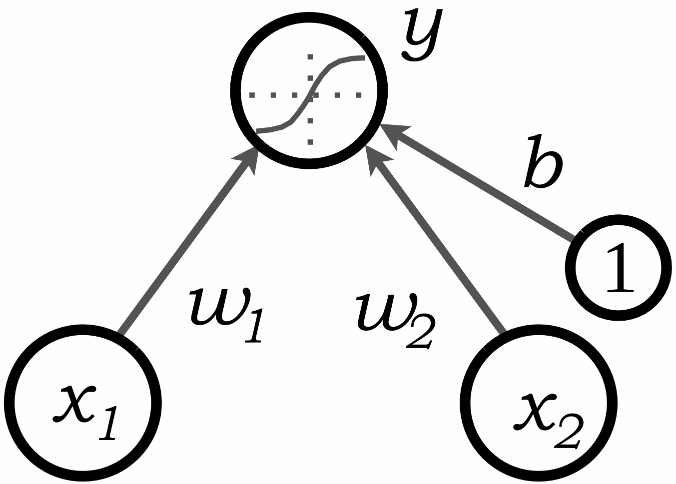
- the activation corresponds to a ‘‘sort of ’’ firing rate
- the weights between neurons model whether neurons excite or inhibit each other
- the activation function and bias model the thresholded behavior of action potentials

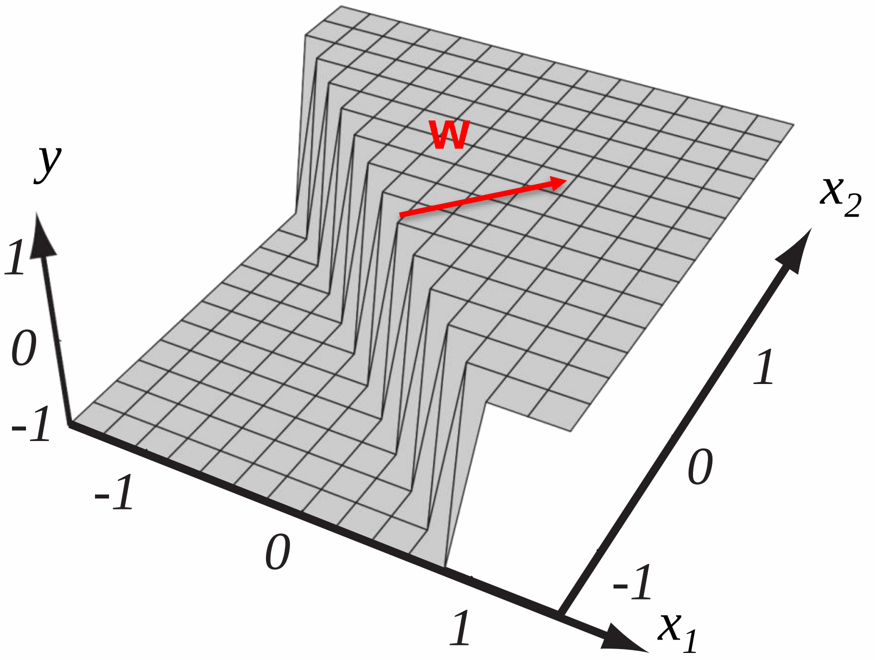




Capacity of a single neuron

- Single neuron can do a binary classification
- With sigmoid activation function it can be an estimation of probability





Capacity of a single hidden layer neural network

- Each neuron is a binary classifier
- By combining several binary classifiers, we can obtain an arbitrary classifier

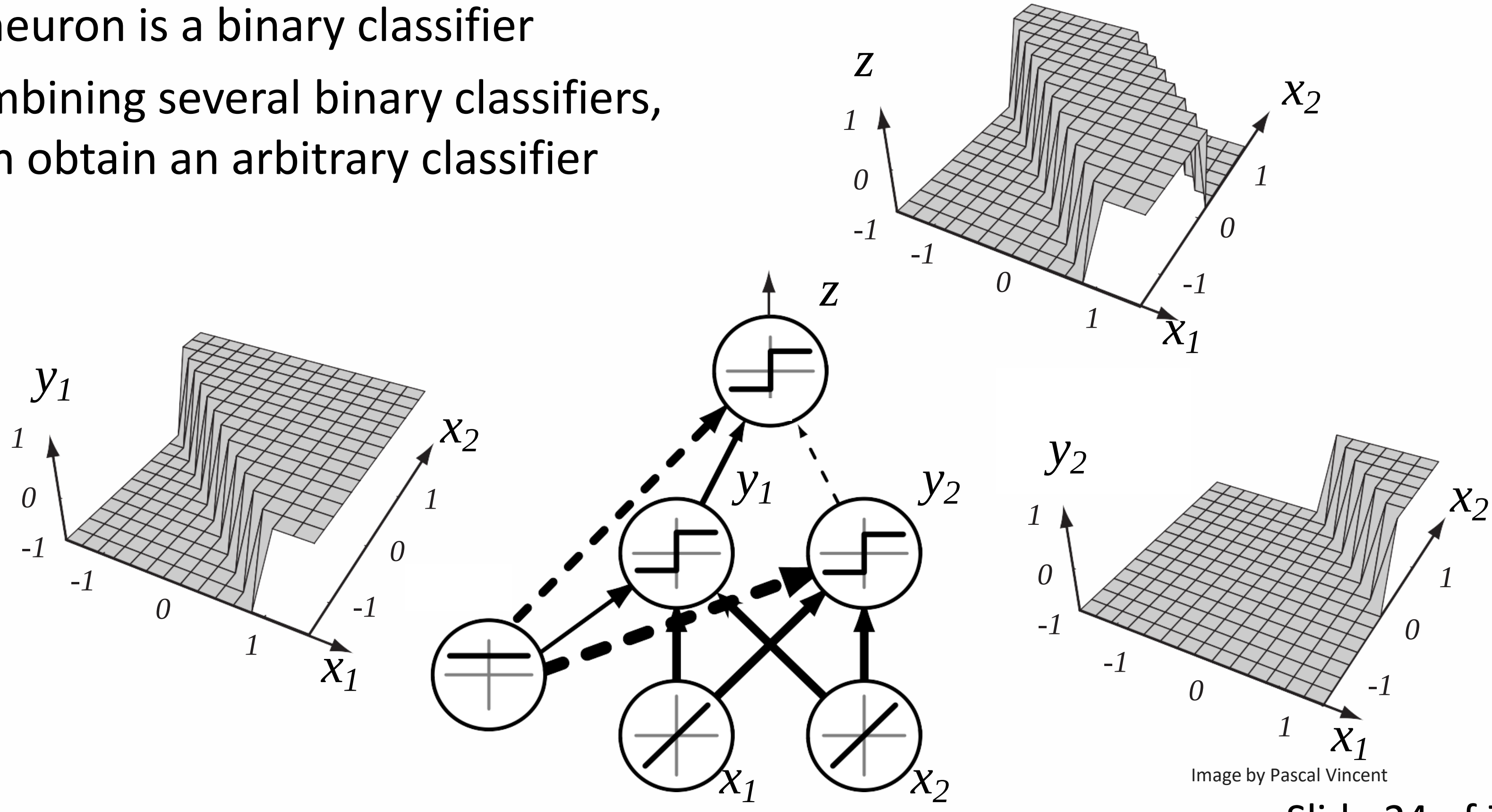


Image by Pascal Vincent

Capacity of a single hidden layer neural network

- Each neuron is a binary classifier
- By combining several binary classifiers, we can obtain an arbitrary classifier

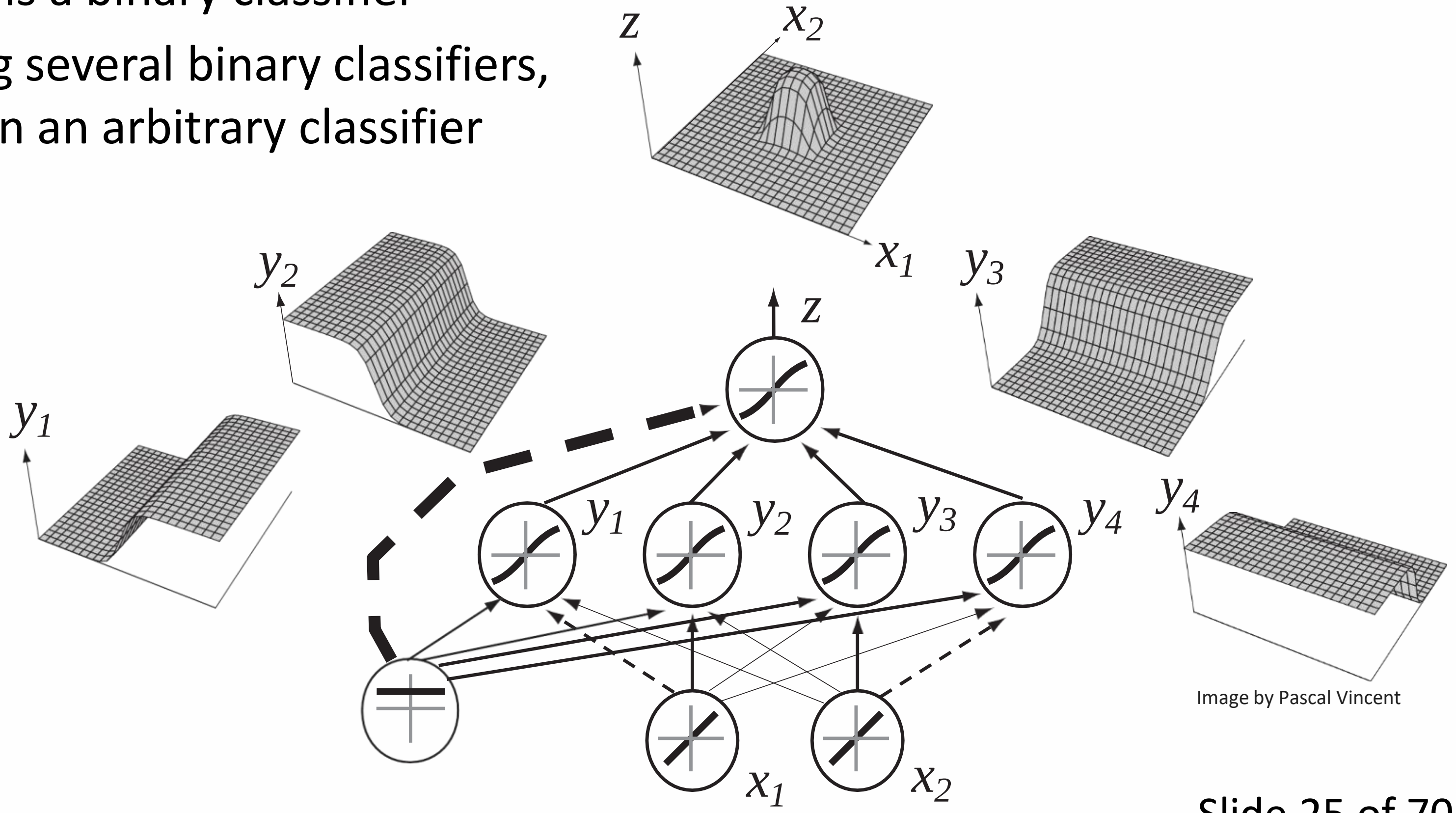
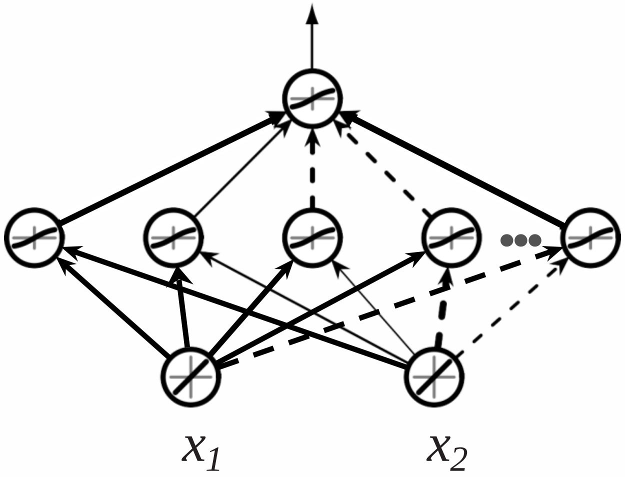


Image by Pascal Vincent

Capacity of a single hidden layer neural network

- Each neuron is a binary classifier
- By combining several binary classifiers, we can obtain an arbitrary classifier



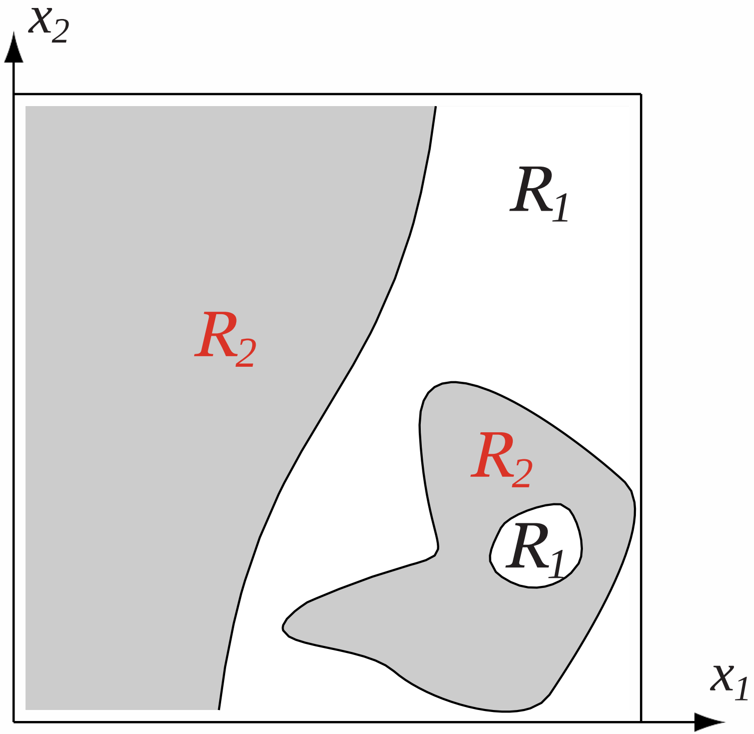


Image by Pascal Vincent

Capacity of a single hidden layer neural network

- Each neuron is a binary classifier
- By combining several binary classifiers, we can obtain an arbitrary classifier
- Universal approximation theorem (Hornik, 1991):
    - "A single hidden layer neural network with a linear output unit can approximate any continuous function arbitrarily well, given enough hidden units"
    - This can be applied for sigmoid, hyperbolic tangent and many other hidden layer activation functions
- It doesn't mean there is a learning algorithm that can find the necessary neural network parameter values

Deep learning

Use the supervised machine learning for empirical risk minimization

- $\arg\min_{\boldsymbol{\theta}} \frac{1}{T}\sum_t l\left(f\left(\boldsymbol{x}^{(t)}, \boldsymbol{\theta}\right), y^{(t)}\right) + \lambda\Omega(\boldsymbol{\theta})$
    - $l\left(f\left(\boldsymbol{x}^{(t)}, \boldsymbol{\theta}\right), y^{(t)}\right)$ is a loss function
    - $\Omega(\boldsymbol{\theta})$ is a regularizer, which penalties some certain values of $\boldsymbol{\theta}$; $\boldsymbol{\theta} = \left\{\boldsymbol{W}_{\boldsymbol{i},\boldsymbol{j}}^{\boldsymbol{k}}, \boldsymbol{b}^{\boldsymbol{k}}\right\}$
    - $\left\{\left(\boldsymbol{x}^{(t)}, y^{(t)}\right)\right\}$ is the dataset
- Learning is the task of optimization of the empirical risk
    - We would like to optimize a classification error, but it’s not smooth
    - Loss function is an upper bound of what we should optimize

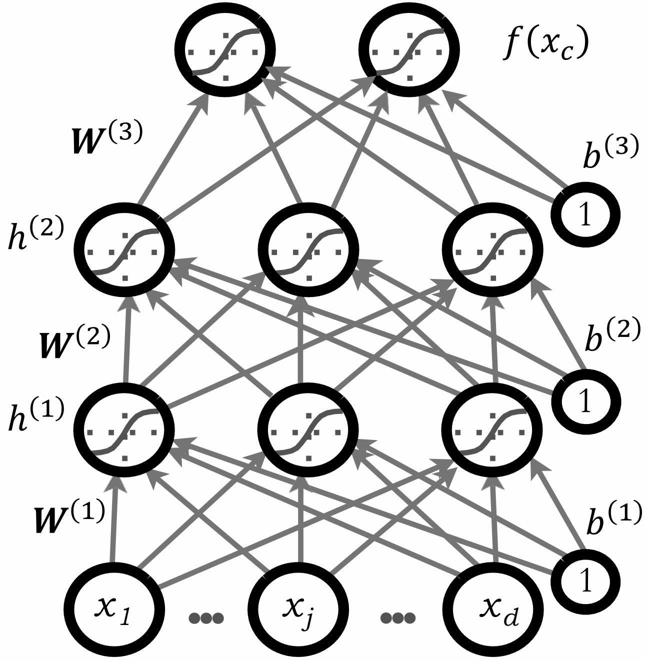


Image by Hugo Larochelle

Deep learning

Stochastic gradient descent (SGD) method

- Initialize $\boldsymbol{\theta}_0$ values: $\boldsymbol{\theta}_0 \equiv \{\boldsymbol{W}^{(1)}, b^{(1)}, \dots, \boldsymbol{W}^{(L+1)}, b^{(L+1)}\}$
- Do N iterations:
  - for each training example $(x^{(t)}, y^{(t)})$
  - $\Delta_e = -\nabla_{\boldsymbol{\theta}} l\left(f\left(\boldsymbol{x}^{(t)}, \boldsymbol{\theta}_e\right), y^{(t)}\right) - \lambda \nabla_\theta \Omega(\boldsymbol{\theta}_e)$
  - $\boldsymbol{\theta}_{\boldsymbol{e}+1} \leftarrow \boldsymbol{\theta}_e + \alpha_e \Delta_e$

Training epoch

To implement the SGD method, we need:

- a loss function $l\left(f\left(\boldsymbol{x}^{(t)}, \boldsymbol{\theta}\right), y^{(t)}\right)$
- a method to compute the parameters gradient $\nabla_\theta l\left(f\left(\boldsymbol{x}^{(t)}, \boldsymbol{\theta}\right), y^{(t)}\right)$
- a regularizer function $\Omega(\boldsymbol{\theta})$ and a method to compute its gradient $\nabla_\theta \Omega(\boldsymbol{\theta})$
- a method to initialize parameter values $\boldsymbol{\theta}$

Loss function

To implement the SGD method, we need:

- a loss function $l\left(f\left(\boldsymbol{x}^{(t)}, \boldsymbol{\theta}\right), y^{(t)}\right)$

- As we already learned:
  - Neural network output estimates the probability $f_c(\boldsymbol{x}) = p(y = c|\boldsymbol{x})$
  - We want to maximize the probabilities $y^{(t)}$ for a training set items $x^{(t)}$
- We will minimize the log-likehood
  - $l(\boldsymbol{f}(\boldsymbol{x}), y) = -\sum_c 1_{(y=c)} \, ln(f(\boldsymbol{x})_c) = -\,ln\left(f(\boldsymbol{x})_y\right)$
  - Natural logarithm is used for numerical stability and math simplicity
  - Sometimes is referred as a cross-entropy

Loss function derivative

To implement the SGD method, we need:

- a method to compute the parameters gradient

Partial derivative of output activation:

- $\frac{\partial}{\partial f(\boldsymbol{x})_c}\left(-\ln\left(f(\boldsymbol{x})_y\right)\right) = \frac{-1_{(y=c)}}{f(\boldsymbol{x})_y}$

Gradient of output activation:

- $\nabla_{f(\boldsymbol{x})}\left(-\ln\left(f(\boldsymbol{x})_y\right)\right) =$

$$= \frac{-1}{f(\boldsymbol{x})_y}\begin{bmatrix} 1_{(y=1)} \\ \cdots \\ 1_{(y=N)} \end{bmatrix} = \frac{-\boldsymbol{e}(y)}{f(\boldsymbol{x})_y}$$

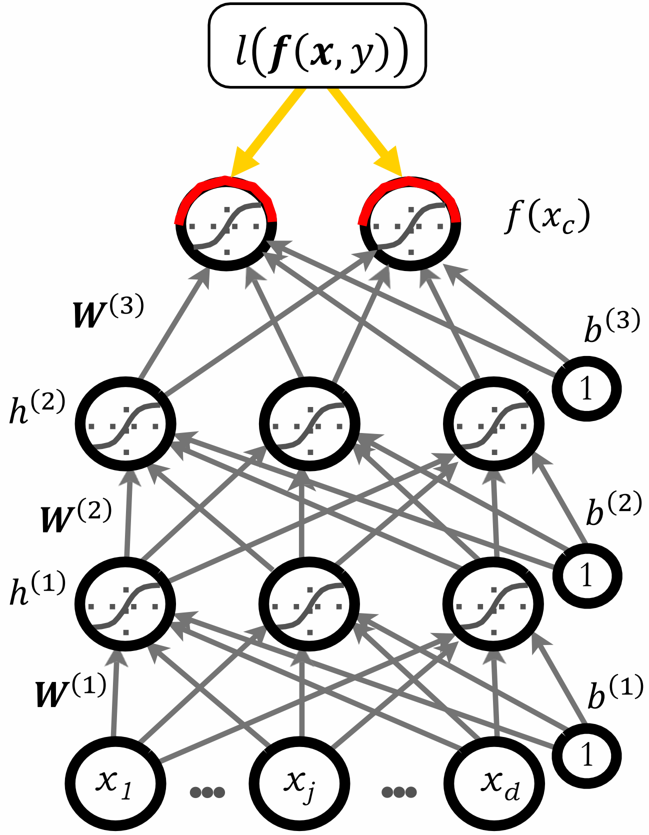


Image by Hugo Larochelle

Loss function derivative

To implement the SGD method, we need:

- a method to compute the parameters gradient

Partial derivative of output pre-activation:

- $$\frac{\partial}{\partial a^{(L+1)}(\boldsymbol{x})_c}\left(-\ln\left(f(\boldsymbol{x})_y\right)\right) =$$
$$= \frac{\partial\left(-\ln\left(f(\boldsymbol{x})_y\right)\right)}{\partial f(\boldsymbol{x})_y}\frac{\partial f(\boldsymbol{x})_y}{\partial a^{(L+1)}(\boldsymbol{x})_c} =$$
$$= \frac{-1}{f(\boldsymbol{x})_y}\frac{\partial f(\boldsymbol{x})_y}{\partial a^{(L+1)}(\boldsymbol{x})_c} =$$
$$= \frac{-1}{f(\boldsymbol{x})_y}\frac{\partial}{\partial a^{(L+1)}(\boldsymbol{x})_c} softmax\left(a^{(L+1)}(\boldsymbol{x})\right)_y =$$

Chain rule: $\frac{\partial y}{\partial x} = \frac{\partial y}{\partial t}\frac{\partial t}{\partial x}$

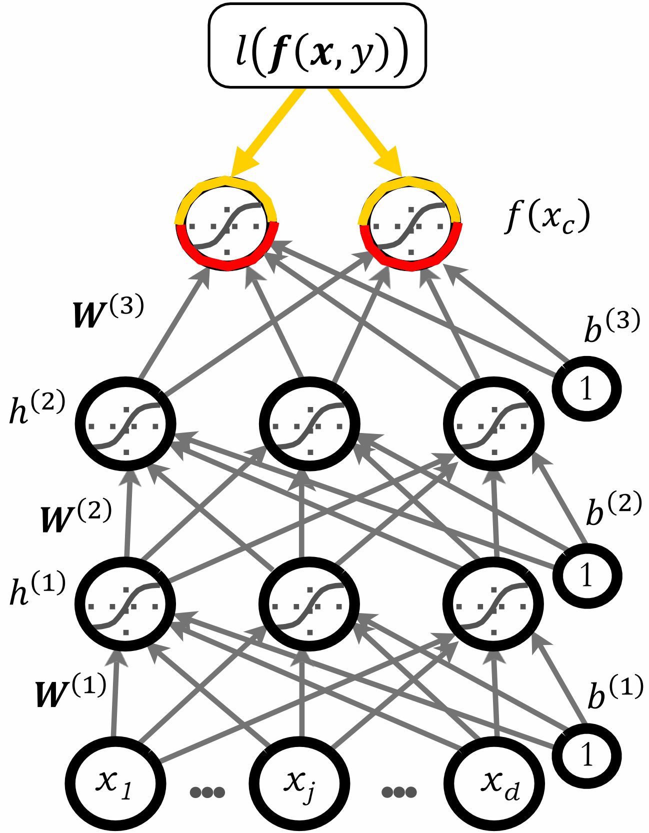


Image by Hugo Larochelle

Loss function derivative

$$= \frac{-1}{f(\boldsymbol{x})_y} \frac{\partial}{\partial a^{(L+1)}(\boldsymbol{x})_c} \frac{e^{\left(a^{(L+1)}(\boldsymbol{x})_y\right)}}{\sum_{c'} e^{\left(a^{(L+1)}(\boldsymbol{x})_{c'}\right)}} =$$

Derivative of ratio: $\frac{\partial}{\partial x}\left(\frac{g(x)}{h(x)}\right) = \frac{\partial g(x)}{\partial x}\frac{1}{h(x)} - \frac{g(x)}{h(x)^2}\frac{\partial h(x)}{\partial x}$

Loss function derivative

$$= \frac{-1}{f(\boldsymbol{x})_y} \frac{\partial}{\partial a^{(L+1)}(\boldsymbol{x})_c} \frac{e^{\left(a^{(L+1)}(\boldsymbol{x})_y\right)}}{\sum_{c'} e^{\left(a^{(L+1)}(\boldsymbol{x})_{c'}\right)}} =$$

$$= \frac{-1}{f(\boldsymbol{x})_y} \left(\frac{\frac{\partial}{\partial a^{(L+1)}(\boldsymbol{x})_c} e^{\left(a^{(L+1)}(\boldsymbol{x})_y\right)}}{\sum_{c'} e^{\left(a^{(L+1)}(\boldsymbol{x})_{c'}\right)}} - \left(\frac{e^{\left(a^{(L+1)}(\boldsymbol{x})_y\right)} \frac{\partial}{\partial a^{(L+1)}(\boldsymbol{x})_c} \sum_{c'} e^{\left(a^{(L+1)}(\boldsymbol{x})_{c'}\right)}}{\left(\sum_{c'} e^{\left(a^{(L+1)}(\boldsymbol{x})_{c'}\right)}\right)^{\mathbf{2}}} \right) \right) =$$

Derivative of ratio: $\frac{\partial}{\partial x}\left(\frac{g(x)}{h(x)}\right) = \frac{\partial g(x)}{\partial x}\frac{1}{h(x)} - \frac{g(x)}{h(x)^2}\frac{\partial h(x)}{\partial x}$

Loss function derivative

$$= \frac{-1}{f(\boldsymbol{x})_y} \frac{\partial}{\partial a^{(L+1)}(\boldsymbol{x})_c} \frac{e^{\left(a^{(L+1)}(\boldsymbol{x})_y\right)}}{\sum_{c'} e^{\left(a^{(L+1)}(\boldsymbol{x})_{c'}\right)}} =$$

$$= \frac{-1}{f(\boldsymbol{x})_y} \left(\frac{\frac{\partial}{\partial a^{(L+1)}(\boldsymbol{x})_c} e^{\left(a^{(L+1)}(\boldsymbol{x})_y\right)}}{\sum_{c'} e^{\left(a^{(L+1)}(\boldsymbol{x})_{c'}\right)}} - \left(\frac{e^{\left(a^{(L+1)}(\boldsymbol{x})_y\right)} \frac{\partial}{\partial a^{(L+1)}(\boldsymbol{x})_c} \sum_{c'} e^{\left(a^{(L+1)}(\boldsymbol{x})_{c'}\right)}}{\left(\sum_{c'} e^{\left(a^{(L+1)}(\boldsymbol{x})_{c'}\right)}\right)^{\mathbf{2}}} \right) \right) =$$

$$= \frac{-1}{f(\boldsymbol{x})_y} \left(\frac{1_{(y=c)} e^{\left(a^{(L+1)}(\boldsymbol{x})_y\right)}}{\sum_{c'} e^{\left(a^{(L+1)}(\boldsymbol{x})_{c'}\right)}} - \frac{e^{\left(a^{(L+1)}(\boldsymbol{x})_y\right)}}{\sum_{c'} e^{\left(a^{(L+1)}(\boldsymbol{x})_{c'}\right)}} \frac{e^{\left(a^{(L+1)}(\boldsymbol{x})_c\right)}}{\sum_{c'} e^{\left(a^{(L+1)}(\boldsymbol{x})_{c'}\right)}} \right) =$$

Derivative of ratio: $\frac{\partial}{\partial x}\left(\frac{g(x)}{h(x)}\right) = \frac{\partial g(x)}{\partial x} \frac{1}{h(x)} - \frac{g(x)}{h(x)^2} \frac{\partial h(x)}{\partial x}$

Loss function derivative

$$= \frac{-1}{f(\boldsymbol{x})_y} \frac{\partial}{\partial a^{(L+1)}(\boldsymbol{x})_c} \frac{e^{\left(a^{(L+1)}(\boldsymbol{x})_y\right)}}{\sum_{c'} e^{\left(a^{(L+1)}(\boldsymbol{x})_{c'}\right)}} =$$

$$= \frac{-1}{f(\boldsymbol{x})_y} \left(\frac{\frac{\partial}{\partial a^{(L+1)}(\boldsymbol{x})_c} e^{\left(a^{(L+1)}(\boldsymbol{x})_y\right)}}{\sum_{c'} e^{\left(a^{(L+1)}(\boldsymbol{x})_{c'}\right)}} - \left(\frac{e^{\left(a^{(L+1)}(\boldsymbol{x})_y\right)} \frac{\partial}{\partial a^{(L+1)}(\boldsymbol{x})_c} \sum_{c'} e^{\left(a^{(L+1)}(\boldsymbol{x})_{c'}\right)}}{\left(\sum_{c'} e^{\left(a^{(L+1)}(\boldsymbol{x})_{c'}\right)}\right)^{\mathbf{2}}} \right) \right) =$$

$$= \frac{-1}{f(\boldsymbol{x})_y} \left(\frac{1_{(y=c)} e^{\left(a^{(L+1)}(\boldsymbol{x})_y\right)}}{\sum_{c'} e^{\left(a^{(L+1)}(\boldsymbol{x})_{c'}\right)}} - \frac{e^{\left(a^{(L+1)}(\boldsymbol{x})_y\right)}}{\sum_{c'} e^{\left(a^{(L+1)}(\boldsymbol{x})_{c'}\right)}} \frac{e^{\left(a^{(L+1)}(\boldsymbol{x})_c\right)}}{\sum_{c'} e^{\left(a^{(L+1)}(\boldsymbol{x})_{c'}\right)}} \right) =$$

$$= \frac{-1}{f(\boldsymbol{x})_y} \left(1_{(y=c)} softmax\left(a^{(L+1)}(\boldsymbol{x})\right)_y - softmax\left(a^{(L+1)}(\boldsymbol{x})\right)_y softmax\left(a^{(L+1)}(\boldsymbol{x})\right)_c \right) =$$

$$= \frac{-1}{f(\boldsymbol{x})_y} \left(1_{(y=c)} f(\boldsymbol{x})_y - f(\boldsymbol{x})_y f(\boldsymbol{x})_c \right) =$$

Derivative of ratio: $\frac{\partial}{\partial x}\left(\frac{g(x)}{h(x)}\right) = \frac{\partial g(x)}{\partial x} \frac{1}{h(x)} - \frac{g(x)}{h(x)^2} \frac{\partial h(x)}{\partial x}$

Loss function derivative

$$= \frac{-1}{f(\boldsymbol{x})_y} \frac{\partial}{\partial a^{(L+1)}(\boldsymbol{x})_c} \frac{e^{\left(a^{(L+1)}(\boldsymbol{x})_y\right)}}{\sum_{c'} e^{\left(a^{(L+1)}(\boldsymbol{x})_{c'}\right)}} =$$

$$= \frac{-1}{f(\boldsymbol{x})_y} \left(\frac{\frac{\partial}{\partial a^{(L+1)}(\boldsymbol{x})_c} e^{\left(a^{(L+1)}(\boldsymbol{x})_y\right)}}{\sum_{c'} e^{\left(a^{(L+1)}(\boldsymbol{x})_{c'}\right)}} - \left(\frac{e^{\left(a^{(L+1)}(\boldsymbol{x})_y\right)} \frac{\partial}{\partial a^{(L+1)}(\boldsymbol{x})_c} \sum_{c'} e^{\left(a^{(L+1)}(\boldsymbol{x})_{c'}\right)}}{\left(\sum_{c'} e^{\left(a^{(L+1)}(\boldsymbol{x})_{c'}\right)}\right)^{\mathbf{2}}} \right) \right) =$$

$$= \frac{-1}{f(\boldsymbol{x})_y} \left(\frac{1_{(y=c)} e^{\left(a^{(L+1)}(\boldsymbol{x})_y\right)}}{\sum_{c'} e^{\left(a^{(L+1)}(\boldsymbol{x})_{c'}\right)}} - \frac{e^{\left(a^{(L+1)}(\boldsymbol{x})_y\right)}}{\sum_{c'} e^{\left(a^{(L+1)}(\boldsymbol{x})_{c'}\right)}} \frac{e^{\left(a^{(L+1)}(\boldsymbol{x})_c\right)}}{\sum_{c'} e^{\left(a^{(L+1)}(\boldsymbol{x})_{c'}\right)}} \right) =$$

$$= \frac{-1}{f(\boldsymbol{x})_y} \left(1_{(y=c)} softmax\left(a^{(L+1)}(\boldsymbol{x})\right)_y - softmax\left(a^{(L+1)}(\boldsymbol{x})\right)_y softmax\left(a^{(L+1)}(\boldsymbol{x})\right)_c \right) =$$

$$= \frac{-1}{f(\boldsymbol{x})_y} \left(1_{(y=c)} f(\boldsymbol{x})_y - f(\boldsymbol{x})_y f(\boldsymbol{x})_c\right) = -\left(1_{(y=c)} - f(\boldsymbol{x})_c\right)$$

Derivative of ratio: $\frac{\partial}{\partial x}\left(\frac{g(x)}{h(x)}\right) = \frac{\partial g(x)}{\partial x} \frac{1}{h(x)} - \frac{g(x)}{h(x)^2} \frac{\partial h(x)}{\partial x}$

Loss function derivative

To implement the SGD method, we need:

- a method to compute the parameters gradient

Partial derivative of output pre-activation:

- $\frac{\partial}{\partial a^{(L+1)}(\boldsymbol{x})_c}\left(-\ln\left(f(\boldsymbol{x})_y\right)\right) =$
$= -\left(1_{(y=c)} - f(\boldsymbol{x})_c\right)$

Gradient of output pre-activation:

- $\nabla_{\boldsymbol{a}^{(L+1)}(\boldsymbol{x})}\left(-\ln\left(f(\boldsymbol{x})_y\right)\right) =$
$= -\left(\boldsymbol{e}(y) - \boldsymbol{f}(\boldsymbol{x})\right)$

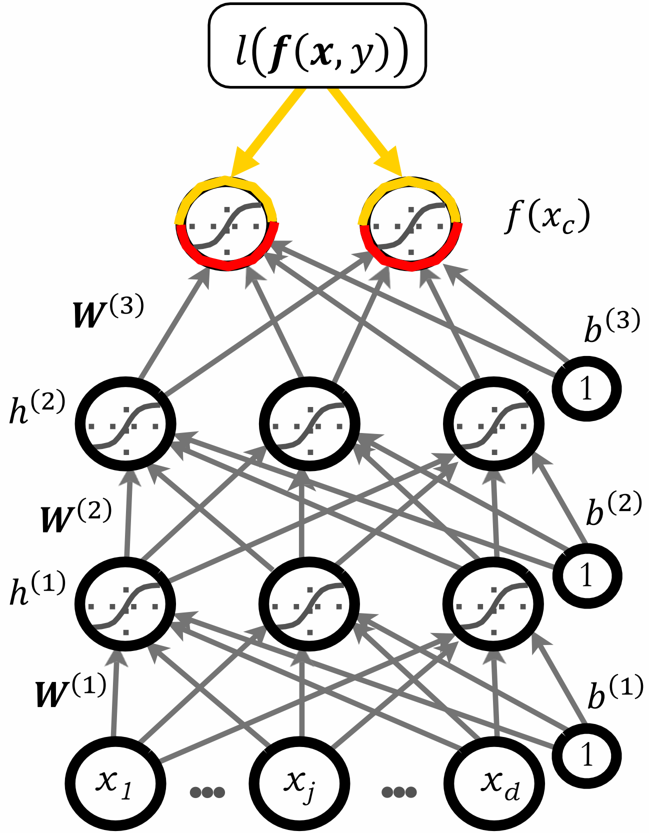


Image by Hugo Larochelle

Loss function derivative

To implement the SGD method, we need:

- a method to compute the parameters gradient

Partial derivative of hidden layer activation:

- $\frac{\partial}{\partial h^{(k)}(\boldsymbol{x})_j}\left(-\ln\left(f(\boldsymbol{x})_y\right)\right)$

$$\boxed{a^{(k+1)}(\boldsymbol{x})_i = b_i^{(k+1)} + \sum_j W_{i,j}^{(k+1)} h^{(k)}(\boldsymbol{x})_j}$$

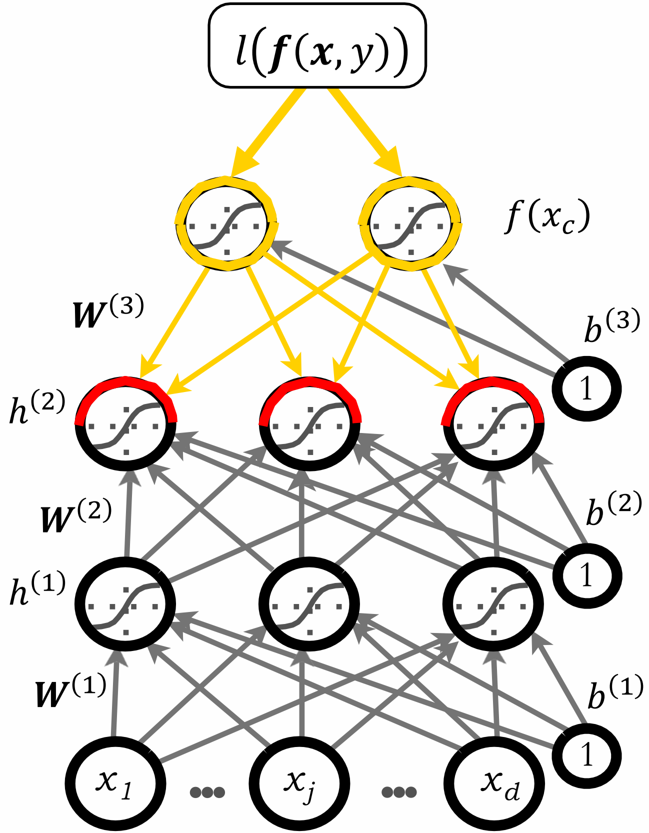


Image by Hugo Larochelle

Loss function derivative

To implement the SGD method, we need:

- a method to compute the parameters gradient

Partial derivative of hidden layer activation:

- $\frac{\partial}{\partial h^{(k)}(\boldsymbol{x})_j}\left(-\ln\left(f(\boldsymbol{x})_y\right)\right)$

Multivariable chain rule:

- $\frac{\partial p(a)}{\partial a} = \frac{\partial p(q_1(a),\ldots,q_n(a))}{\partial a} =$
  $= \sum_i \frac{\partial p(a)}{\partial q_i(a)} \frac{\partial q_i(a)}{\partial a}$

where a is a unit in layer

- $q_i(a)$ is a pre-activation in a layer above
- $p(a)$ is a loss function

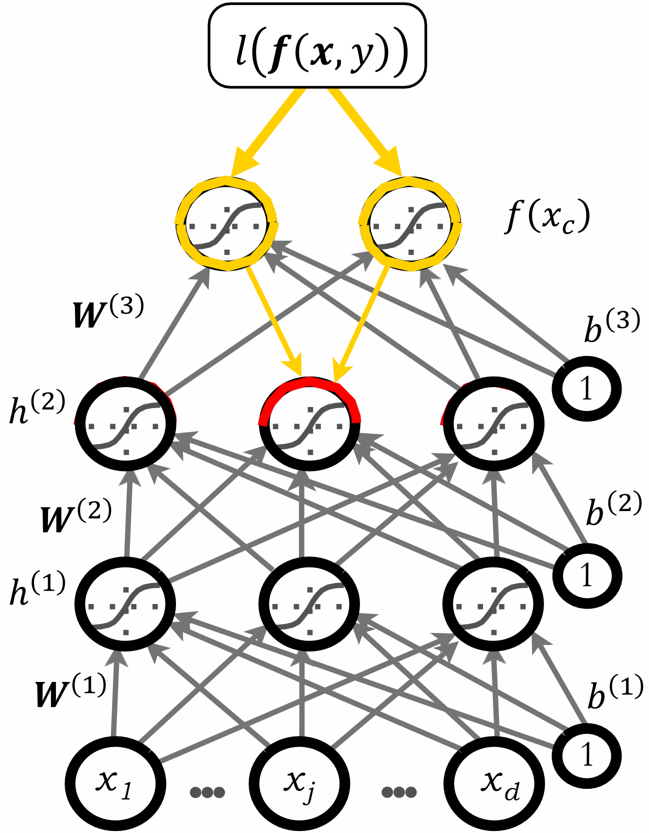


Image by Hugo Larochelle

Loss function derivative

To implement the SGD method, we need:

- a method to compute the parameters gradient

Partial derivative of hidden layer activation:

- $$\frac{\partial}{\partial h^{(k)}(\boldsymbol{x})_j}\left(-\ln\left(f(\boldsymbol{x})_y\right)\right) =$$

$$= \sum_i \frac{\partial\left(-\ln\left(f(\boldsymbol{x})_y\right)\right)}{\partial a^{(k+1)}(\boldsymbol{x})_i} \frac{\partial a^{(k+1)}(\boldsymbol{x})_i}{\partial h^{(k)}(\boldsymbol{x})_j} =$$

$$= \sum_i \frac{\partial\left(-\ln\left(f(\boldsymbol{x})_y\right)\right)}{\partial a^{(k+1)}(\boldsymbol{x})_i} \boldsymbol{W}_{i,j}^{(k+1)} =$$

$$= \boldsymbol{W}_{\cdot,j}^{(k+1)^T}\left(\nabla_{a^{(k+1)}(\boldsymbol{x})}\left(-\ln\left(f(\boldsymbol{x})_y\right)\right)\right)$$

$$\boxed{a^{(k+1)}(\boldsymbol{x})_i = b_i^{(k+1)} + \sum_j W_{i,j}^{(k+1)} h^{(k)}(\boldsymbol{x})_j}$$

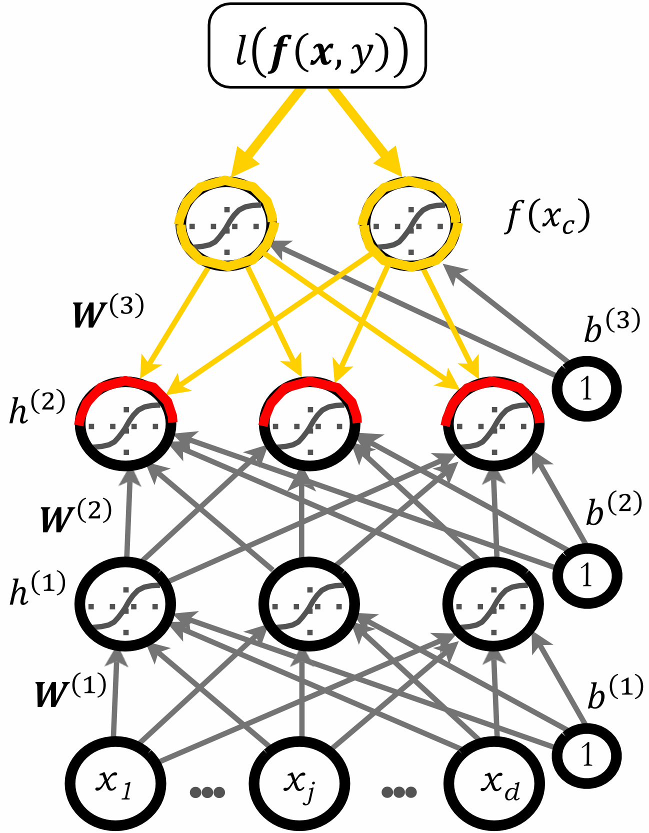


Image by Hugo Larochelle

Loss function derivative

To implement the SGD method, we need:

- a method to compute parameters gradient

Gradient of hidden layer activation:

- $\nabla_{\boldsymbol{h}^{(k)}(\boldsymbol{x})}\left(-\ln\left(f(\boldsymbol{x})_y\right)\right) =$
  $= \boldsymbol{W}^{(k+1)^T}\left(\nabla_{\boldsymbol{a}^{(k+1)}(\boldsymbol{x})}\left(-\ln\left(f(\boldsymbol{x})_y\right)\right)\right)$

$$\boxed{a^{(k+1)}(\boldsymbol{x})_i = b_i^{(k+1)} + \sum_j W_{i,j}^{(k+1)} h^{(k)}(\boldsymbol{x})_j}$$

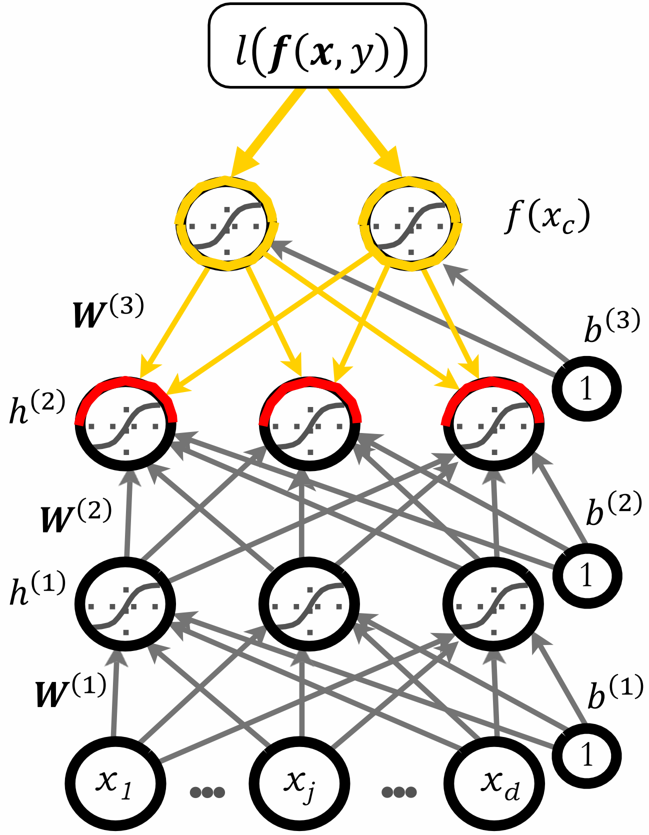


Image by Hugo Larochelle

Loss function derivative

To implement the SGD method, we need:

- a method to compute the parameters gradient

Partial derivative of hidden layer pre-activation:

- $$\frac{\partial}{\partial a^{(k)}(\boldsymbol{x})_j}\left(-\ln\left(f(\boldsymbol{x})_y\right)\right) =$$
$$= \frac{\partial\left(-\ln\left(f(\boldsymbol{x})_y\right)\right)}{\partial h^{(k)}(\boldsymbol{x})_j}\frac{\partial h^{(k)}(\boldsymbol{x})_j}{\partial a^{(k)}(\boldsymbol{x})_j} =$$
$$= \frac{\partial\left(-\ln\left(f(\boldsymbol{x})_y\right)\right)}{\partial h^{(k)}(\boldsymbol{x})_j} g'\left(a^{(k)}(\boldsymbol{x})_j\right)$$

$$\boxed{h^{(k)}(\boldsymbol{x})_j = g\left(a^{(k)}(\boldsymbol{x})_j\right)}$$

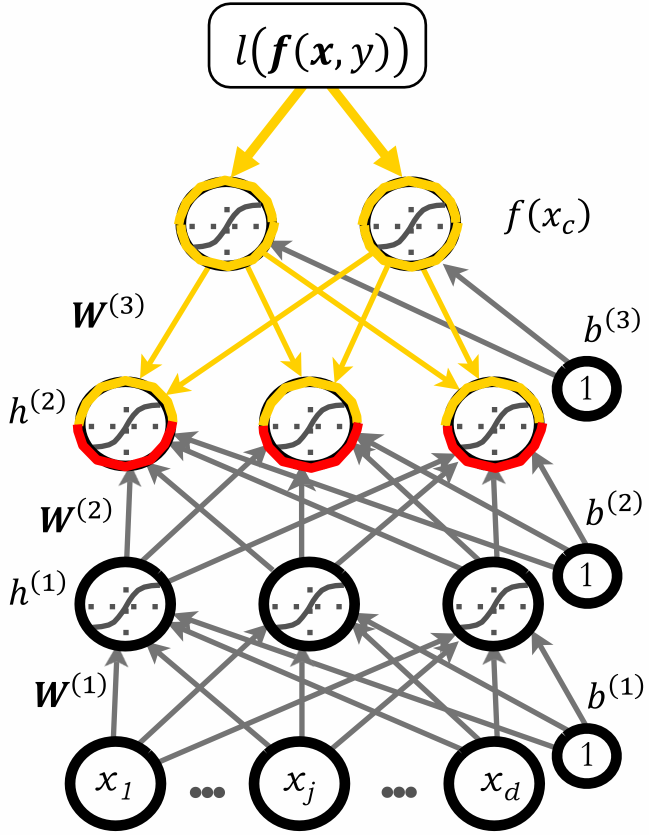


Image by Hugo Larochelle

Loss function derivative

To implement the SGD method, we need:

- a method to compute the parameters gradient

Partial gradient of hidden layer pre-activation:

- $\nabla_{a^{(k)}(\boldsymbol{x})}\left(-ln\left(f(\boldsymbol{x})_y\right)\right) =$

$$= \left(\nabla_{h^{(k)}(\boldsymbol{x})}\left(-ln\left(f(\boldsymbol{x})_y\right)\right)\right)^T \nabla_{a^{(k)}(\boldsymbol{x})} h^{(k)}(x) =$$

$$= \left(\nabla_{h^{(k)}(\boldsymbol{x})}\left(-ln\left(f(\boldsymbol{x})_y\right)\right)\right)^T \cdot$$

$$\cdot \left[\dots \quad g'\left(a^{(k)}(\boldsymbol{x})_j\right) \quad \dots\right]$$

$$\boxed{h^{(k)}(\boldsymbol{x})_j = g\left(a^{(k)}(\boldsymbol{x})_j\right)}$$

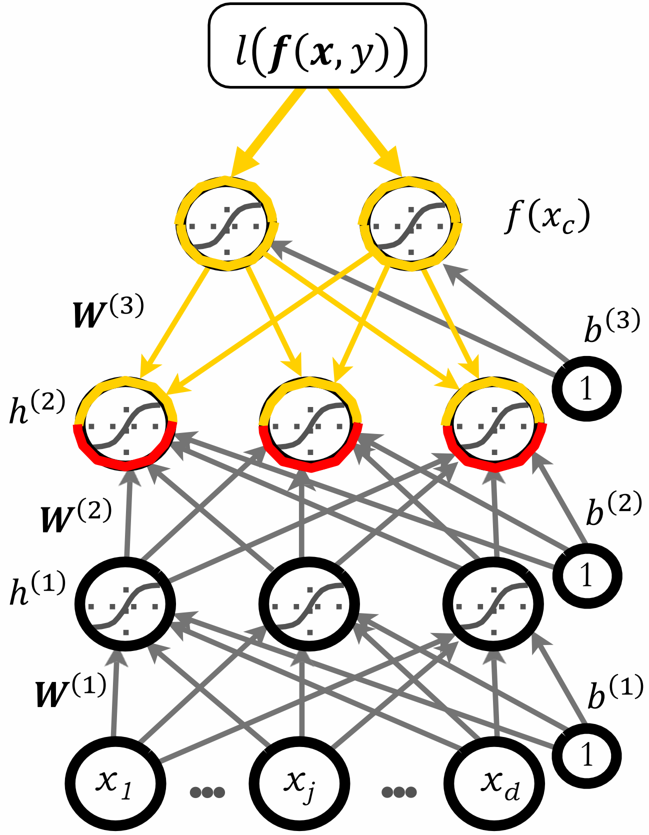


Image by Hugo Larochelle

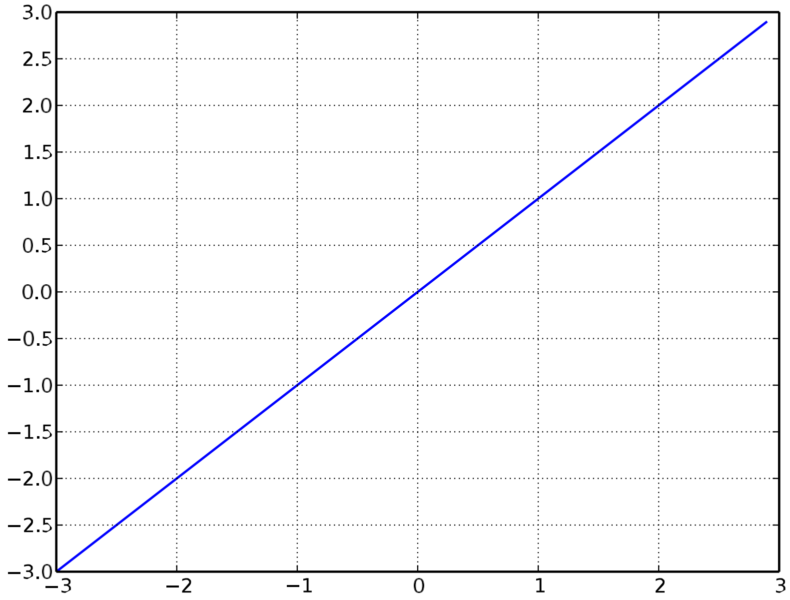
Activation function

To implement the SGD method, we need:

- a method to compute the parameters gradient

Linear activation function

- Partial derivative
- $g'(a) = 1$



$$g(a) = a$$

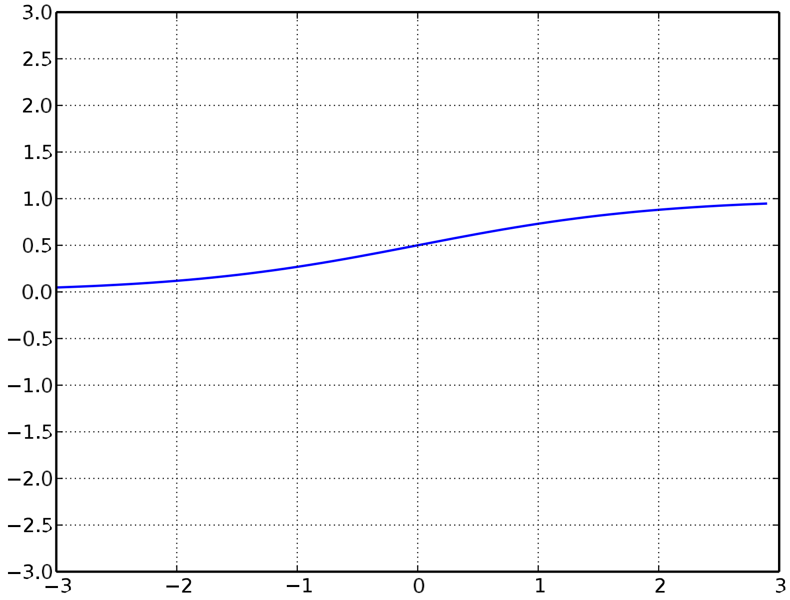
Activation function

To implement the SGD method, we need:

- a method to compute the parameters gradient

Sigmoid activation function

- Partial derivative
- $g'(a) = g(a)\big(1 - g(a)\big)$



$$g(a) = sigm(a) = \frac{1}{1 + e^{-a}}$$

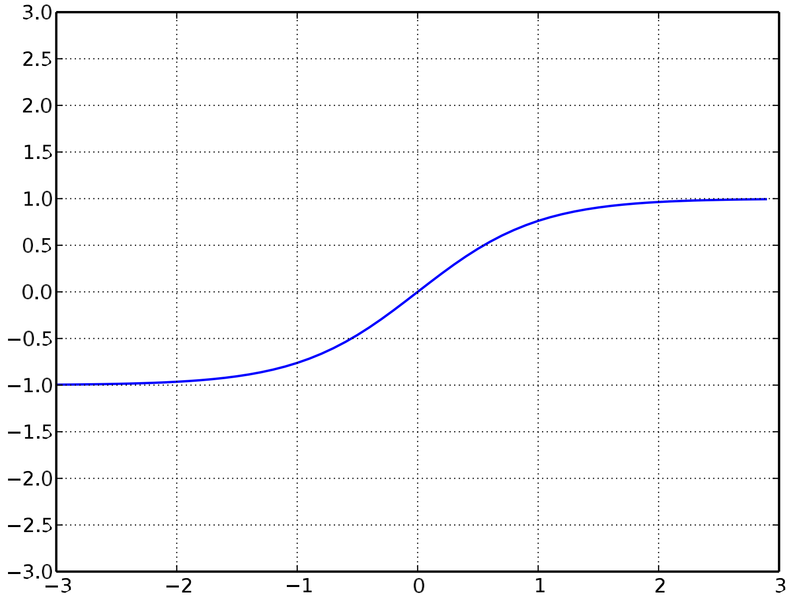
Activation function

To implement the SGD method, we need:

- a method to compute the parameters gradient

Hyperbolic tangent activation function

- Partial derivative
- $g'(a) = 1 - g(a)^2$



$$g(a) = tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}} = \frac{e^{2a} - 1}{e^{2a} + 1}$$

Loss function derivative

To implement the SGD method, we need:

- a method to compute the parameters gradient

Partial derivative for weights:

- $$\frac{\partial}{\partial W_{i,j}^{(k)}}\left(-ln\left(f(\boldsymbol{x})_y\right)\right) =$$
$$= \frac{\partial\left(-ln\left(f(\boldsymbol{x})_y\right)\right)}{\partial a^{(k)}(x)_i}\frac{\partial a^{(k)}(\boldsymbol{x})_i}{\partial W_{i,j}^{(k)}} =$$
$$= \frac{\partial\left(-ln\left(f(\boldsymbol{x})_y\right)\right)}{\partial a^{(k)}(x)_i} h^{(k-1)}(\boldsymbol{x})_j$$

$$\boxed{a^{(k)}(\boldsymbol{x})_i = b_i^{(k)} + \sum\nolimits_j W_{i,j}^{(1)} h^{(k-1)}(\boldsymbol{x})_j}$$

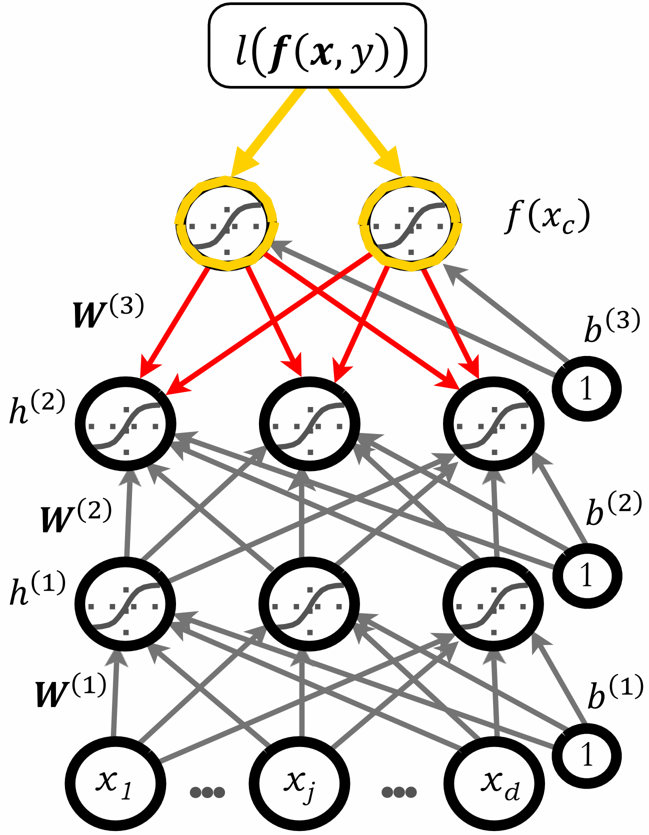


Image by Hugo Larochelle

Loss function derivative

To implement the SGD method, we need:

- a method to compute the parameters gradient

Gradient for weights:

- $\nabla_{\boldsymbol{W}^{(k)}}\left(-ln\left(f(\boldsymbol{x})_y\right)\right) =$
  $= \nabla_{\boldsymbol{a}^{(k)}(\boldsymbol{x})}\left(-ln\left(f(\boldsymbol{x})_y\right)\right) h^{(k-1)}(\boldsymbol{x})^{\mathrm{T}}$

$$\boxed{a^{(k)}(\boldsymbol{x})_i = b_i^{(k)} + \sum_j W_{i,j}^{(1)} h^{(k-1)}(\boldsymbol{x})_j}$$

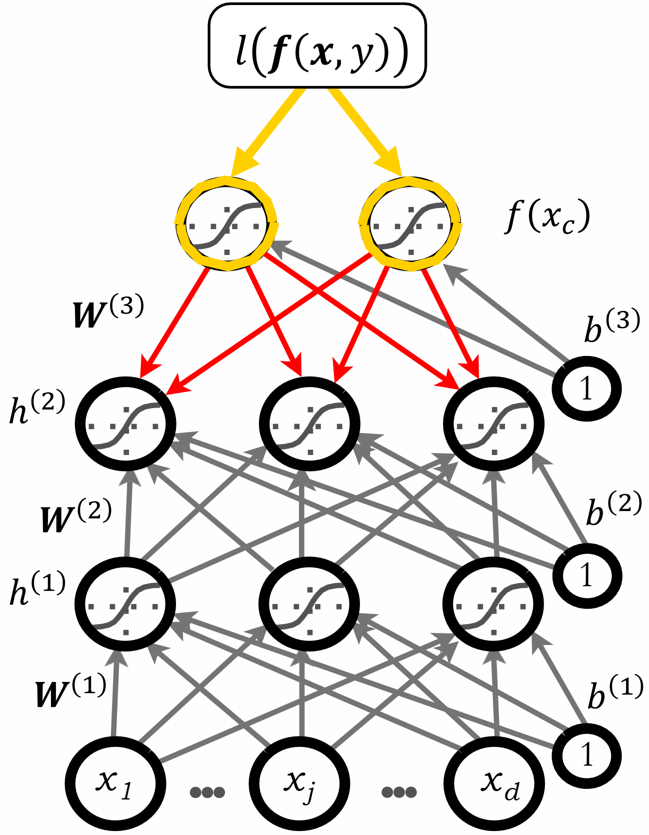


Image by Hugo Larochelle

Loss function derivative

To implement the SGD method, we need:

- a method to compute the parameters gradient

Partial derivative for bias:

- $$\frac{\partial}{\partial b_i^{(k)}}\left(-\ln\left(f(\boldsymbol{x})_y\right)\right) =$$
$$= \frac{\partial\left(-\ln\left(f(\boldsymbol{x})_y\right)\right)}{\partial a^{(k)}(x)_i}\frac{\partial a^{(k)}(x)_i}{\partial b_i^{(k)}} =$$
$$= \frac{\partial\left(-\ln\left(f(\boldsymbol{x})_y\right)\right)}{\partial a^{(k)}(x)_i}$$

$$\boxed{a^{(k)}(\boldsymbol{x})_i = b_i^{(k)} + \sum_j W_{i,j}^{(1)} h^{(k-1)}(\boldsymbol{x})_j}$$

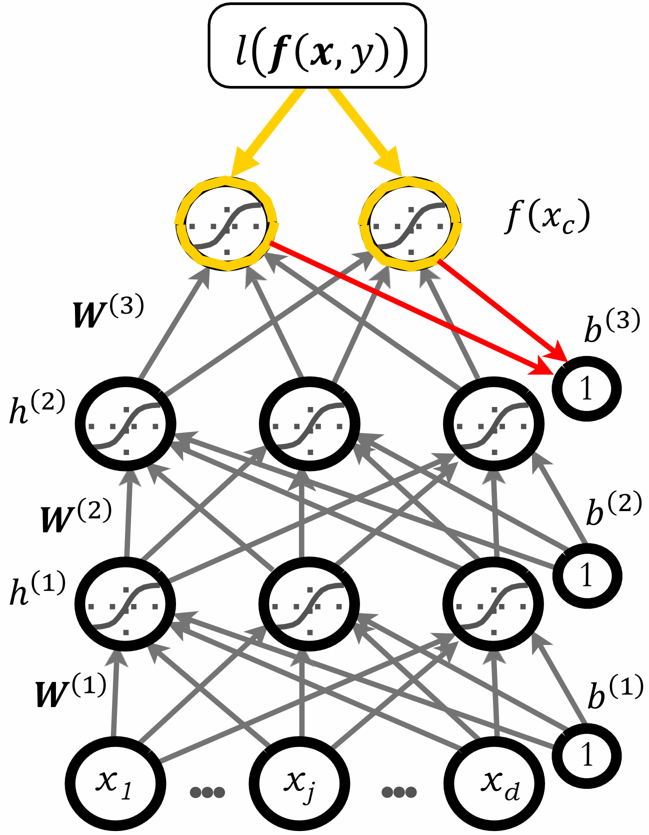


Image by Hugo Larochelle

Loss function derivative

To implement the SGD method, we need:

- a method to compute the parameters gradient

Gradient for bias:

- $\nabla_{\boldsymbol{b}^{(k)}}\left(-\ln\left(f(\boldsymbol{x})_y\right)\right) =$
  $= \nabla_{\boldsymbol{a}^{(k)}(\boldsymbol{x})}\left(-\ln\left(f(\boldsymbol{x})_y\right)\right)$

$$\boxed{a^{(k)}(\boldsymbol{x})_i = b_i^{(k)} + \sum_j W_{i,j}^{(1)} h^{(k-1)}(\boldsymbol{x})_j}$$

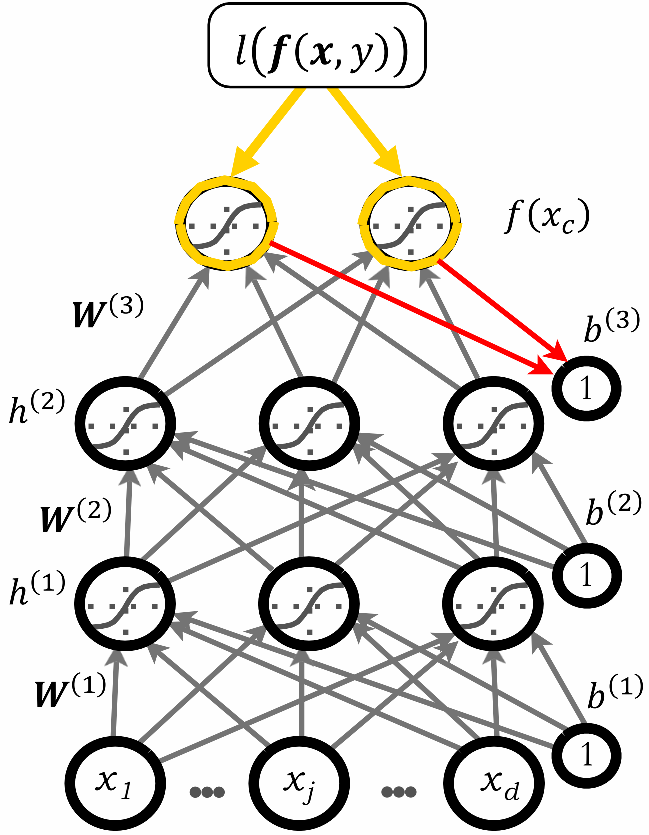


Image by Hugo Larochelle

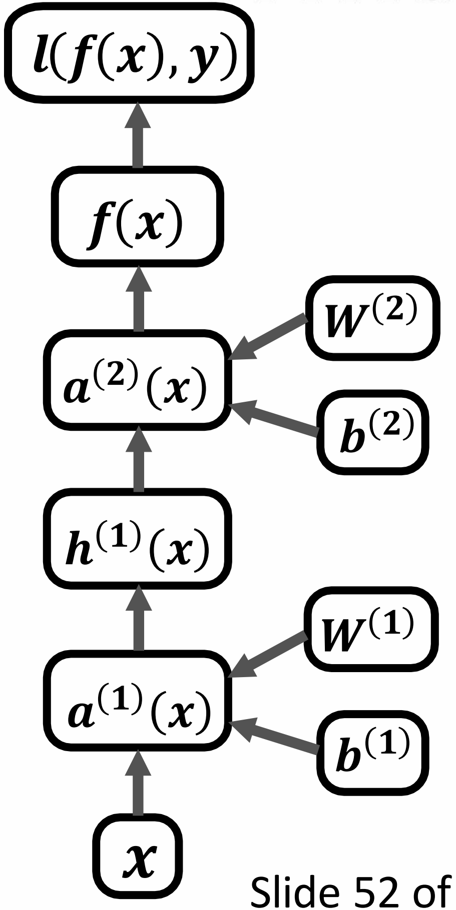
Forward propagation

Calculation of the neural network is a forward propagation

It can be represented as an acyclic flow graph

Can be easily implemented in modular way:

- Each block is a function with arguments of its children
- Calling functions in a right order implements a forward propagation



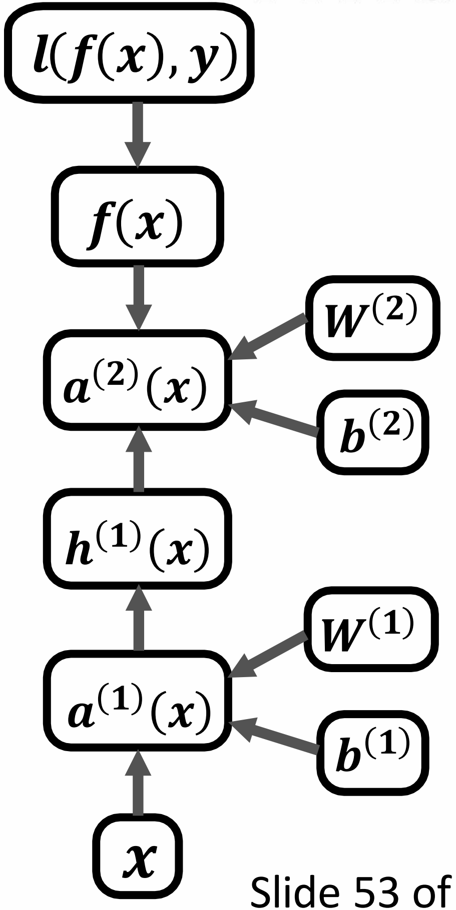
Backward propagation

Calculation of the gradient is a backward propagation

It calculates the gradient of the loss with respect to each children

1. Compute the output layer gradient
   - $\nabla_{\boldsymbol{a}^{(L+1)}(\boldsymbol{x})}(-\ln(f(\boldsymbol{x})_y)) = -(\boldsymbol{e}(y) - \boldsymbol{f}(\boldsymbol{x}))$
2. For each k from $L+1$ to 1
   a. Compute the gradients of hidden layer parameters
   b. Compute the gradient of the hidden layer below

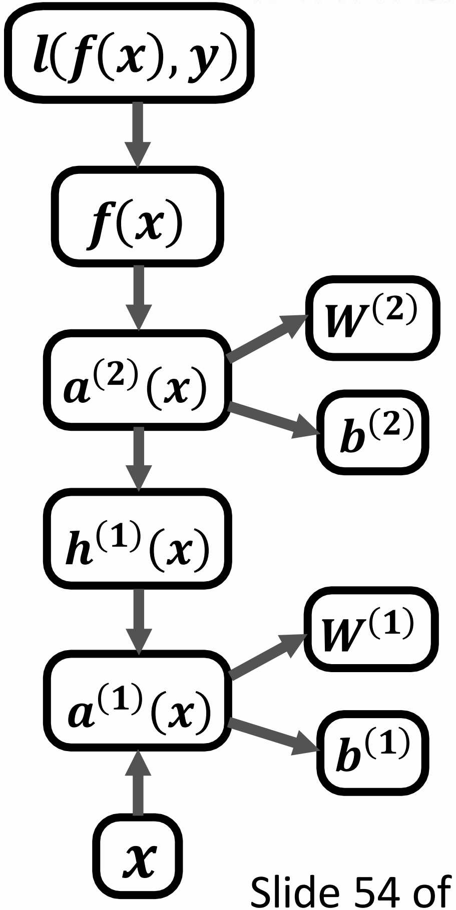
- Backward propagation is executed until parameters θ are reached



Backward propagation

For each k from $L+1$ to 1

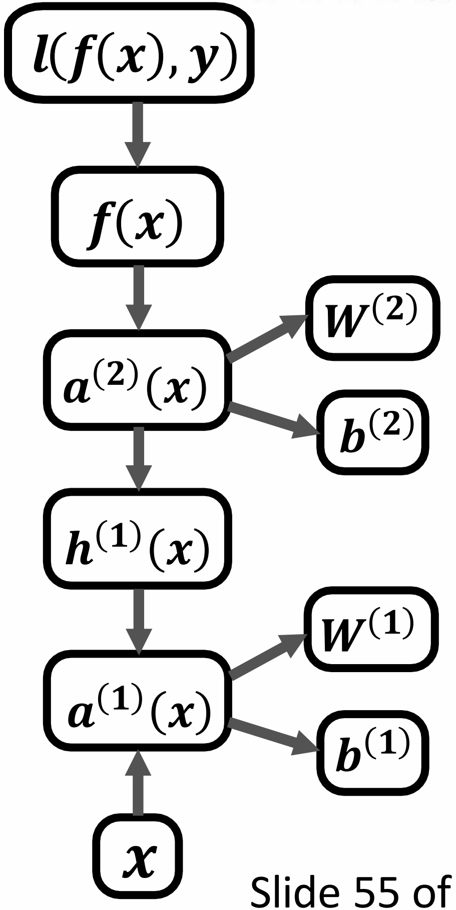
- Compute the gradients of hidden layer parameters
  - $\nabla_{\boldsymbol{W}^{(k)}}\left(-ln\left(f(\boldsymbol{x})_y\right)\right) = \nabla_{\boldsymbol{a}^{(k)}(\boldsymbol{x})}\left(-ln\left(f(\boldsymbol{x})_y\right)\right)h^{(k-1)}(\boldsymbol{x})^{\mathrm{T}}$
  - $\nabla_{\boldsymbol{b}^{(k)}}\left(-ln\left(f(\boldsymbol{x})_y\right)\right) = \nabla_{\boldsymbol{a}^{(k)}(\boldsymbol{x})}\left(-ln\left(f(\boldsymbol{x})_y\right)\right)$
- Compute the gradient of the hidden layer below
  - activation: $\nabla_{\boldsymbol{h}^{(k-1)}(\boldsymbol{x})}\left(-ln\left(f(\boldsymbol{x})_y\right)\right) =$
    $= \boldsymbol{W}^{(k)^T}\left(\nabla_{\boldsymbol{a}^{(k)}(\boldsymbol{x})}\left(-ln\left(f(\boldsymbol{x})_y\right)\right)\right)$
  - pre-activation: $\nabla_{a^{(k)}(\boldsymbol{x})}\left(-ln\left(f(\boldsymbol{x})_y\right)\right) =$
    $= \left(\nabla_{h^{(k+1)}(\boldsymbol{x})}\left(-ln\left(f(\boldsymbol{x})_y\right)\right)\right)^T \cdot$
    $\cdot \begin{bmatrix} \dots & g'\left(a^{(k)}(\boldsymbol{x})_j\right) & \dots \end{bmatrix}$



Backward propagation

The consistency of the forward and backward propagation methods

- The gradient can be estimated as:
  - $\frac{\partial f(x)}{\partial x} \approx \frac{f(x+\epsilon)-f(x-\epsilon)}{2\epsilon}$
  - where $f(x)$ is a loss function
  - x is a parameter
  - $f(x+\epsilon)$ is a loss value is the parameter x is increased by ϵ
  - $f(x-\epsilon)$ is a loss value is the parameter x is decreased by ϵ



Regularization

To implement the SGD method, we need:

- a regularizer function $\Omega(\boldsymbol{\theta})$ and a method to compute its gradient

L2 regularization

- $\Omega(\boldsymbol{\theta}) = \sum_k \sum_i \sum_j \left(W_{i,j}^{(k)}\right)^2 = \sum_k \left\|W^{(k)}\right\|_F^2$

Gradient

- $\nabla_{\boldsymbol{W}^{(k)}} \Omega(\boldsymbol{\theta}) = 2\boldsymbol{W}^{(k)}$

Can be applied on weights, but not biases

Regularization

To implement the SGD method, we need:

- a regularizer function $\Omega(\boldsymbol{\theta})$ and a method to compute its gradient

L1 regularization

- $\Omega(\boldsymbol{\theta}) = \sum_k \sum_i \sum_j \left| W_{i,j}^{(k)} \right|$

Gradient

- $\nabla_{\boldsymbol{W}^{(k)}} \Omega(\boldsymbol{\theta}) = sign\left(\boldsymbol{W}^{(k)}\right)$
- where $sign\left(\boldsymbol{W}^{(k)}\right) = 1_{W_{i,j}^{(k)} > 0} - 1_{W_{i,j}^{(k)} < 0}$

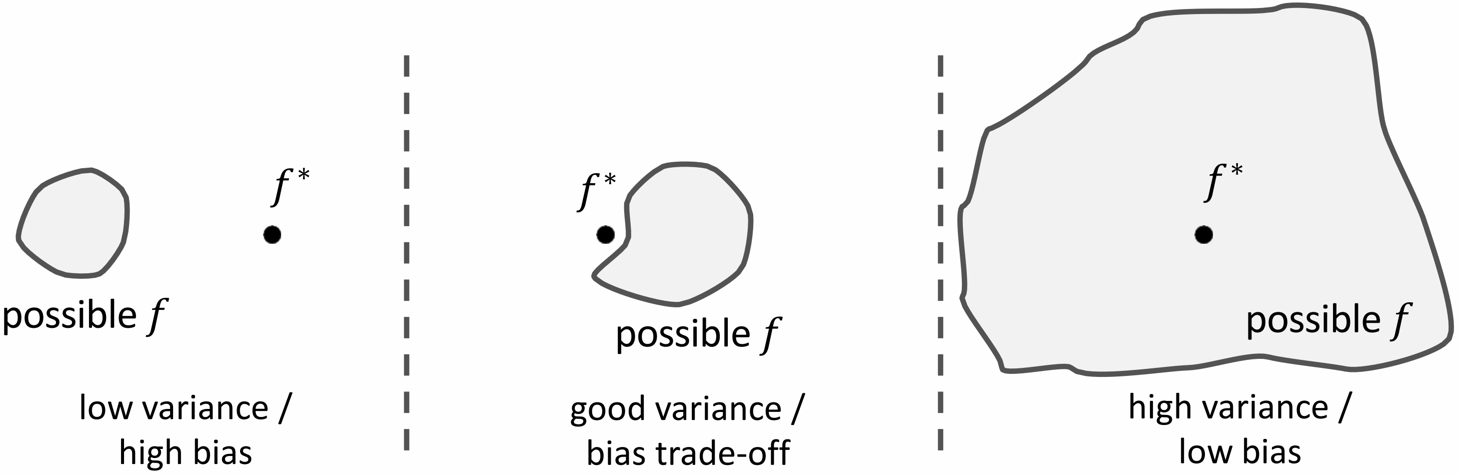
Can be applied on weights, but not biases

Unlike L2 it will push certain weights to be exactly 0

Regularization

Regularization parameter

- Variance of trained model – does it vary a lot if the training set changes
- Bias of trained model – is the average model close to the true solution f^*
- Variance / bias ratio is controlled by the λ multiplier



Parameters initialization

To implement the SGD method, we need:

- a method to initialize parameter values

Biases

- $b_i^{(k)} = 0$

Weights

- Can't initialize weights to 0 with $tanh$ activation
  - as all gradients would then be equal to 0
- Can't initialize all weights to the same value
  - all hidden units in a layer will always behave the same
- Need to break symmetry

Parameters initialization

To implement the SGD method, we need:

- a method to initialize parameter values

Biases

- $b_i^{(k)} = 0$

Weights

- Idea: sample around 0 but break symmetry
  - sample each $W_{i,j}^{(k)}$ uniformly from a range $[-b, b]$
  - where $b = \frac{\sqrt{6}}{\sqrt{H_k + H_{k+1}}}$, and H_k is the number of units in $h^{(k)}(x)$
  - other values of *b* could also work well

Model parameters and hyper-parameters

Stochastic gradient descent method allows training a neural network

- Choose model parameters ($W_{i,j}^{(k)}$, $b^{(k)}$)

Model hyper-parameters are parameters that define a neural network

- The neural network shape:
  - the number of hidden layers
  - the number of units in each hidden layer
- Regularization parameters
- Number of epochs
- Learning coefficients

How to select hyper-parameters?

Model dataset

Dataset

- Training dataset
  - used to train a model
- Validation dataset
  - used to select a model hyper-parameters
- Testing dataset
  - used to estimate the generalization of a trained neural network

Generalization is a behavior of the neural network on unseen examples

- the goal of the machine learning

Choosing hyper-parameters

Grid search

- Specify a list of possible values for each hyper-parameter
- Check all possible combinations of selected hyper-parameters
- Choose the best combination

Random search

- Specify a range for each hyper-parameter
- Sample a combination of hyper-parameters taking their ranges into an account
- Choose the best sample

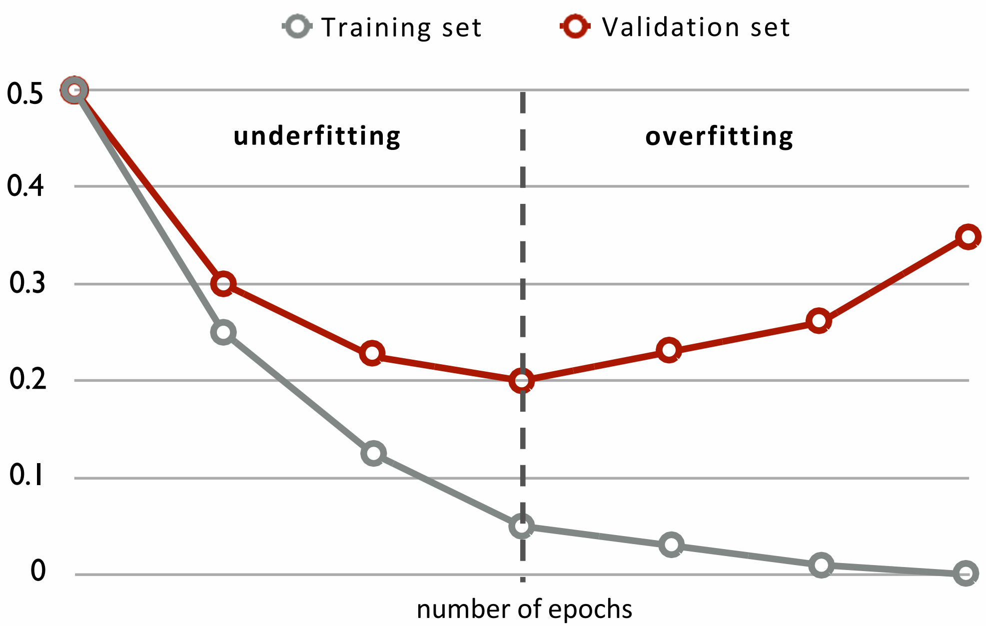
Validation set is used to estimate the neural network performance

Choosing hyper-parameters

Stop increasing a hyper-parameter when validation set error increases

Overfitting – model captured too much specific parameters of the training set and looses generalization

Underfitting – model has not captured enough training set parameters



Validation set is used to estimate the neural network performance

Model convergence

SGD convergence conditions

The learning coefficients α_t should satisfy the condition

- $\sum_{t=1}^{\infty} \alpha_t = \infty$
- $\sum_{t=1}^{\infty} \alpha_t^2 < \infty$

Decreasing strategy

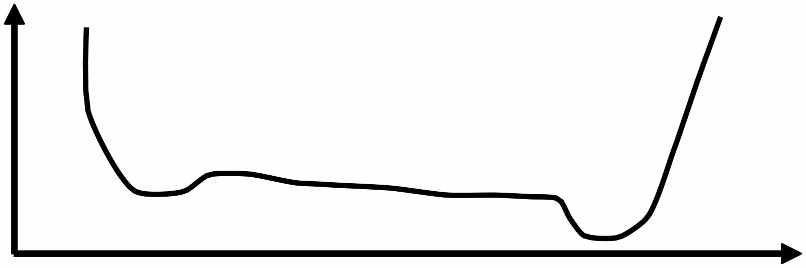
- $\alpha_t = \frac{\alpha}{1+\delta t}$
- $\alpha_t = \frac{\alpha}{t^{\delta}}, \delta \in (0.5; 1]$
- where α_t is a learning coefficient of the i^{th} epoch

It's advised to use the constant learning coefficient for first few epoches

Model convergence

SGD optimization

- Optimization can get stuck in local minima or plateau
  - there is no single global minimum



- The gradient is getting vanished with increasing the number of layers

Possible optimization strategies

- Use batch of examples instead of one when calculating a gradient
  - gradient is an average of batch gradients
- Use an exponential average of previous gradients
  - $\overline{\nabla_\theta^{(t)}} = \nabla_\theta l\left(f\left(x^{(t)}\right), y^{(t)}\right) + \beta \overline{\nabla_\theta^{(t-1)}}$

Neural Networks and Deep Learning

Neural networks are mathematical model of biological neural networks

They can be used for wide range of the categorization and decision-making tasks

There are other learning strategies as well

- Unsupervised learning
- Reinforcement learning
- etc.

There are various neural network models

- Convolutional neural networks
- Recurrent neural networks
- Long short-term memory networks
- etc.

Test

Lecture 13-14 Test

Please scan the code to start the test

THANK YOU
FOR YOUR TIME!

iT's MOre than a UNIVERSITY

Andrei Zhdanov
adzhdanov@itmo.ru