Machine Learning L+Pr

Béla J. Szekeres, PhD

Lecture 6

ELTE Faculty of Informatics, Szombathely, Hungary

Topics of this day

- Neural networks introduction
- Go into deeper
- Practice: PyTorch

Brief Syllabus

- Neural networks, activation functions, teaching methods, back propagation
- Convolutional Neural Network, transfer learning
- Regularization techniques: dropout, maxout, Relu, batch normalization
- Recurrent neural networks: LSTM, NLP, time series forecasting

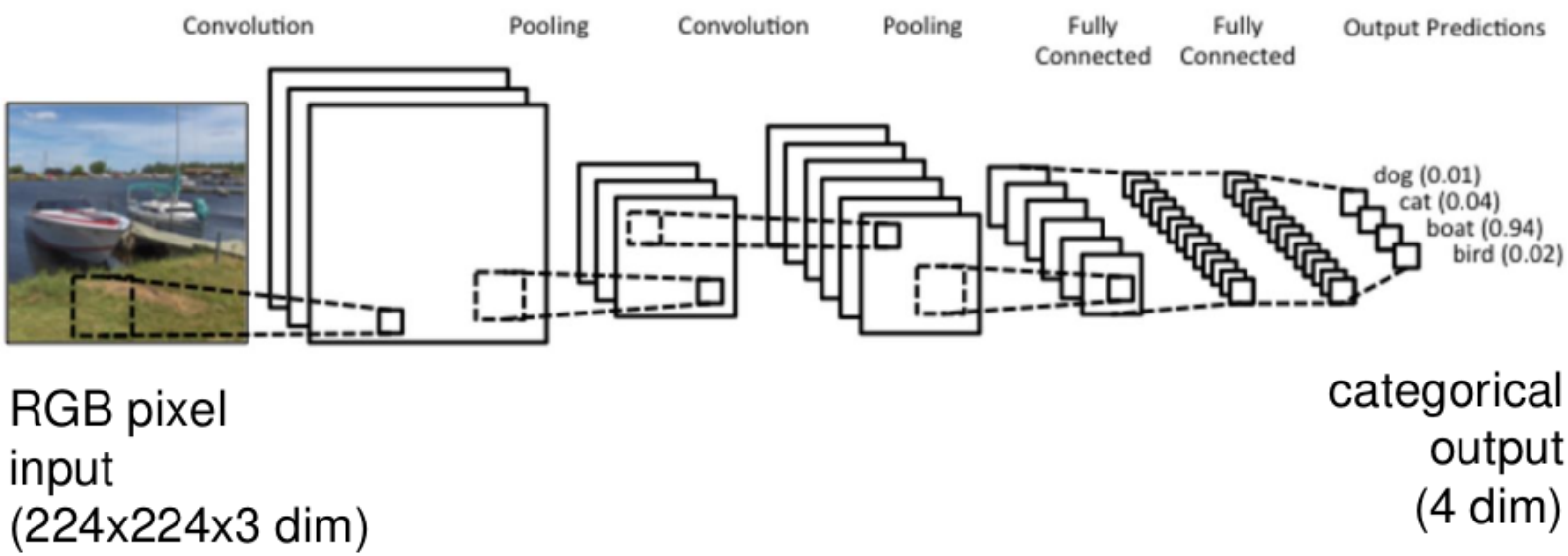
Introduction

Typical Applications

- Image classification
- Object detection
- Natural Language processing

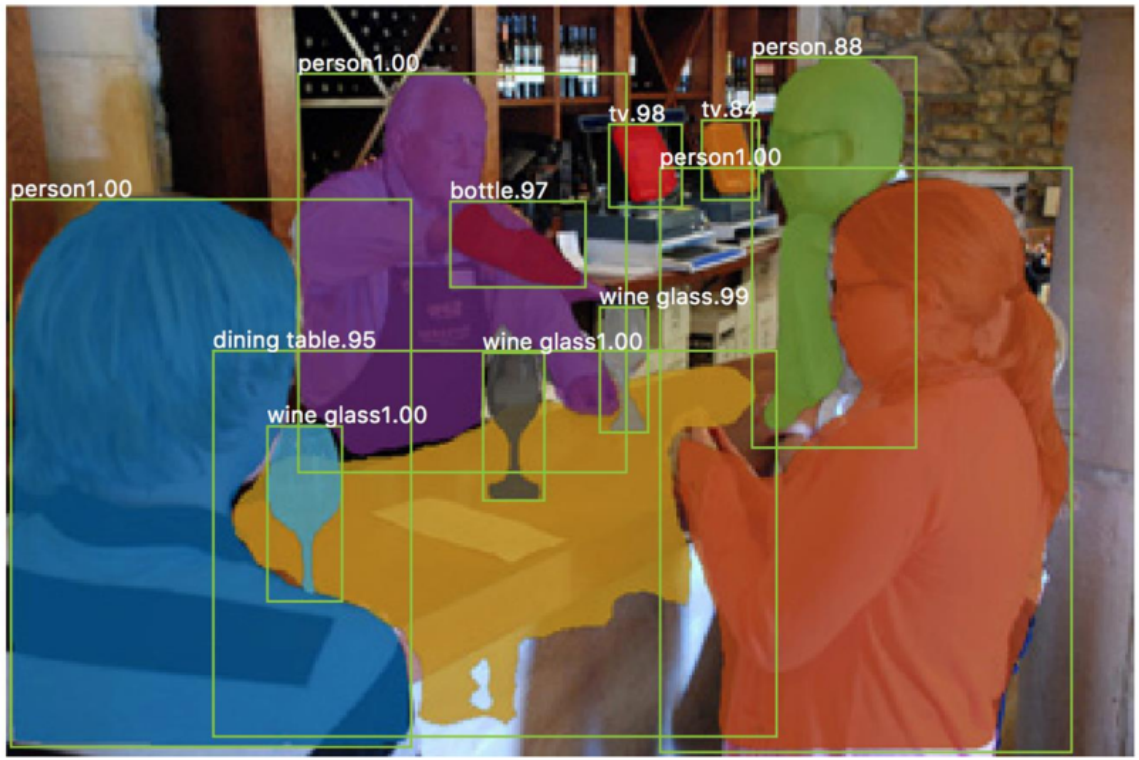
Image classification

Convolutional networks



Object detection

Convolutional networks



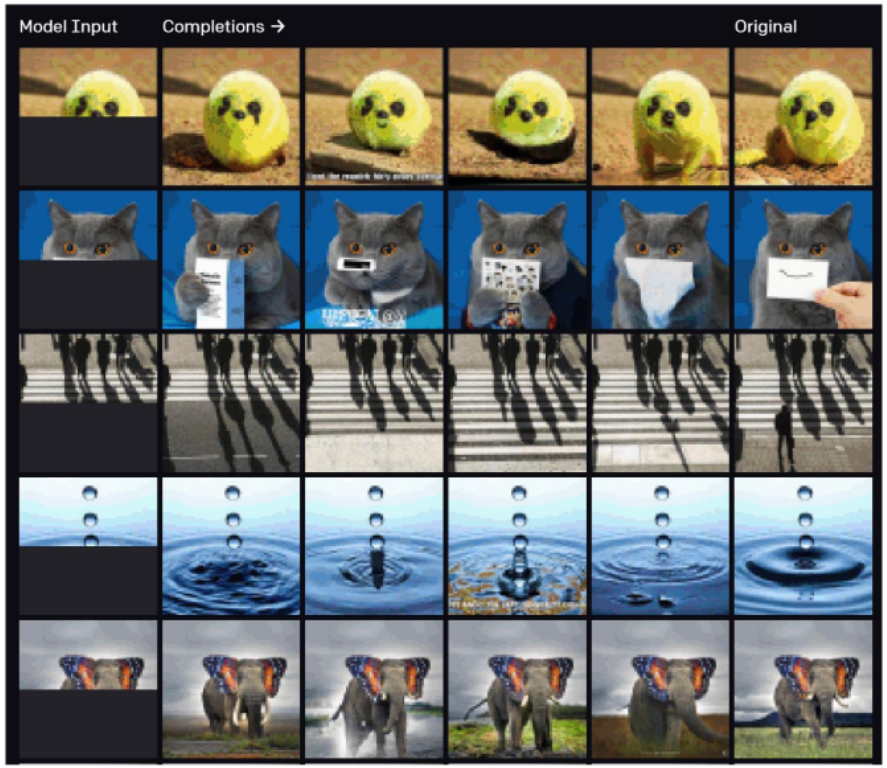
Natural Language Processing

Text2pic

https://arxiv.org/pdf/1605.05396.pdf

Image GPT, completions

Inpainting



https://openai.com/blog/image-gpt/

Pic2pic transformations

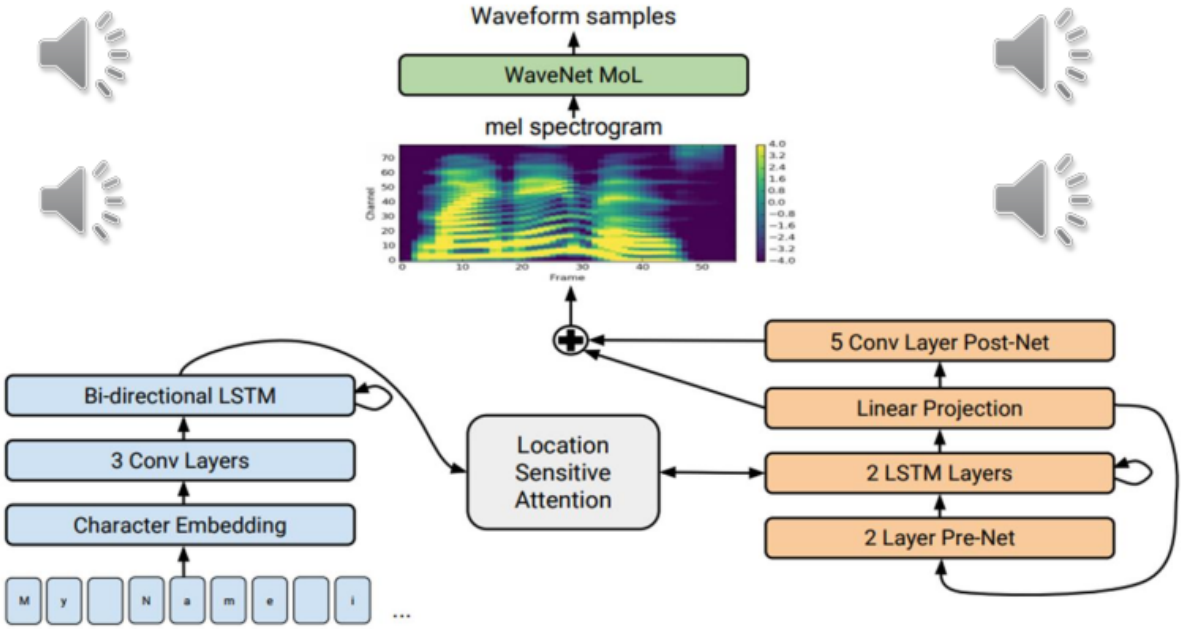
GAN

NLP

Speech synthesis

"She earned a doctorate in sociology at Columbia University."

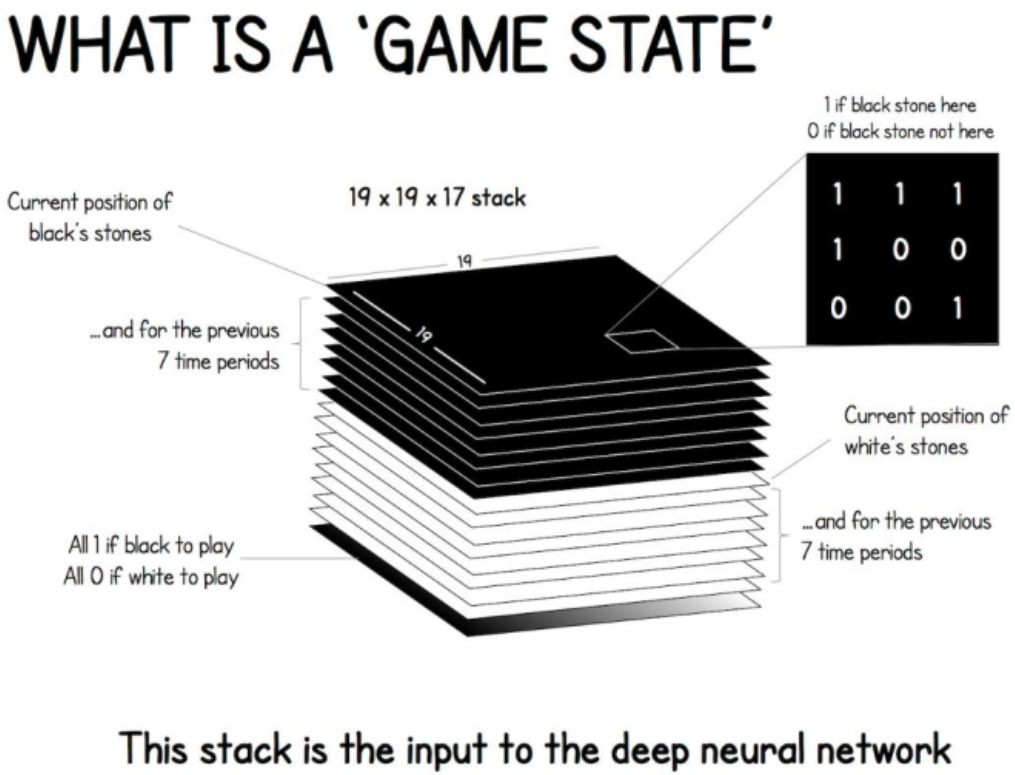
"George Washington was the first President of the United States."



https://arxiv.org/pdf/1712.05884.pdf

Reinforcement learning

AlphaGo and AlphaGo Zero



AI winters and summers

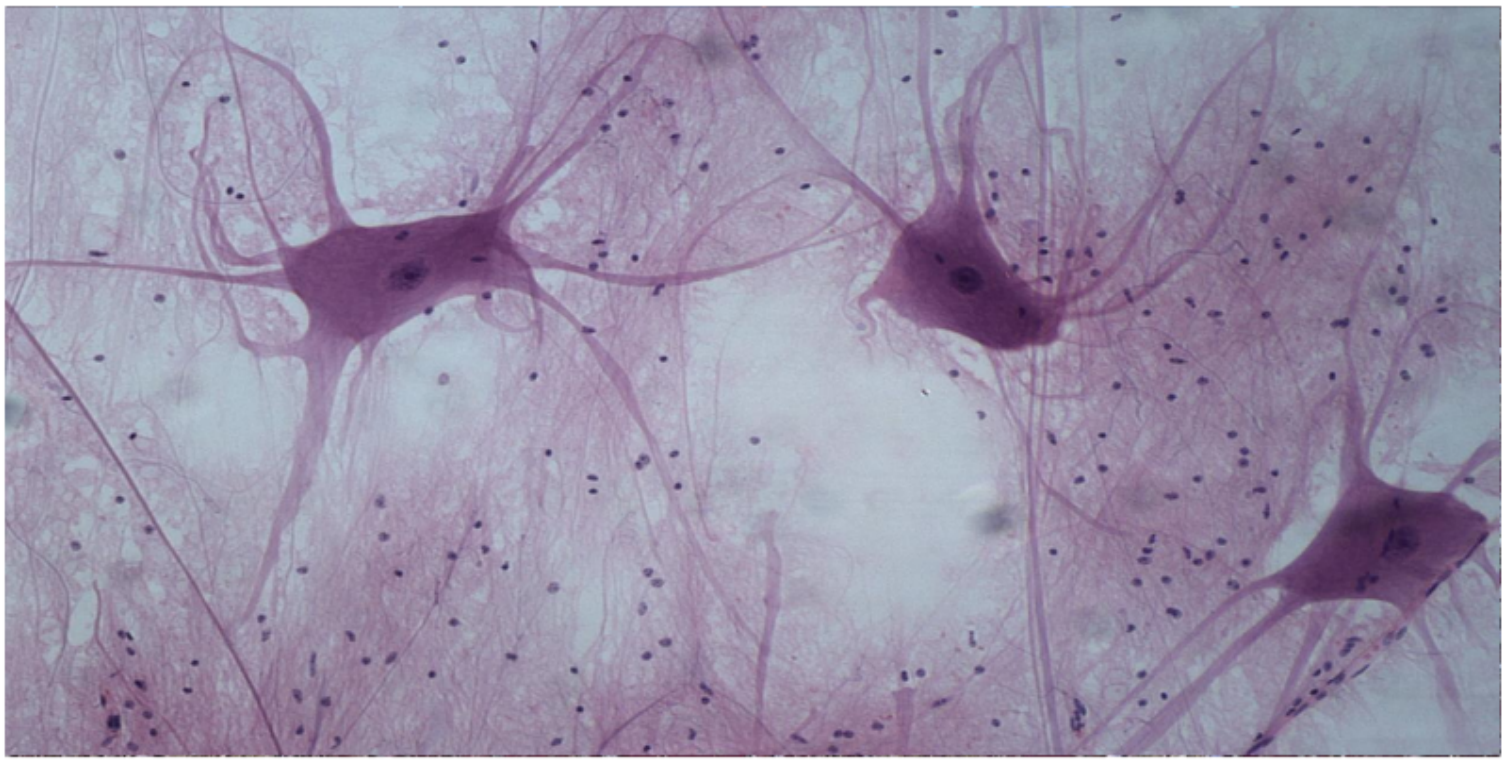
- 1956: Dartmouth meeting, NN has been already mentioned
- 1957: Perceptron (Rosenblatt)
- 1969: Minsky & Papert showed Perceptron is not very powerful
- 1980s: Multi-layer perceptron
- 1986: Back-propagation, hard to train more than 3 layers
- 1989: 1 hidden layer can do all, why deep?
- 2006: Breakthrough in training multilayered NN – RBM initialization
- from 2009: Game industry has pushed the growth of GPU's
- 2012: AlexNet is the winner of the ImageNet Large Scale Visual Recognition Competition

Today

- Artificial neuron, artificial neural network
- Feed-forward networks
- Activation functions
- Loss functions
- Teaching neural networks
- (Stochastic) Gradient Descent
- Batch, learning rate
- Back propagation
- Teaching/Validation/Testing data
- Training error and generalisation error

Neuron

Biological inspiration



Neuron

Mathematical definition

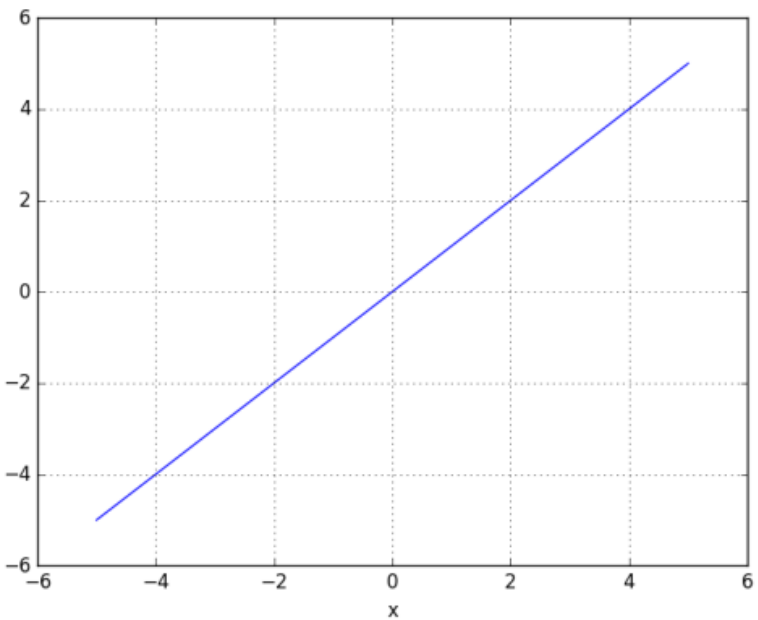
- They compute a linear weighted sum of their inputs: $a = b + \sum_i y_i w_i$
- The output is a nonlinear function of the total input: $y = f(a)$
- b is also called bias

What is f?

- Activation function

Activation functions

Identity



linear activation function $f(x) = x$

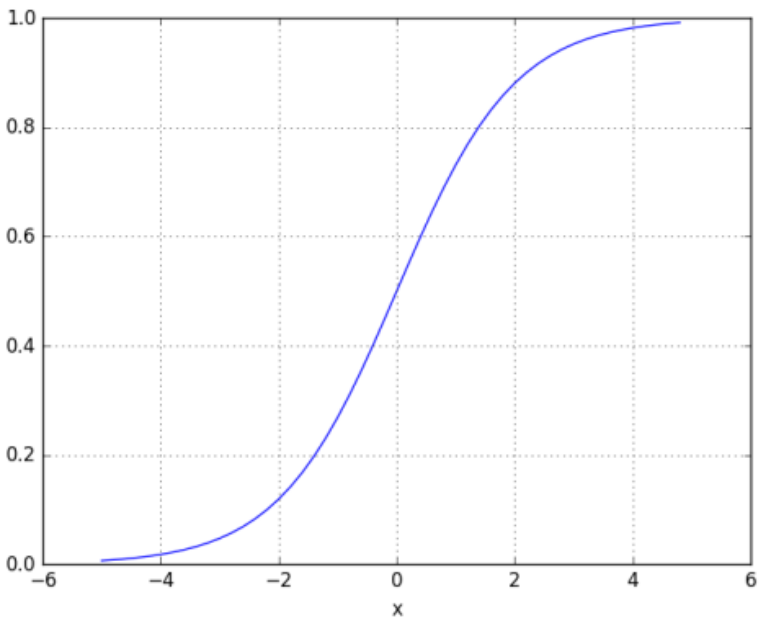
- it is the identity operator

basically: it means the function passes the signal through unchanged

- usually we do not change the input when dealing with the **input layer**: so we can say that the input layer has linear activation function

Activation functions

Sigmoid



Transforms the data

into the range $[0, 1]$ $f(x) = \frac{1}{1+\exp(-x)}$

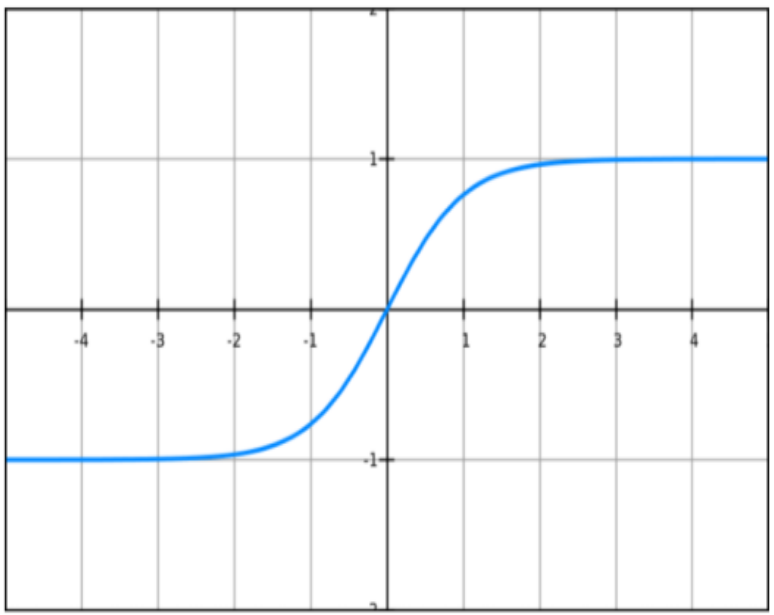
- interpret the results as probabilities

- it reduce extreme values and outliers

in the data without removing them

Activation functions

Hyperbolic tangent

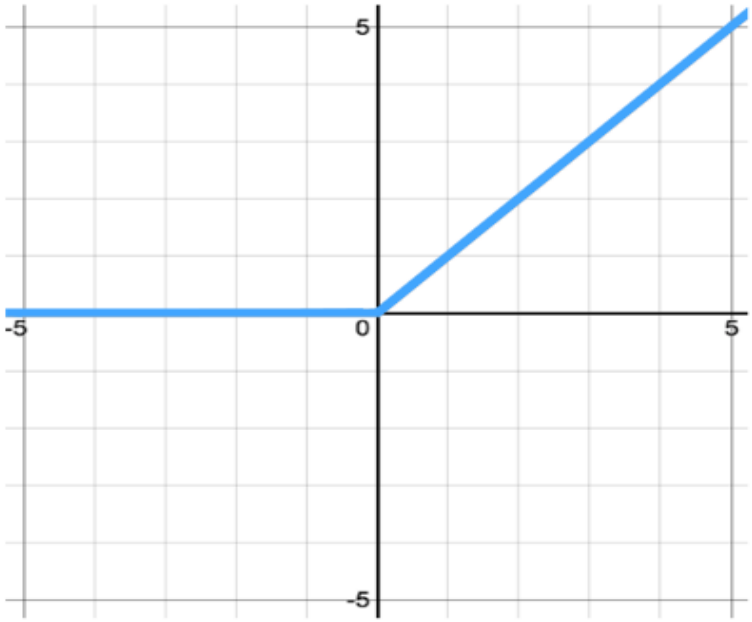


Transforms the data into the range $[-1, 1]$ $f(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$

- it can handle negative values as well

Activation functions

ReLU

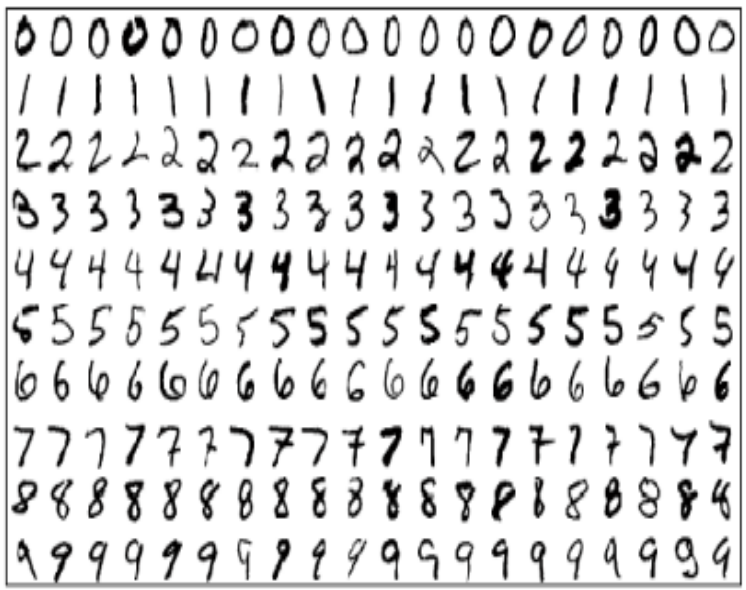


It activates a neuron only if the input is above a threshold $f(x) = \max(0, x)$

- most popular: the gradient is either zero or a constant (1.0), so it can solve the vanishing gradient problem

Activation functions

Softmax: $\text{softmax}(z)_j = \frac{e^{z_j}}{\sum_k e^{k}}, j = 1, \ldots, k, \quad \text{softmax} : \mathbb{R}^K \to \mathbb{R}^K$



Transforms the data into
the range $[-1, 1]$ in each dimension
-the sum of them is 1!
- we use the softmax activation
function in the last layer if we
want to solve a **multiclassification** task
- for
example recognize handwritten digits

We need nonlinearity!

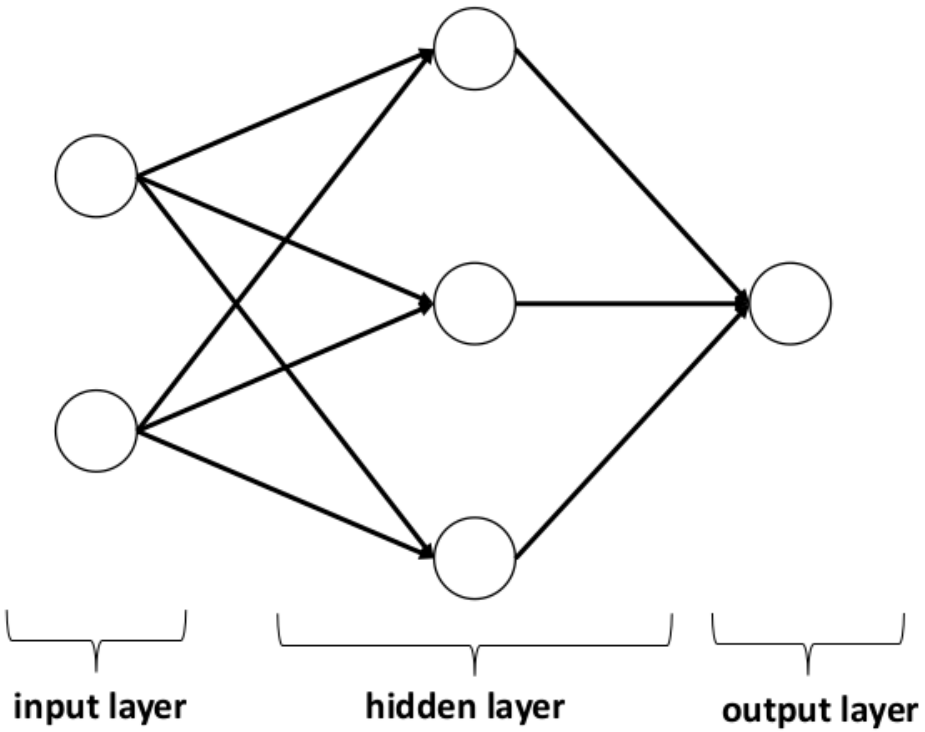
Activation functions

Identity	$f(x) = x$	$f'(x) = 1$
Binary step	$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)	$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1-f(x))$
TanH	$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan	$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear	$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus	$f(x) = \log_e(1+e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$
Bent identity	$f(x) = \frac{\sqrt{x^2+1}-1}{2} + x$	$f'(x) = \frac{x}{2\sqrt{x^2+1}} + 1$
SoftExponential	$f(\alpha, x) = \begin{cases} -\frac{\log_e(1-\alpha(x+\alpha))}{\alpha} & \text{for } \alpha < 0 \\ x & \text{for } \alpha = 0 \\ \frac{e^{\alpha x}-1}{\alpha} + \alpha & \text{for } \alpha > 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} \frac{1}{1-\alpha(\alpha+x)} & \text{for } \alpha < 0 \\ e^{\alpha x} & \text{for } \alpha \geq 0 \end{cases}$
Sinusoid	$f(x) = \sin(x)$	$f'(x) = \cos(x)$
Sinc	$f(x) = \begin{cases} 1 & \text{for } x = 0 \\ \frac{\sin(x)}{x} & \text{for } x \neq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x = 0 \\ \frac{\cos(x)}{x} - \frac{\sin(x)}{x^2} & \text{for } x \neq 0 \end{cases}$
Gaussian	$f(x) = e^{-x^2}$	$f'(x) = -2xe^{-x^2}$

Multilayer networks

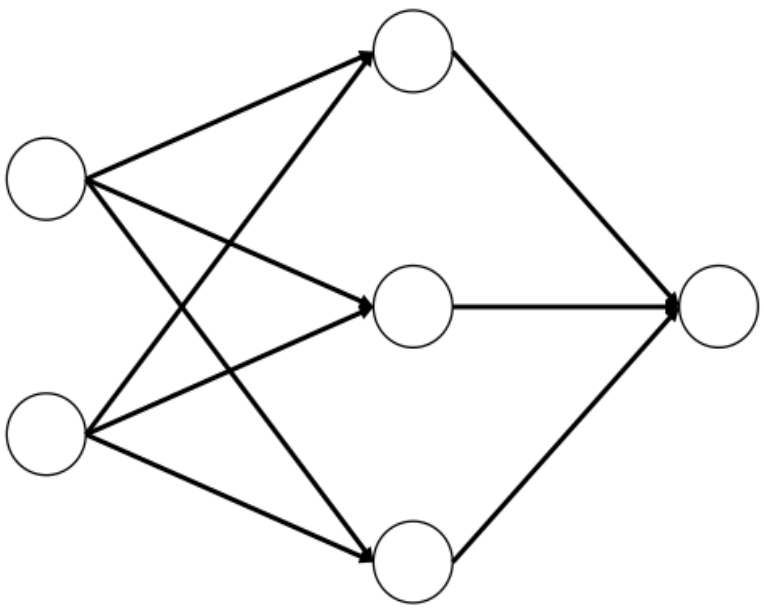
The structure

A simple example: 2 input, 3 hidden, 1 output neurons



Multilayer networks

A simple example

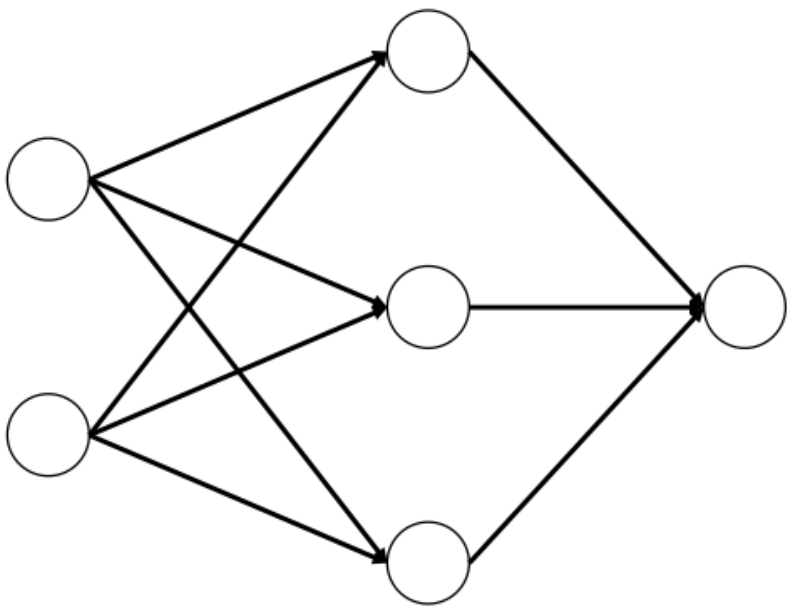


- every node is connected
to every node in the next layer
- a little change in the edge
weights results in change in the output
- **Training a**
network: means adjusting the weights
- hyperparameters

Exact definition?

Multilayer networks

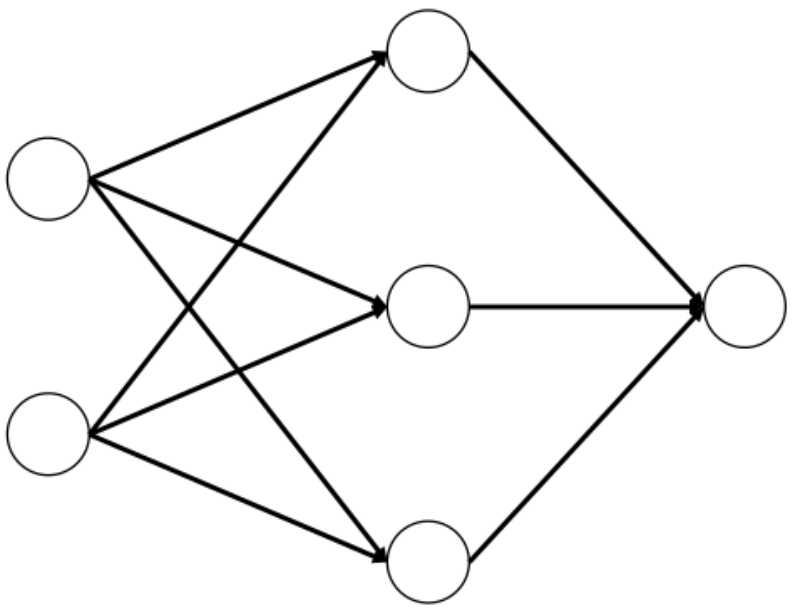
A simple example



- $a_j^{(i)}$: input value
of the j-th neuron in the i-th layer
- $y_j^{(i)}$: output value
of the j-th neuron in the i-th layer
(activation value)
- $W_{l,k}^{(i)}$: weights
between the i-th and the $i+1$-th layer
from the k-th (i-th
layer) to the l-th neuron (i+1-th layer)
- $b_j^{(i)}$: bias
in the j-th neuron in the i-th layer

Multilayer networks

Vectorization

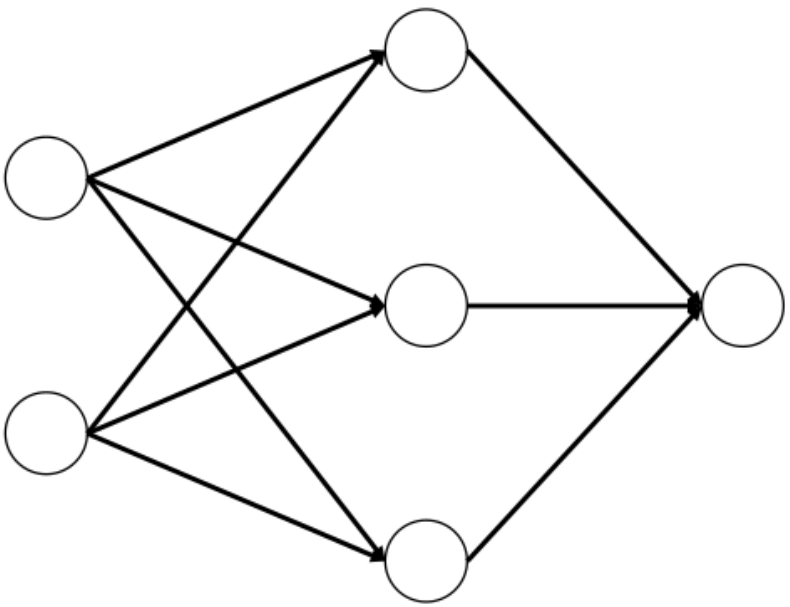


Collect to vectors (and matrices)

- $a_j^{(i)} \rightarrow a^{(i)}$
- $y_j^{(i)} \rightarrow y^{(i)}$
- $W_{l,k}^{(i)} \rightarrow W^{(i)}$
- $b_j^{(i)} \rightarrow b^{(i)}$

Multilayer networks

Calculating Network on $x \in \mathbb{R}^2$? Feeding forward example



- $a^{(1)} \in \mathbb{R}^2$: $a^{(1)} = x \quad \rightarrow y^{(1)} = a^{(1)}$
- $a^{(2)} \in \mathbb{R}^3$: $a^{(2)} = W^{(1)} y^{(1)} + b^{(2)}$

$\rightarrow y^{(2)} = f(a^{(2)})$

- $a^{(3)} \in \mathbb{R}^1$: $a^{(3)} = W^{(2)} y^{(2)} + b^{(3)}$

$\rightarrow y^{(3)} = f(a^{(3)})$

x is the input of the network

$y^{(3)}$ is the output of the network

How to change the weights?

Multilayer networks

We need a loss function!

Let y denote the output of the network (in the previous example $y^{(3)}$)

- **Regression** MSE: $L(W) = \frac{1}{N}\sum_{i=1}^{N} \|y(x^{(i)}) - t^{(i)}\|^2$
- **Binary classification** BCE: $L(W) = \frac{1}{N}\sum_{i=1}^{N} t \log y + (1-t)\log(1-y)$ (only 1 output)
- **Multi classification** MCE: $L(W) = \frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} t_k \log y_k$ (K classes)

We have to calculate the gradients $\frac{\partial L}{\partial W}$ and $\frac{\partial L}{\partial b}$ and apply for example gradient descent

$$W_{new}^{(i)} = W_{old}^{(i)} - \alpha \frac{\partial L}{\partial W^{(i)}} \quad b_{new}^{(i)} = b_{old}^{(i)} - \alpha \frac{\partial L}{\partial b^{(i)}}$$

Back propagation example

Consider the example with MSE loss

on the (x, t) input-output pair

1. Calculating $\frac{\partial L}{\partial a^{(3)}} = f'(a^{(3)}) \odot (y^{(3)} - t) \quad \rightarrow \quad \delta^{(3)}$
2. Calculating $\frac{\partial L}{\partial a^{(2)}} = f'(a^{(2)}) \odot W^{(2),T} \delta^{(3)} \quad \rightarrow \quad \delta^{(2)}$

Apply again the chain rule

We know that $a^{(2)} = W^{(1)} y^{(1)} + b^{(2)}$ and $a^{(3)} = W^{(2)} y^{(2)} + b^{(3)}$

1. Calculating $\frac{\partial L}{\partial W^{(2)}} = \frac{\partial L}{\partial a^{(3)}} \frac{\partial a^{(3)}}{\partial W^{(2)}} \quad \rightarrow \quad \frac{\partial L}{\partial W^{(2)}} = \delta^{(3)} y^{(2),T}$
2. Calculating $\frac{\partial L}{\partial W^{(1)}} = \frac{\partial L}{\partial a^{(2)}} \frac{\partial a^{(2)}}{\partial W^{(1)}} \quad \rightarrow \quad \frac{\partial L}{\partial W^{(1)}} = \delta^{(2)} y^{(1),T}$

Back propagation example

Consider the example with MSE loss

on the (x, t) input-output pair

1. Calculating $\frac{\partial L}{\partial a^{(3)}} = f'(a^{(3)}) \odot (y^{(3)} - t) \quad \rightarrow \quad \delta^{(3)} = \frac{\partial L}{\partial b^{(3)}}$
2. Calculating $\frac{\partial L}{\partial a^{(2)}} = f'(a^{(2)}) \odot W^{(2),T} \delta^{(3)} \quad \rightarrow \quad \delta^{(2)} = \frac{\partial L}{\partial b^{(2)}}$

Apply again the chain rule

We know that $a^{(2)} = W^{(1)} y^{(1)} + b^{(2)}$ and $a^{(3)} = W^{(2)} y^{(2)} + b^{(3)}$

1. Calculating $\frac{\partial L}{\partial W^{(2)}} = \frac{\partial L}{\partial a^{(3)}} \frac{\partial a^{(3)}}{\partial W^{(2)}} \quad \rightarrow \quad \frac{\partial L}{\partial W^{(2)}} = \delta^{(3)} y^{(2),T}$
2. Calculating $\frac{\partial L}{\partial W^{(1)}} = \frac{\partial L}{\partial a^{(2)}} \frac{\partial a^{(2)}}{\partial W^{(1)}} \quad \rightarrow \quad \frac{\partial L}{\partial W^{(1)}} = \delta^{(2)} y^{(1),T}$

Summary

Initialize the weights

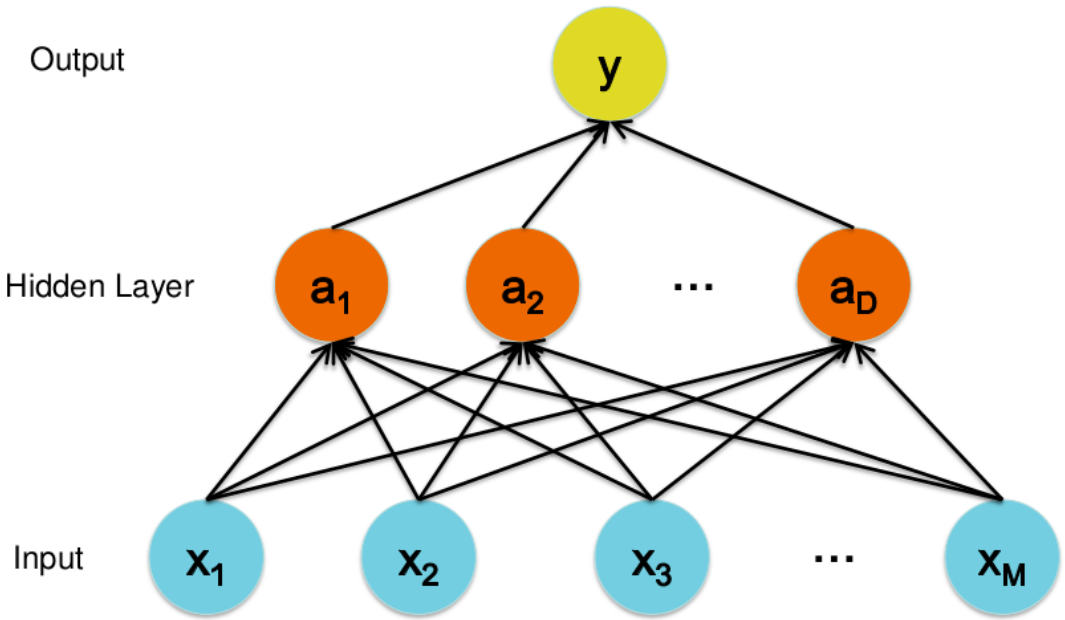
Iterative process:

1. Calculate the network
2. Modify the weights (for example by gradient descent)

How powerful can be a NN?

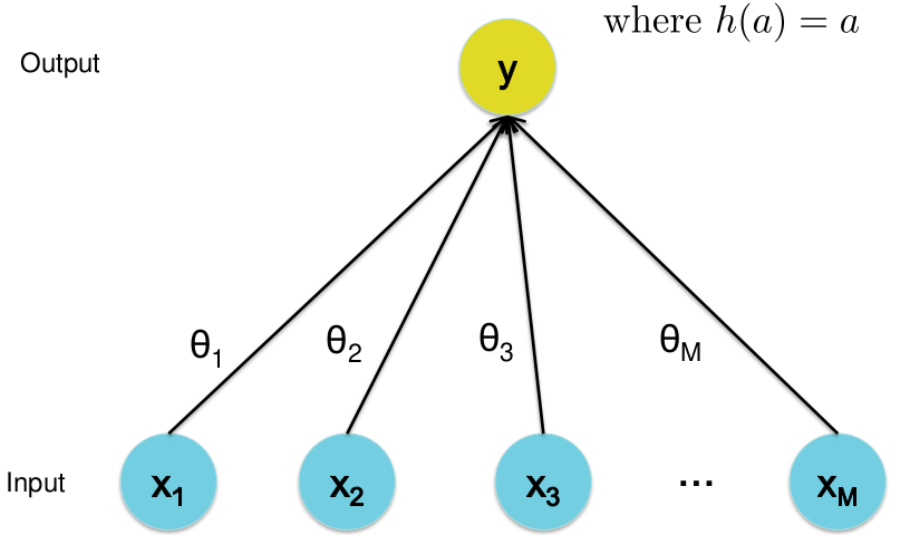
- UAT:
  `https://en.wikipedia.org/wiki/Universal_approximation_theorem`
- Still under development

Neural network



Linear Regression

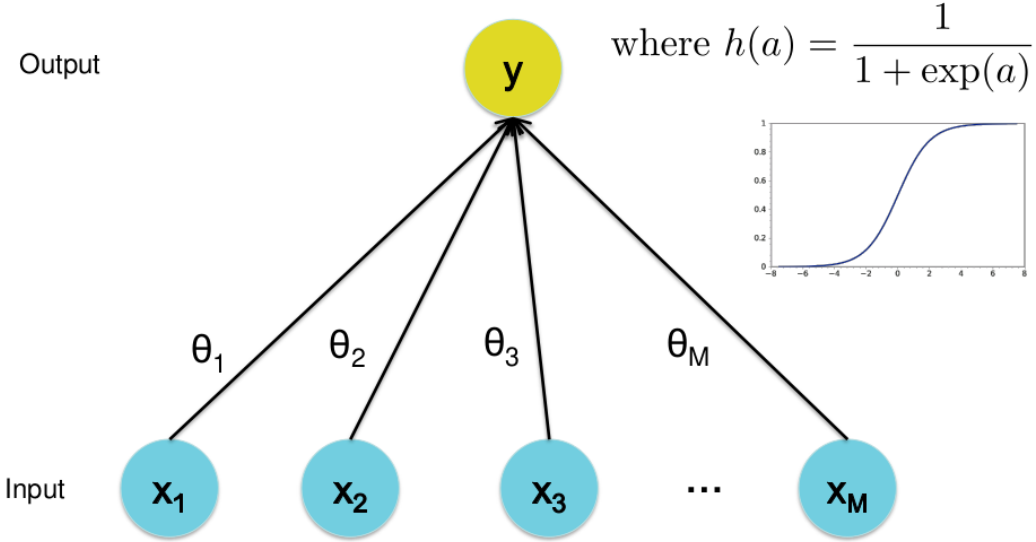
$$y = f_{\boldsymbol{\theta}}(\boldsymbol{x}) = h(\boldsymbol{\theta} \cdot \boldsymbol{x})$$
$$\text{where } h(a) = a$$



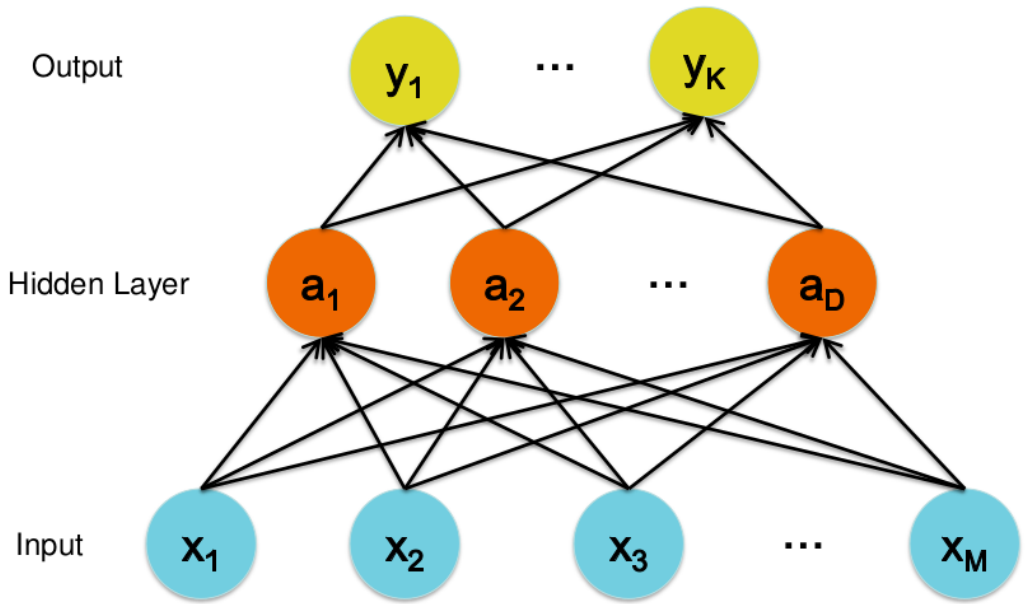
Logistic Regression

$$y = f_{\boldsymbol{\theta}}(\boldsymbol{x}) = h(\boldsymbol{\theta} \cdot \boldsymbol{x})$$

$$\text{where } h(a) = \frac{1}{1 + \exp(a)}$$



Multi-output



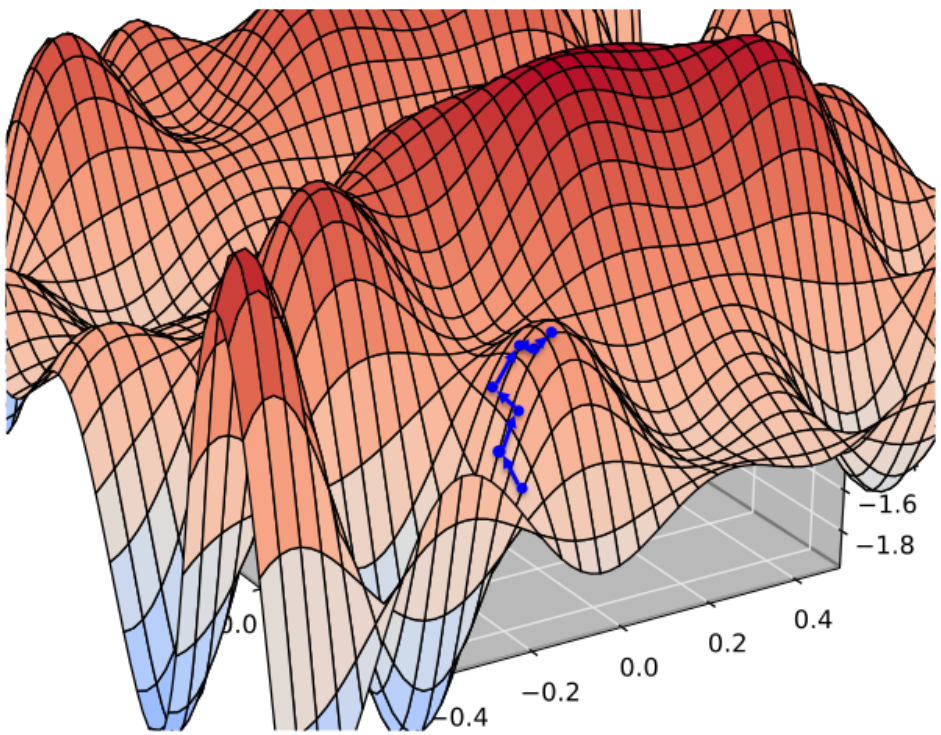
Deeper Networks

- We have several layers (more than 3 usually)
- Breakthrough in 2006: 8-layer network

What can go wrong?

- Gets stuck in local optima
  - Nonconvex loss function
  - usually start at a bad point parameter space
- gradient is progressively getting more weak (deeper in the NN) during doing back propagation layer by layer (backwards) $\rightarrow$ Vanishing gradients

Nonconvexity



- stochastic

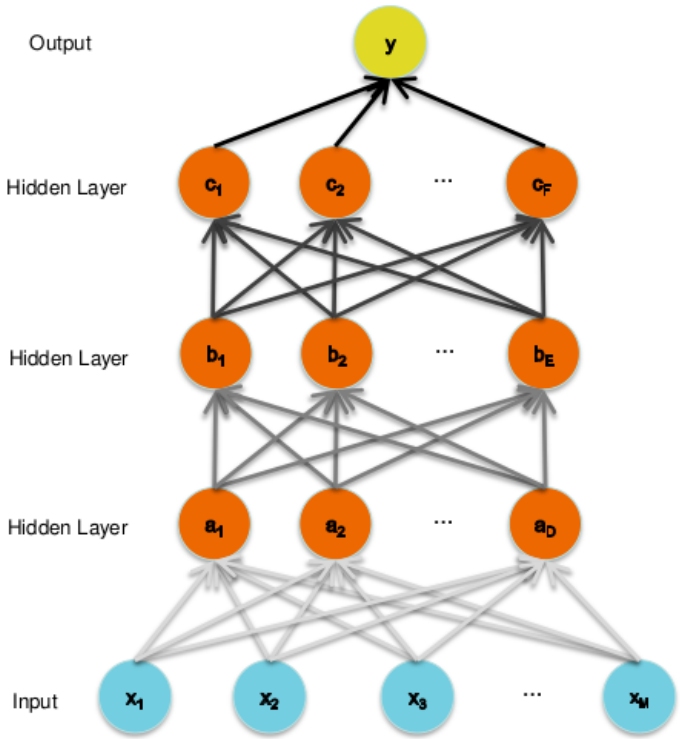
gradient descent climbs

to the top of the nearest hill

- which might not

lead to the top of the mountain

Vanishing gradients



- The gradient for an edge at the base of the network depends on the gradients of many edges above it
- The chain rule multiplies many of these partial derivatives together

Fine-tuning the learning process

- Control learning rate
- Momentum method: use the old values of parameters
- Use the second momentum of the gradients

Optimizers

- Nesterov momentum
- Adaptive Gradient Algorithm (AdaGrad)
- Root Mean Square Propagation (RMSProp)
- **Adam**

Mini-batch gradient method

We divide our dataset into mini-batches with (almost) the same size

Mini-batch gradient algorithm:

for mbatch in mini-batches:

1. calculate the network on mbatch
2. calculate the gradient on mbatch
3. update the weights

- if BATCH_SIZE == size of the dataset: (batch) gradient descent
- if 1< BATCH_SIZE < size of the dataset: mini-batch gradient descent
- if BATCH_SIZE == 1: stochastic gradient descent

Softwares

We get these softwares as a gift (free)

	Restrictions	Wrapper	Architectures	Notes
~~Chainer~~	~~Both feed forward and recurrent nets~~	~~Python~~	~~Multicore(?)/ CUDA~~	~~Multiple optimization~~
~~GraphLab~~	~~Feed Forward: CNN, DBN~~	~~Python/C++~~	~~Multicore/CUDA/ distributed~~	~~Compact, Hadoop~~
~~Caffe~~	~~Feed Forward: CNN, DBN~~	~~Python/Matlab~~	~~Multicore/CUDA~~	~~Blob~~
~~Theano~~	~~--~~	~~Python~~	~~Multicore/CUDA~~	~~Multiple optimization~~
TensorFlow & Keras	--	Python/C++	Multicore/CUDA/ distributed	Graphical interface and multiple optimization
MXNet	??	C++, Python, Julia, Matlab, JavaScript, R, ...	Multicore/CUDA/ distributed	Several languages, platforms supp.
PyTorch	--	Python	Multicore/CUDA	Developed for computer vision, now also for NLP
JAX/FLAX	??	Python/NumPy	GPU/TPU	

Features

- DL needs and can use more data for the models than ever before
- DL require more computing capacity ever before (GPU, spec. hardwares, like TPU)
- Sophisticated solutions based on heuristic experiments of long years (architectures, optimization methods)
- Prefabricated models
- Nearly all progress is public, open source software

Why go deeper?

Neural Nets (One hidden layer)	Deep Networks (Two or more hidden layers)
• Already universal function approximators	• Can be representationally efficient • Fewer computational units for the same function
• Can represent non-linear combinations of the input features	• Might allow for a hierarchy • Allows non-local generalizations
• Work well	• Have been shown to work even better (vision, audio, NLP, etc.)