practical deep learning

artificial neural networks

Alex Honchar
University of Verona

Day 2 goals

- You understand artificial neural networks structure
- You understand backpropagation algorithm (the most important thing)
- You can train your own neural network and continuously improve performance
- You can use Python **Keras** framework

How it learns? (machine learning)

Program is said to learn from **experience E** (data set with pictures) with respect to some class of **tasks T** (logistic regression) and performance **measure P** (mean squared error) if performance **improves** (with gradient descent iterations)

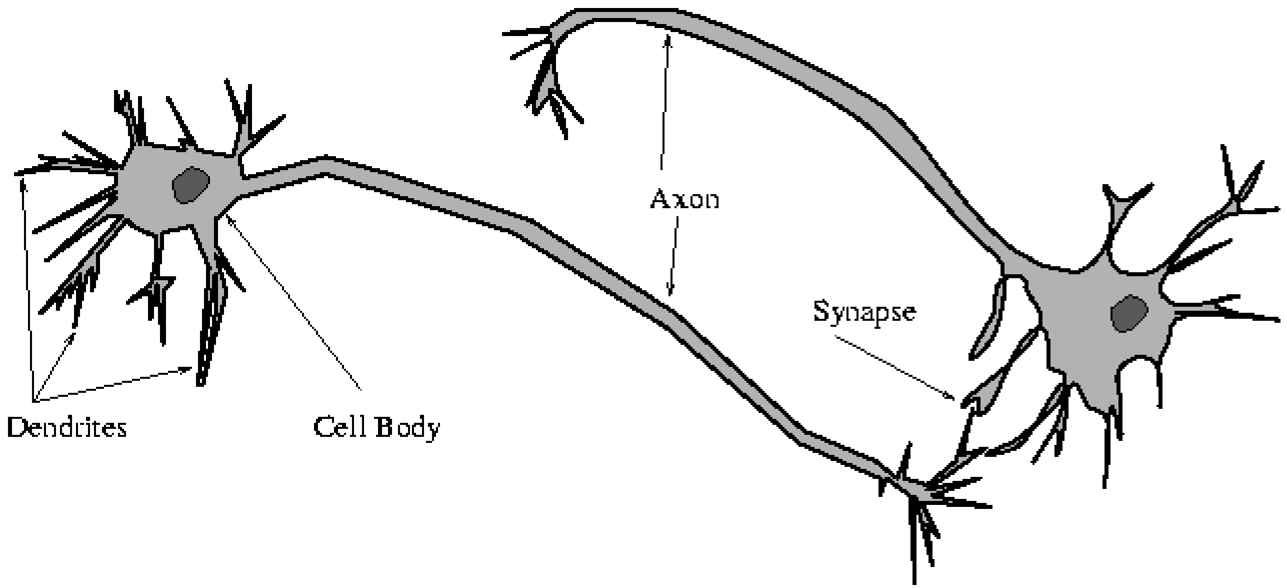
Mitchell, 1997

How it learns? (deep learning)

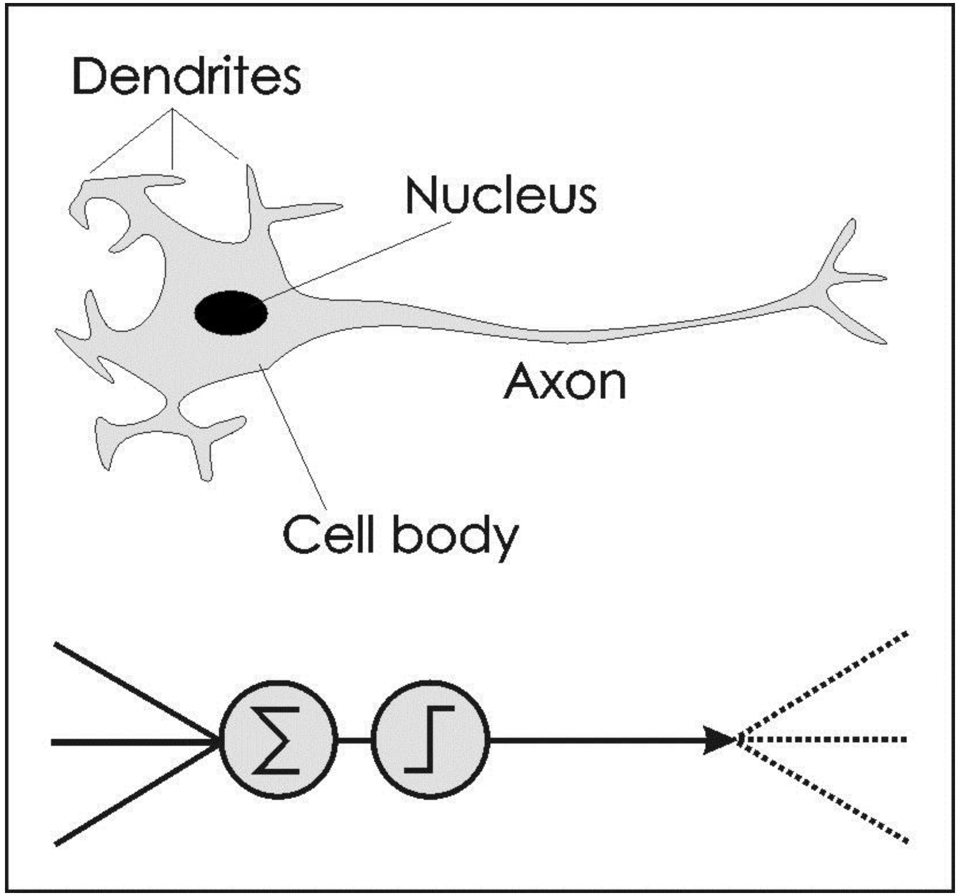
Program is said to learn from **learned representation R** (convolutional layers) of **experience E** (data set with pictures) with respect to some class of **tasks T** (logistic regression) and performance **measure P** (mean squared error) if performance **improves** (with gradient descent iterations)

biological neurons

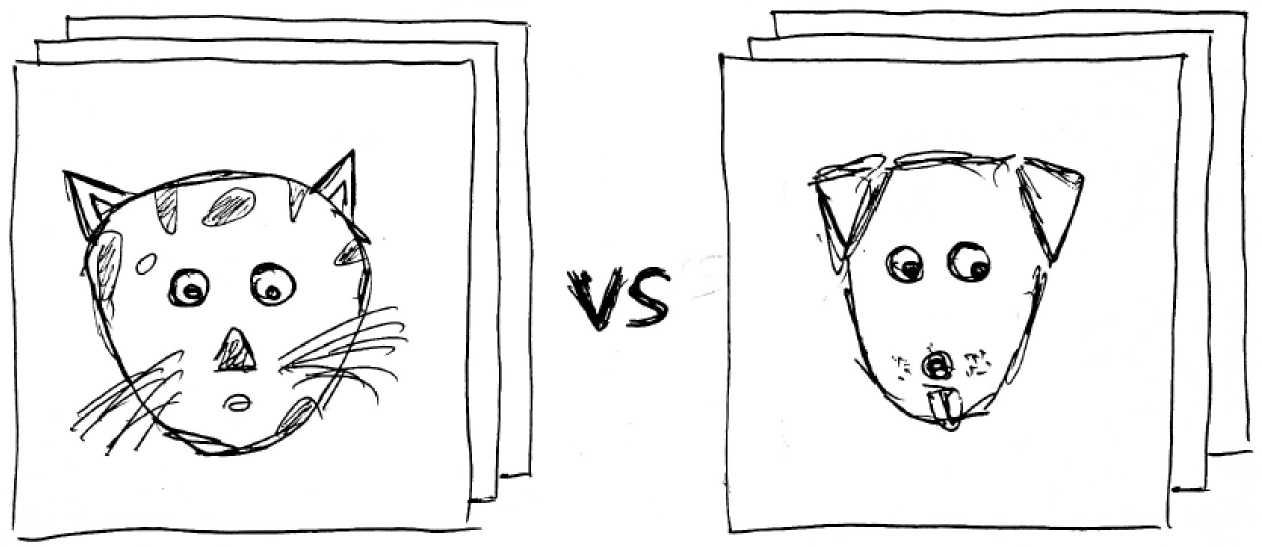
Biological neuron



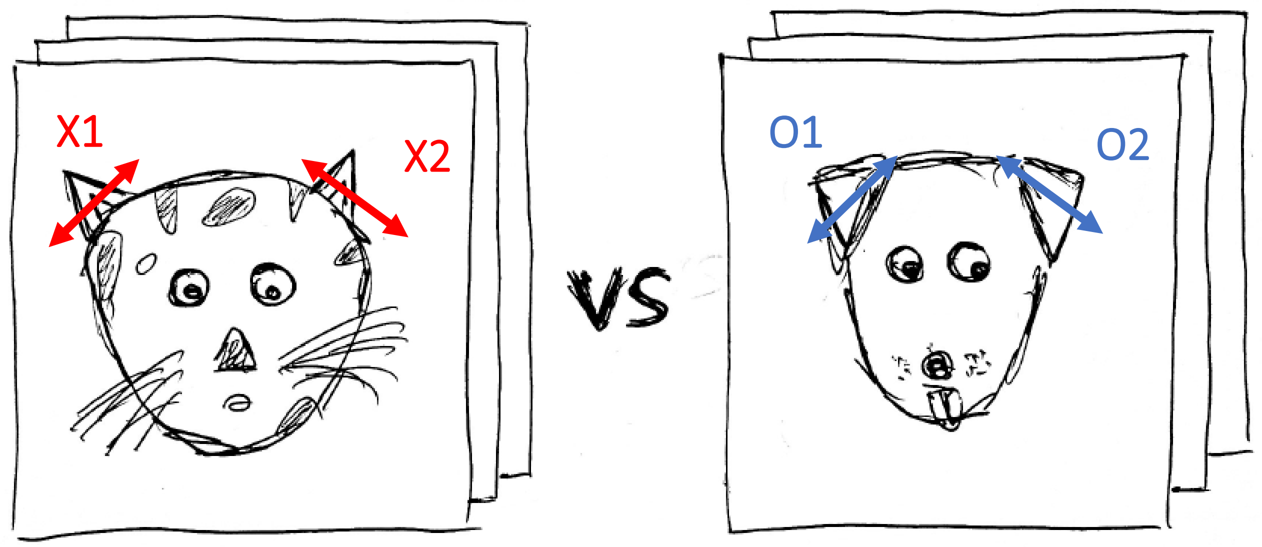
Biological neuron



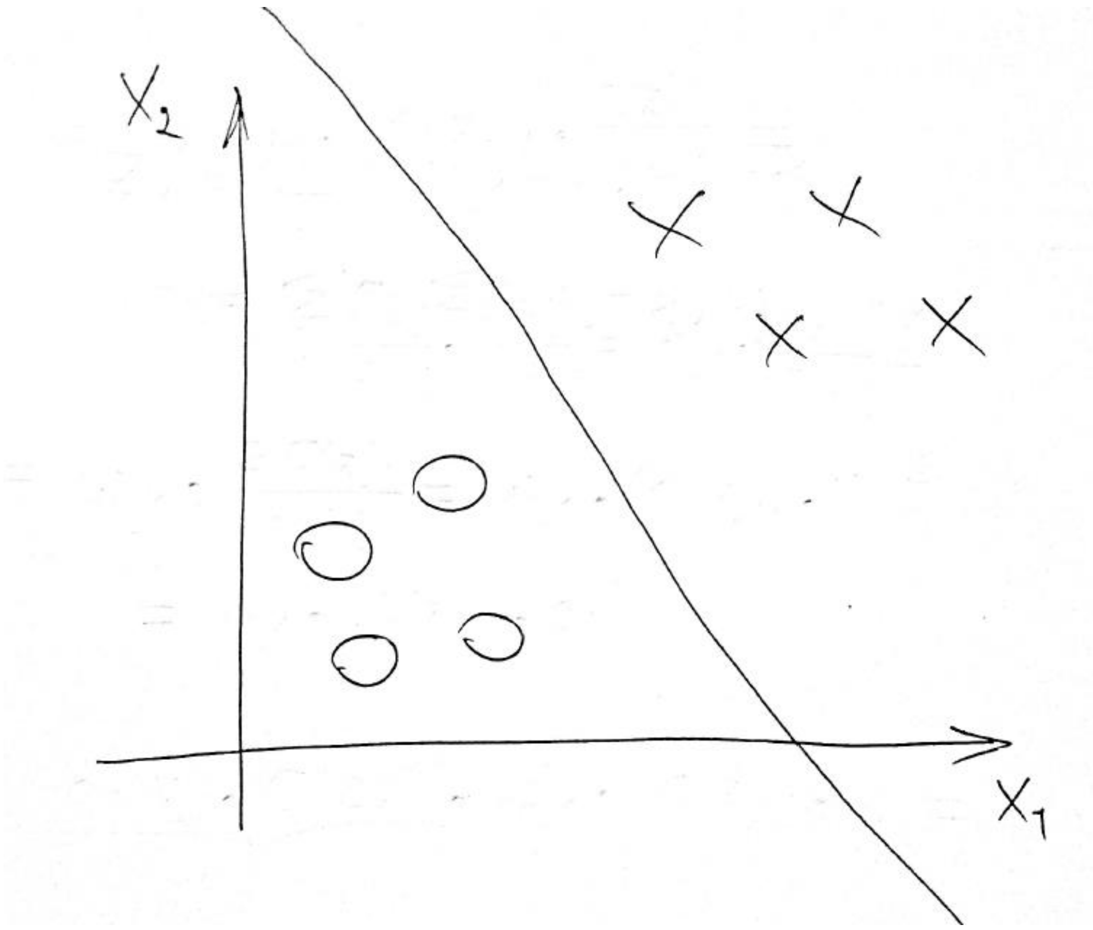
Cats vs Dogs



Cats vs Dogs

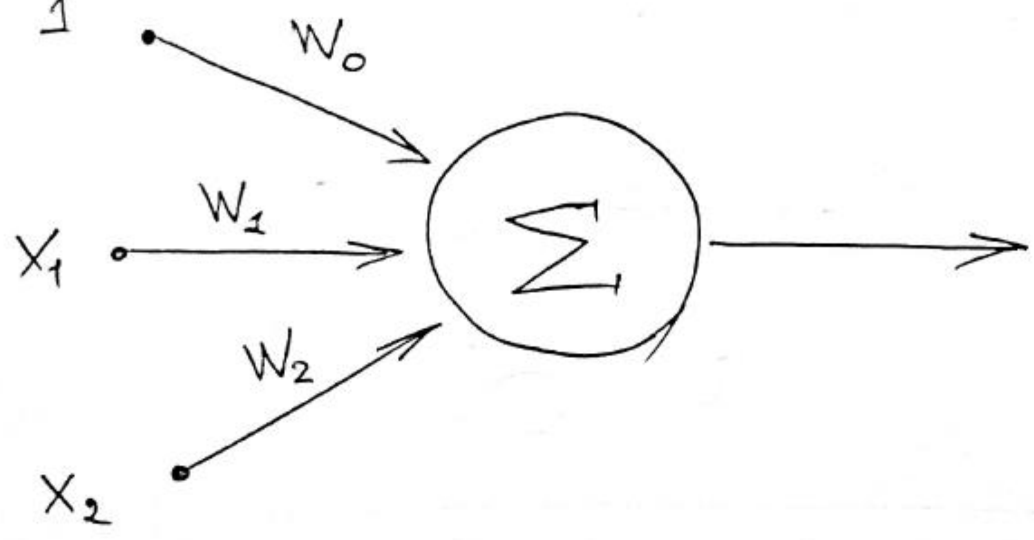


Cats vs Dogs



Logistic regression

$$W_0 + W_1 x_1 + W_2 x_2$$



Logistic regression

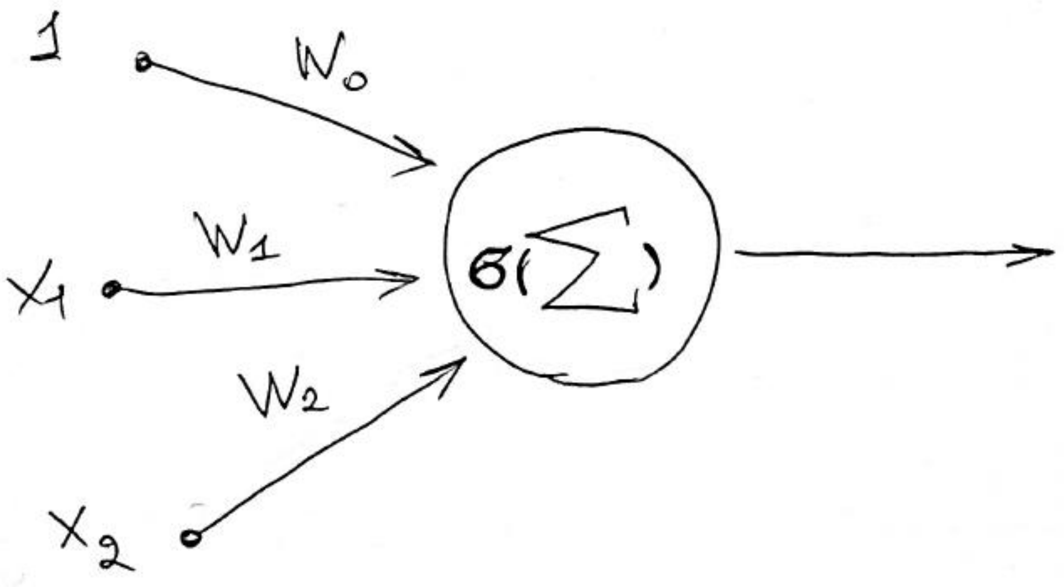
$$h_w(x) \in [0, 1]$$

$$h_w(x) = g\left(W^T x\right)$$

$$g(z) = \frac{1}{e^{-z} + 1} \Longrightarrow h_w(x) = \frac{1}{1 + e^{-W^T x}}$$

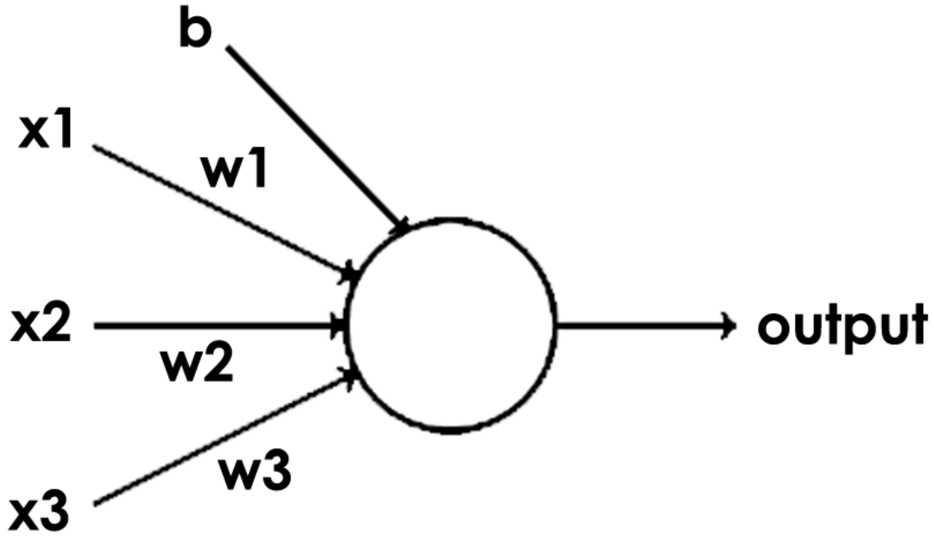
Logistic regression

$$\sigma(W_0 + W_1 x_1 + W_2 x_2)$$



artificial neurons

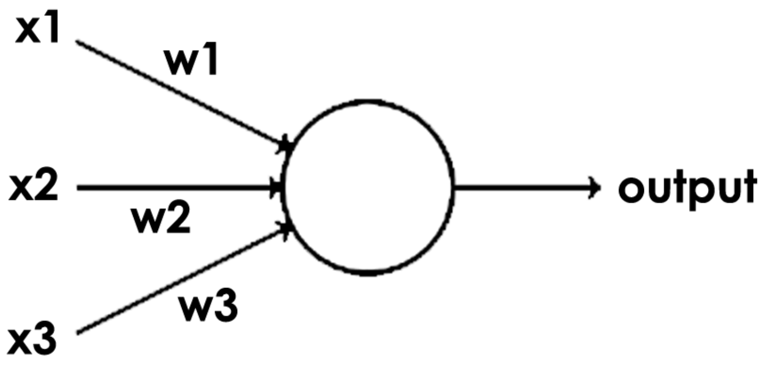
Logistic regression -> Artificial neuron



$$\text{output} = \begin{cases} 1, & \sum_i w_i * x_i + b > 0 \\ 0, & \sum_i w_i * x_i + b \leq 0 \end{cases}$$

We just move treshold to the right and call it b (bias) and we get Rozenblatt's perceptron

Logistic regression -> Artificial neuron



Real-world problem:

- Neuron deciding going to party or not (parameters X1 – X3)
- If event Xi happens, it equals 1, otherwise 0
- Calculate the output on the formula – 1 – party all the night, 0 – staying at home

X1 – you have important exam tomorrow
X2 – free drinks on the party
X3 – girl of your dream asking you to go out tonight

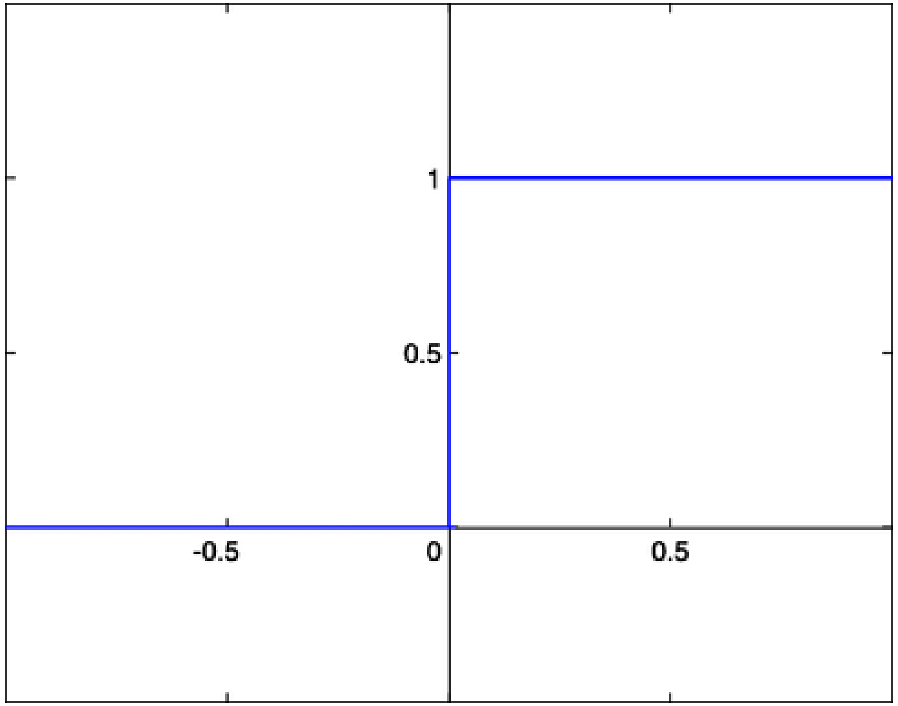
W1 = -6
W2 = 3
W3 = 4

$$output = \begin{cases} 1, \sum_i w_i * x_i > threshold \\ 0, \sum_i w_i * x_i \leq threshold \end{cases}$$

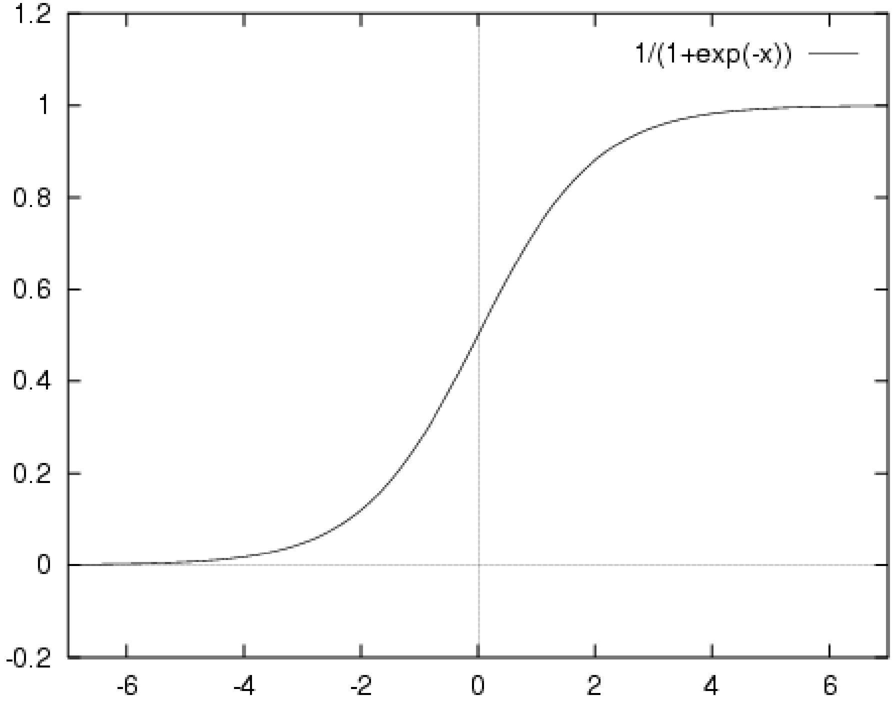
$threshold$ = 0

1. What happens if we don't have an exam and we have free drinks?
2. What happens if exam tomorrow, but free drinks are so tempting?
3. What happens if you have an exam, but free drinks and you dream girl are calling you?

Activation functions



$$\sigma(z) = \begin{cases} 1, & z > 0 \\ 0, & z \leq 0 \end{cases}$$



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

Sigmoidal neuron

We had perceptron

$$\text{output} = \begin{cases} 1, \sum_i w_i * x_i + b > 0 \\ 0, \sum_i w_i * x_i + b < 0 \end{cases}$$

$$z = \sum_i w_i * x_i + b$$

We turn into sigmoidal

$$\text{output} = \begin{cases} 1, \sigma(z) > 0.5 \\ 0, \sigma(z) \leq 0.5 \end{cases}$$

What is $\boldsymbol{\sigma(z)}$:

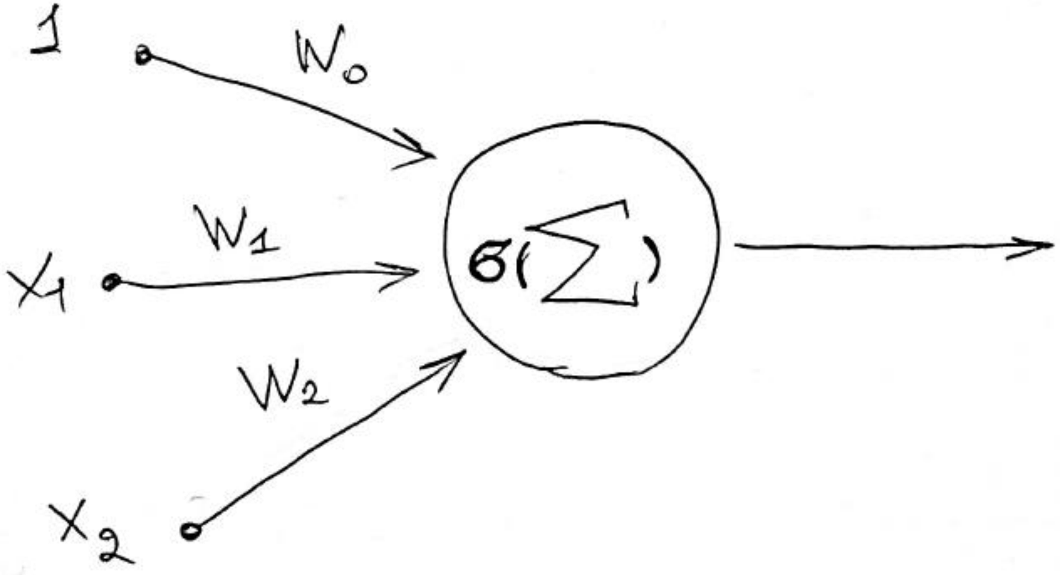
$$\sigma(z) = \frac{1}{1 + e^{-z}};$$

$$\sigma'(z) = \sigma(z) * (1 - \sigma(z))$$

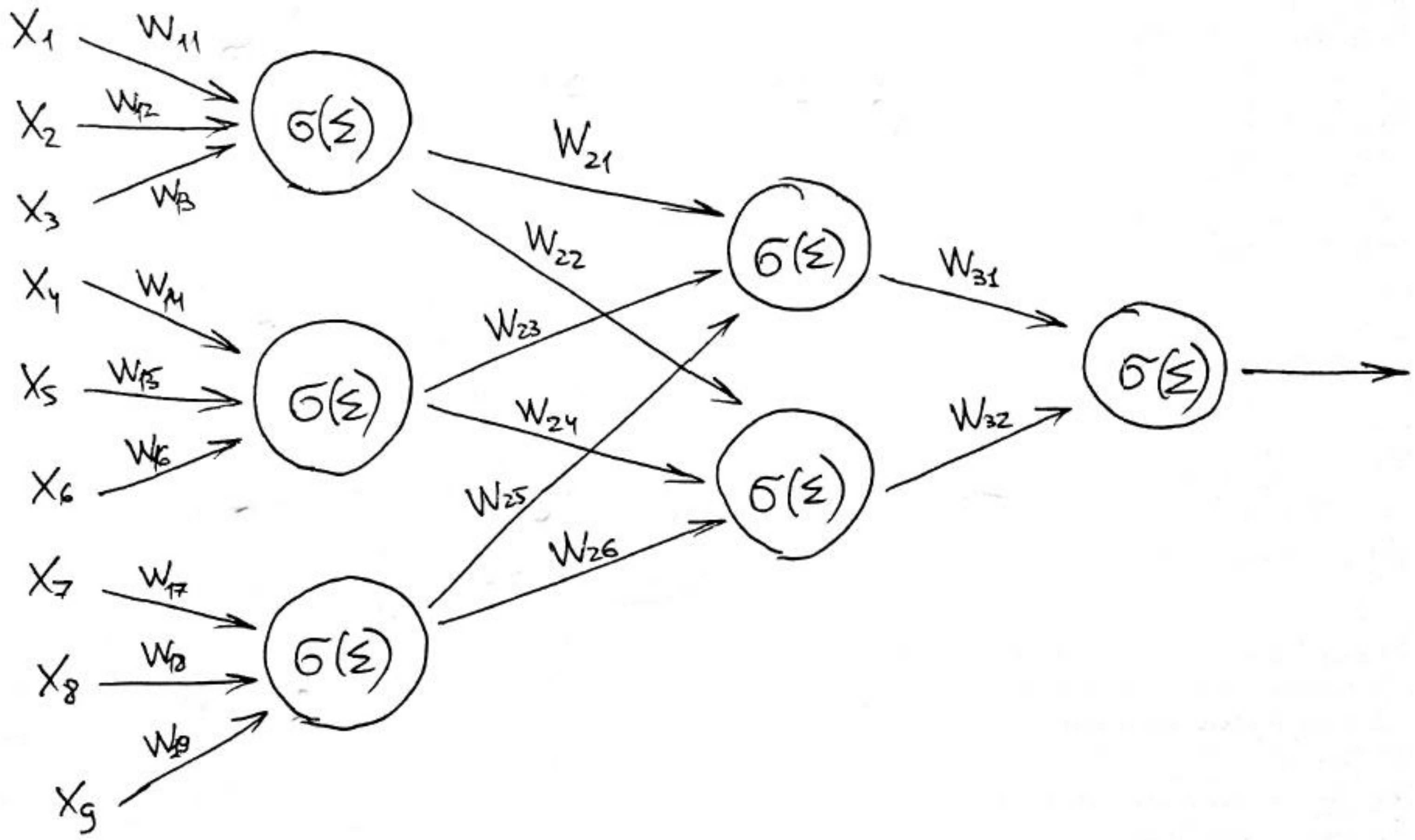
neural networks

Perceptron

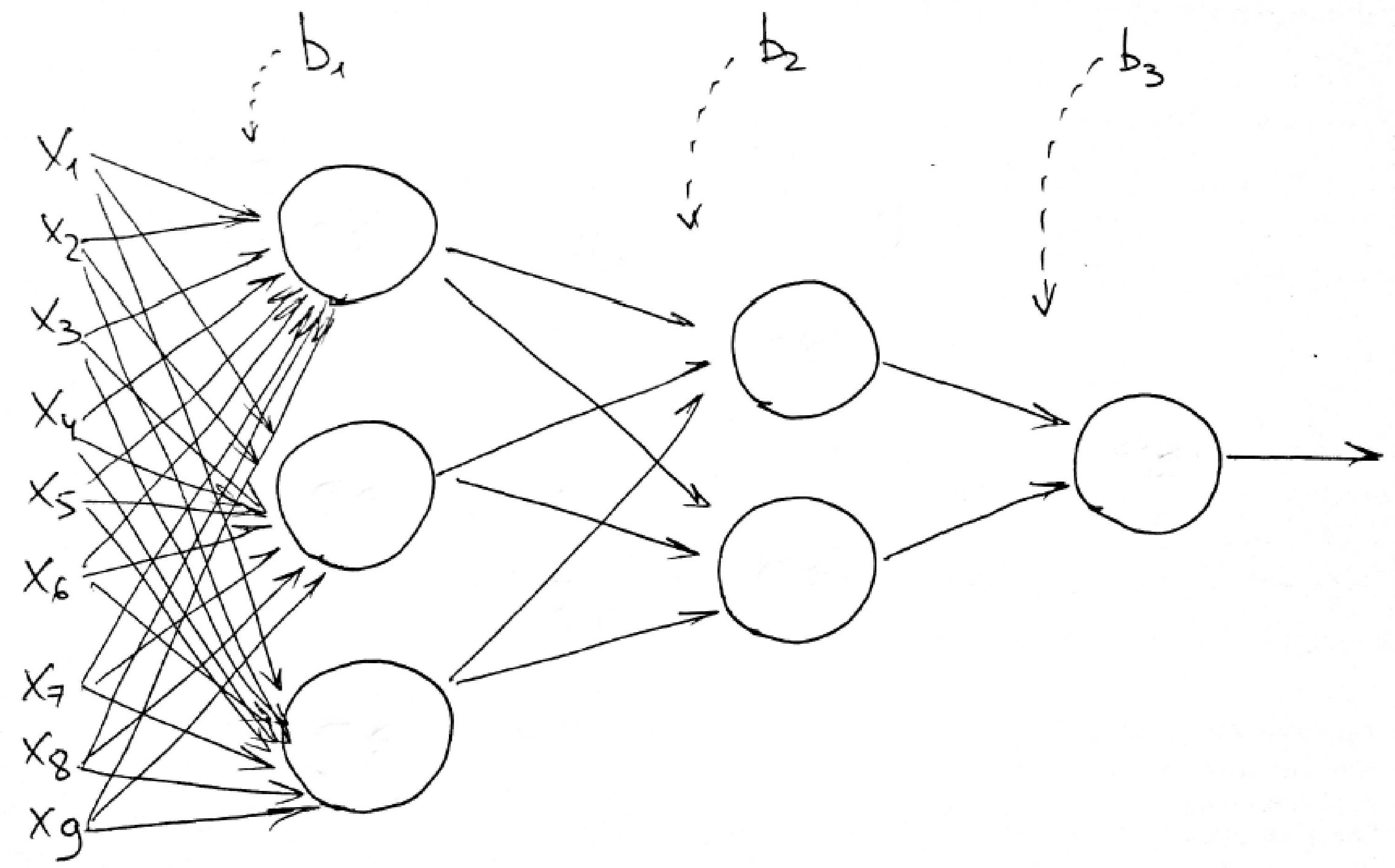
$$\sigma(W_0 + W_1 x_1 + W_2 x_2)$$



Neural networks



Fully-connected neural networks



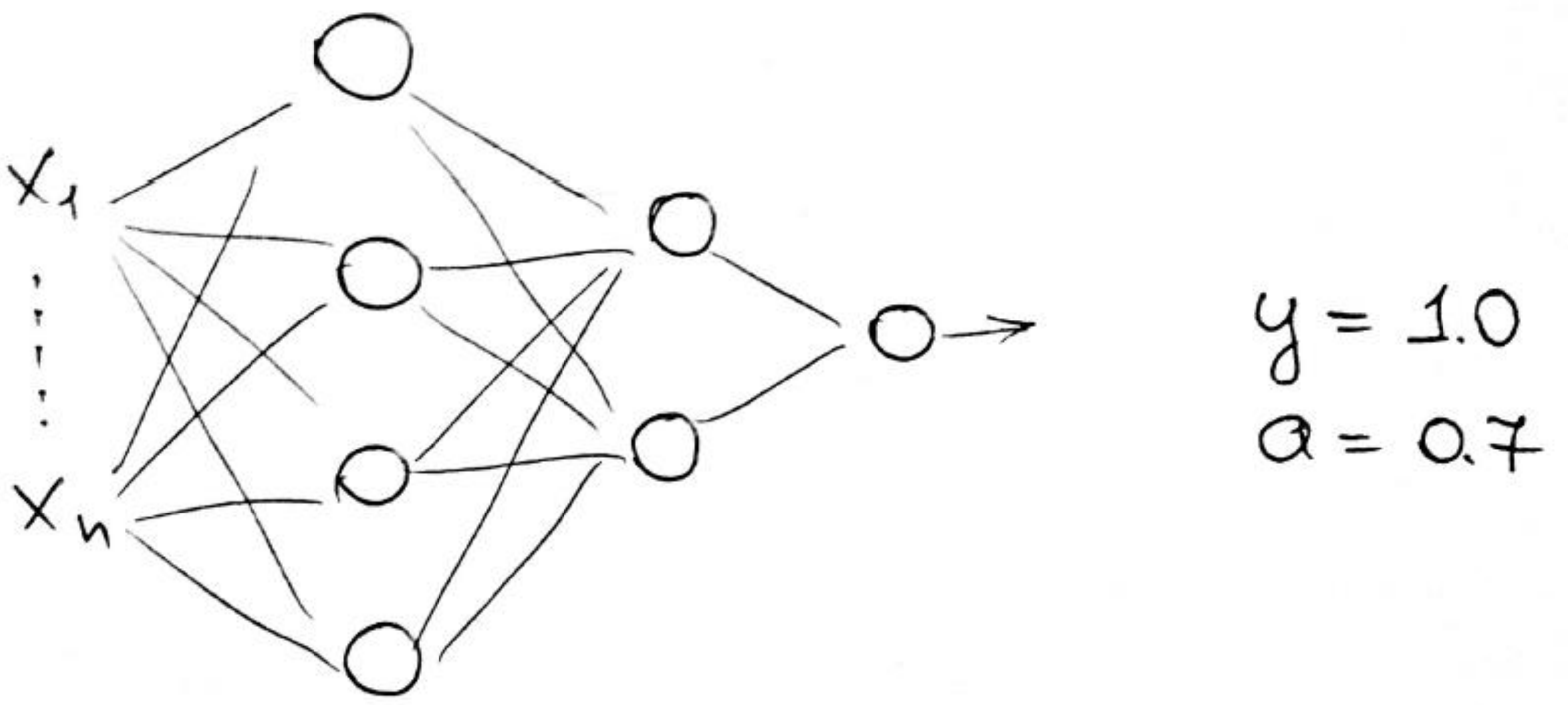
Fully-connected neural networks

Output: $NN(w, b, x) = a$

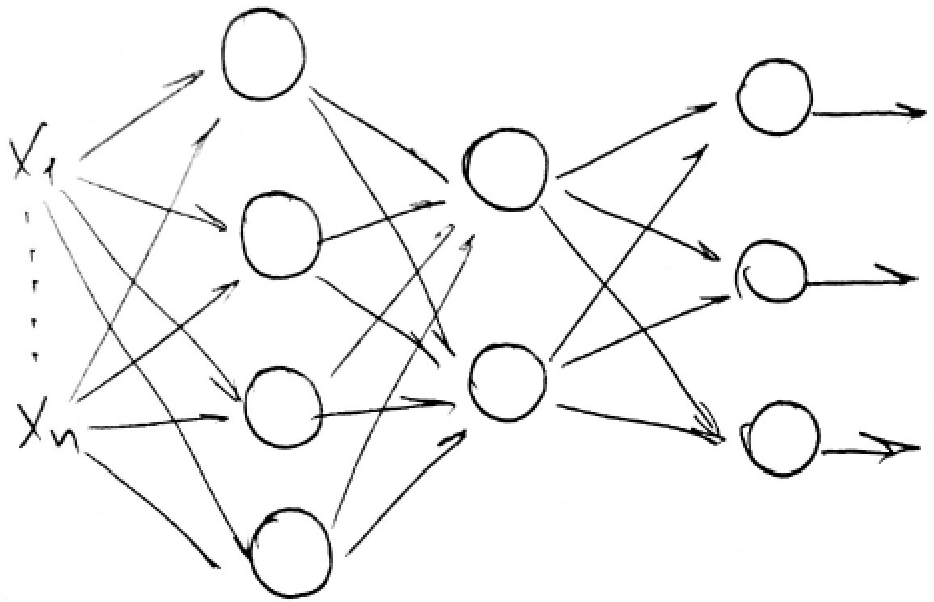
Real: $y(x)$

Error: $\frac{1}{2n} \sum_x \| a - y \|^2$

Fully-connected neural networks



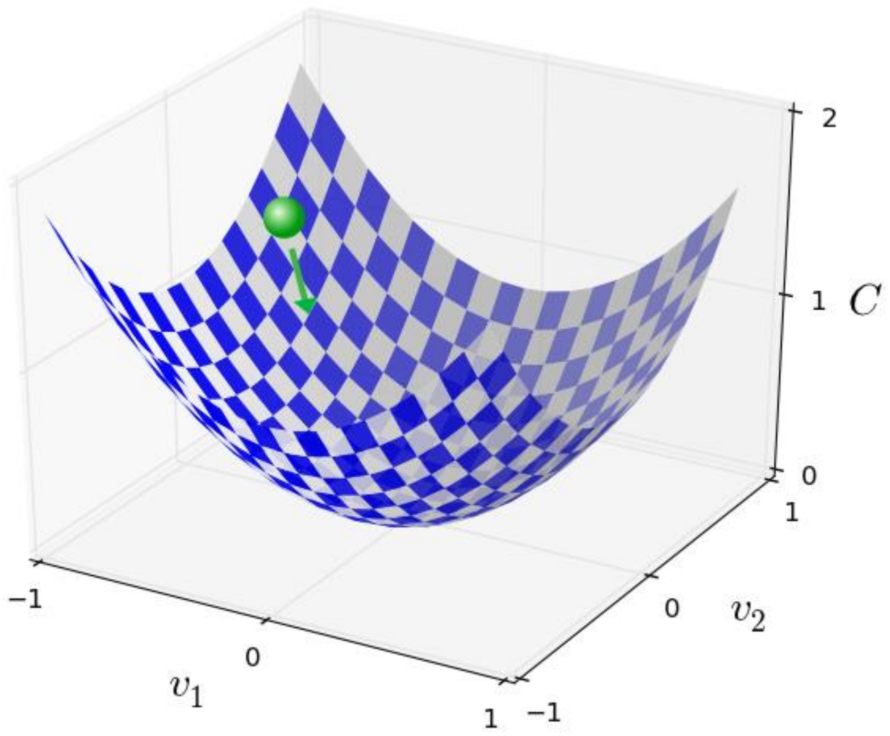
Fully-connected neural networks



$$y = \begin{pmatrix} 0.0 \\ 1.0 \\ 0.0 \end{pmatrix} \quad a = \begin{pmatrix} 0.2 \\ 0.6 \\ 0.2 \end{pmatrix}$$

gradient descent
+
backpropagation

How it learns?



- We want to minimize some

$$J(x1, x2)$$

- **Gradient** (∇J) – vector, showing the direction of the fastest **increasing** of **J.**
- To minimize, we have to move in the direction of **antigradient** $-\nabla C$ with some **step** λ.

Gradient descent

$$\mathcal{J}(x_1, x_2) = x_1^2 + x_2^2$$

$$x^{(0)} = (5,5)^T$$

$$\lambda = 0.1$$

UPDATE RULE

$$x^{(k+1)} = x^{(k)} - \lambda \cdot \nabla \mathcal{J}(x^{(k)})$$

Gradient descent

$$\nabla J(x^{(0)}) = \begin{pmatrix} 2 \cdot x_1^{(0)} \\ 2 \cdot x_2^{(0)} \end{pmatrix} = \begin{pmatrix} 10 \\ 10 \end{pmatrix}$$

$$x^{(1)} = \begin{pmatrix} 5 \\ 5 \end{pmatrix} - \lambda \cdot \begin{pmatrix} 10 \\ 10 \end{pmatrix} = \begin{pmatrix} 5 \\ 5 \end{pmatrix} - 0.1 \cdot \begin{pmatrix} 10 \\ 10 \end{pmatrix} =$$

$$= \begin{pmatrix} 5 \\ 5 \end{pmatrix} - \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 4 \\ 4 \end{pmatrix}$$

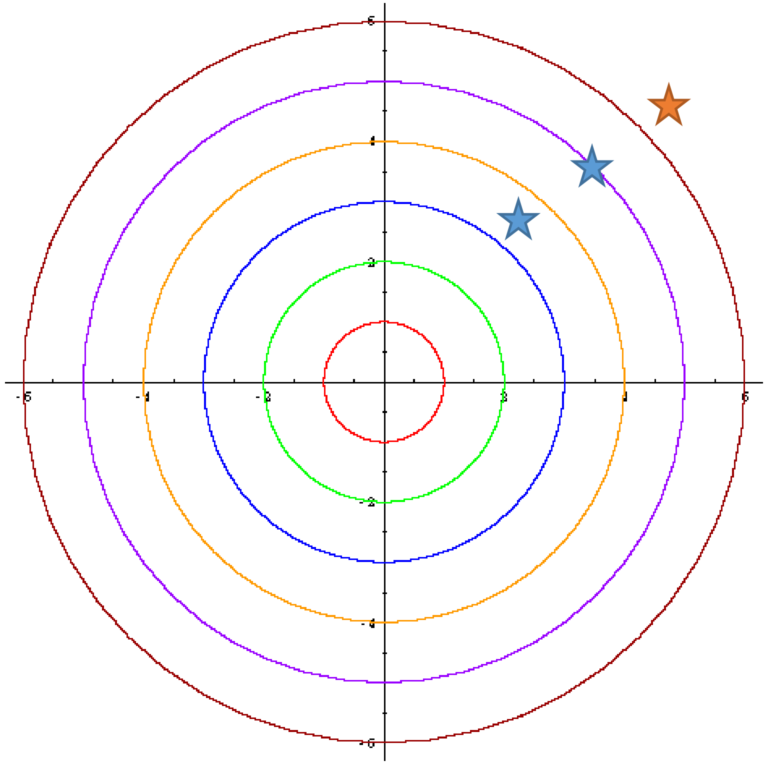
Gradient descent

$$\nabla J\left(x^{(1)}\right) = \begin{pmatrix} 2 \cdot x_1^{(1)} \\ 2 \cdot x_2^{(1)} \end{pmatrix} = \begin{pmatrix} 8 \\ 8 \end{pmatrix}$$

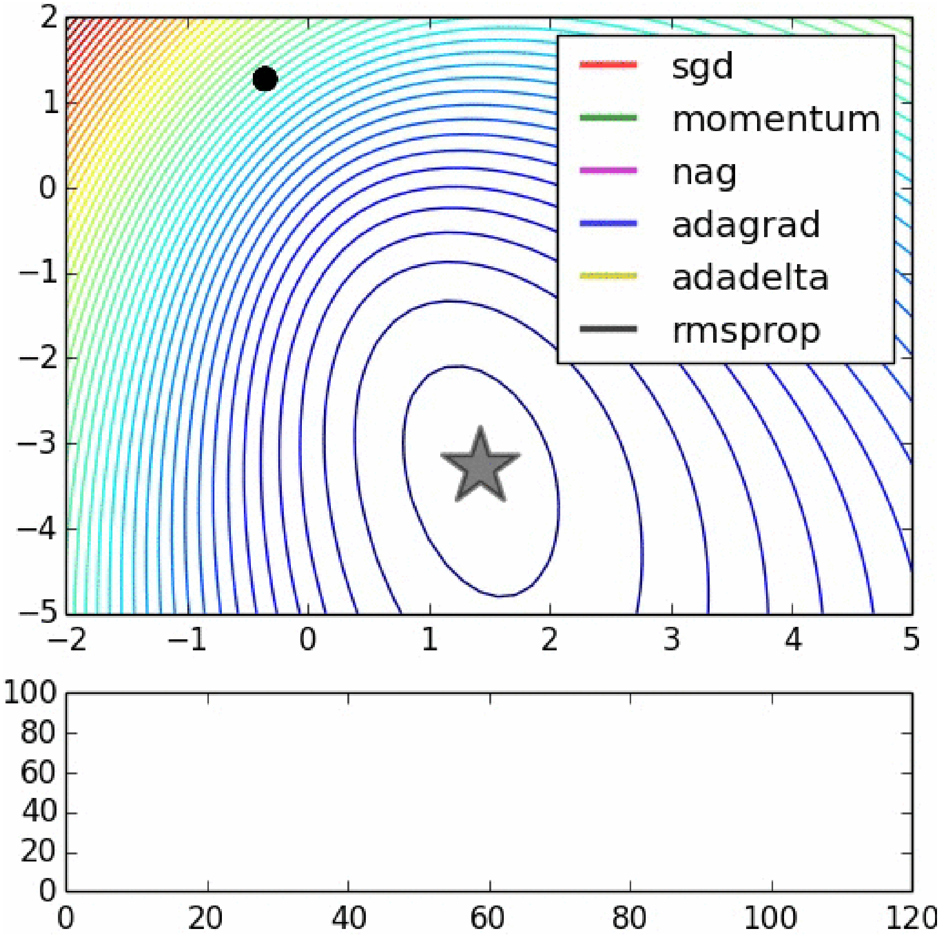
$$x^{(2)} = \begin{pmatrix} 4 \\ 4 \end{pmatrix} - \lambda \cdot \begin{pmatrix} 8 \\ 8 \end{pmatrix} = \begin{pmatrix} 4 \\ 4 \end{pmatrix} - 0.1 \cdot \begin{pmatrix} 8 \\ 8 \end{pmatrix} =$$

$$= \begin{pmatrix} 4 \\ 4 \end{pmatrix} - \begin{pmatrix} 0.8 \\ 0.8 \end{pmatrix} = \begin{pmatrix} 3.2 \\ 3.2 \end{pmatrix}$$

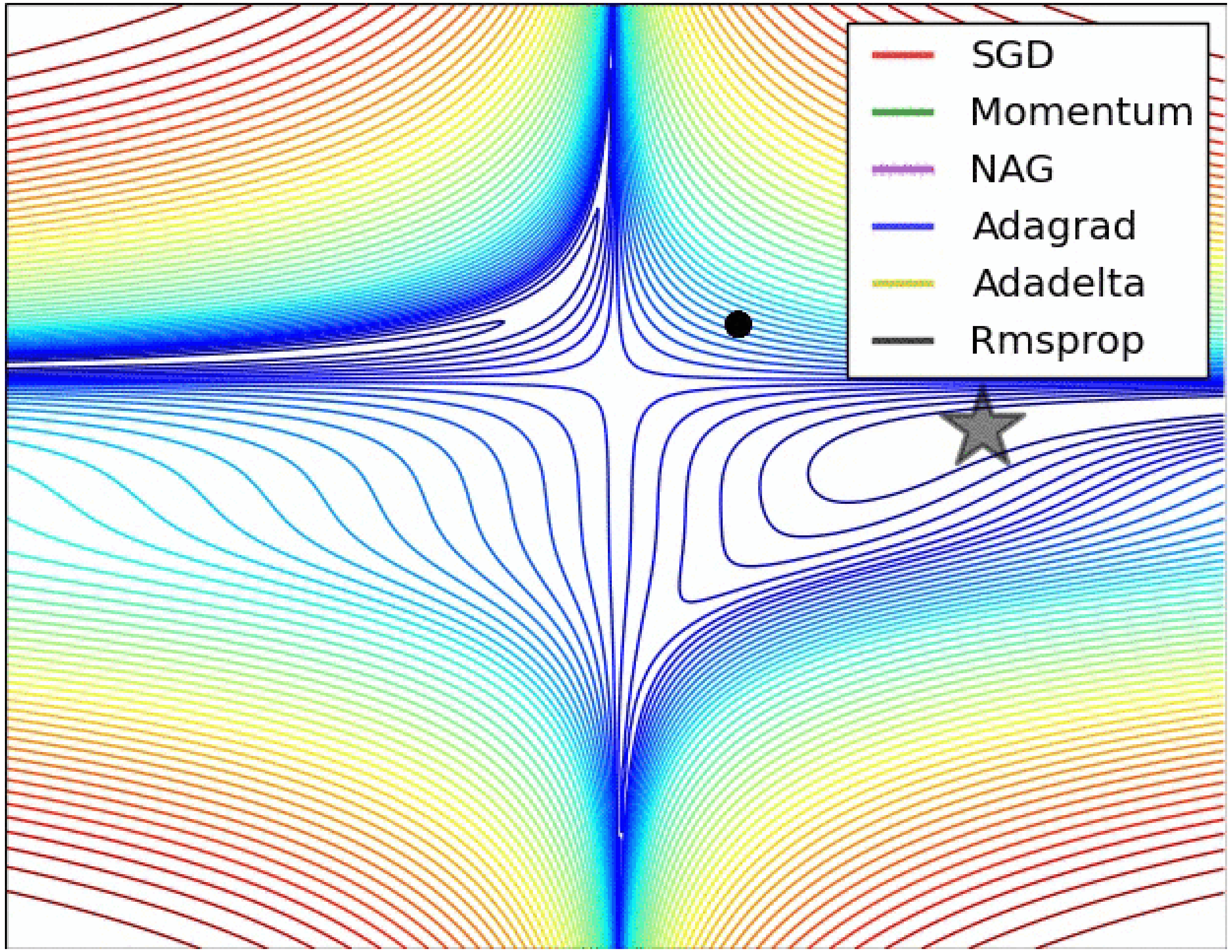
Gradient descent



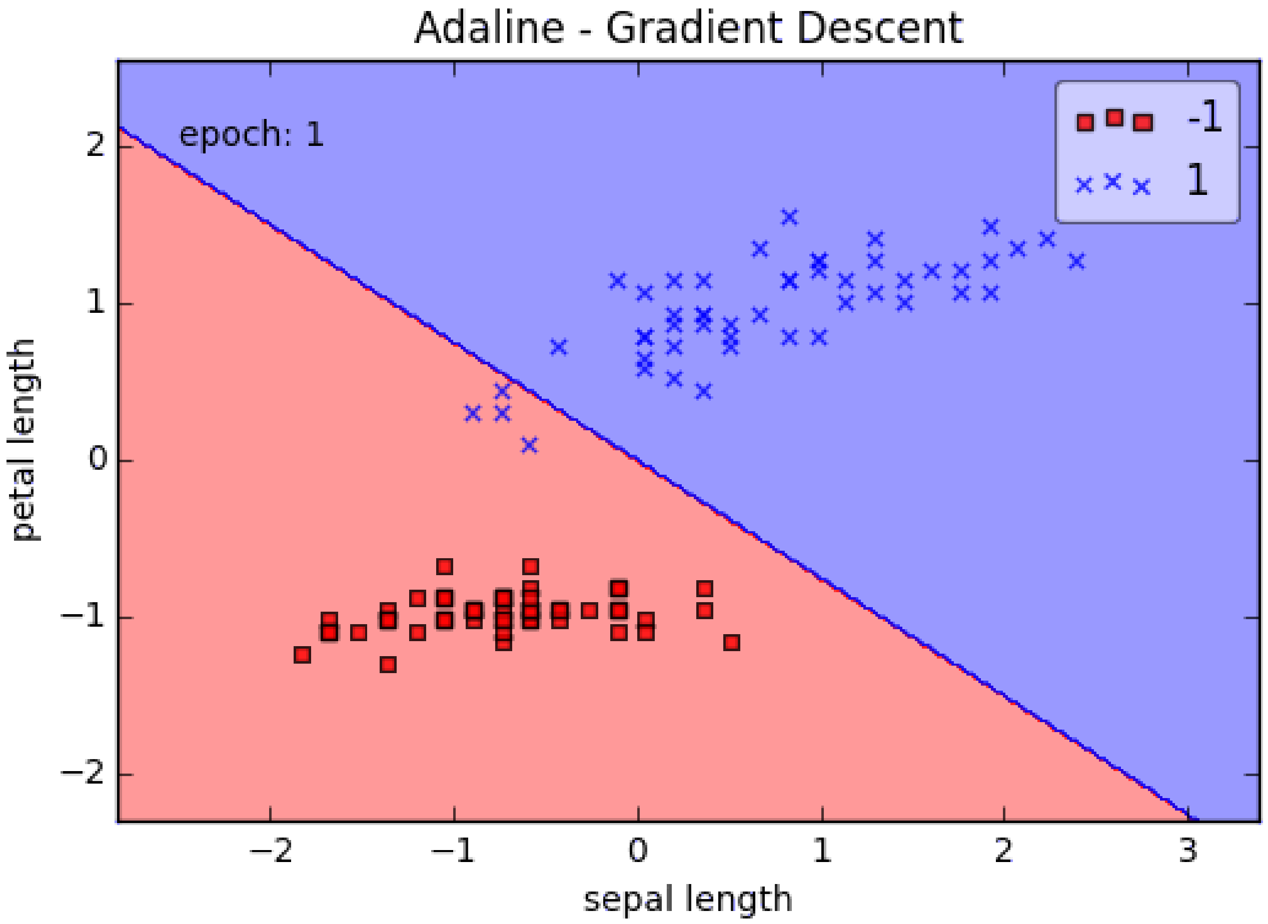
Gradient descent



Gradient descent



Gradient descent



Stochastic gradient descent

One example error

$$C_x = \frac{\| y(x) - a \|^2}{2}$$

All examples gradient

$$\nabla C = \frac{1}{n} \sum_x \nabla C_x$$

Subsamples gradient

$$\frac{\sum_{j=1}^{m} \nabla C_{x_j}}{m} \approx \frac{\sum_x \nabla C_x}{n} = \nabla C$$

$$\boxed{m \ll n}$$

Gradients we need

<u>For gradient descent we need:</u>

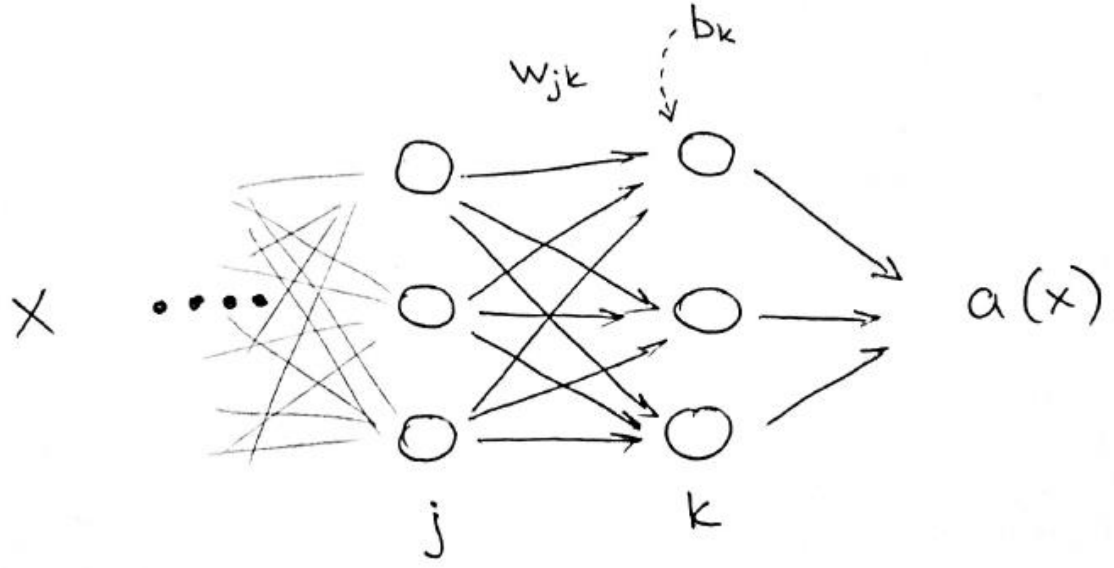
$$W^{(k+1)} = W^{(k)} - \lambda \frac{\partial C}{\partial W_k}$$

$$b^{(\ell+2)} = b^{(\ell)} - \lambda \frac{\partial C}{\partial b_\ell}$$

Notations

$$C = \frac{(a(x) - y(x))^2}{2}$$

$$z_k = \sum a_j w_{jk} + b_k$$



Backpropagation 1

$$\frac{\partial C}{\partial \omega_{jk}} = \frac{\partial}{\partial \omega_{jk}} \left(\frac{(a-y)^2}{2} \right) =$$

$$= (a-y) \cdot \frac{\partial a}{\partial \omega_{jk}} = (a-y) \cdot \frac{\partial}{\partial \omega_{jk}} \left(\sigma(z_k) \right) =$$

$$= (a-y) \cdot (\sigma(z_k))' \cdot \frac{\partial z}{\partial \omega_{jk}} =$$

$$= \underbrace{(a-y) \cdot (\sigma(z_k))'}_{\delta_k} \cdot a_j =$$

$$= \delta_k a_j$$

Backpropagation 2

$$\frac{\partial C}{\partial b_k} = (a-y)\cdot\frac{\partial a}{\partial b_k} =$$

$$= (a-y)\cdot(\sigma(z_k))'\cdot\frac{\partial z}{\partial b_k} =$$

$$= \underbrace{(a-y)\cdot(\sigma(z_k))'}_{\delta_k}\cdot 1 =$$

$$= \delta_k$$

Backpropagation 3

$$z_k = \sum_j a_j \omega_{jk} + b_k =$$

$$= \sum_j \sigma\left(\sum_i z_i \omega_{ij} + b_i\right) \omega_{jk} + b_k$$

Backpropagation 3

$$\frac{\partial C}{\partial \omega_{ij}} = (a-y) \cdot (\sigma(z_k))' \cdot \frac{\partial z}{\partial \omega_{ij}} =$$

$$\frac{\partial z_j}{\partial \omega_{ij}} = \frac{\partial z_j}{\partial a_j} \cdot \frac{\partial a_j}{\partial \omega_{ij}} = \omega_{jk} \cdot \frac{\partial a_j}{\partial \omega_{ij}} =$$

$$= \omega_{jk} \cdot \frac{\partial \sigma(z_j)}{\partial \omega_{ij}} = \omega_{jk} \cdot (\sigma(z_j))' \cdot \frac{\partial \left(\sum_i a_i \cdot \omega_{ij} + b_i \right)}{\partial \omega_{ij}} =$$

$$= \omega_{jk} \cdot (\sigma(z_j))' \cdot a_i$$

$$= \underbrace{(a-y) \cdot (\sigma(z_k))'} \cdot \omega_{jk} \cdot (\sigma(z_j))' \cdot a_i =$$

$$= a_i \cdot (\sigma(z_j))' \cdot \delta_k \cdot \omega_{jk}$$

Backpropagation 4

$$\frac{\partial C}{\partial b_i} = (a-y)\cdot\frac{\partial a}{\partial \omega_{jk}} =$$

$$= (a-y)\cdot(\sigma(z_k))'\cdot\frac{\partial z}{\partial b_i} =$$

$$= (a-y)\cdot(\sigma(z_k))'\cdot(\sigma(z_j))'\cdot\omega_{jk} =$$

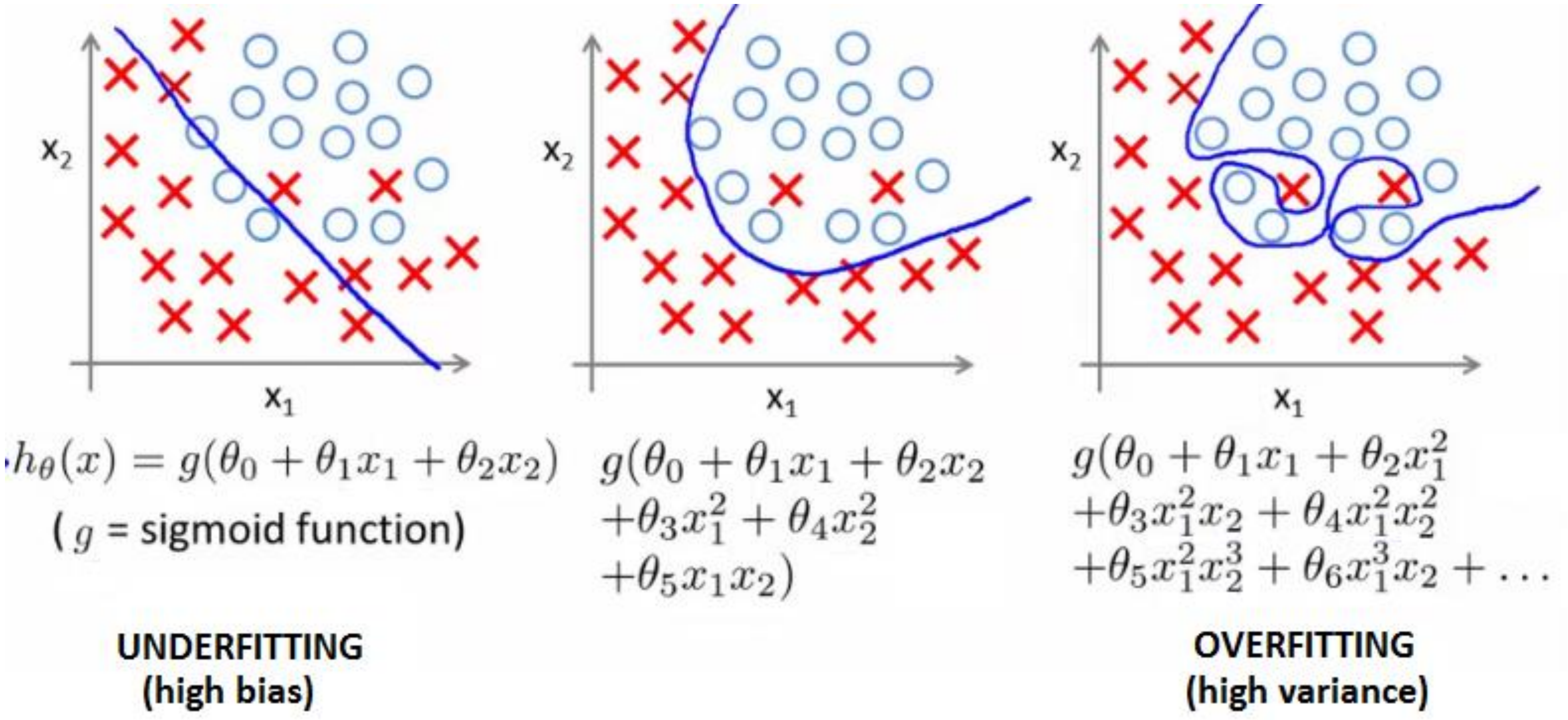
$$= (\sigma(z_j))'\cdot\delta_k\cdot\omega_{jk}$$

Backpropagation full

- We define training set
- For every training example x:
  - For every layer $l = 2,3 \ldots L$
    - calculate neuron outputs $z^{x,l} = w^l a^{x,l-1} + b^l$
    - calculate neuron outputs $a^{x,l} = \sigma(z^{x,l})$
  - Calculate output error:
    - $\delta^{x,L} = \nabla C_x \odot \sigma(z^{x,L})'$
  - Backpropagate the error through the layers $l = L-1, L-2 \ldots 2$:
    - $\delta^{x,l} = \left(\left(w^{l+1}\right)^T * \delta^{x,l+1} \right) \odot \sigma(z^{x,l})'$
  - Launch gradient descent updates:
    - $w^l \rightarrow w^l - \frac{\lambda}{m} \sum_x \delta^{x,l} \left(a^{x,l-1}\right)^T$
    - $b^l \rightarrow b^l - \frac{\lambda}{m} \sum_x \delta^{x,l}$

training tricks

Overfitting



Training tricks

1. Regularization
2. Dropout
3. Weights initialization
4. Learning slowdown
5. Gradient descent variations
6. Different activation functions
7. Data augmentation
8. Injecting noise into input / output
9. Ensemble methods
10. Hyperparameters optimization

Training tricks

- L2 Regularization:

$$C = C_0 + \frac{\lambda}{2n} \sum_w w^2,$$

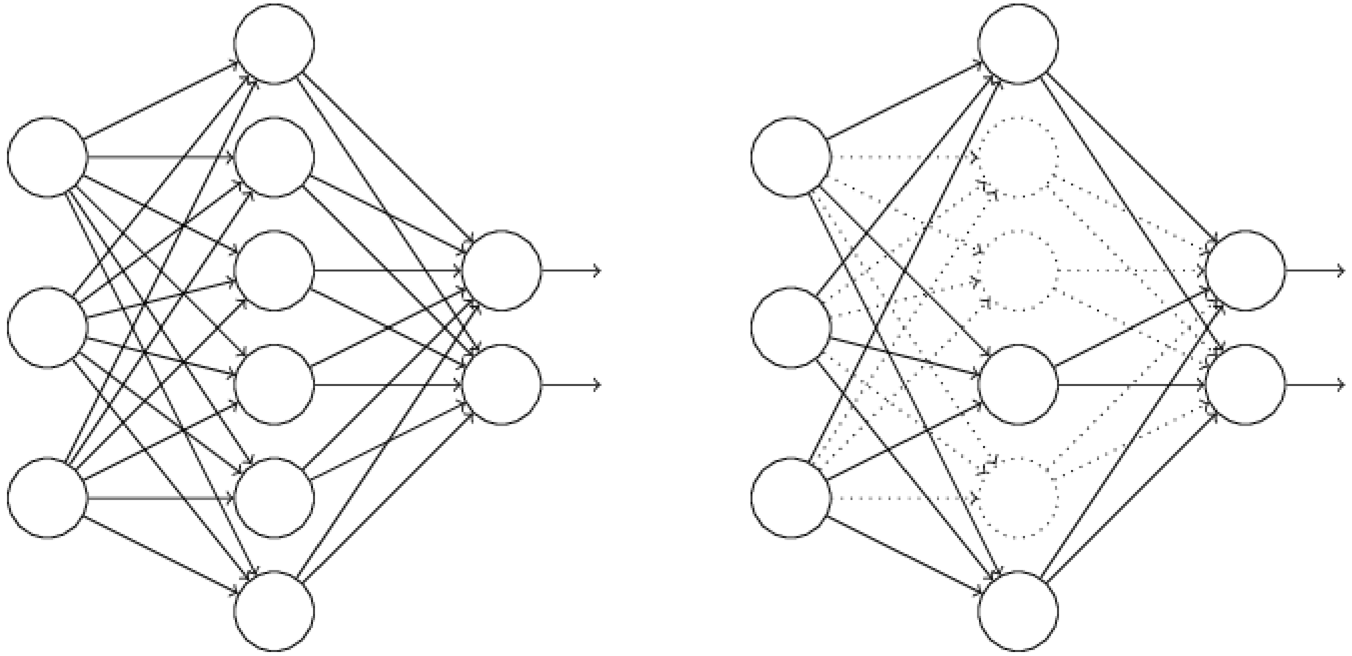
$$\frac{\partial C}{\partial w} = \frac{\partial C_0}{\partial w} + \frac{\lambda}{n} w \qquad w \rightarrow \left(1 - \frac{\eta \lambda}{n}\right) w - \frac{\eta}{m} \sum_x \frac{\partial C_x}{\partial w}$$

- L1 Regularization:

$$C = C_0 + \frac{\lambda}{n} \sum_w |w|.$$

$$\frac{\partial C}{\partial w} = \frac{\partial C_0}{\partial w} + \frac{\lambda}{n} \operatorname{sgn}(w) \qquad w \rightarrow w' = w \left(1 - \frac{\eta \lambda}{n}\right) - \eta \frac{\partial C_0}{\partial w}.$$

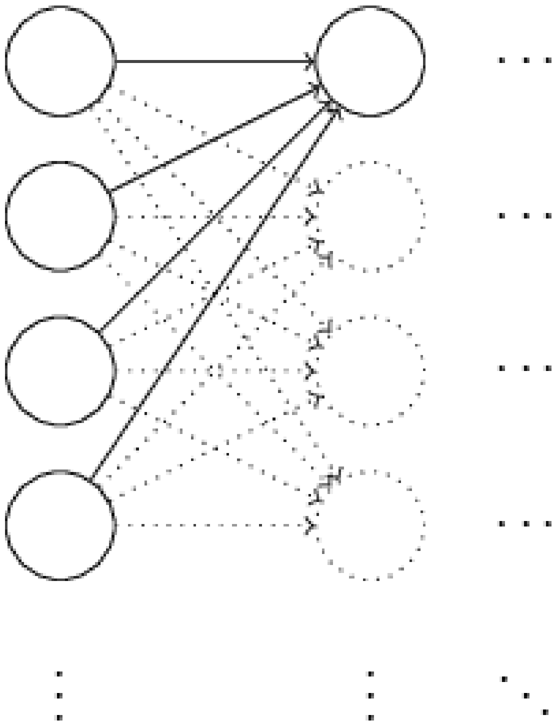
Dropout



- We reduce the “co-adaptation” of neurons
- We kinda train lots of different architectures

Weights initialization

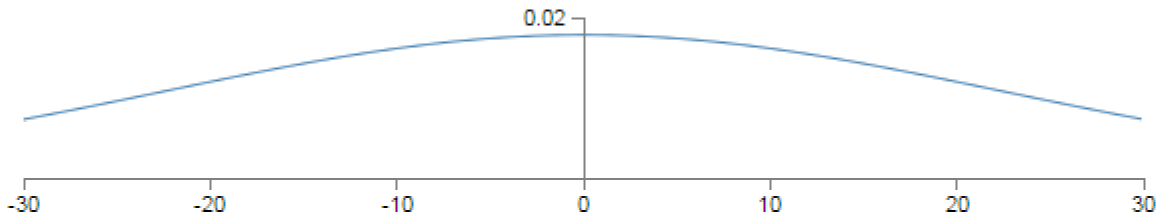
- Usually we initialize with Gaussian distribution with *EV* = 0, *SD* (standard deviation) = 1



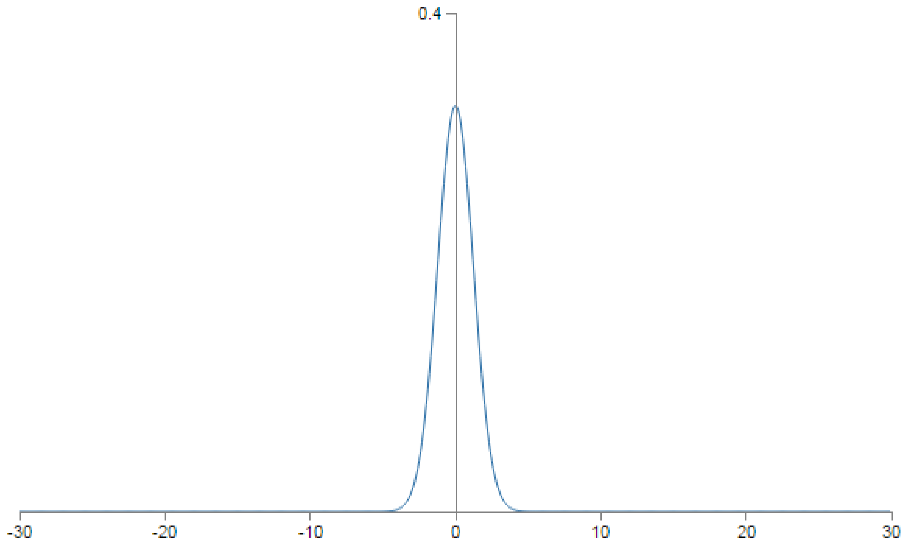
Example:

- 1000 inputs, half equals 1, other half – 0.
- We have variable $z = \sum_i w_i * x_i + b$; and it has $SD = \sqrt{501} \approx 22.4$

Weights initialization



- $\sqrt{501} \approx 22.4$

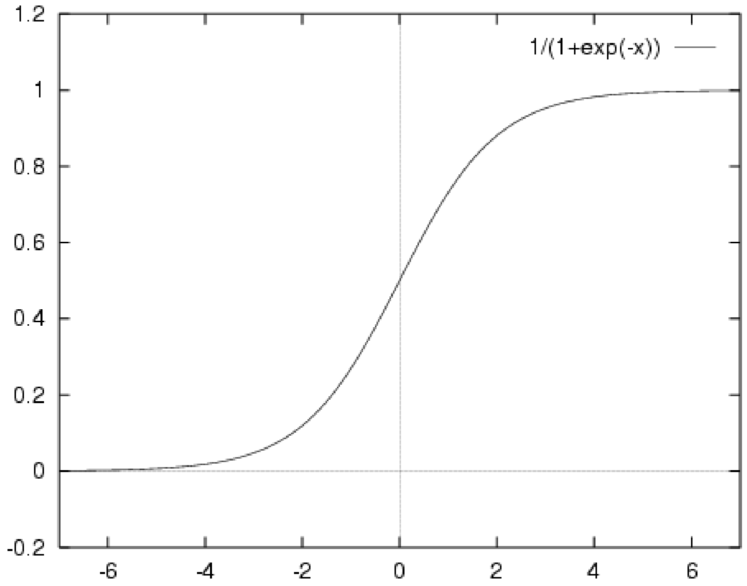


- Initialize with the same distribution, but
- $EV = 0$, SD (standard deviation) = $\frac{1}{\sqrt{n_i}}$
- $\sqrt{1.5} \approx 1.22$

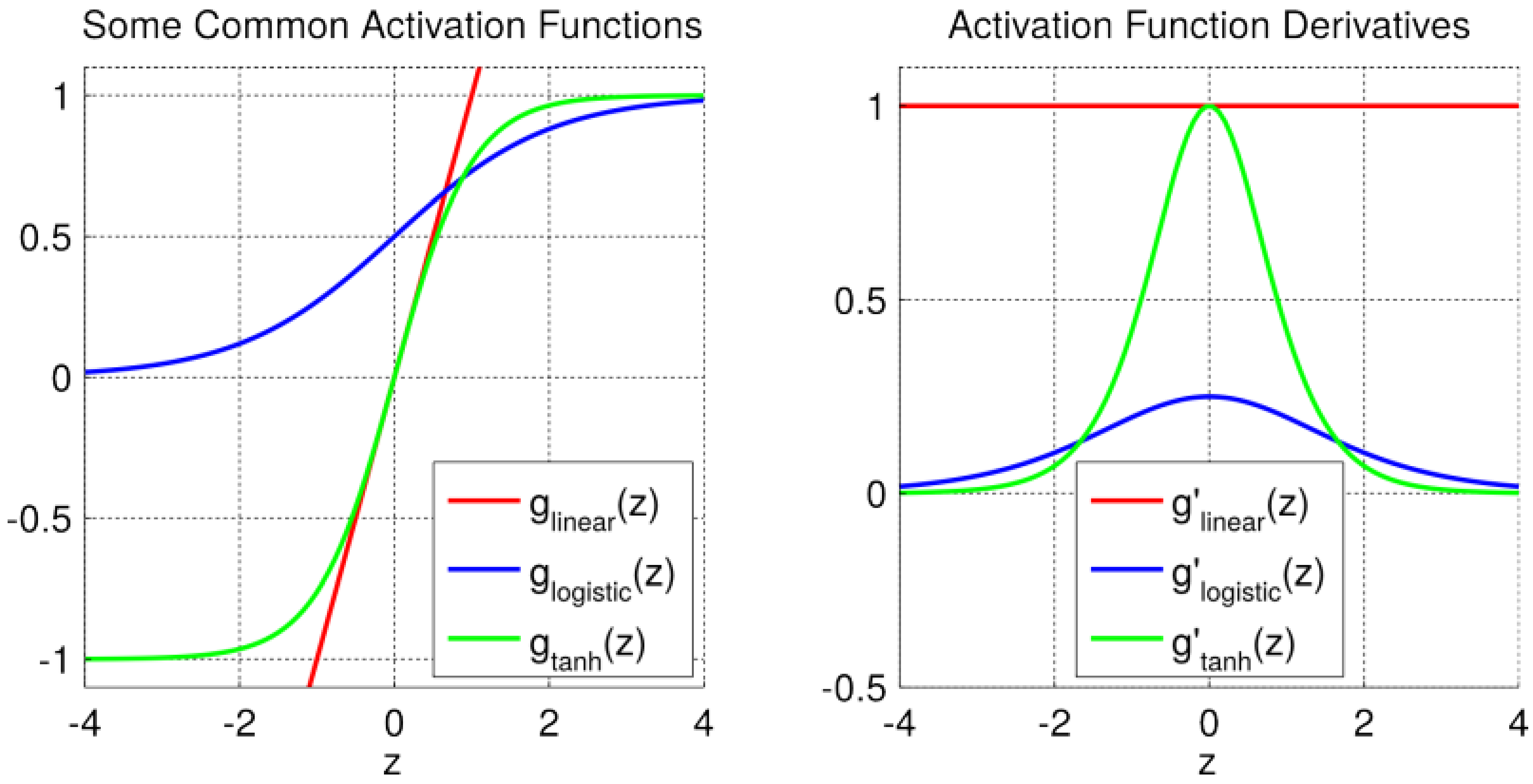
Learning slowdown

- Let's remember formula for output error in backpropagation:

$$\delta^L = \nabla C \odot \sigma(z)'$$



Learning slowdown



Learning slowdown

- Influence on partial derivatives $\frac{\partial C}{\partial w_k}, \frac{\partial C}{\partial b_k}$ is too small
- We want to have "huge error – huge influence" correspondence
- Cross-entropy cost function:
  - $C = -\frac{1}{n}\sum_x[yln a + (1-y)\ln(1-a)]$
  - $\frac{\partial C}{\partial w_j} = \frac{1}{n}\sum_x x_j(\sigma(z) \ - y)$ - now speed of learning depends only on the output
- Softmax cost function:

$$a_j^L = \frac{e^{z_j^L}}{\sum_k e^{z_k^L}},$$

Gradient descent variations

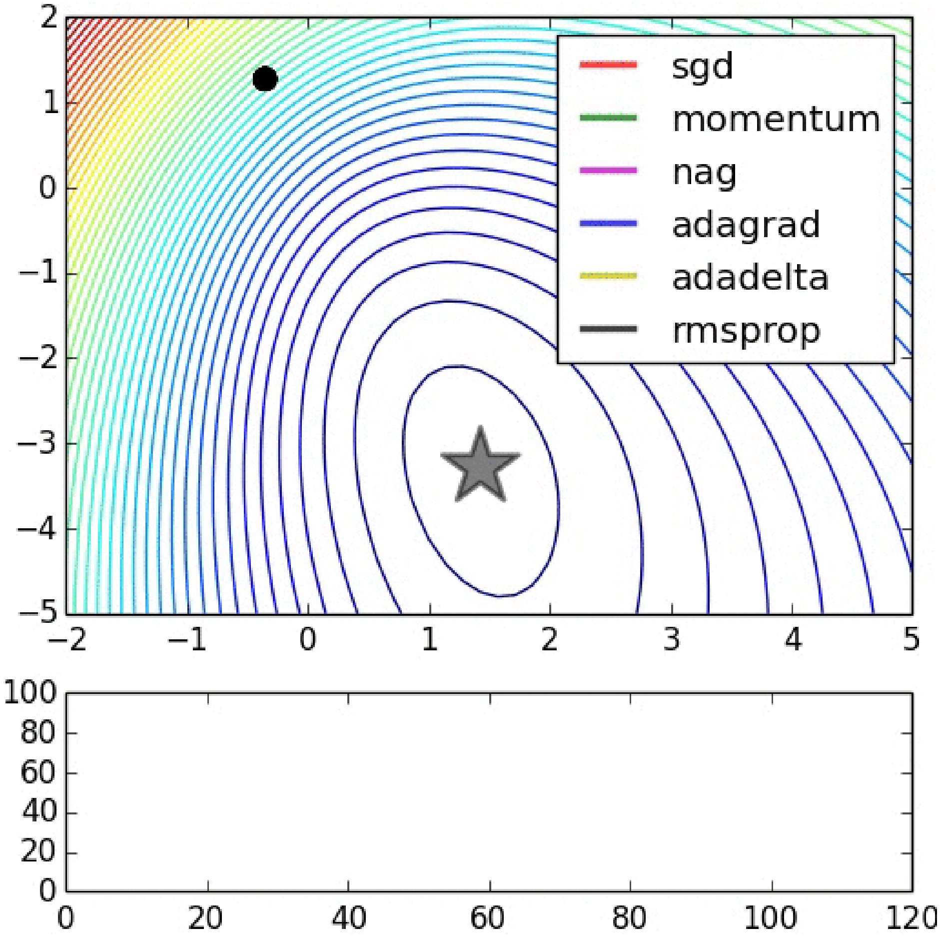
Gradient descent:

- $x^{(k+1)} = x^{(k)} - \lambda \nabla C\left(x^{(k)}\right)$

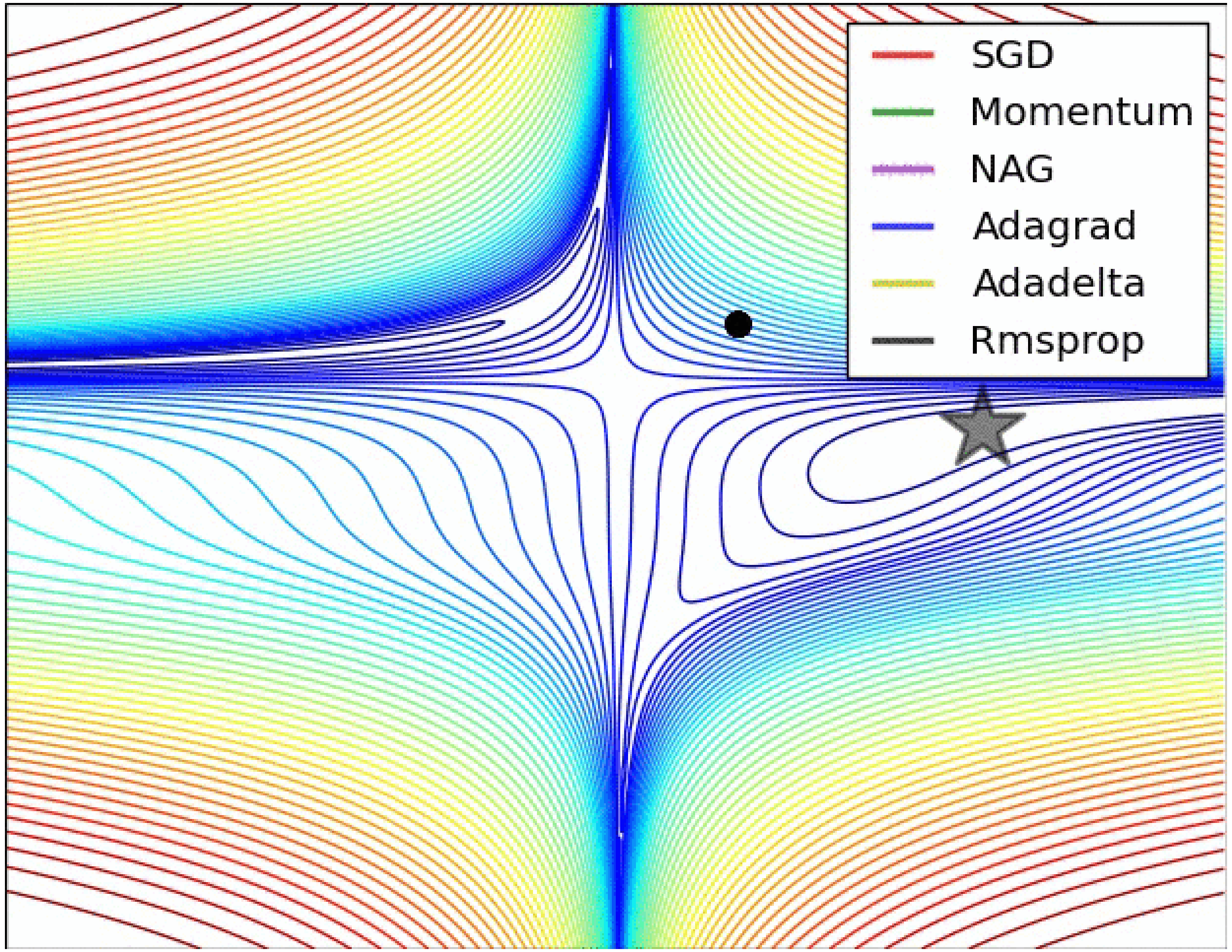
Gradient descent with momentum:

- $v^{(k+1)} = (1 - \mu) v^{(k)} - \lambda \nabla C\left(x^{(k)}\right)$
- $x^{(k+1)} = x^{(k)} + v^{(k+1)}$
- $(1 - \mu)$ - “slippage coefficient”

Gradient descent variations

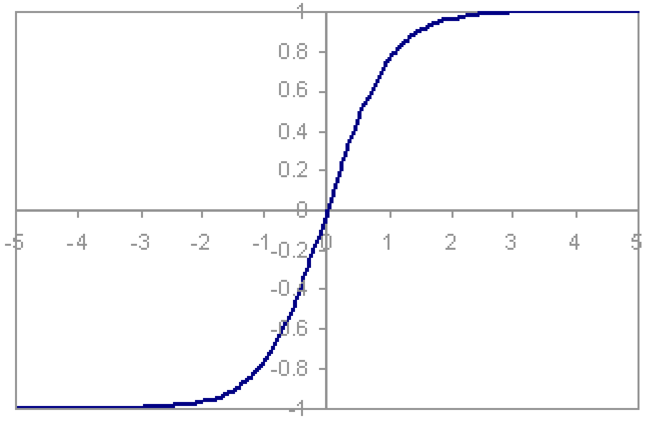


Gradient descent variations

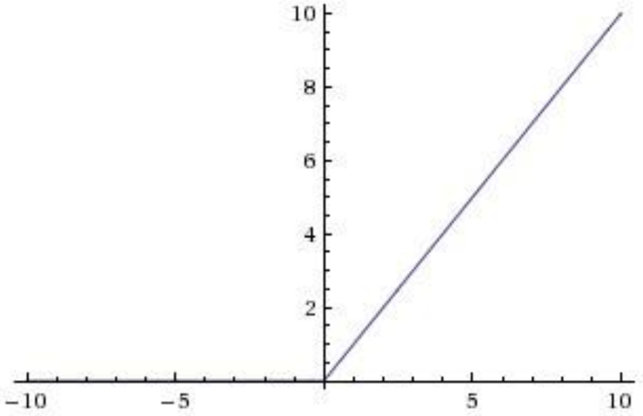


Activation functions

- Hyperbolic tangent:
  - tanh = $\frac{1+\tanh(\frac{z}{2})}{2}$



- Rectified linear unit:
  - $\sigma(z) = \max(0, w * x + b)$



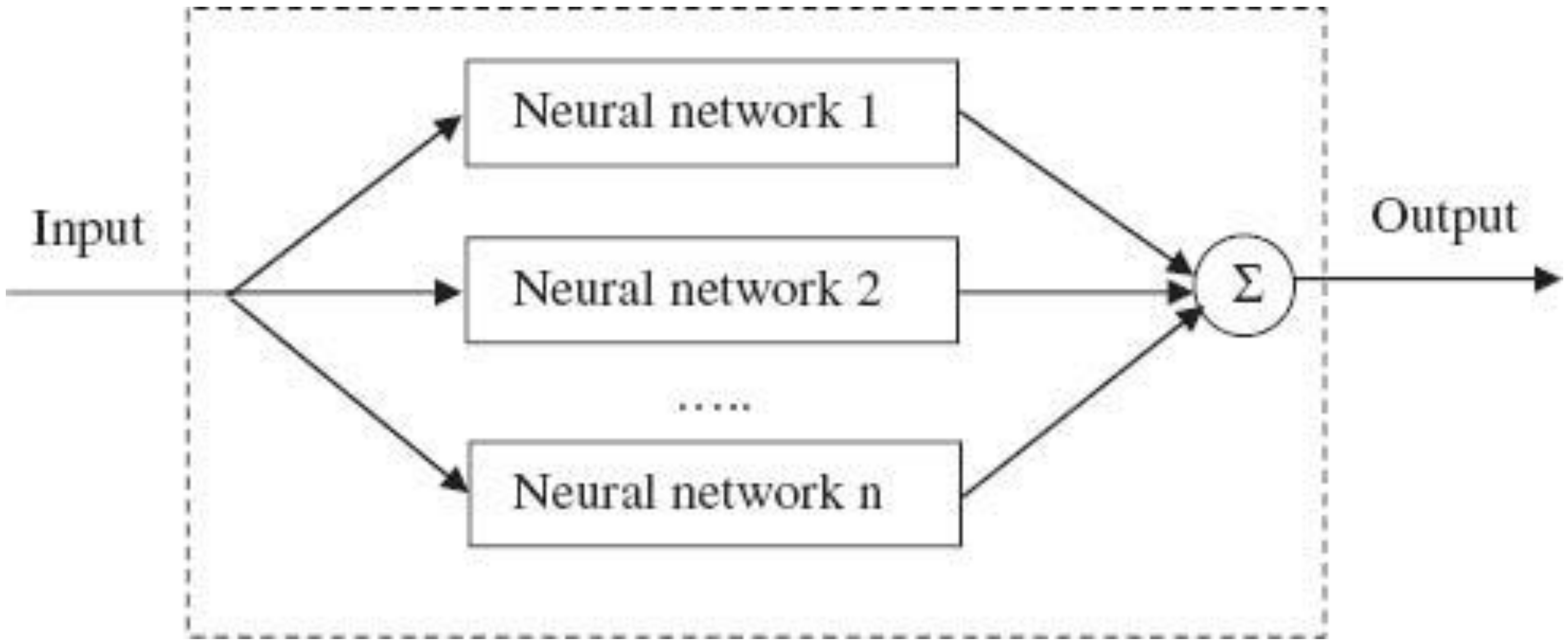
Data augmentation

- Scaling to the range [0; 1] or [-1; 1]
- Polynomial features
- Mirroring (for images)
- Rotating (for images)
- Cropping images
- Zoom-in / zoom-out
- Contrast / brightness normalization
- Changing color scheme(HSV)
- Whitening images (PCA etc)
- Fourier transform (time series)
- Wavelet decomposition (time series)

Noise injecting

- Adding to the input (for noisy data reconstruction)
- Adding to the weights (basically it’s dropout)
- Adding to the output (when y’s have errors)

Esnemble methods



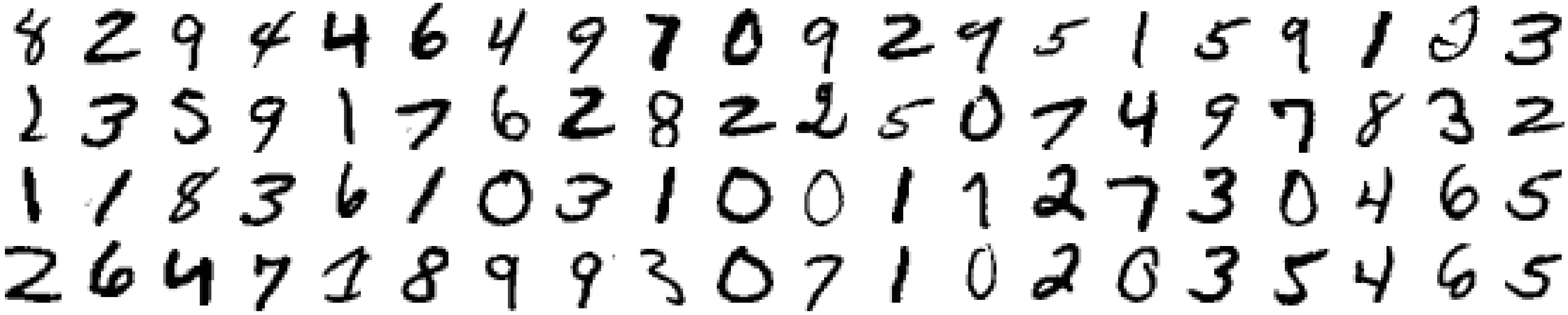
Hyperparameters optimization

- Number of layers
- Number of neurons in every layer
- Regularizations and their parameters
- Gradient descent step
- Different optimization algorithms
- Number of epochs
- Batch size in stochastic gradient descent

- In CNNs / RNNs much more!

coding session

MNIST dataset



Our first network