The optimization behind deep neural networks

Marleen Balvert

Tilburg University

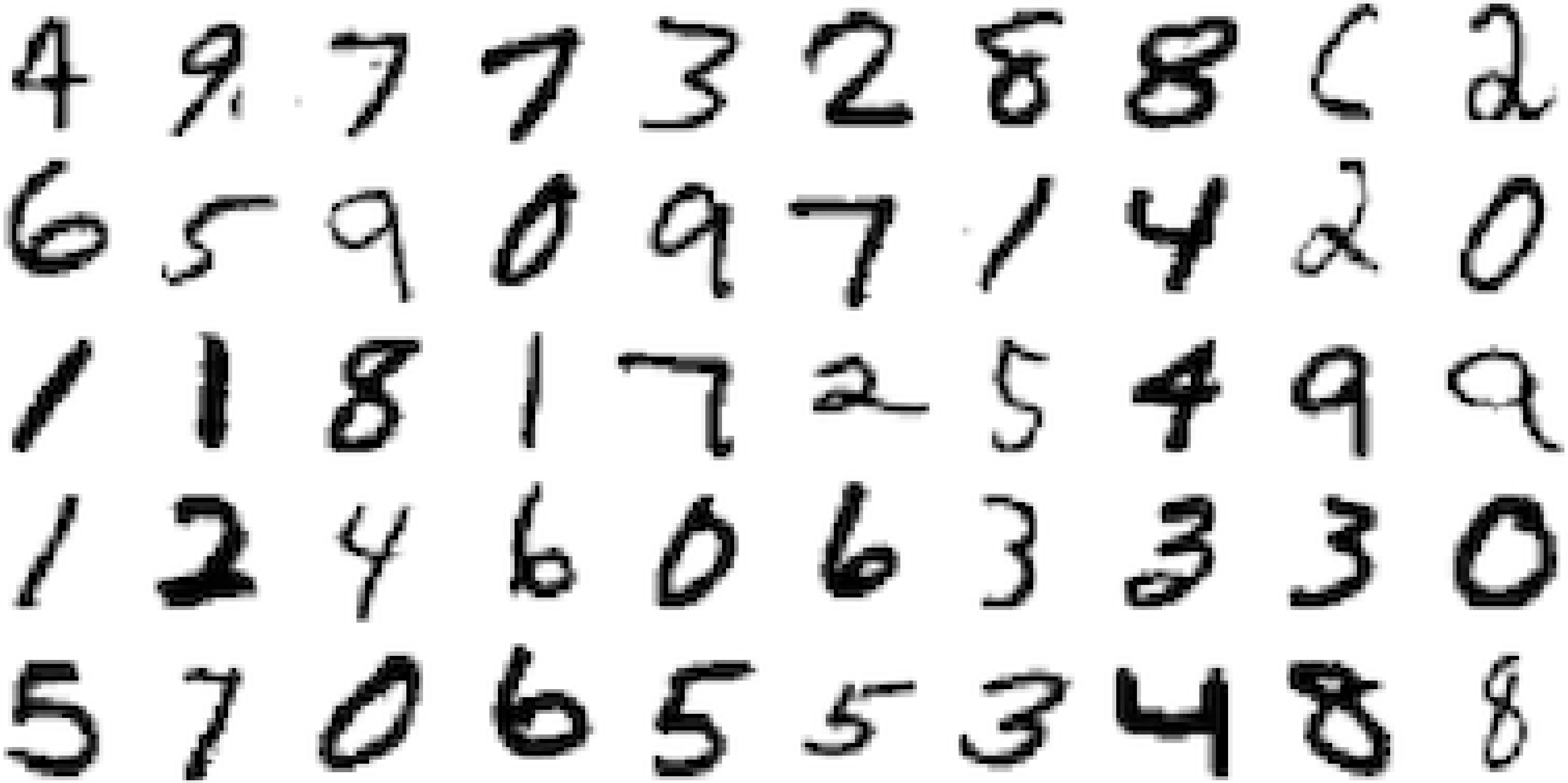
Disclaimer

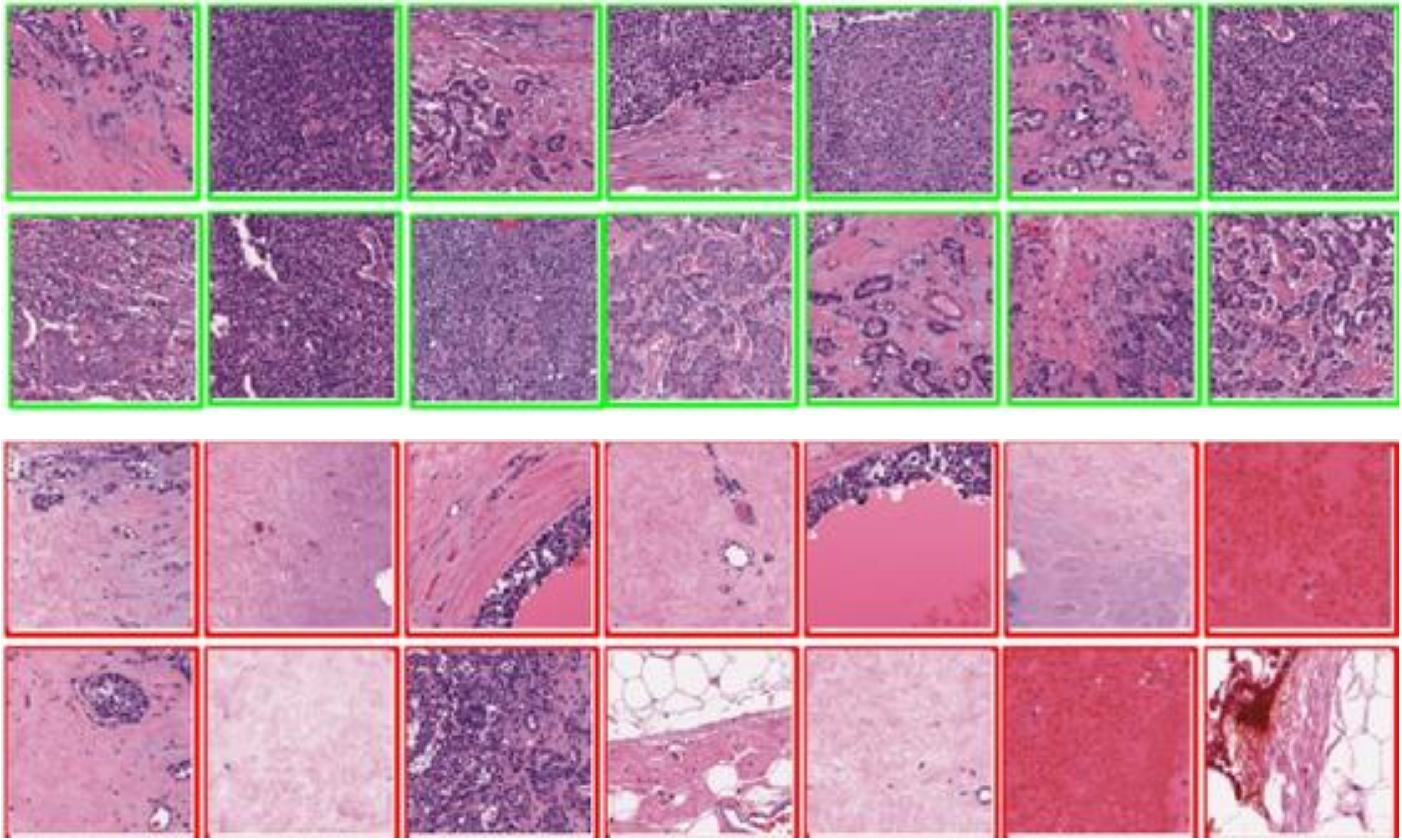
This is a tutorial,

not a research talk

Course material

- www.neuralnetworksanddeeplearning.com
- www.deeplearningbook.org





Deep neural network basics

The classification problem (recap)

Given data:

- Samples $1, \ldots, N$
- Independent variables $x_1, \ldots, x_N \in \mathbb{R}^P$
- Dependent variables $y_1, \ldots, y_N \in \{0, \ldots, K\}$

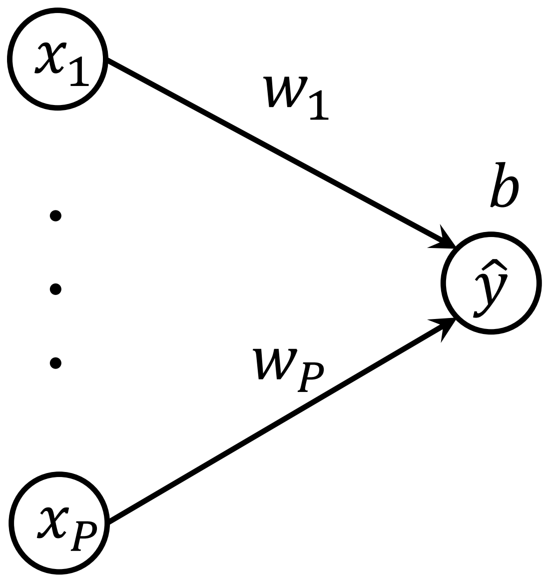
    → Often converted into K dummy variables

Goal: find the relation $y = f(x)$

Goal: make a model that determines the class of a new sample

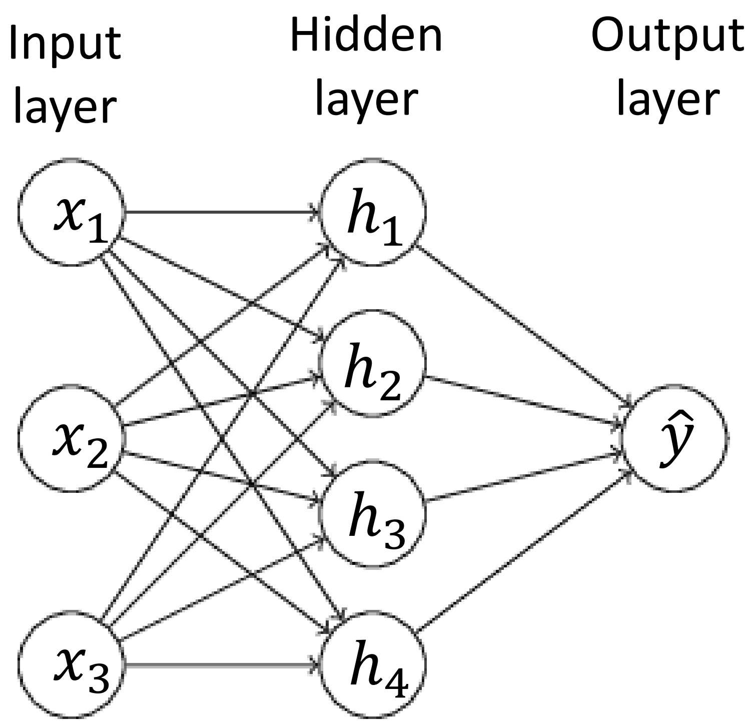
Basic element: neuron or perceptron

Notation: $x \in \mathbb{R}^P$ is feature vector of single sample



$$\hat{y} = \begin{cases} 1 & \text{if } w^T x + b \geq 0 \\ 0 & \text{if } w^T x + b < 0 \end{cases}$$

Neural network (NN)

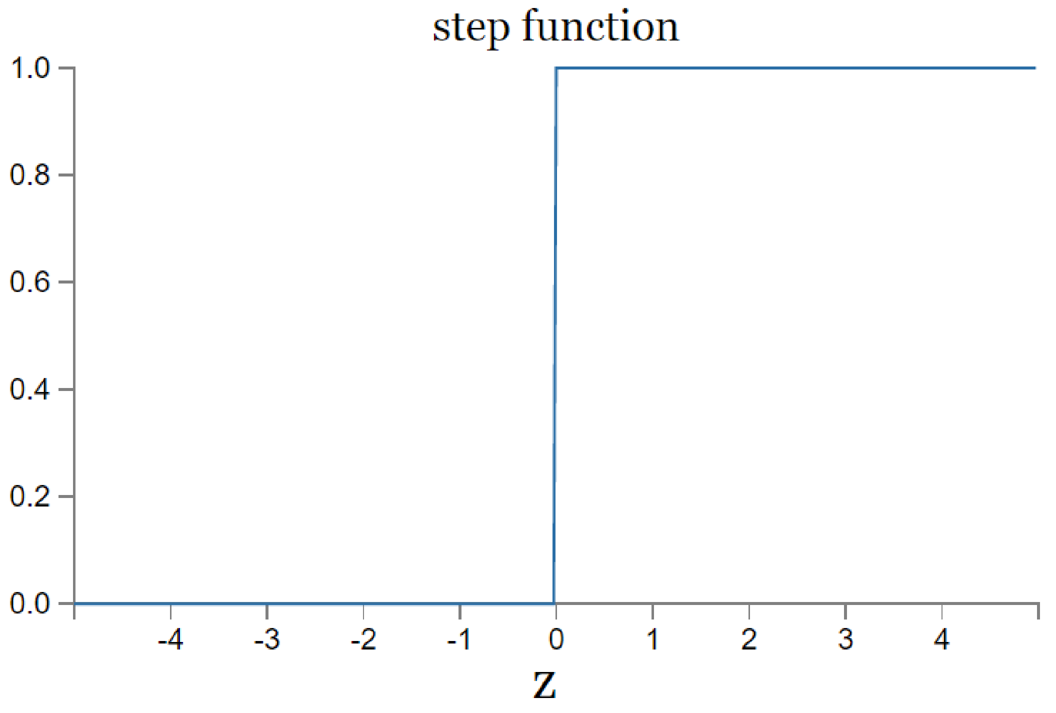


$$h_i = \begin{cases} 1 & \text{if } {w^i}^T x + b_i \geq 0 \\ 0 & \text{if } {w^i}^T x + b_i < 0 \end{cases}$$

$$\hat{y} = \begin{cases} 1 & \text{if } {w^5}^T h + b_5 \geq 0 \\ 0 & \text{if } {w^5}^T h + b_5 < 0 \end{cases}$$

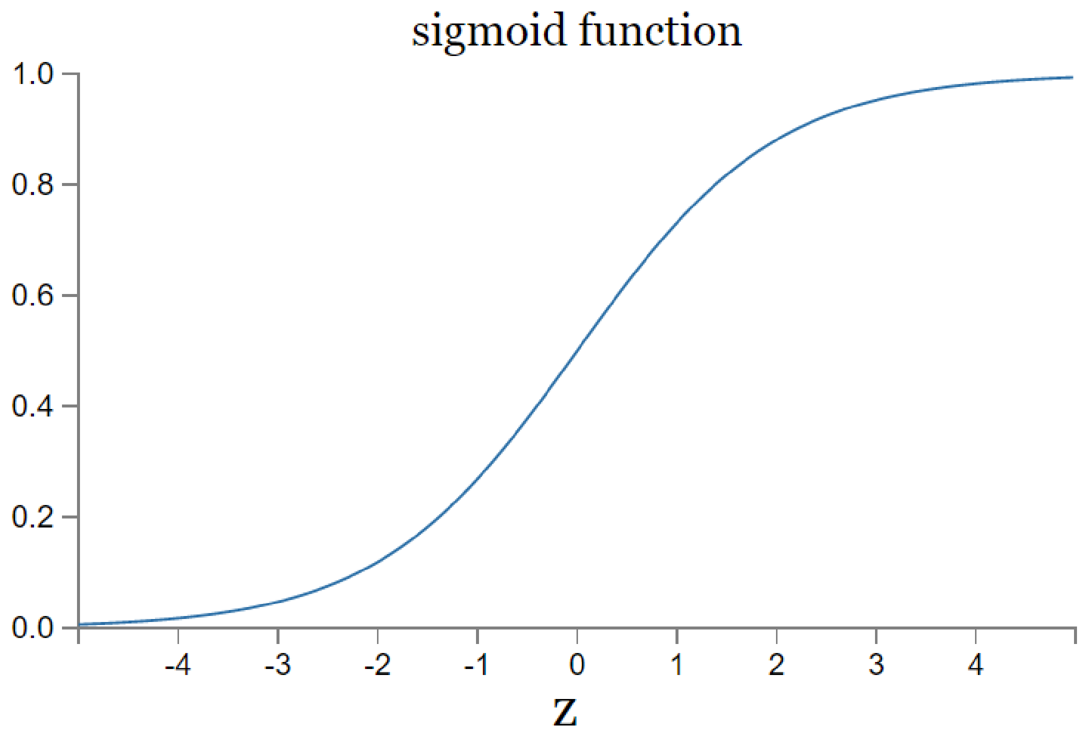
The activation function: step

$$\hat{y} = \begin{cases} 1 & \text{if } w^T x + b \geq 0 \\ 0 & \text{if } w^T x + b < 0 \end{cases}$$



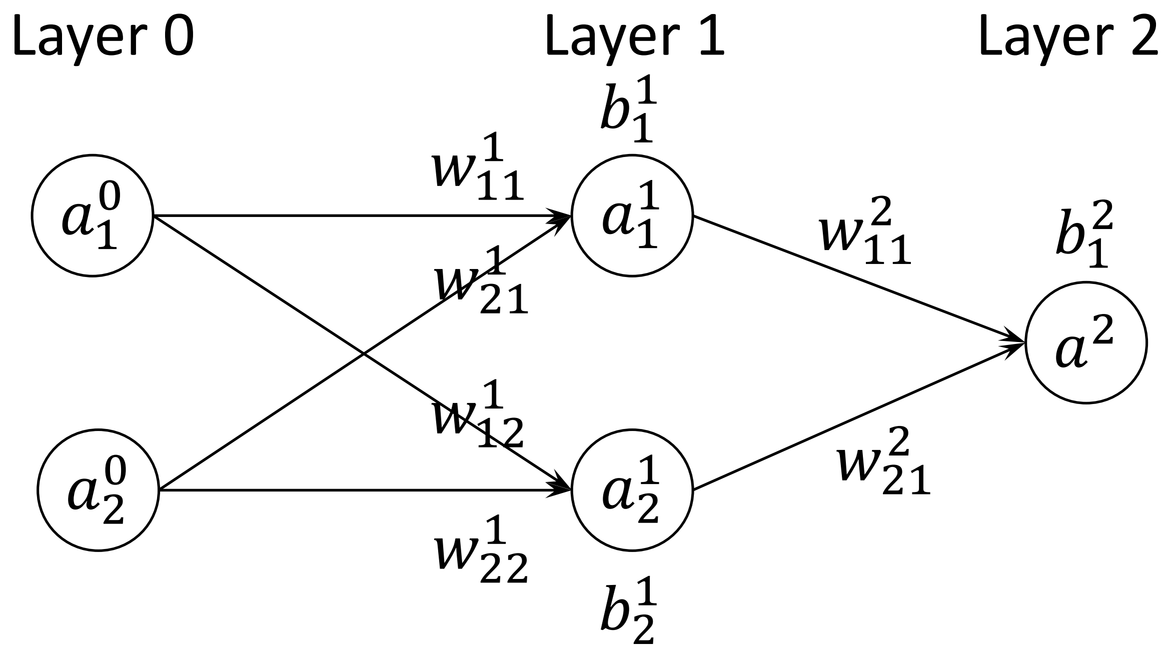
The activation function: sigmoid

$$\hat{y} = \frac{1}{1 + e^{-w^T x - b}} = \sigma(w^T x + b)$$



Optimizing a (deep) neural network

Notation



Where:

- w_{jk}^l weight from node j in layer $l-1$ to node k in layer l
- b_j^l bias in node j of layer l
- $a_j^l = \sigma\left(w^l a^{l-1} + b^l\right), l = 2, 3$
- $a_j^0 = x_j$

Optimizing the weights and biases

- Cost function, e.g.

$$C(y, \hat{y}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

Optimizing the weights and biases

- Cost function, e.g.

$$C(y, X, w, b) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i(w, b, x_i))^2$$

- Minimize the cost function using gradient descent:

$$w_{jk}^l \leftarrow w_{jk}^l - \eta \boxed{\frac{\partial C}{\partial w_{jk}^l}}$$

$$b_j^l \leftarrow b_j^l - \eta \boxed{\frac{\partial C}{\partial b_j^l}}$$

Neural networks (NNs)

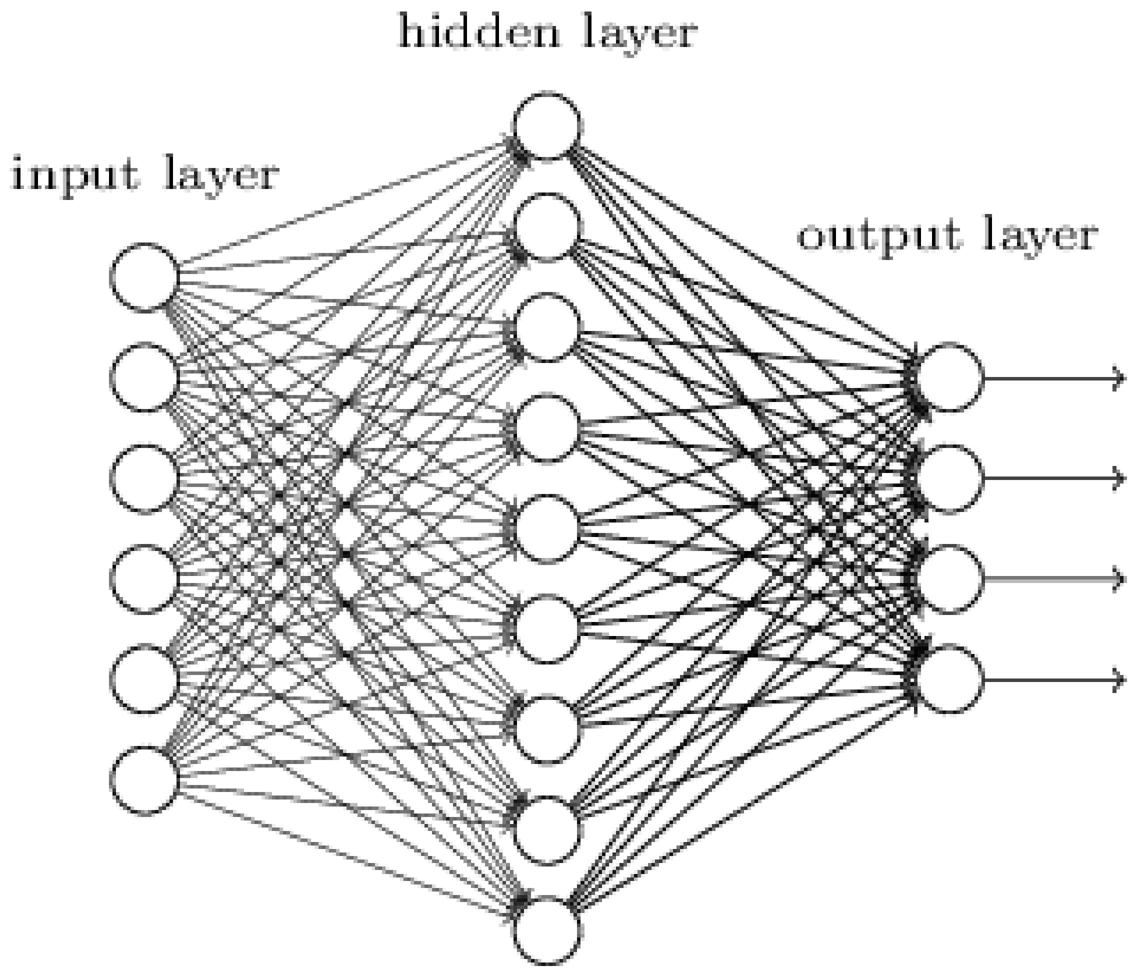


Image from neuralnetworksanddeeplearning.com

Deep neural networks (DNNs)

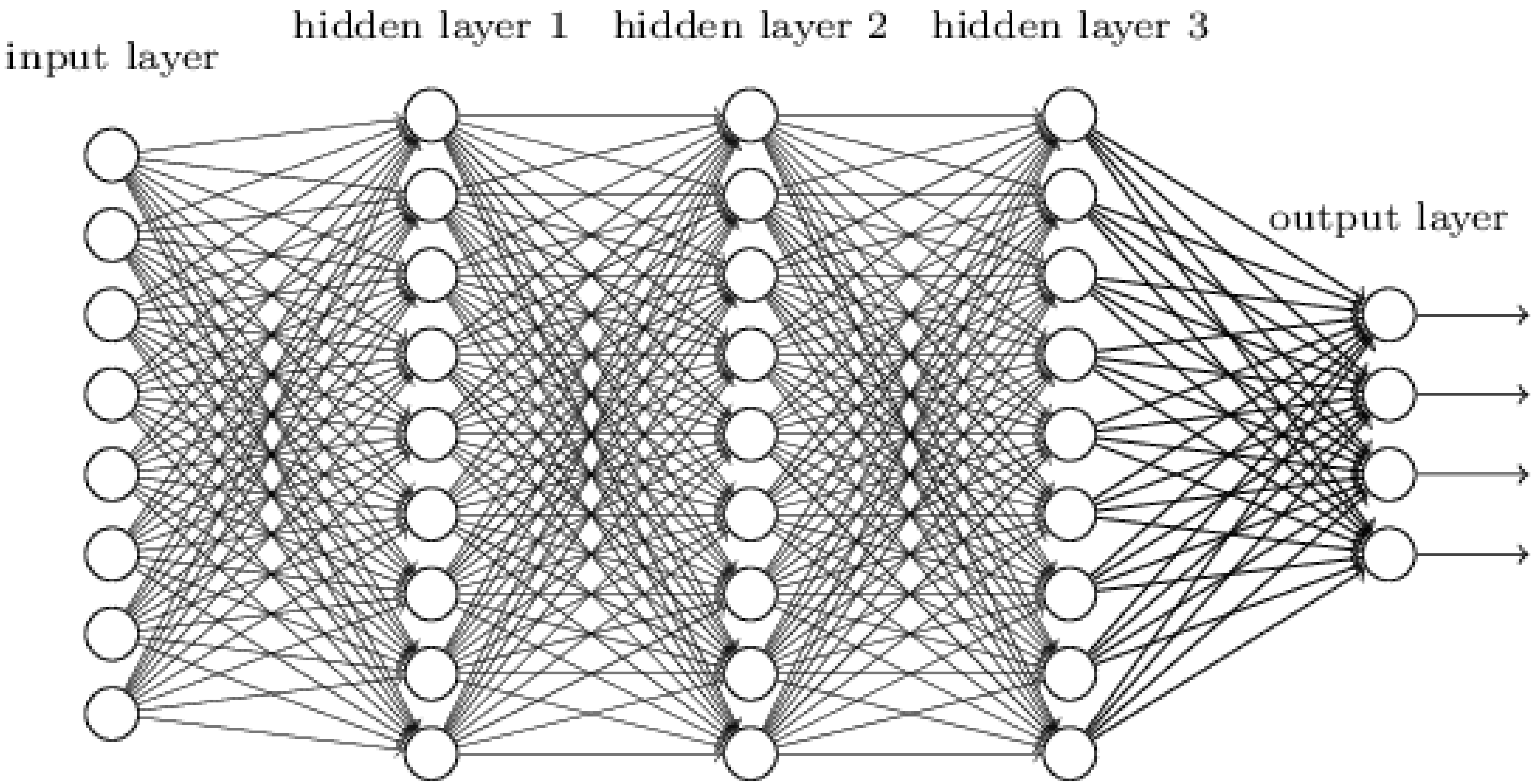
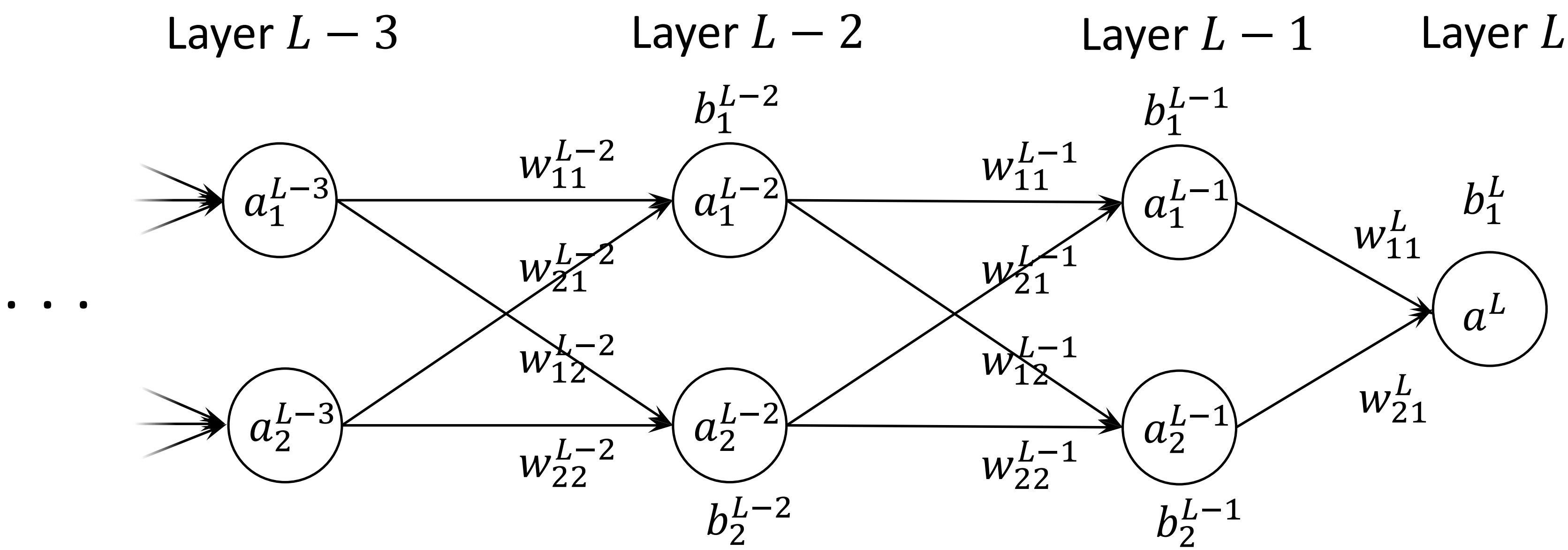


Image from neuralnetworksanddeeplearning.com



Define:

- $a^l = \sigma(w^l a^{l-1} + b^l)$ activation of layer l
- $z^l = w^l a^{l-1} + b^l$ weighted input to layer l
- $\delta_j^l = \frac{\partial C}{\partial z_j^l}$ error in node j of layer l

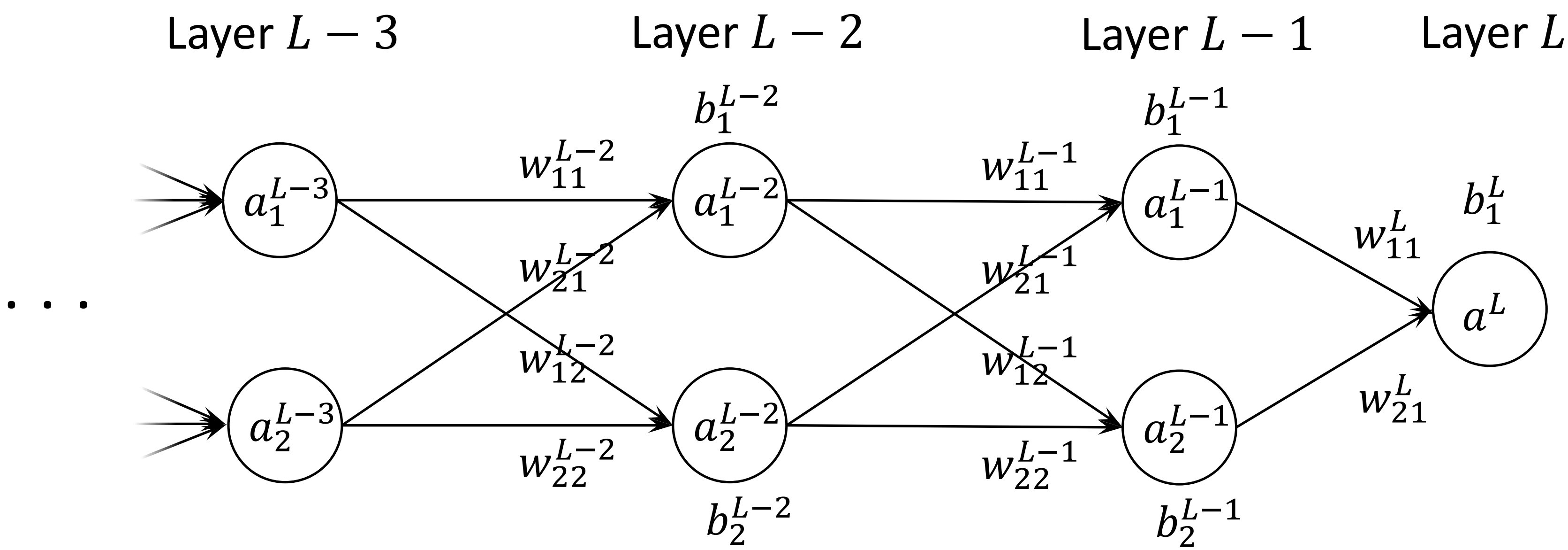
We can now compute:

- $\delta^L = \nabla_{a^L} C \cdot \sigma'(z^L)$
- $\delta^l = \left(\left(w^{l+1}\right)^T \delta^{l+1}\right) \odot \sigma'(z^l)$

→

$$\frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l} \cdot \frac{\partial z_j^l}{\partial b_j^l}$$

$$\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial C}{\partial z_j^l} \cdot \frac{\partial z_j^l}{\partial w_{jk}^l}$$



Backpropagation

Set the input $a^0 = x_j$. Then:

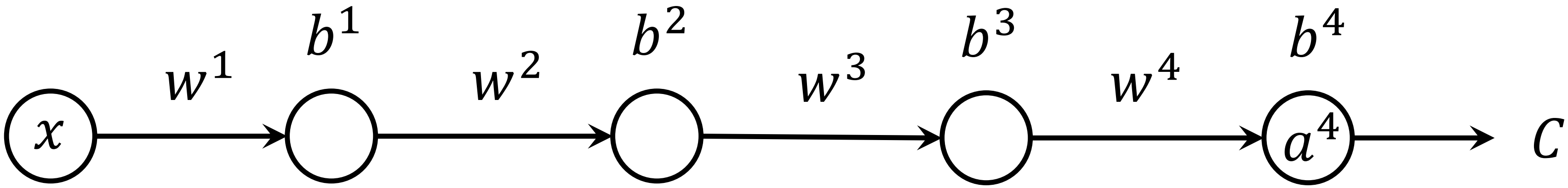
1. Feedforward: compute $z^l = w^l a^{l-1} + b^l$, $a^l = \sigma(z^l)$
2. Output error: compute $\delta^L = \nabla_a C \odot \sigma'(z^L)$
3. Backpropagate the error:

$$\delta^l = \left(\left(w^{l+1}\right)^T \delta^{l+1}\right) \odot \sigma'(z^l)$$

4. Compute the gradients: $\frac{\partial C}{\partial b_j^l} = \delta_j^l$ and $\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l$

Challenge: Unstable gradients

A simple example

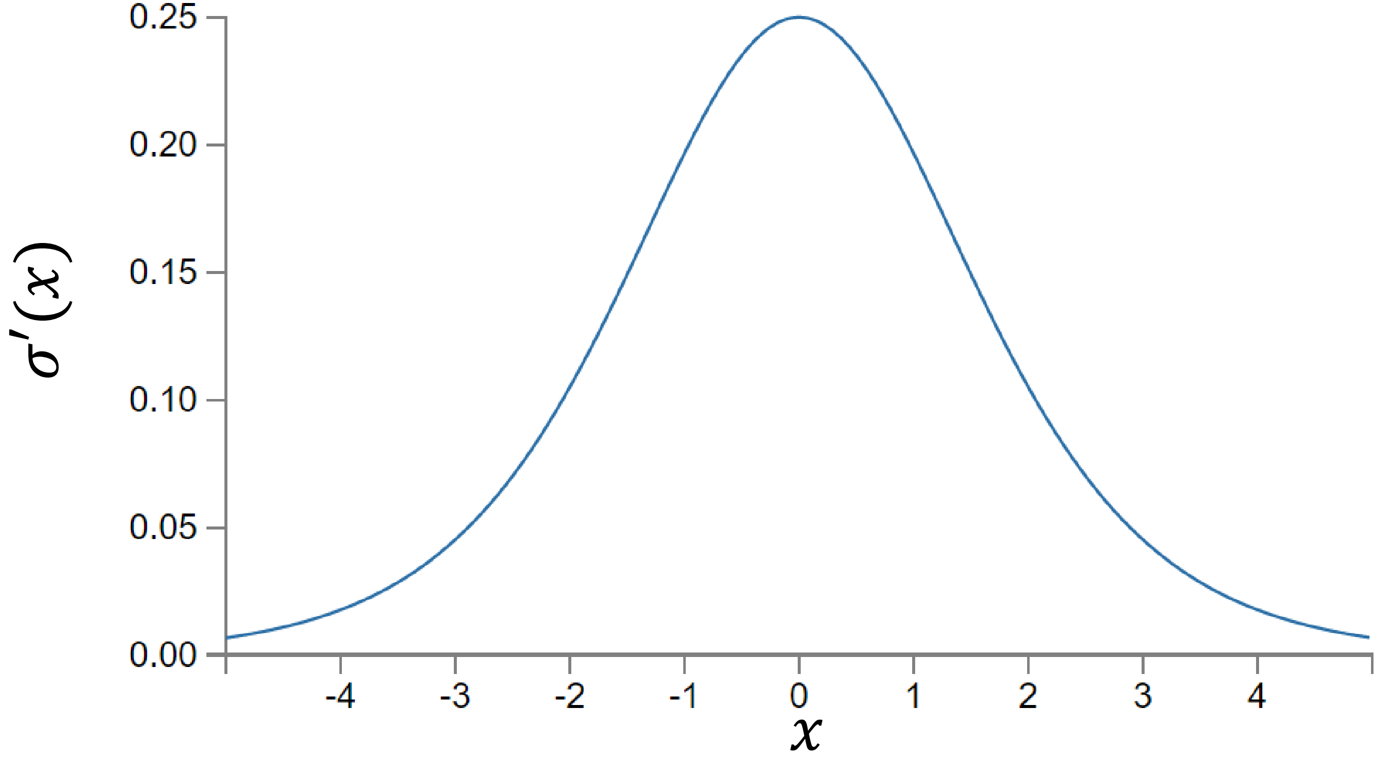


$$\frac{\partial C}{\partial b^4} = \sigma'(z^4)\nabla_{a^4}C$$

$$\frac{\partial C}{\partial b^3} = \sigma'(z^3) \cdot w^4 \cdot \sigma'(z^4)\nabla_{a^4}C$$

$$\frac{\partial C}{\partial b^2} = \sigma'(z^2) \cdot w^3 \cdot \sigma'(z^3) \cdot w^4 \cdot \sigma'(z^4)\nabla_{a^4}C$$

$$\frac{\partial C}{\partial b^1} = \sigma'(z^1) \cdot w^2 \cdot \sigma'(z^2) \cdot w^3 \cdot \sigma'(z^3) \cdot w^4 \cdot \sigma'(z^4)\nabla_{a^4}C$$


0.25
0.20
0.15
0.10
0.05
0.00
σ'(x)
-4
-3
-2
-1
0
1
2
3
4
x

Vanishing gradients

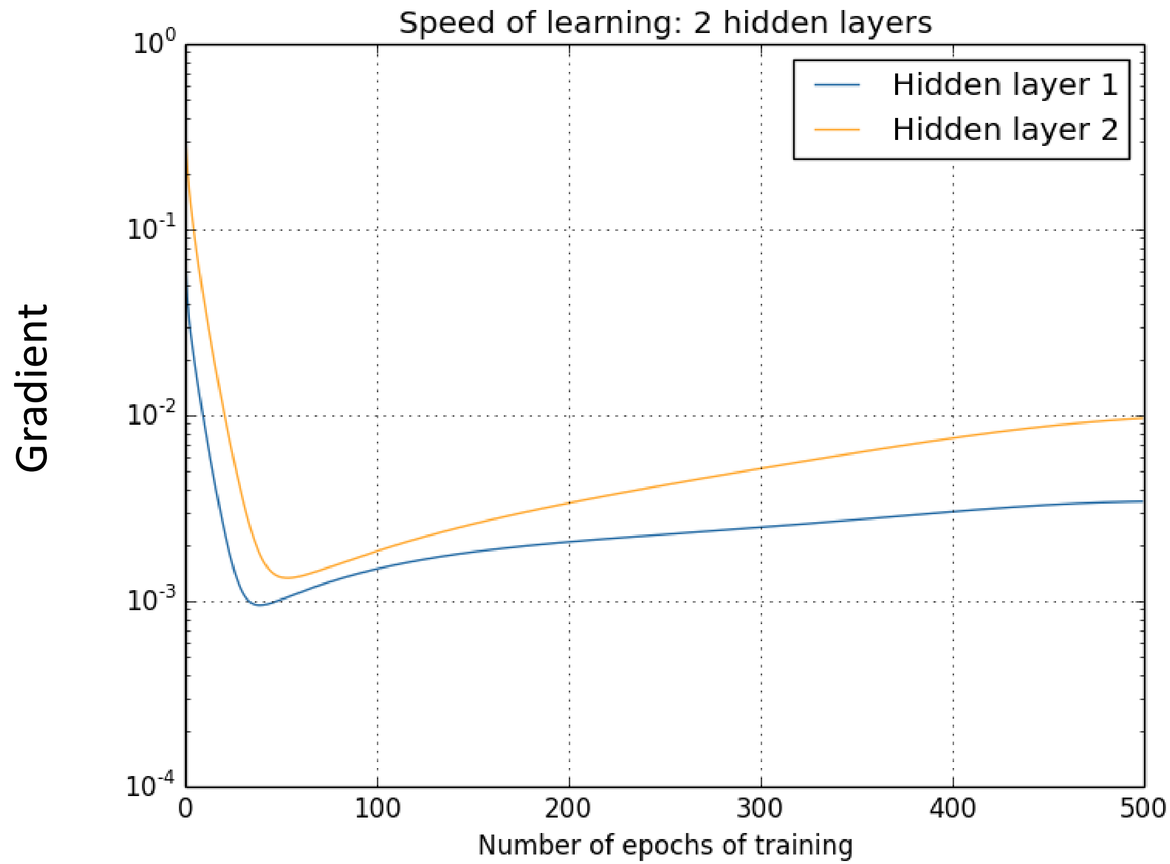


Image from neuralnetworksanddeeplearning.com

Vanishing gradients

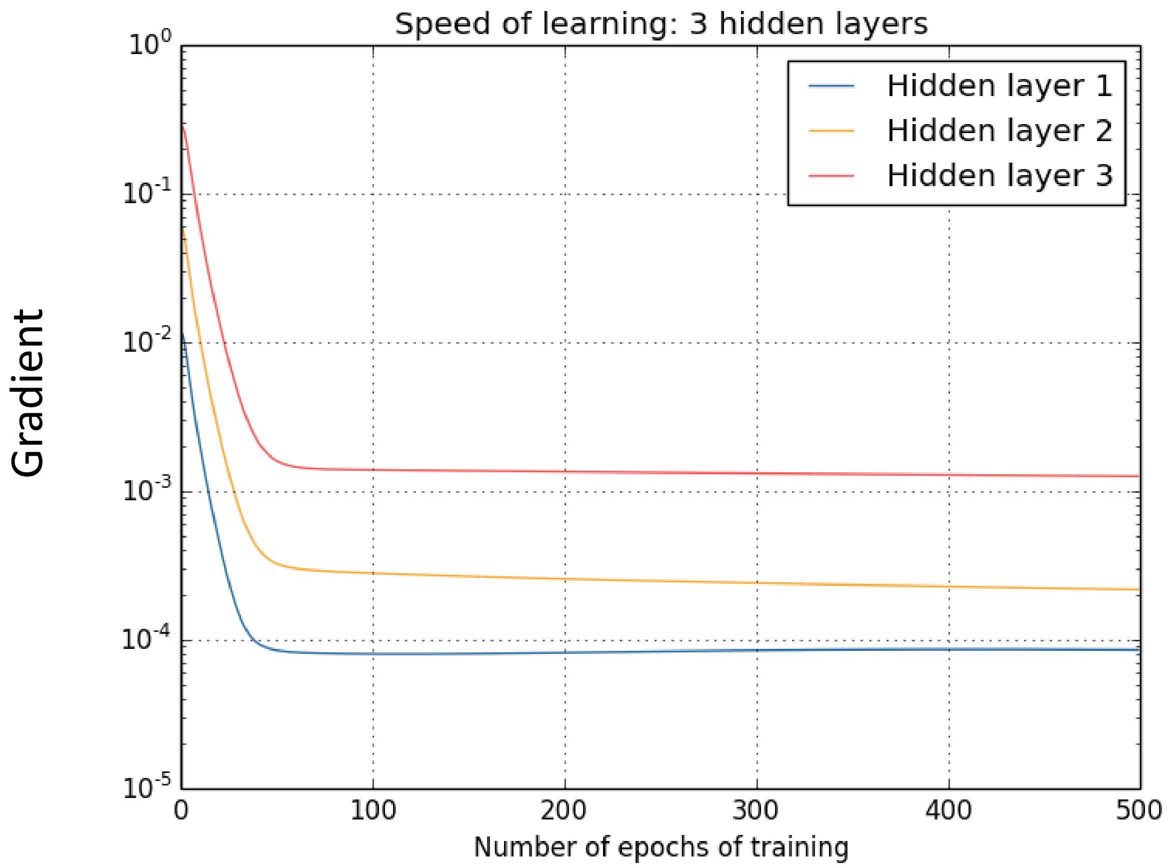


Image from neuralnetworksanddeeplearning.com

Vanishing gradients

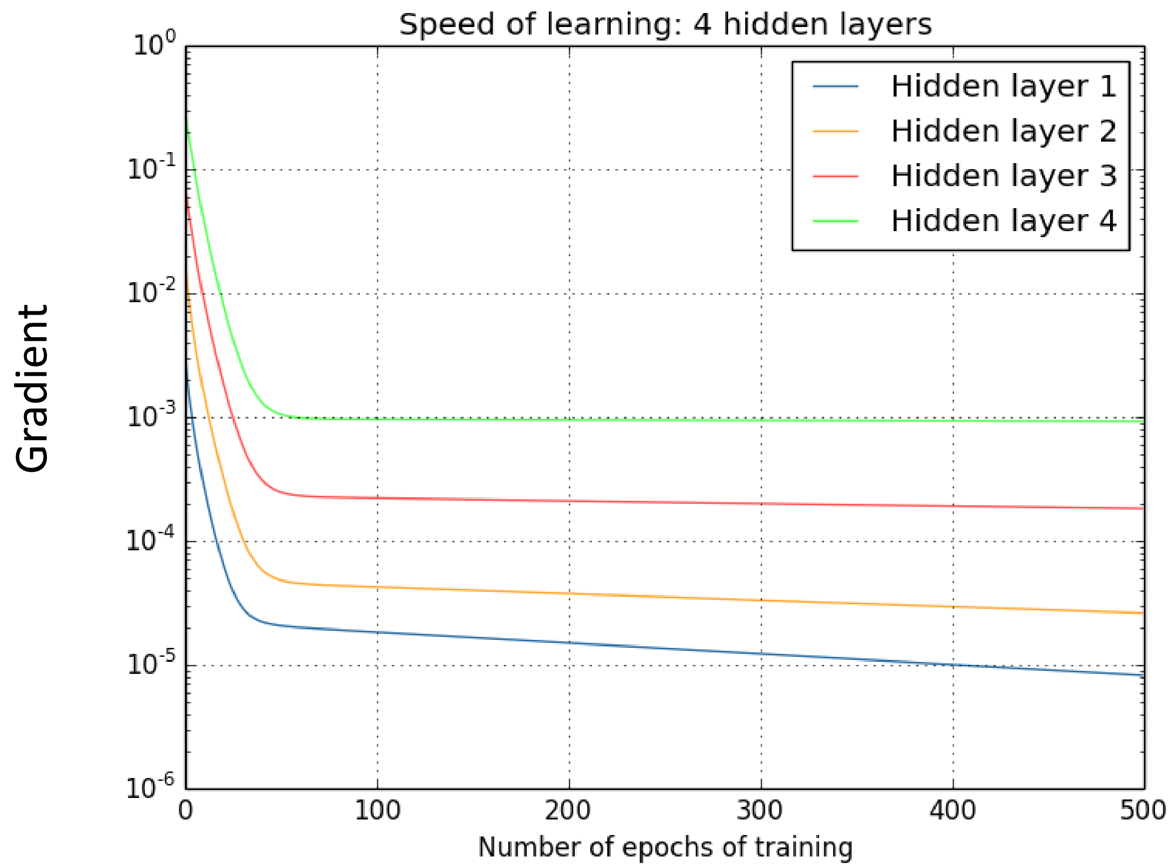


Image from neuralnetworksanddeeplearning.com

Exploding gradients

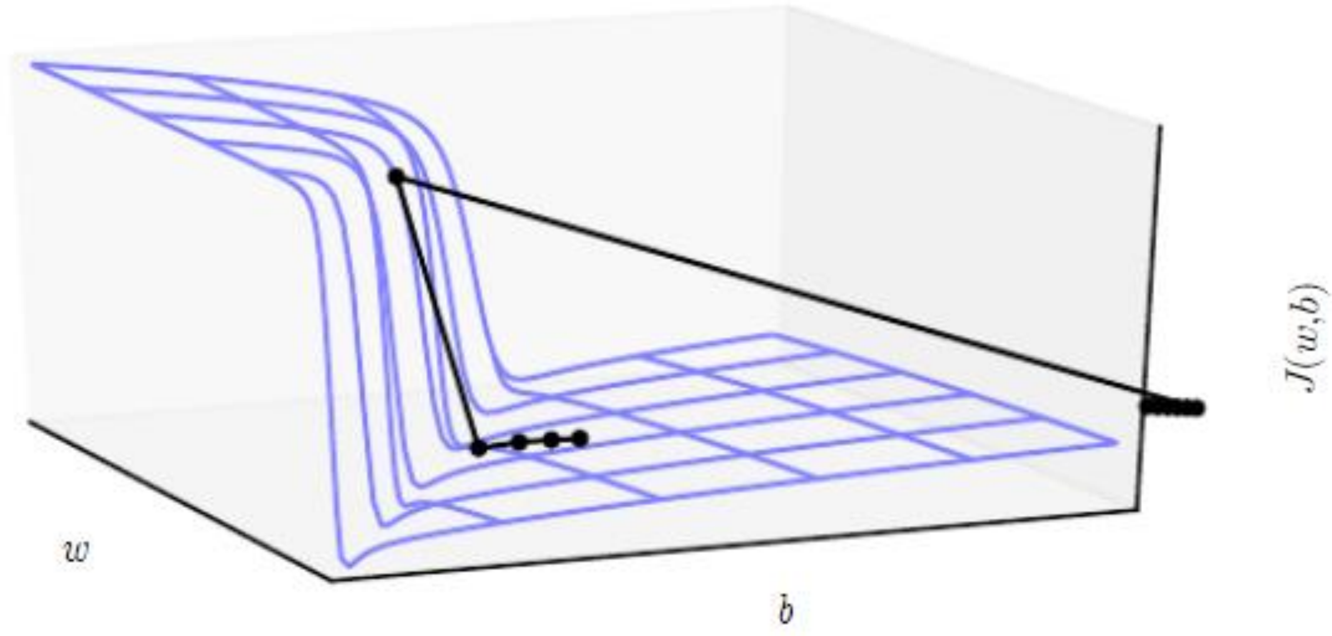


Image from deeplearningbook.org

Solutions

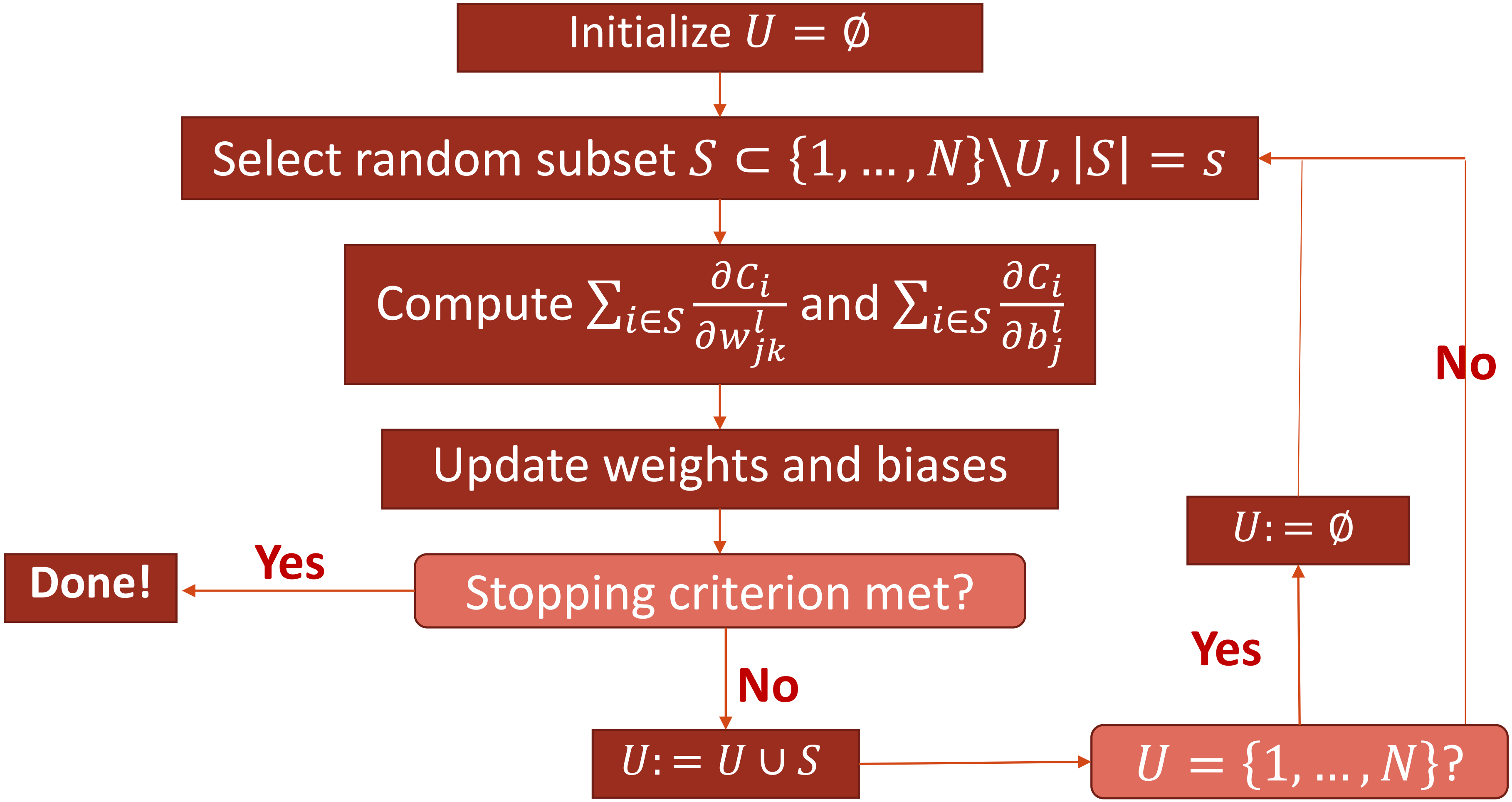
- Other activation functions
- Batch normalization
- Gradient clipping (for exploding gradients)
- Other?

Challenge: large number of samples

$$C(\cdot) = \sum_{i=1}^{N} C_i(\cdot)$$

$$\frac{\partial C}{\partial w_{jk}^{l}} = \sum_{i=1}^{N} \frac{\partial C_i}{\partial w_{jk}^{l}}$$

Stochastic gradient descent



Challenge: valleys

Or: the Hessian is ill-conditioned

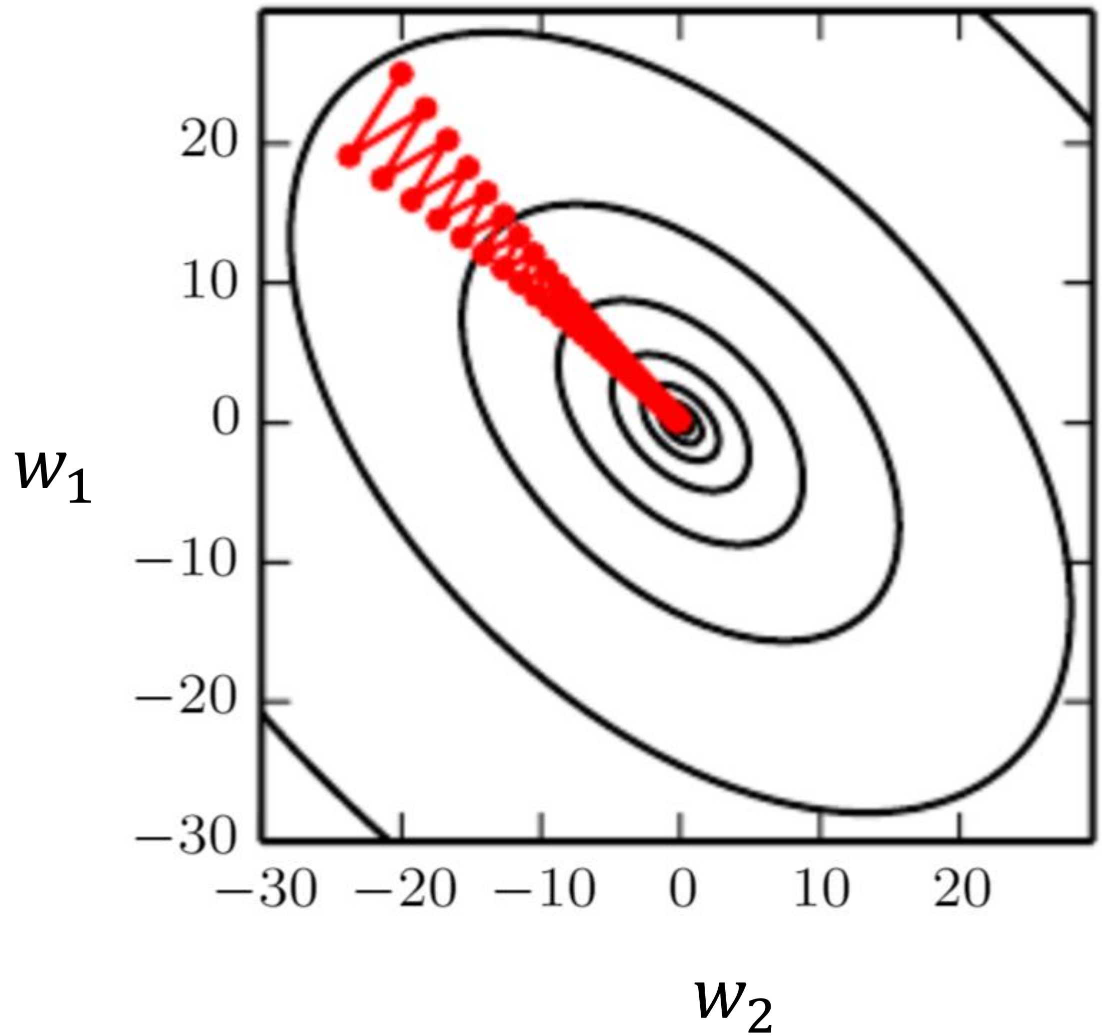


Image from deeplearningbook.org

Solution: momentum

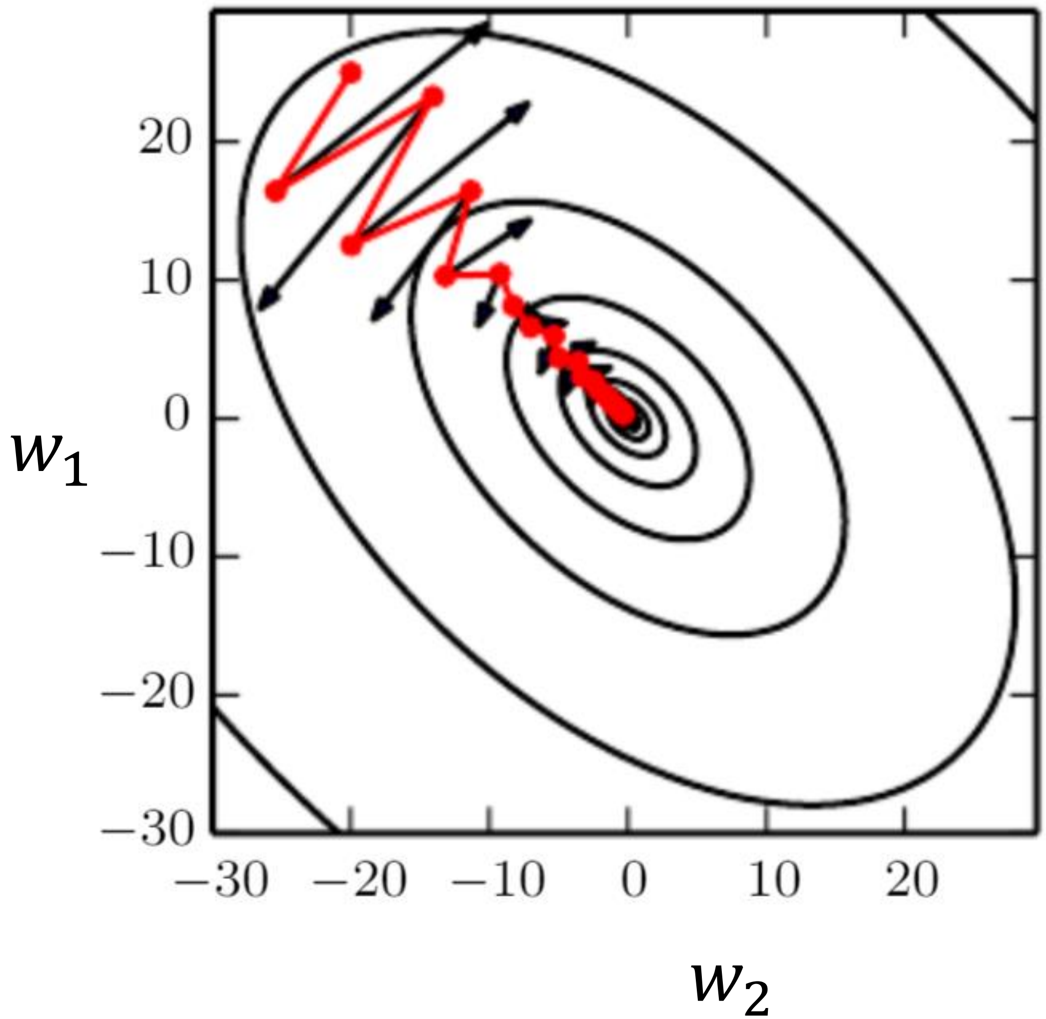
- Gradient descent:

$$w \leftarrow w - \eta \nabla_w C$$

- Momentum:

$$v \leftarrow \alpha \cdot v - \eta \nabla_w C$$

$$w \leftarrow w + v$$



20
10
0
−10
−20
−30
w_1
−30
−20
−10
0
10
20
w_2

Solution: AdaGrad

- Gradient descent:

$$w \leftarrow w - \eta \nabla_w C$$

- AdaGrad:

$$r \leftarrow r + \nabla_w C^T \nabla_w C$$

$$w \leftarrow w - \frac{\eta}{\sqrt{\delta + r}} \nabla_w C$$

Solution: RMSProp

- Gradient descent:

$$w \leftarrow w - \eta \nabla_w C$$

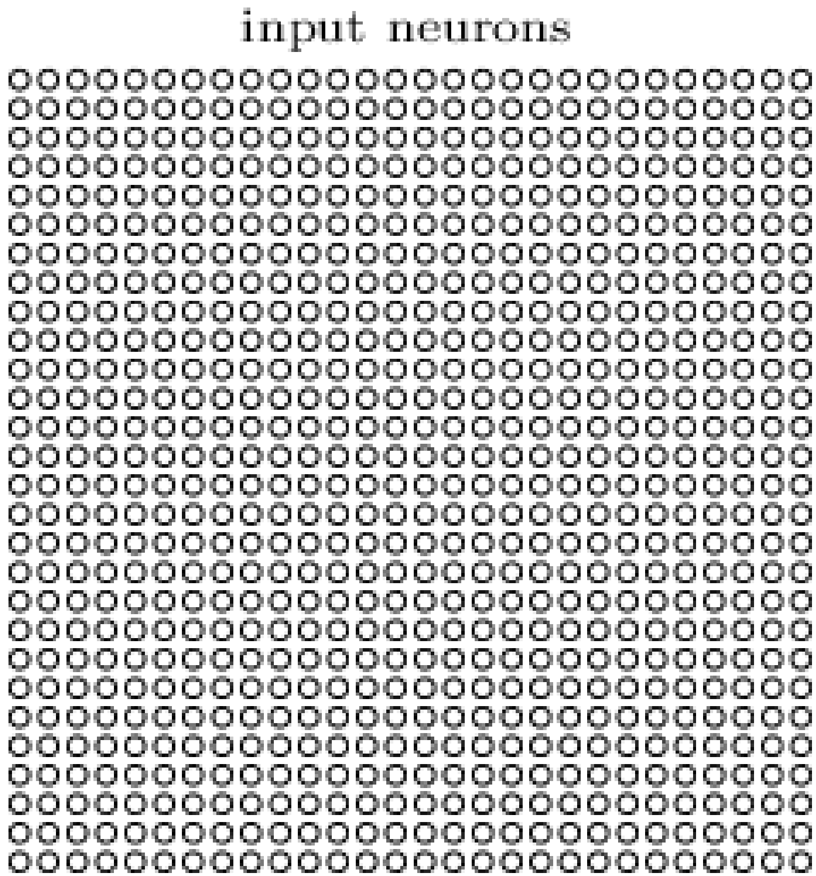
- RMSProp:

$$r \leftarrow \alpha r + (1 - \alpha) \nabla_w C^T \nabla_w C$$

$$w \leftarrow w - \frac{\eta}{\sqrt{\delta + r}} \nabla_w C$$

Challenge: architecture design

Convolution layers



Convolution layers: Local receptive fields

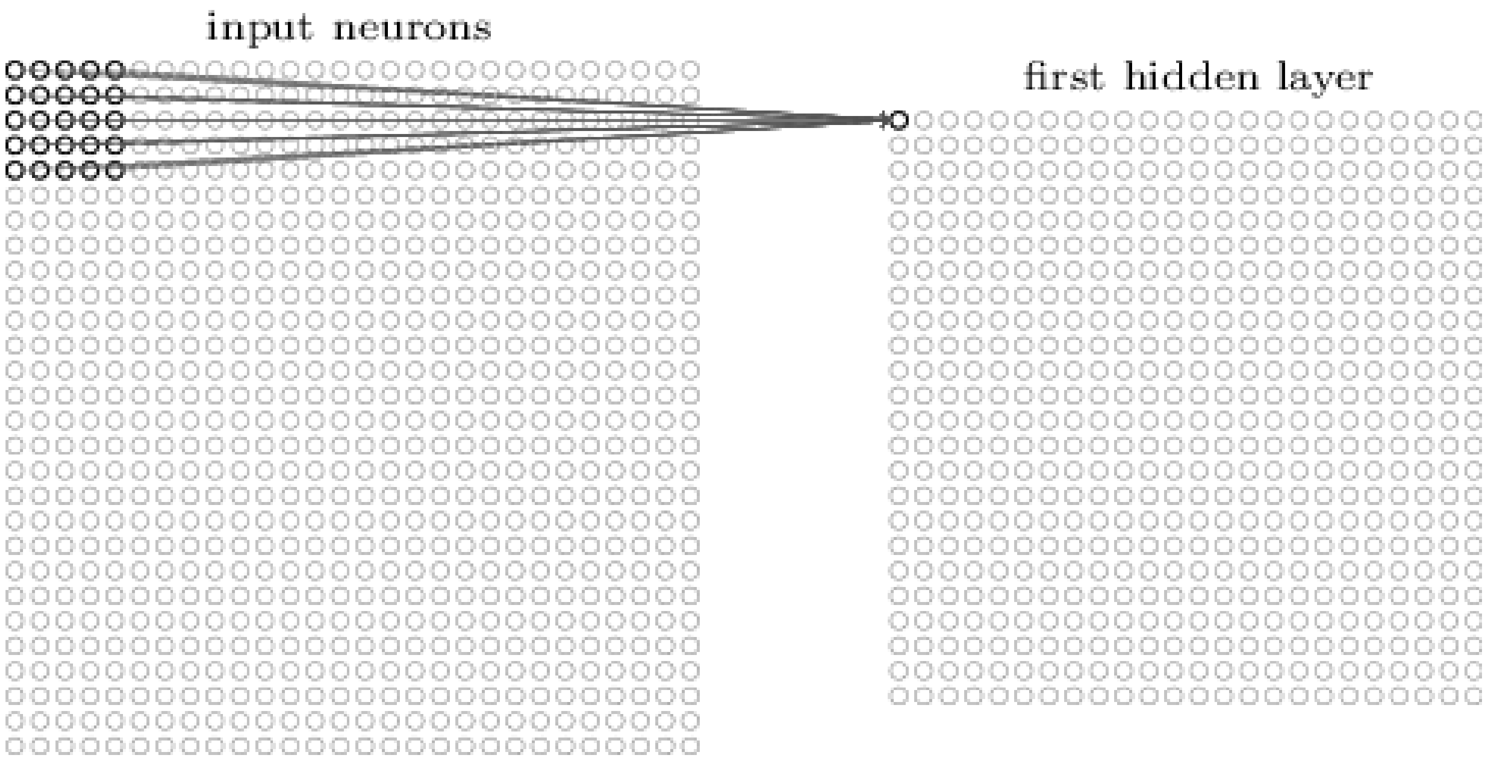


Image from neuralnetworksanddeeplearning.com

Convolution layers: Local receptive fields

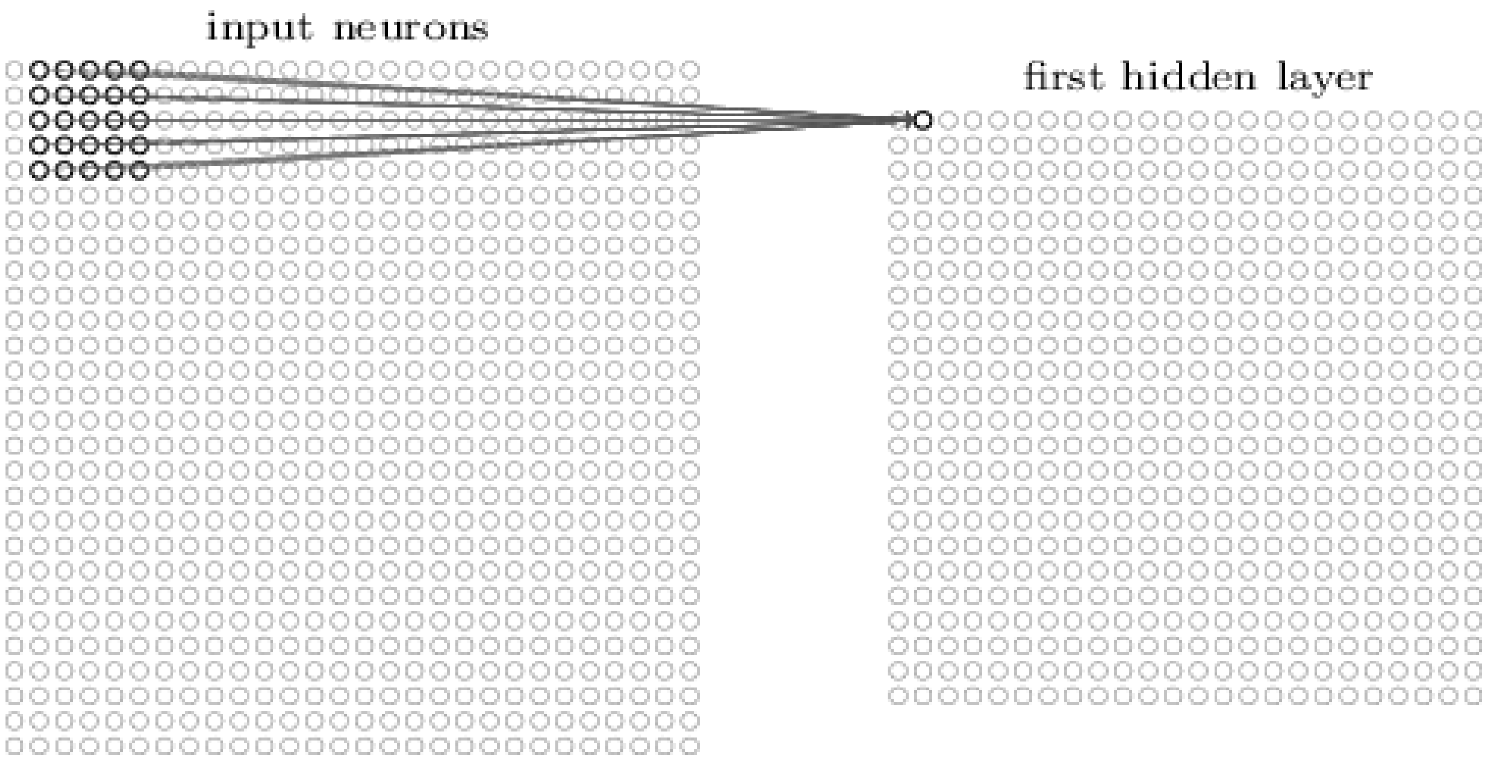


Image from neuralnetworksanddeeplearning.com

Learning features

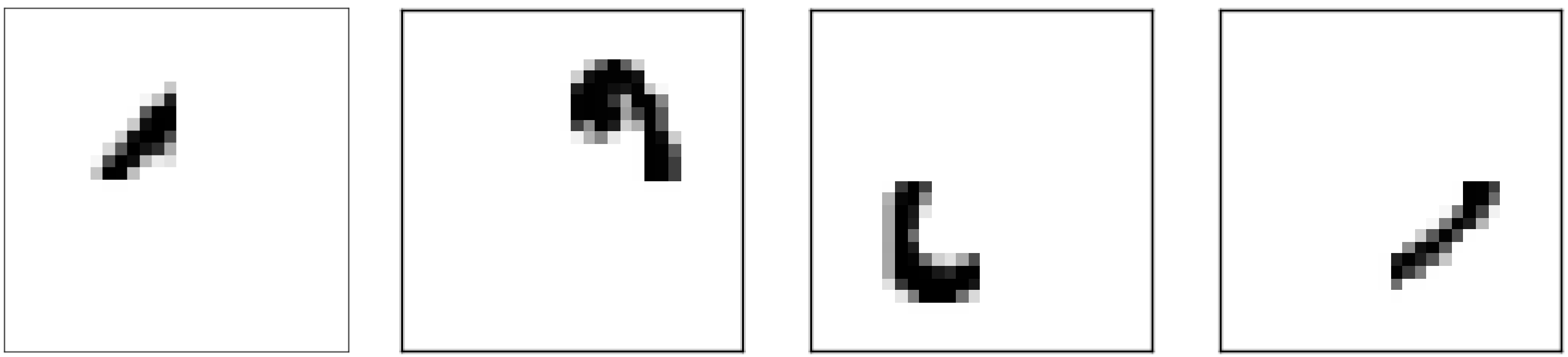
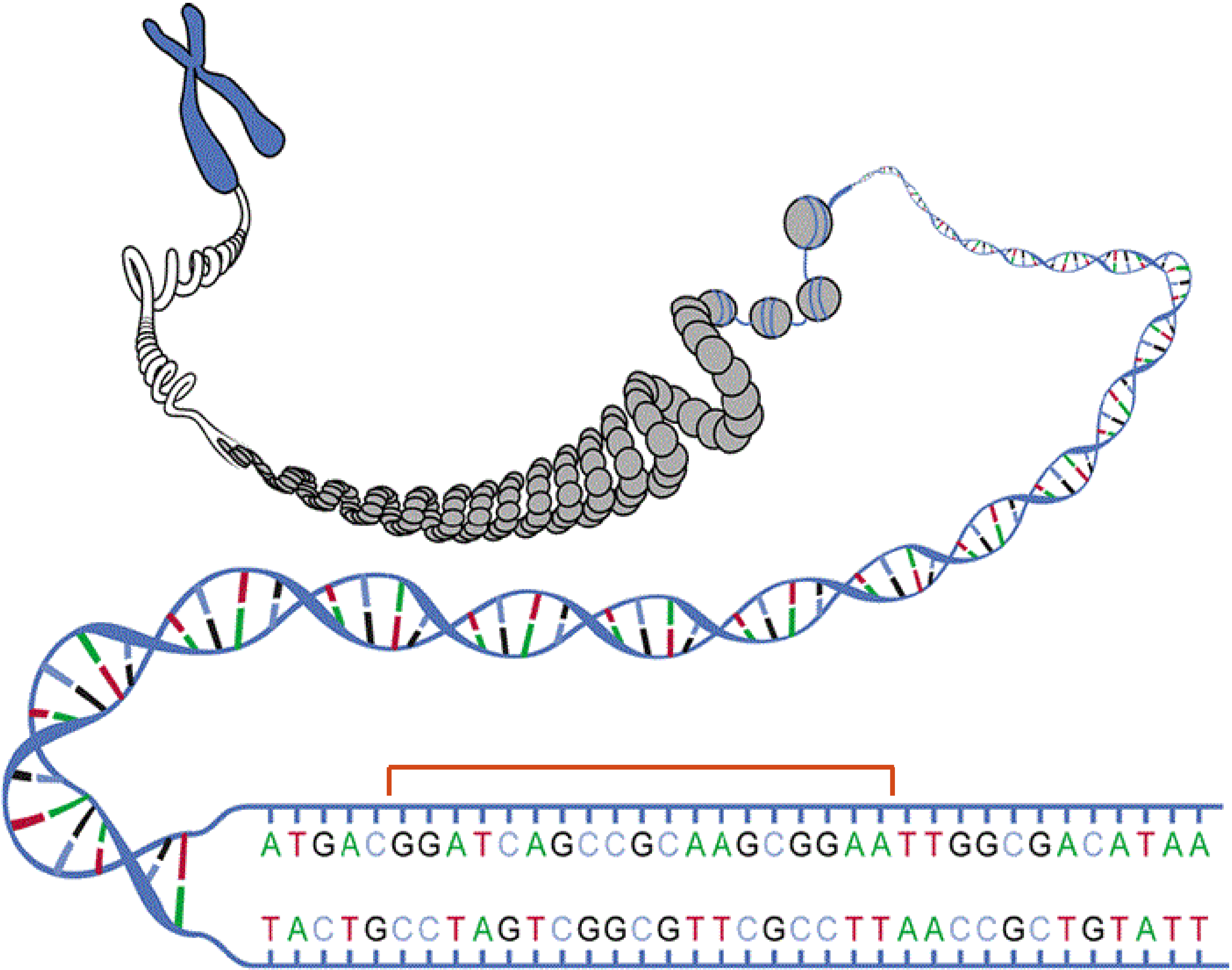
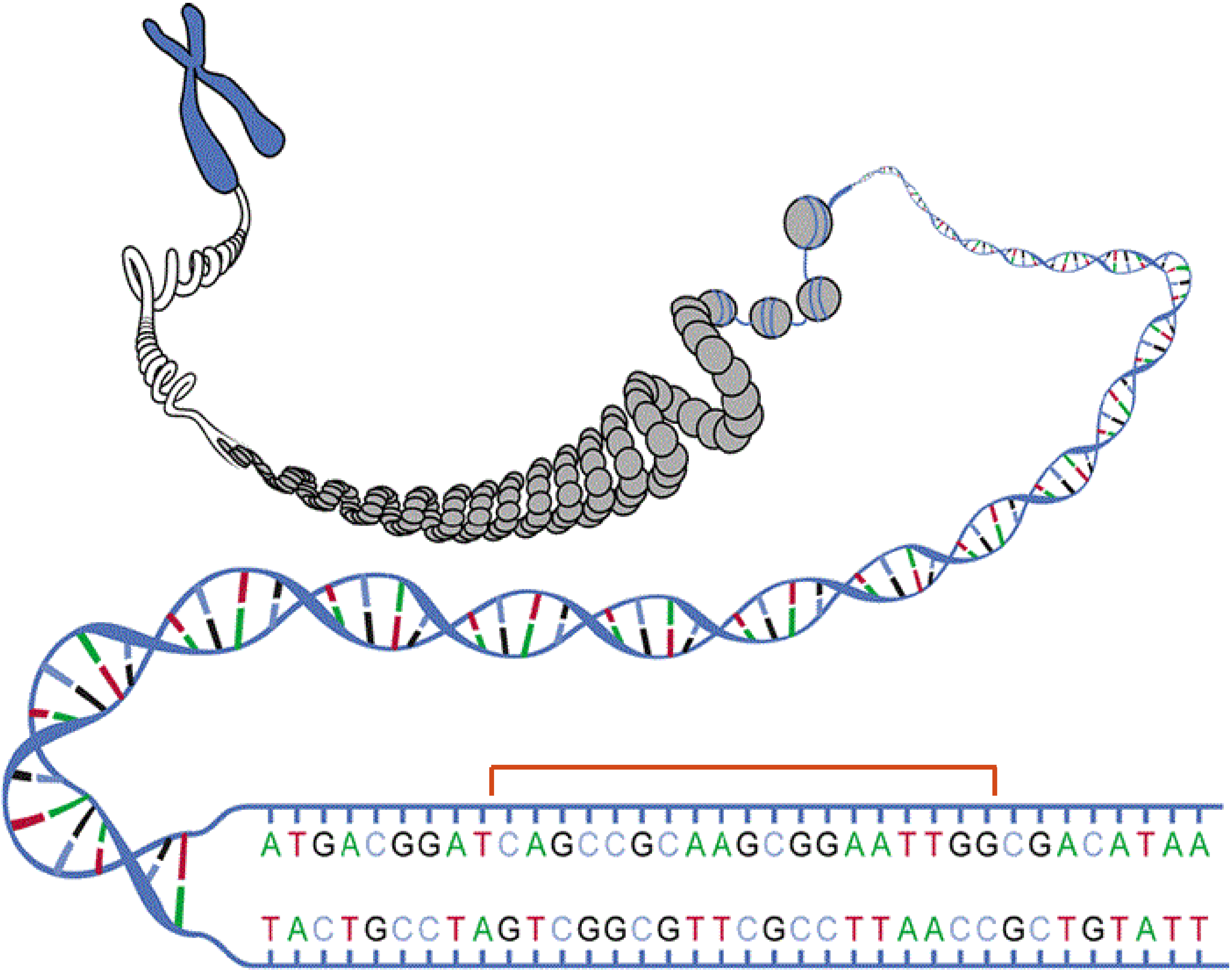
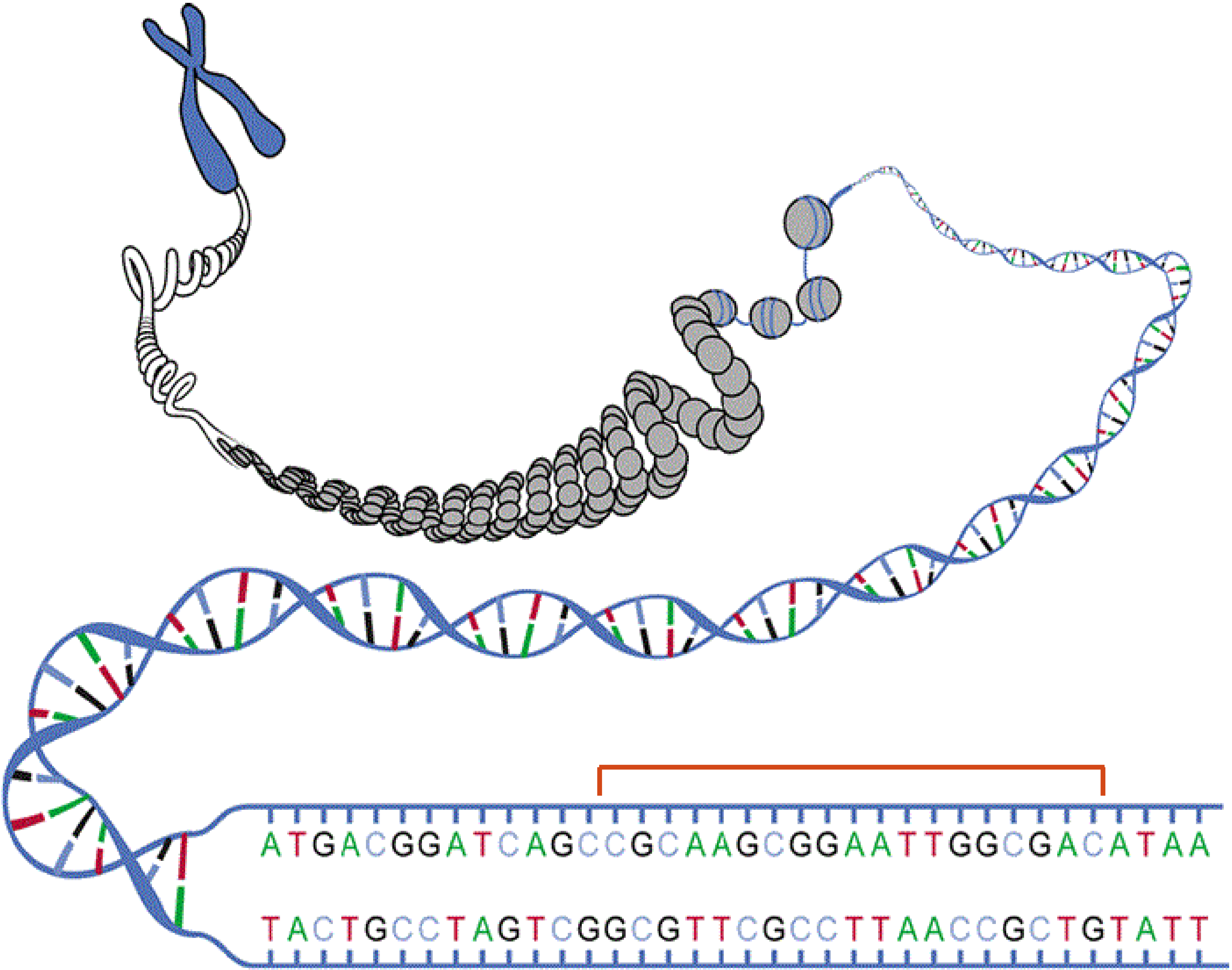
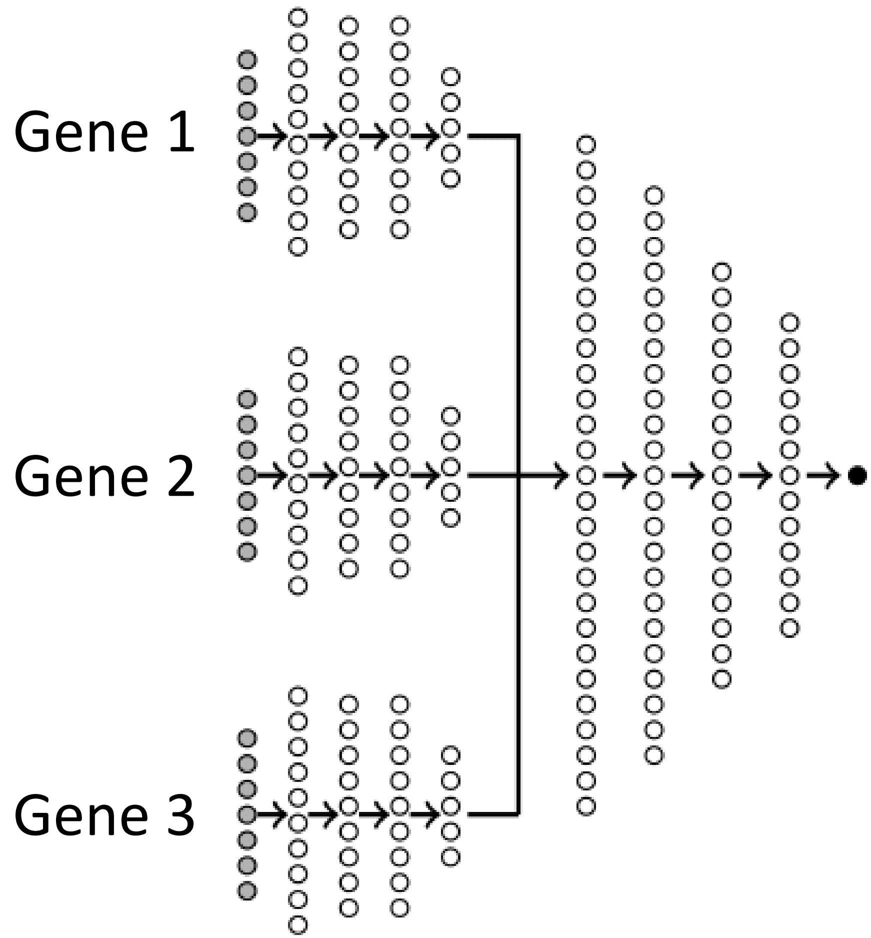


Image from neuralnetworksanddeeplearning.com

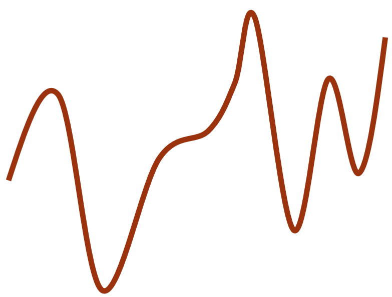

ATGACGGATCAGCCGCAAGCGGAATTGGCGACATAA
TACTGCCTAGTCGGCGTTCGCCTTAACCGCTGTATT


A T G A C G G A T C A G C C G C A A G C G G A A T T G G C G A C A T A A
T A C T G C C T A G T C G G C G T T C G C C T T A A C C G C T G T A T T

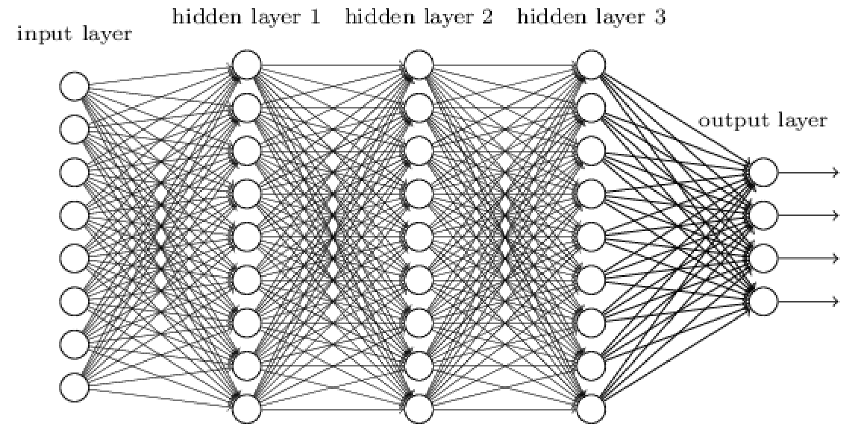

ATGACGGATCAGCCGCAAGCGGAATTGGCGACATAA
TACTGCCTAGTCGGCGTTCGCCTTAACCGCTGTATT


Gene 1
Gene 2
Gene 3

Some more challenges...



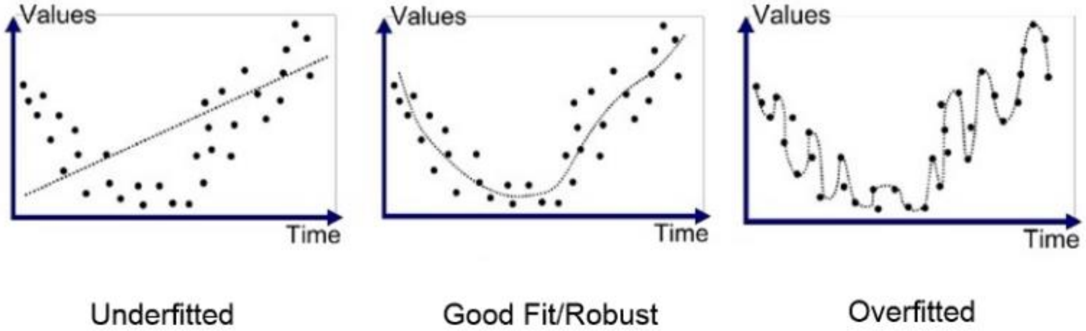
Local minima and saddle points



Network design is a manual process

$$H_x = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_1 x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n x_1} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

Second order methods do not scale well



Risk of overfitting

To conclude

Tilburg University

m.Balvert@uvt.nl