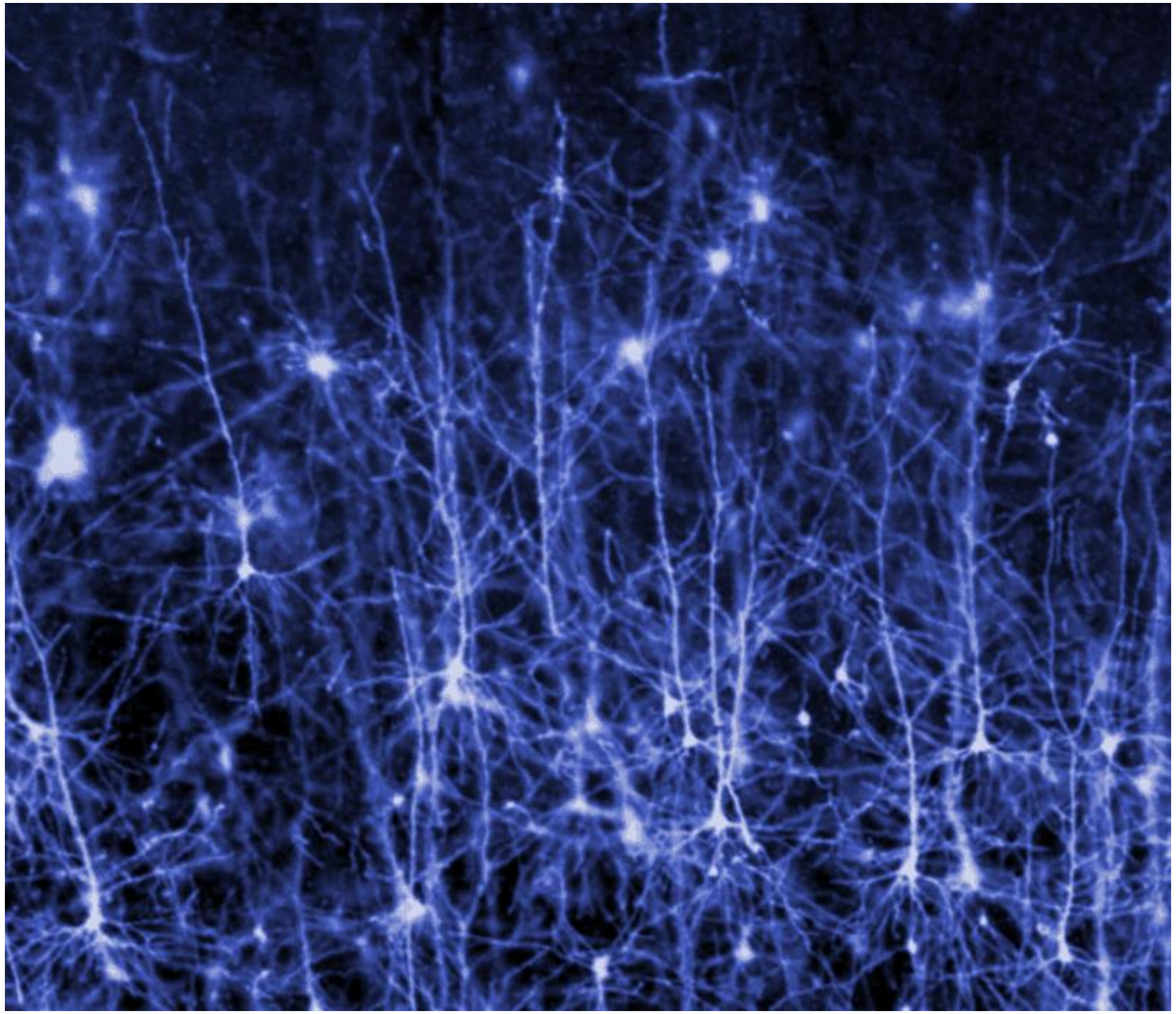
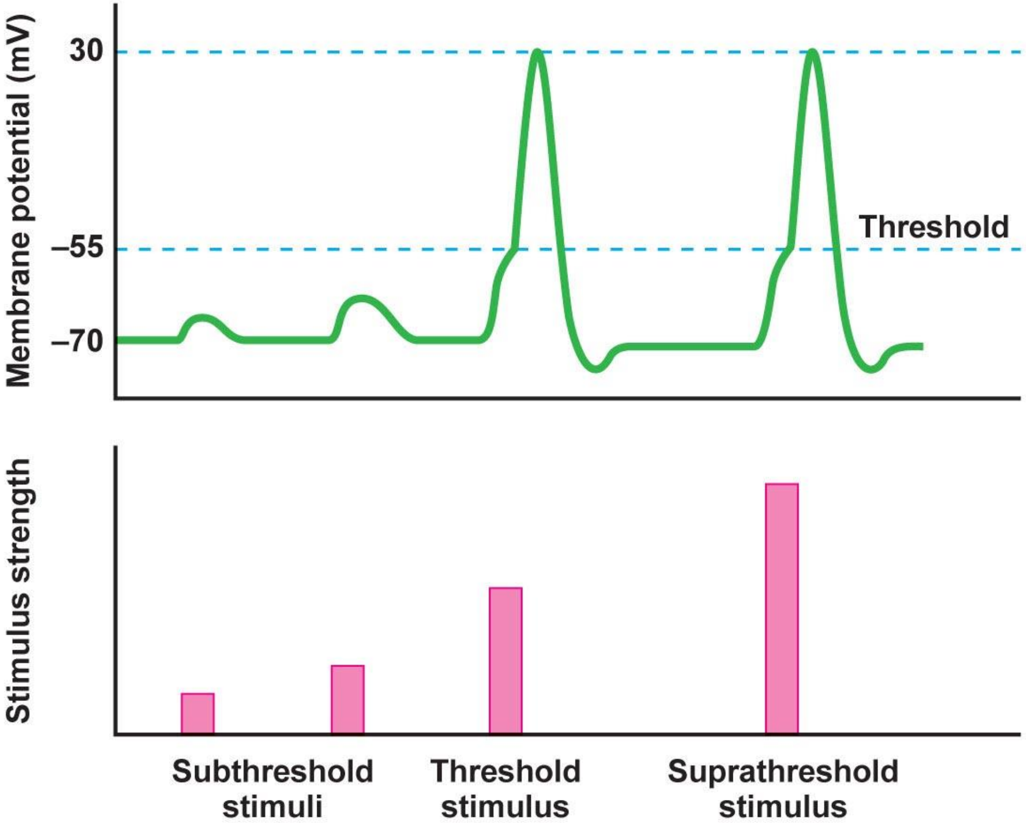
Neural networks

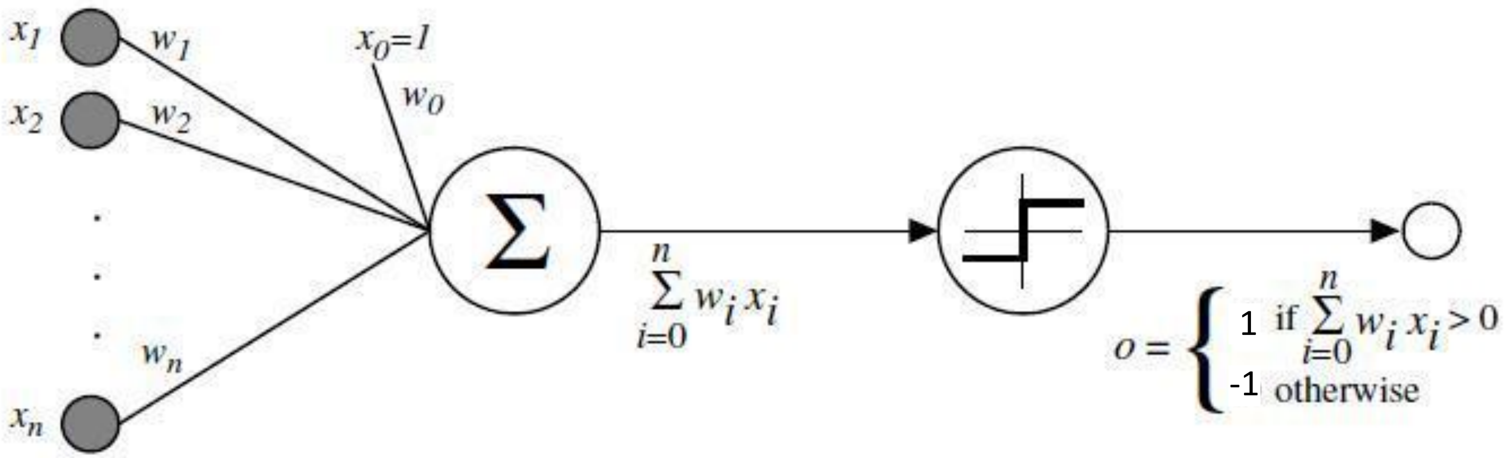
Biological neural networks





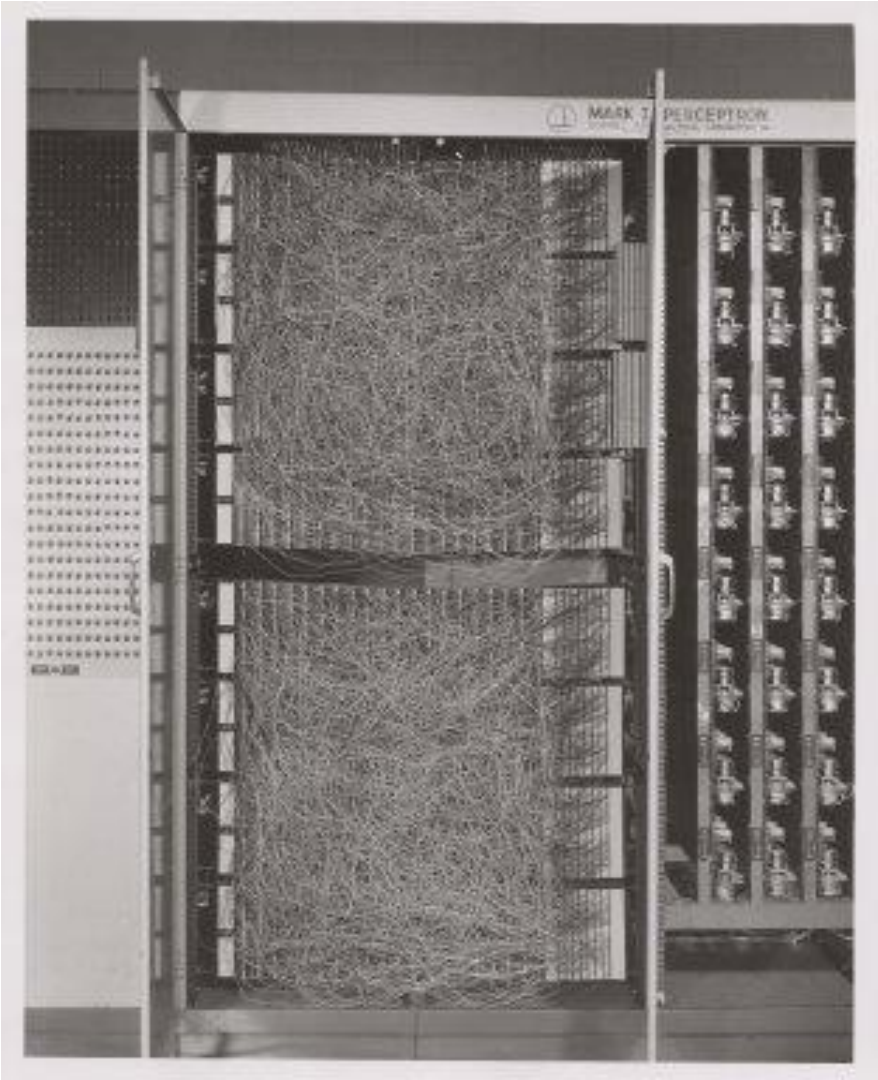
All-or-none law

Perceptron (Frank Rosenblatt, 1957)

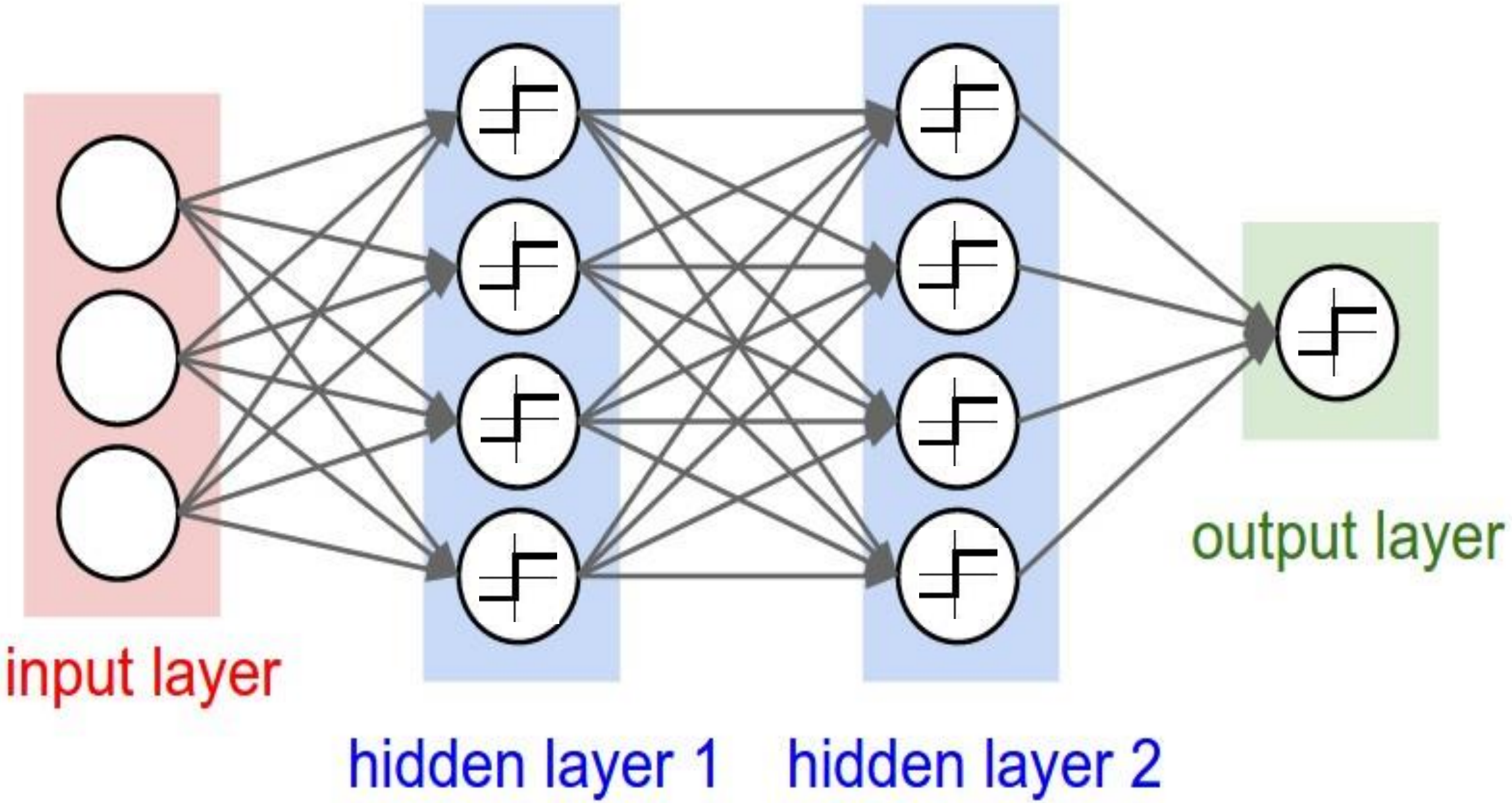


$$\text{sign}(\mathbf{w}^\intercal \mathbf{x}_n) \neq y_n$$

$$\mathbf{w} \leftarrow \mathbf{w} + y_n \mathbf{x}_n$$

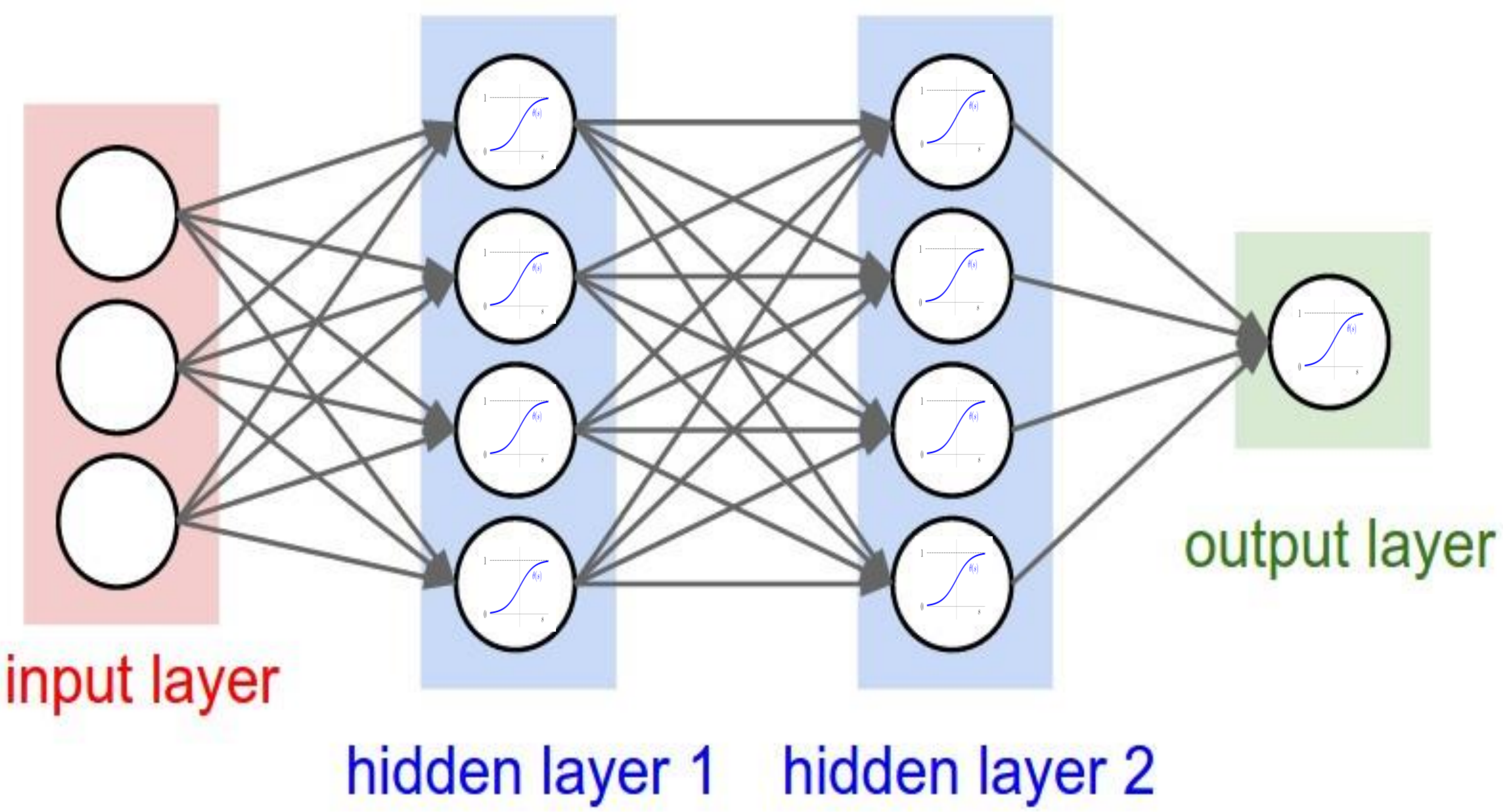


Multi-layer perceptron (MLP)



Neural networks are very limited - Marvin Minsky and Seymour Papert, 1969

Multi-layer perceptron (MLP)



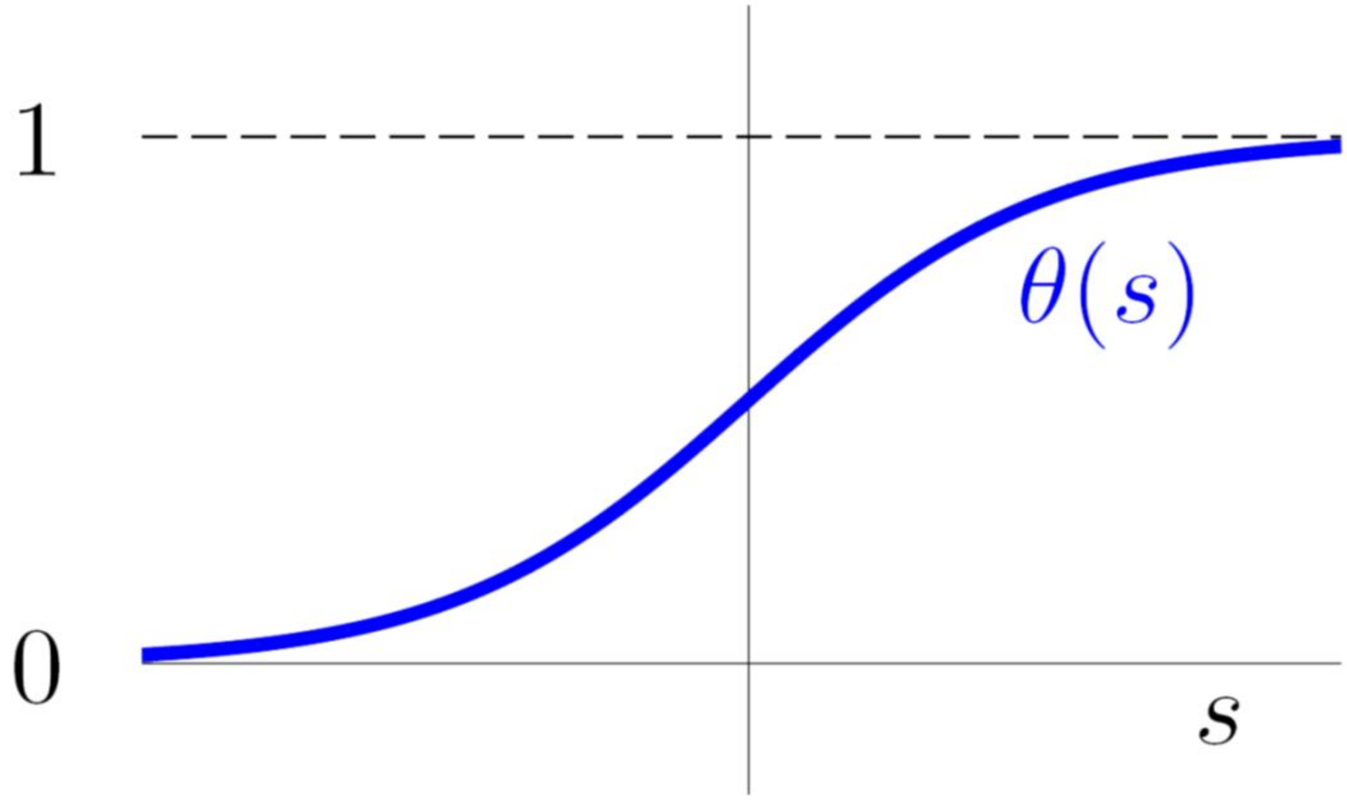
Neural networks are very limited - Marvin Minsky and Seymour Papert, 1969

Logistic function (sigmoid)

$$P(y \mid \mathbf{x}) = \begin{cases} f(\mathbf{x}) & \text{for } y = +1; \\ 1 - f(\mathbf{x}) & \text{for } y = -1. \end{cases}$$

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

$$\sigma(-s) = 1 - \sigma(s)$$



Logistic regression

$$P(y \mid \mathbf{x}) = \begin{cases} f(\mathbf{x}) & \text{for } y = +1; \\ 1 - f(\mathbf{x}) & \text{for } y = -1. \end{cases}$$

$$\sigma(-s) = 1 - \sigma(s)$$

$$P(y \mid \mathbf{x}) = \sigma(y\ \mathbf{w}^\intercal \mathbf{x})$$

Logistic regression

$$P(y \mid \mathbf{x}) = \begin{cases} f(\mathbf{x}) & \text{for } y = +1; \\ 1 - f(\mathbf{x}) & \text{for } y = -1. \end{cases}$$

$$\sigma(-s) = 1 - \sigma(s)$$

$$P(y \mid \mathbf{x}) = \sigma(y\ \mathbf{w}^{\intercal}\mathbf{x})$$

Likelihood: $\prod_{i=1}^{N} P(y_i|\mathrm{x}_i) = \prod_{i=1}^{N} \sigma\left(y_i \mathrm{w}^{\mathrm{T}} \mathrm{x}_i\right)$

Logistic regression loss function

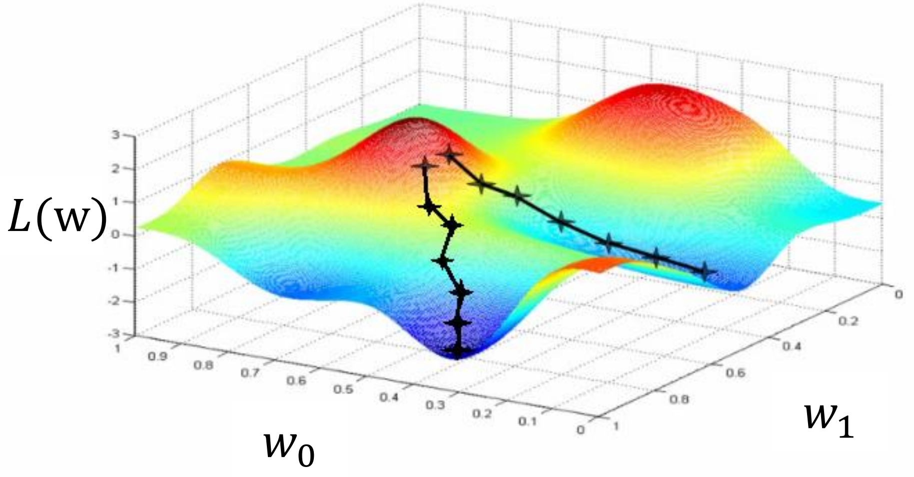
Likelihood: $$\prod_{i=1}^{N} P(y_i|\mathrm{x}_i) = \prod_{i=1}^{N} \sigma\left(y_i \mathrm{w}^{\mathrm{T}} \mathrm{x}_i\right)$$

$$L(\mathrm{w}) = -\frac{1}{N} \ln\left(\prod_{i=1}^{N} \sigma\left(y_i \mathrm{w}^{\mathrm{T}} \mathrm{x}_i\right)\right) = \frac{1}{N} \sum_{i=1}^{N} \ln\left(\frac{1}{\sigma\left(y_i \mathrm{w}^{\mathrm{T}} \mathrm{x}_i\right)}\right)$$

$$= \frac{1}{N} \sum_{i=1}^{N} \ln\left(1 + e^{-y_i \mathrm{w}^{\mathrm{T}} \mathrm{x}_i}\right)$$

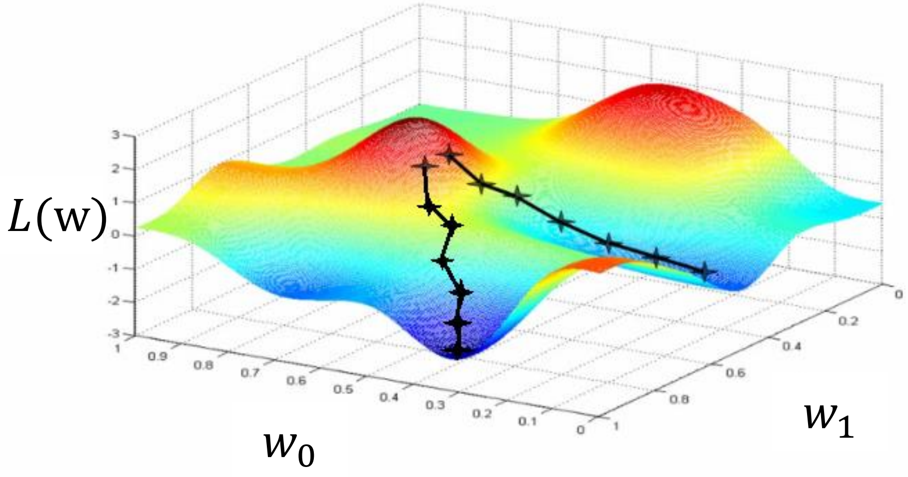
Gradient Descent

$$\mathrm{w}(t+1) = \mathrm{w}(t) - \eta \frac{\partial L(\mathrm{w})}{\partial \mathrm{w}}$$



Gradient Descent

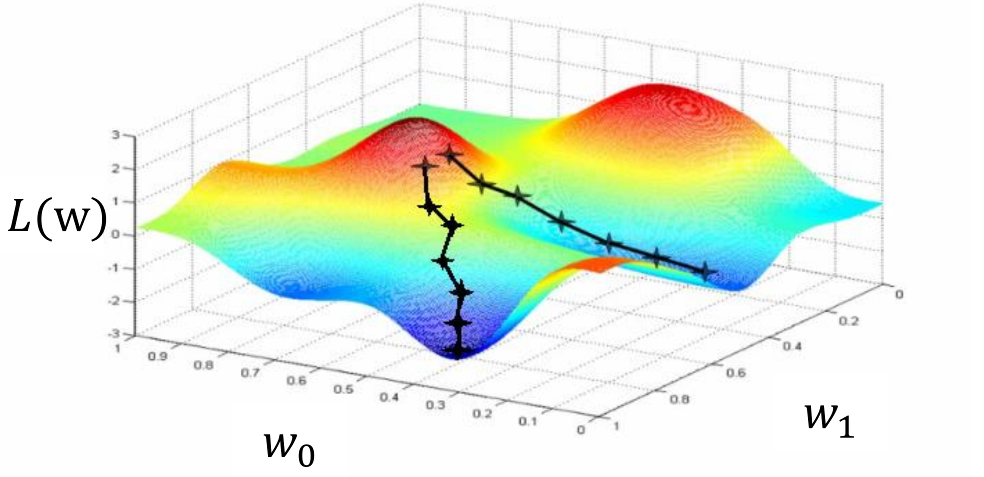
$$\mathrm{w}(t+1) = \mathrm{w}(t) - \eta \frac{\partial L(\mathrm{w})}{\partial \mathrm{w}}$$



Stochastic gradient descent (SGD) calculates *L(w)* on a sample of the data (*batch*).

Gradient Descent

$$\mathrm{w}(t+1) = \mathrm{w}(t) - \eta \frac{\partial L(\mathrm{w})}{\partial \mathrm{w}}$$



Stochastic gradient descent (SGD) calculates *L(*w*)* on a sample of the data (*batch*).

Once we go through all batches we complete an *epoch*.

Gradient descent

$$\mathrm{w} \leftarrow \mathrm{w} - \eta\left(-\frac{1}{N}\sum_{i=1}^{N}\frac{y_i\mathrm{x}_i}{1+e^{y_i\mathrm{w}^{\mathrm{T}}\mathrm{x}_i}}\right)$$

Stochastic gradient descent

$$\mathrm{w} \leftarrow \mathrm{w} - \eta\left(-\frac{1}{N_{batch}}\sum_{x_i\in batch}\frac{y_i\mathrm{x}_i}{1+e^{y_i\mathrm{w}^{\mathrm{T}}\mathrm{x}_i}}\right)$$

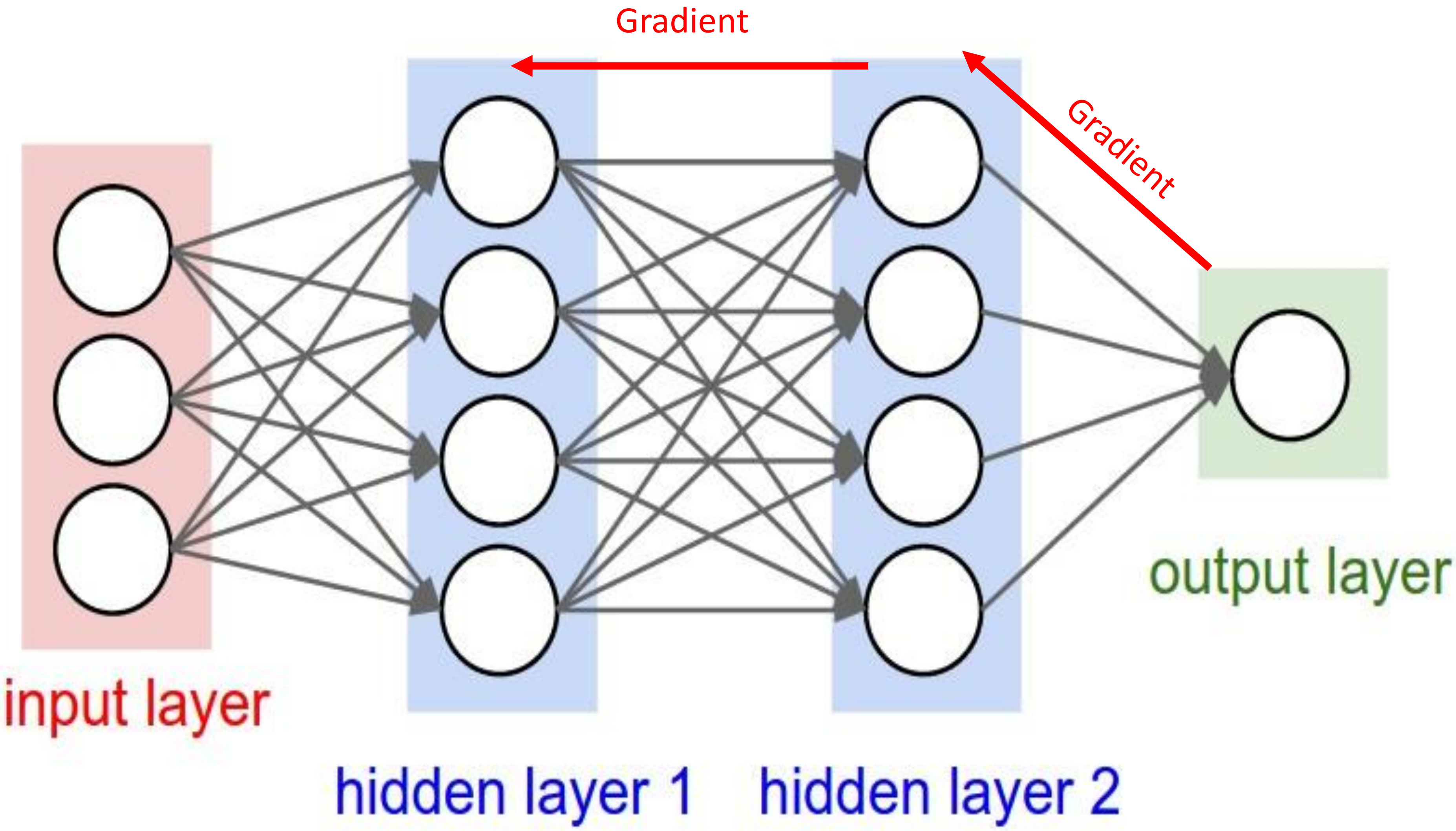
Gradient descent

$$\mathrm{w} \leftarrow \mathrm{w} - \eta \left(-\frac{1}{N} \sum_{i=1}^{N} \frac{y_i \mathrm{x}_i}{1 + e^{y_i \mathrm{w}^{\mathrm{T}} \mathrm{x}_i}} \right)$$

Very stochastic gradient descent

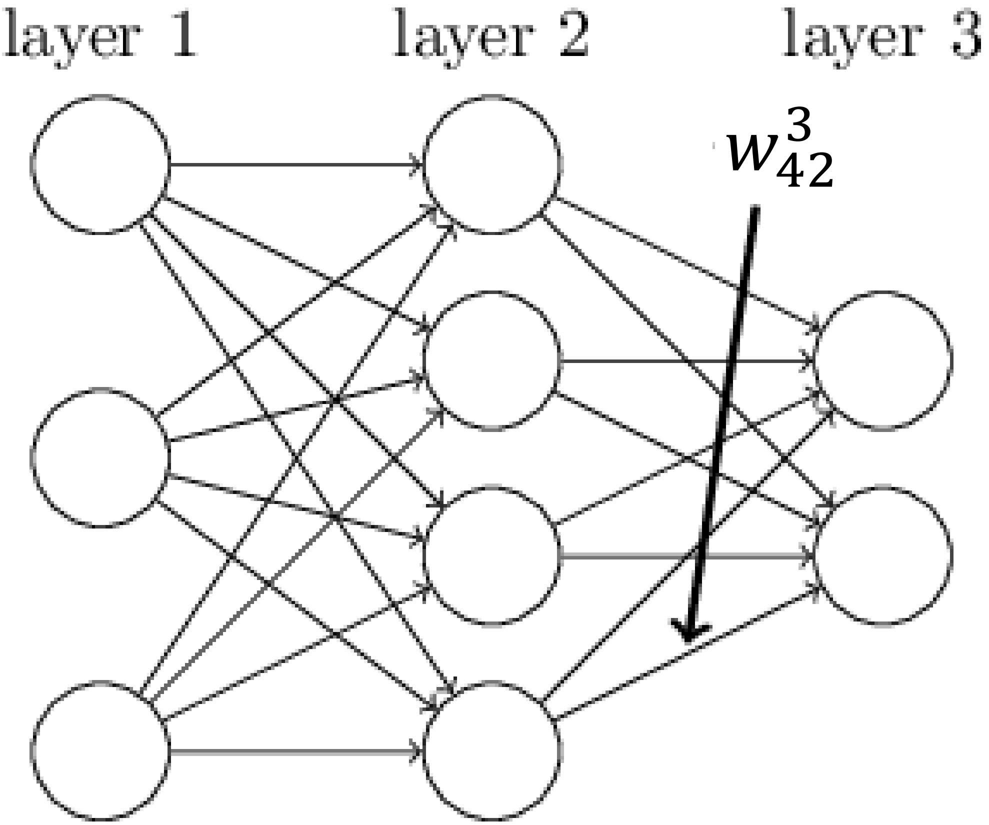
$$\mathrm{w} \leftarrow \mathrm{w} - \eta \frac{\partial L\left(\mathrm{w}^{\mathrm{T}} \mathrm{x}_i, y_i\right)}{\partial \mathrm{w}}$$

Back-propagation



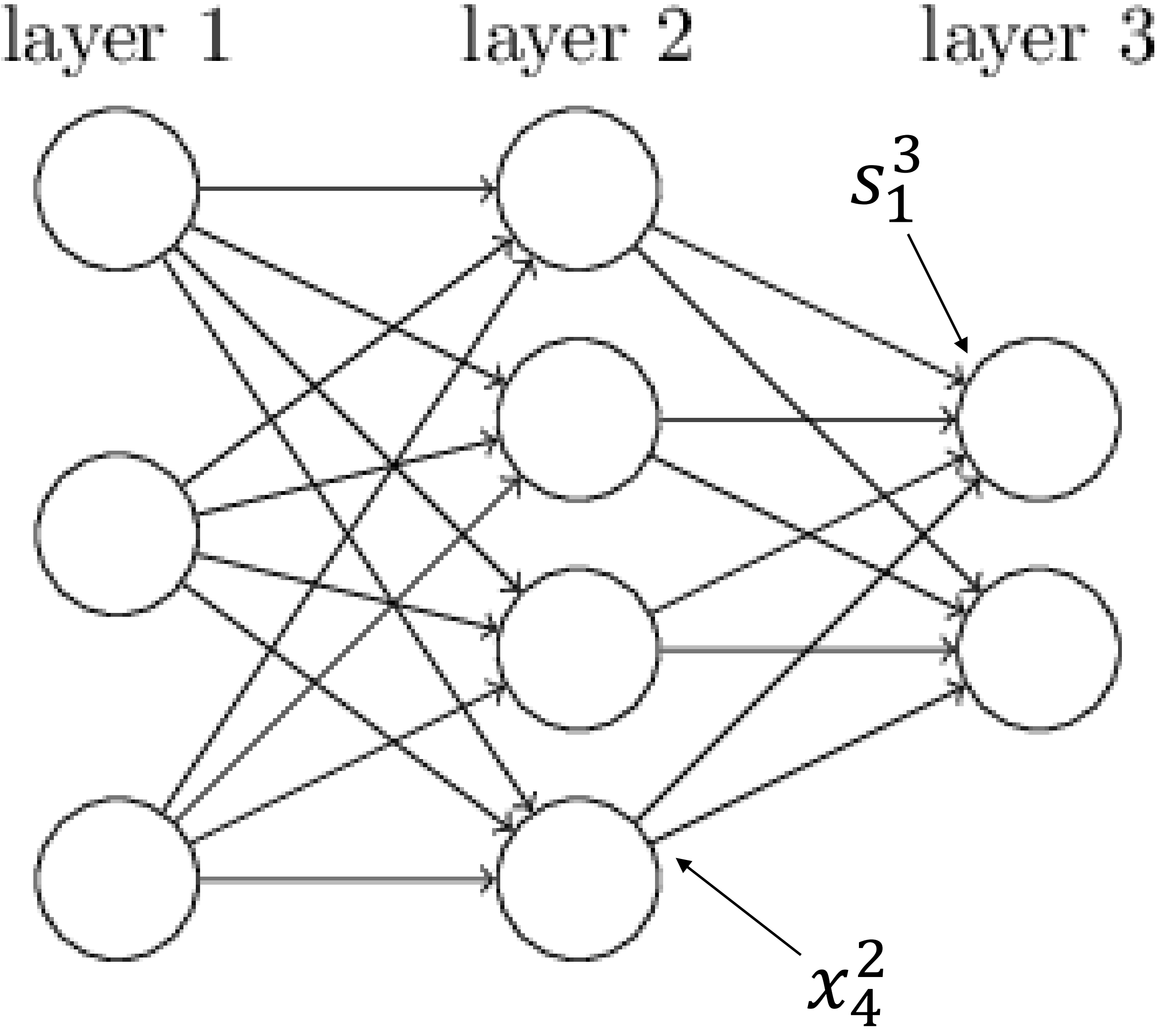
David Rumelhart, Geoffrey Hinton and Rondald Williams, 1986

Back-propagation



w_{jk}^l is the weight between neuron j in layer $l-1$ and neuron k in layer l.

Back-propagation



s_j^l - incoming "signal" to neuron j in layer l.

$$s_j^l = \sum w_{ij}^l x_i^{l-1}$$

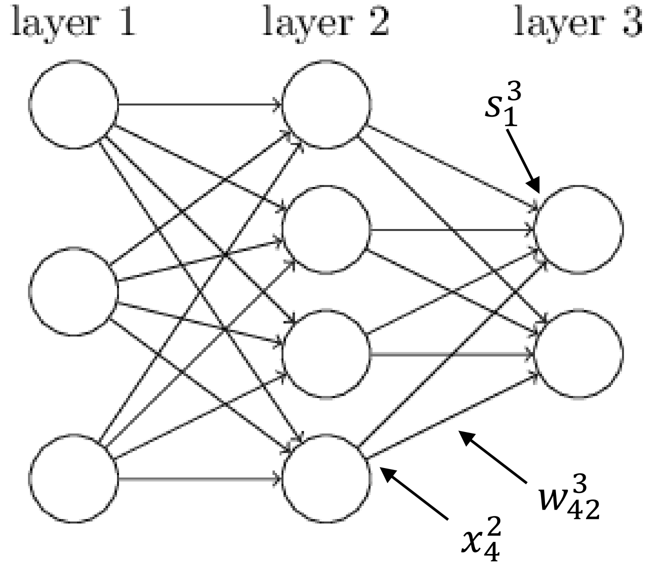
x_j^l - outcoming "signal" after the activation function.

$$x_j^l = \sigma(s_j^l) = \sigma\left(\sum w_{ij}^l x_i^{l-1}\right)$$

$$x_j^1 = x_j$$

Back-propagation

$$\nabla L_{w_{ij}^l}(\mathrm{w}): \frac{\partial L(\mathrm{w})}{\partial w_{ij}^l} = \frac{\partial L(\mathrm{w})}{\partial s_j^l} \times \frac{\partial s_j^l}{\partial w_{ij}^l}$$

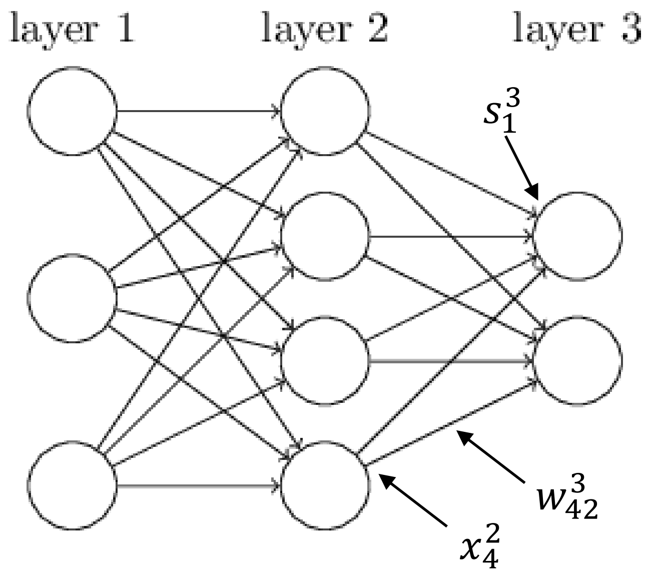


$$\frac{\partial s_j^l}{\partial w_{ij}^l} = x_i^{l-1} \qquad \frac{\partial L(\mathrm{w})}{\partial s_j^l} = \delta_j^l$$

Back-propagation

Last layer:

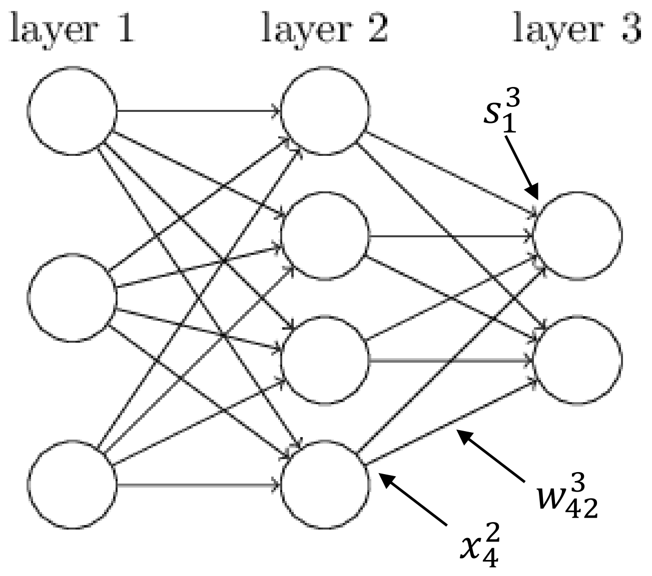
$$\delta_i^L = \frac{\partial L(\mathrm{w})}{\partial s_i^L}, \qquad L(\mathrm{w}) = f(\mathrm{x}^L), \qquad \mathrm{x}_i^L = \sigma(s_i^L)$$



Back-propagation

Last layer:

$$\delta_i^L = \frac{\partial L(\mathrm{w})}{\partial s_i^L}, \qquad L(\mathrm{w}) = f(\mathrm{x}^L), \qquad \mathrm{x}_i^L = \sigma(s_i^L)$$



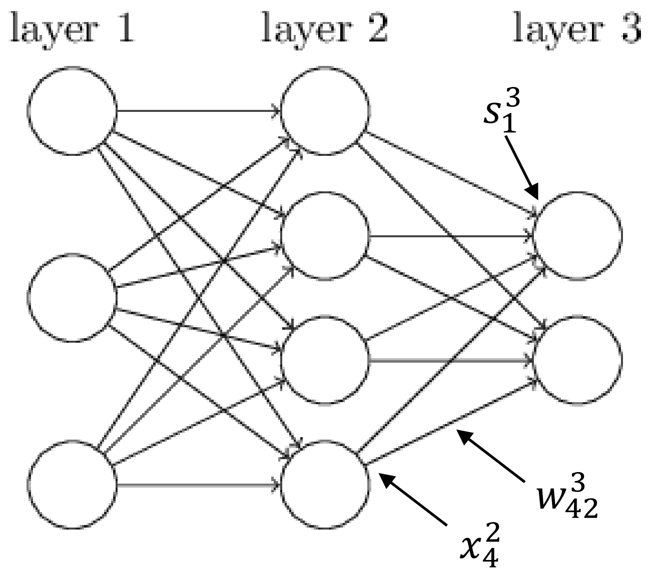
Previous layers:

$$\delta_i^{l-1} = \frac{\partial L(\mathrm{w})}{\partial s_i^{l-1}}$$

Back-propagation

Last layer:

$$\delta_i^L = \frac{\partial L(\mathrm{w})}{\partial s_i^L}, \qquad L(\mathrm{w}) = f(\mathrm{x}^L), \qquad \mathrm{x}_i^L = \sigma(s_i^L)$$



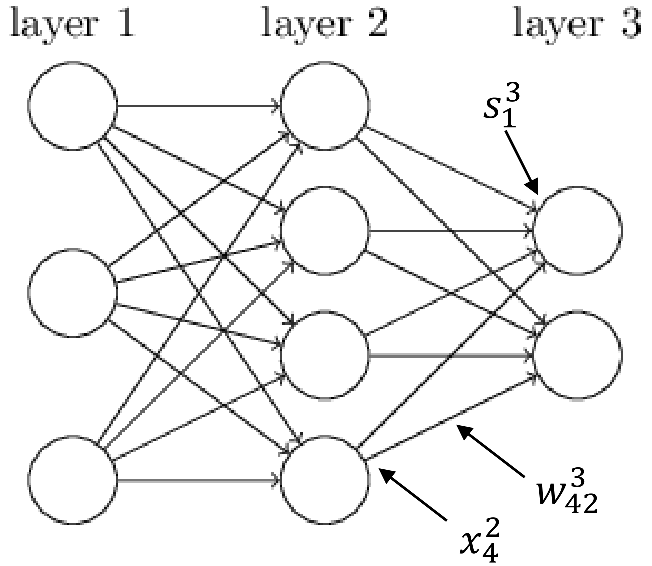
Previous layers:

$$\delta_i^{l-1} = \frac{\partial L(\mathrm{w})}{\partial s_i^{l-1}} = \sum_j \frac{\partial L(\mathrm{w})}{\partial s_j^l} \times \frac{\partial s_j^l}{\partial x_i^{l-1}} \times \frac{\partial x_i^{l-1}}{\partial s_i^{l-1}}$$

Back-propagation

Last layer:

$$\delta_i^L = \frac{\partial L(\mathrm{w})}{\partial s_i^L}, \qquad L(\mathrm{w}) = f(\mathrm{x}^L), \qquad \mathrm{x}_i^L = \sigma(s_i^L)$$



Previous layers:

$$\delta_i^{l-1} = \frac{\partial L(\mathrm{w})}{\partial s_i^{l-1}} = \sum_j \frac{\partial L(\mathrm{w})}{\partial s_j^l} \times \frac{\partial s_j^l}{\partial x_i^{l-1}} \times \frac{\partial x_i^{l-1}}{\partial s_i^{l-1}} = \sum_j \delta_j^l \times w_{ij}^l \times \sigma'\left(s_i^{l-1}\right)$$

Back-propagation

1. Initialize weights randomly*

2. Forward: calculate x and s

3. Backward: calculate δ

4. $w_{ij}^{l} \leftarrow w_{ij}^{l} - \eta x_{i}^{l-1} \delta_{j}^{l}$

-> 2.

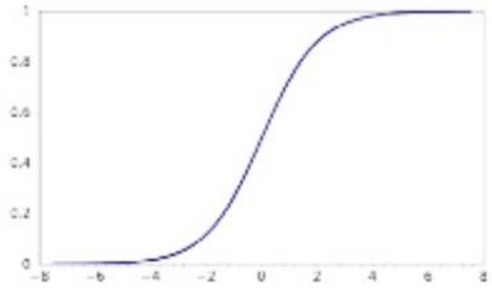
Activation functions

Sigmoid: $\sigma(s) = \frac{1}{1+e^{-s}}$

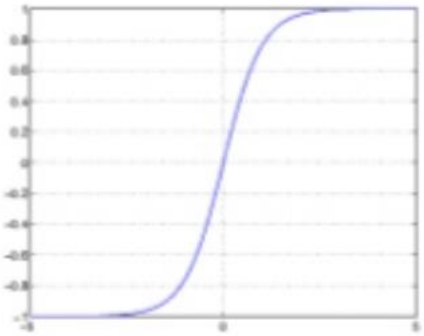
Tanh: $\sigma(s) = \frac{e^{s}-e^{-s}}{e^{s}+e^{-s}}$

Rectified Linear Unit (ReLU):

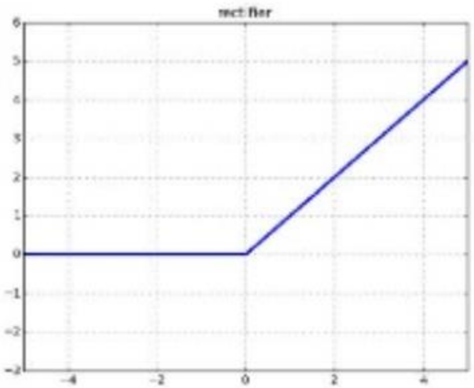
$$\sigma(s) = \max(0, s)$$



Sigmoid

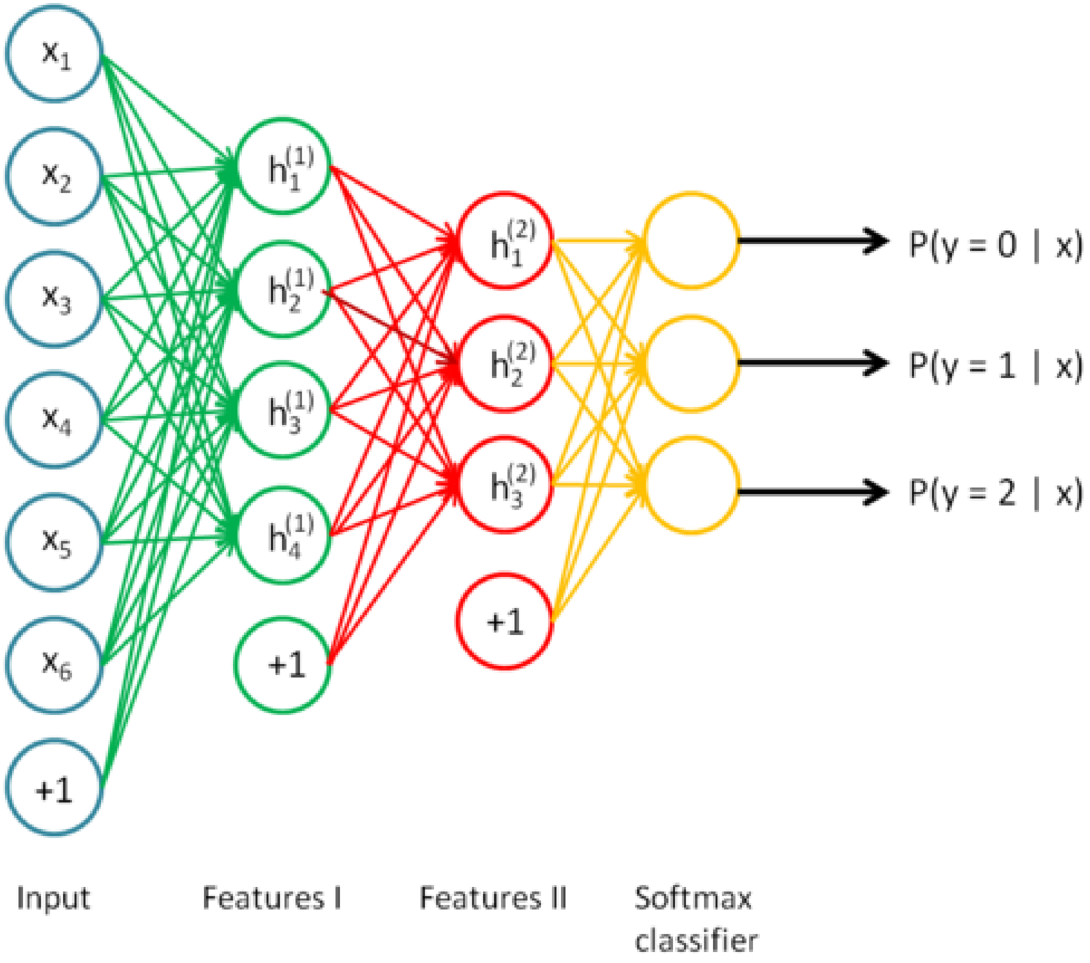


Tanh



ReLU

Softmax and Cross-Entropy



$$x_j^L = \sigma^L(s_j^L) = \frac{e^{s_j^L}}{\sum e^{s_i^L}}$$

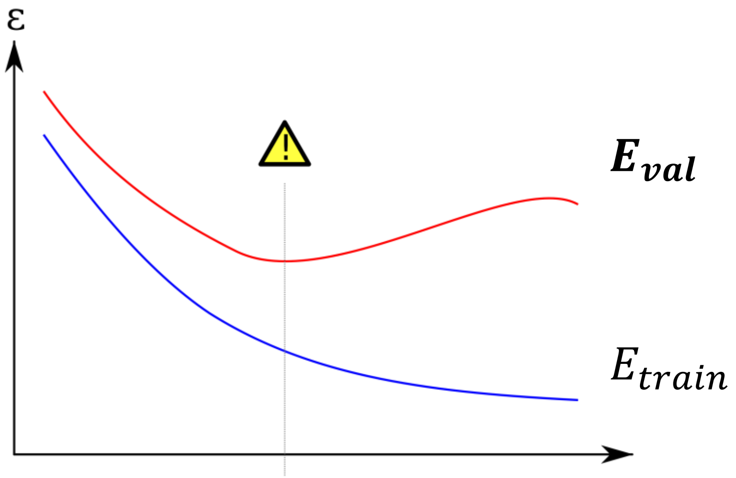
$$L(\mathrm{w}) = -\sum o_i \log(x_i^L)$$

$$o_i = \begin{cases} 1, if\ y = i \\ 0, if\ y \neq i \end{cases}$$

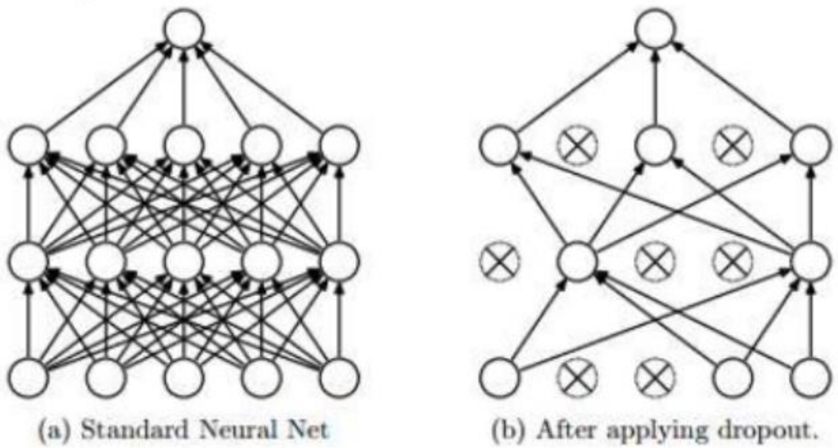
Regularization

L2 regularization — $L'(w) = L(w) + \frac{\lambda}{2N}||\mathrm{w}||_2^2$

Early Stopping:



Dropout:



Data augmentation

Deep Learning Libraries

TensorFlow (www.tensorflow.org) + Keras (keras.io)

Torch (torch.ch) - PyTorch

Deeplearning4j (deeplearning4j.org)