Introduction to

Neural Networks

Saeed Reza Kheradpisheh

Dept. Computer Science
Shahid Behesti University

Typical ML

Feature Extractions

Classification

dog

Data independent

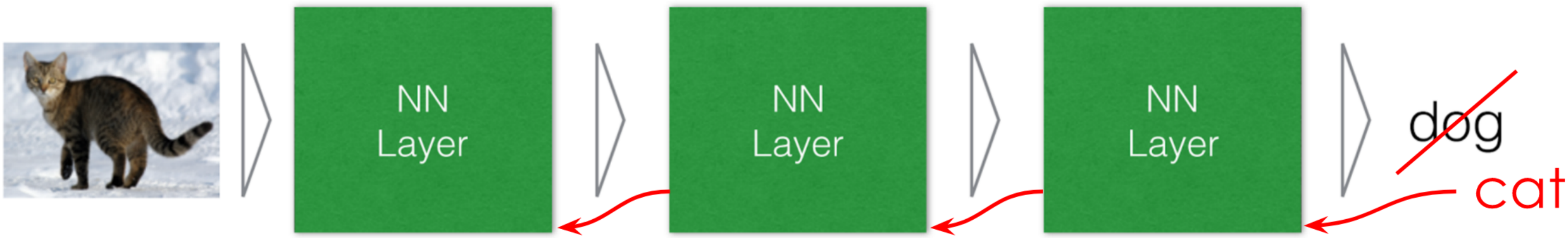
Supervised Learning

Typical ML

Data independent

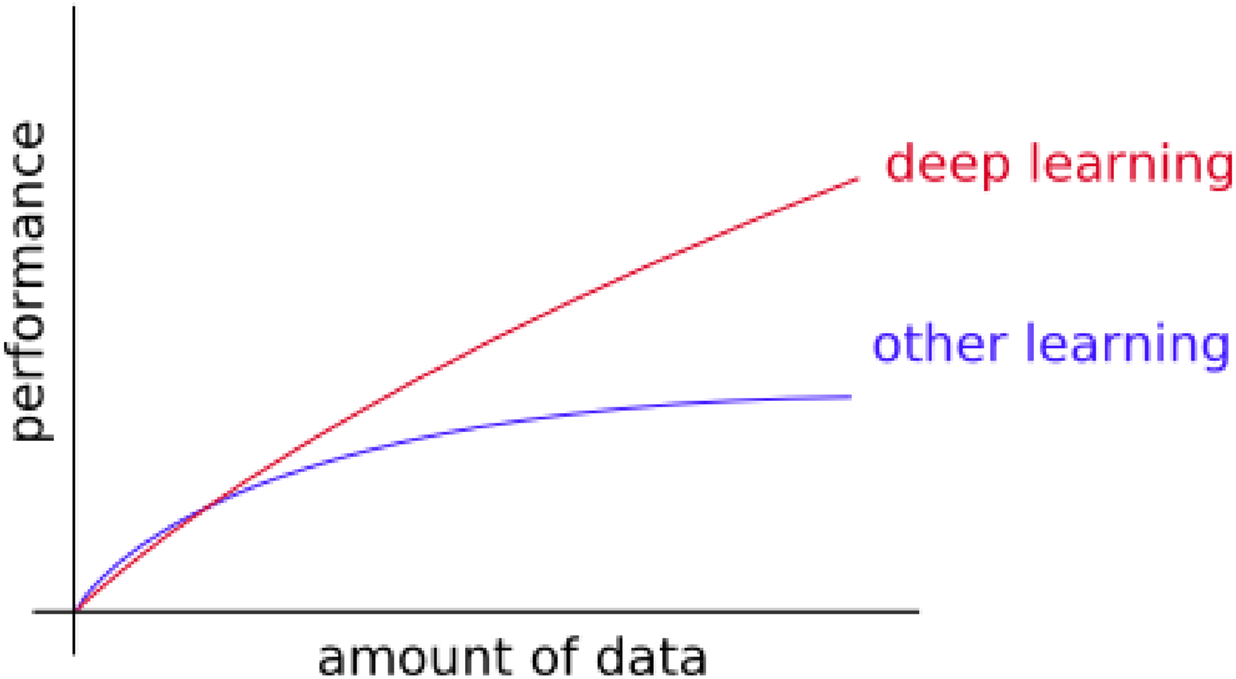
Supervised Learning

Deep Learning

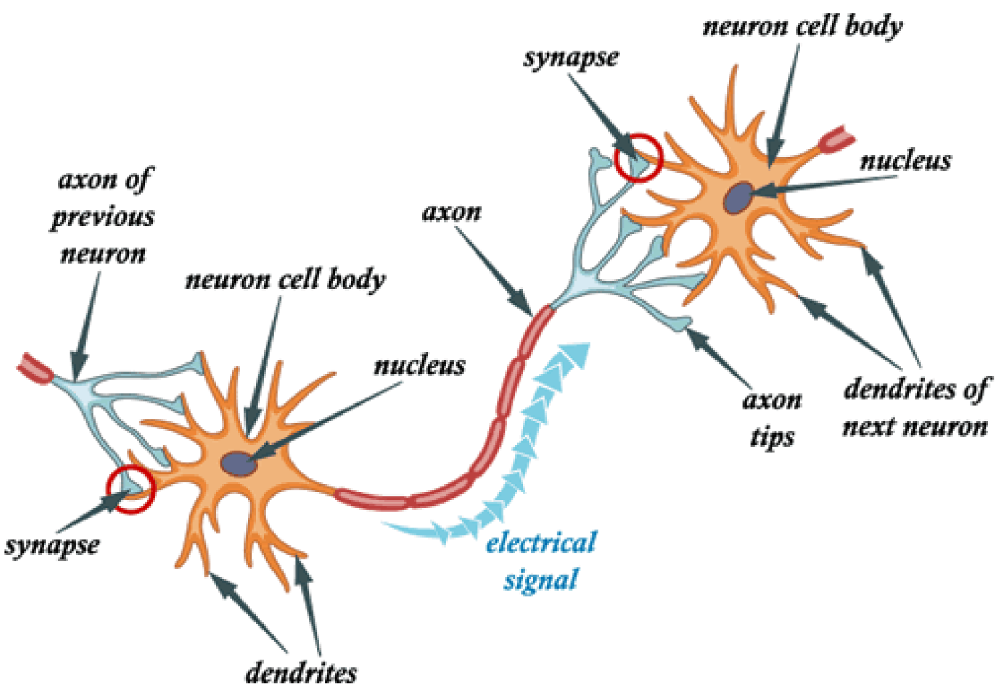


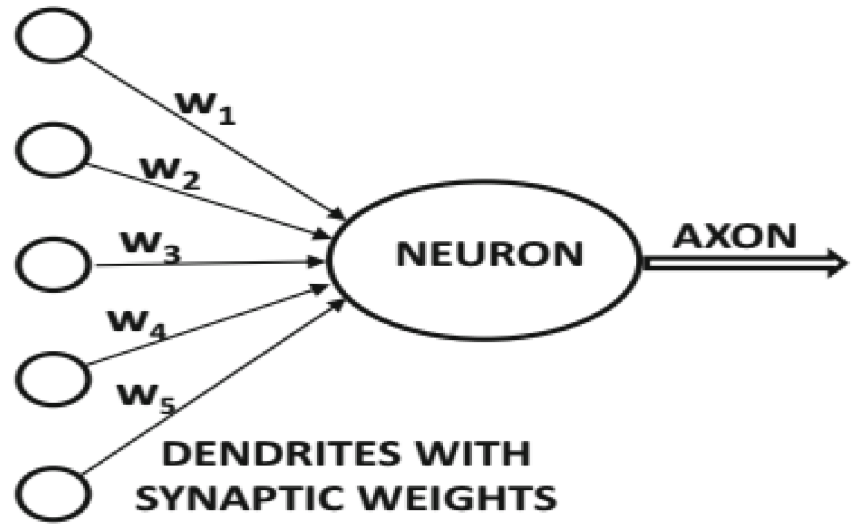
Supervised
Learning

Why DL

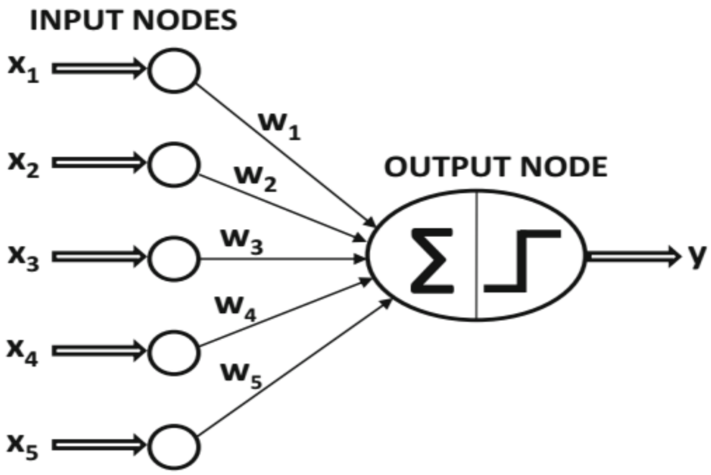


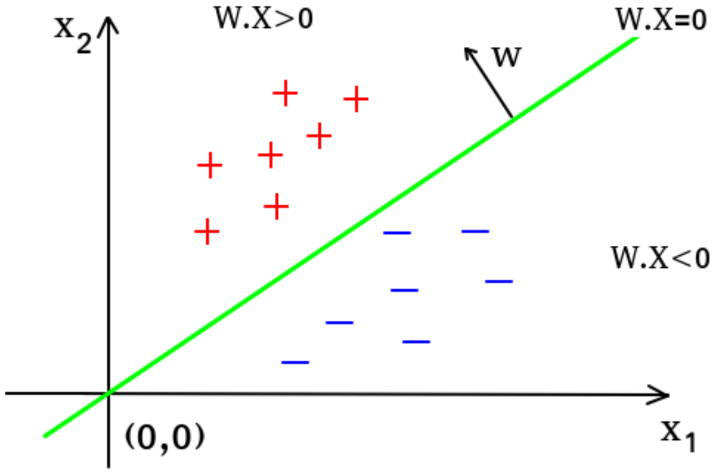
Biological v.s. Arificial Neuron





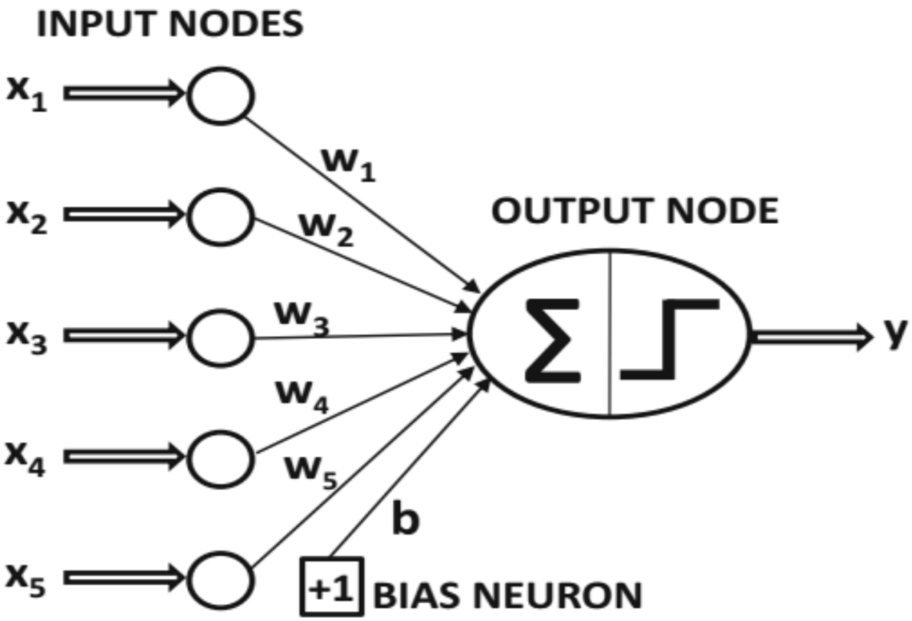
Perceptron

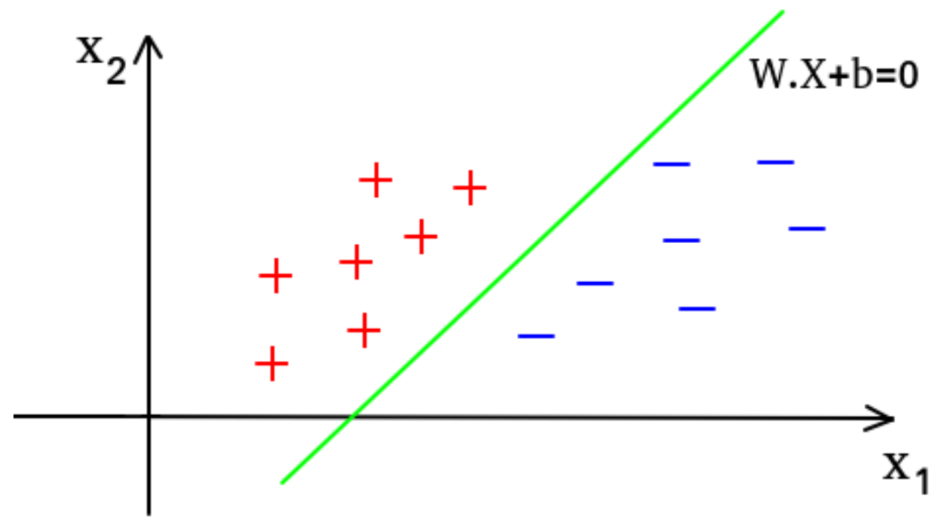




- Input vector: $X = [x_1, \ldots, x_d]$
- Target output: $Y \in \{-1, +1\}$
- Input weights: $W = [w_1, \ldots, w_d]$
- Predicted output: $y = sign\{W.X\} = sign\{\sum_{i=1}^{d} w_i x_i\}$

Perceptron with bias





Loss Function

Consider a d-dimensional binary classification problem:

- Training set: $D = \{(X_i, Y_i) | i = 1 : N\}$
- Training sample: $X_i = [X_{i1}, ..., X_{id}]$, $Y_i \in \{-1, +1\}$
- Perceptron predicts: $y_i = sign\{W.X_i\}$
- *Loss Function*:

$$L = \sum_{(X_i, Y_i) \in D} (Y_i - y_i)^2 = \sum_{(X_i, Y_i) \in D} (Y_i - sign\{W.X_i\})^2$$

Learning in Perceptron

- *Loss Function*:

$$L = \sum_{(X_i,Y_i)\in D} (Y_i - y_i)^2 = \sum_{(X_i,Y_i)\in D} (Y_i - sign\{W.X_i\})^2$$

- Loss function depends on W and D.
- As D is given, hence, learning is to find W^* minimizing the loss function:

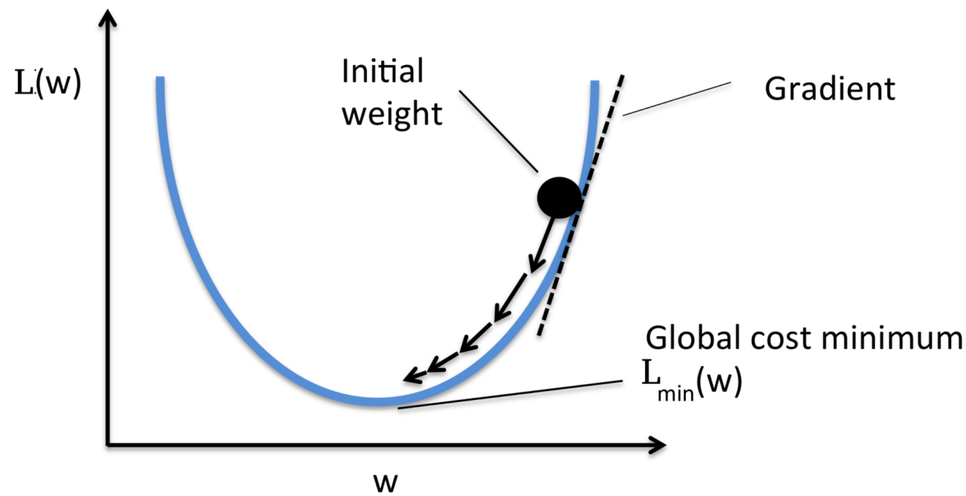
$$W^* = \underset{W}{\operatorname{argmin}} \sum_{(X_i,Y_i)\in D} (Y_i - sign\{W.X_i\})^2$$

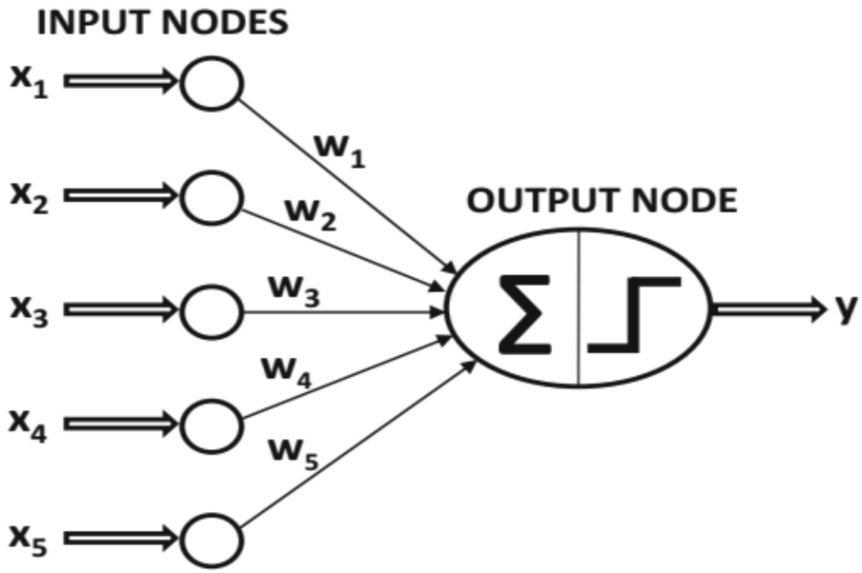
How to find the optimum weight

- For an arbitrary weight vector, $-\nabla_w L$ shows the direction of the steepest descent of the loss function.

$$\nabla_w L = [\frac{\partial L}{\partial w_1} ..., \frac{\partial f}{\partial w_d}]$$

- Find W^* by starting from a random weight vector and an iterative use of gradient:





$$\frac{\partial L}{\partial w_j} = -\sum_{(X_i,Y_i)\in D} (Y_i - y_i)x_{ij}$$

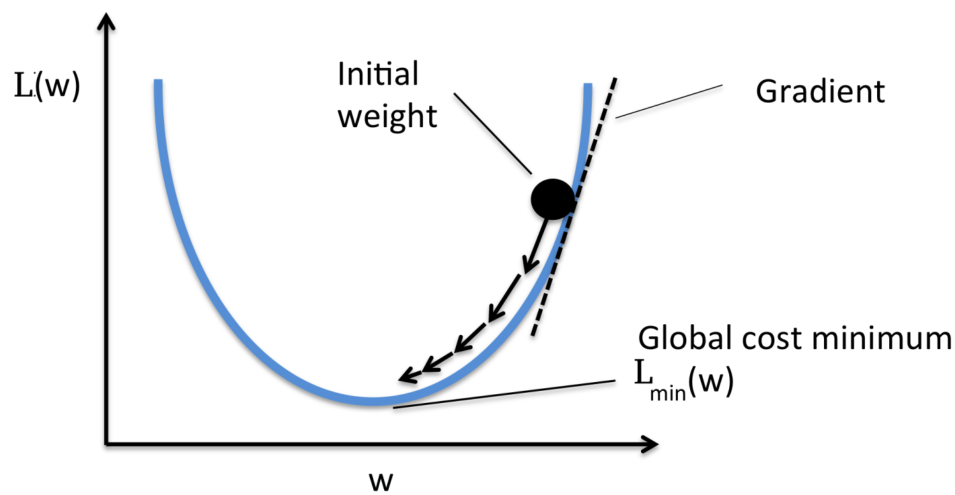
Gradient Descent

To find optimum weights (W^*):

- Start from a random initial weight vector, W^0.
- Through an iterative manner, use gradients and update the weights:

$$W^{t+1} = W^t + \eta \sum_{(X_i,Y_i)\in D} (Y_i - y_i)X_i$$

- The laerning rate is controled by η.

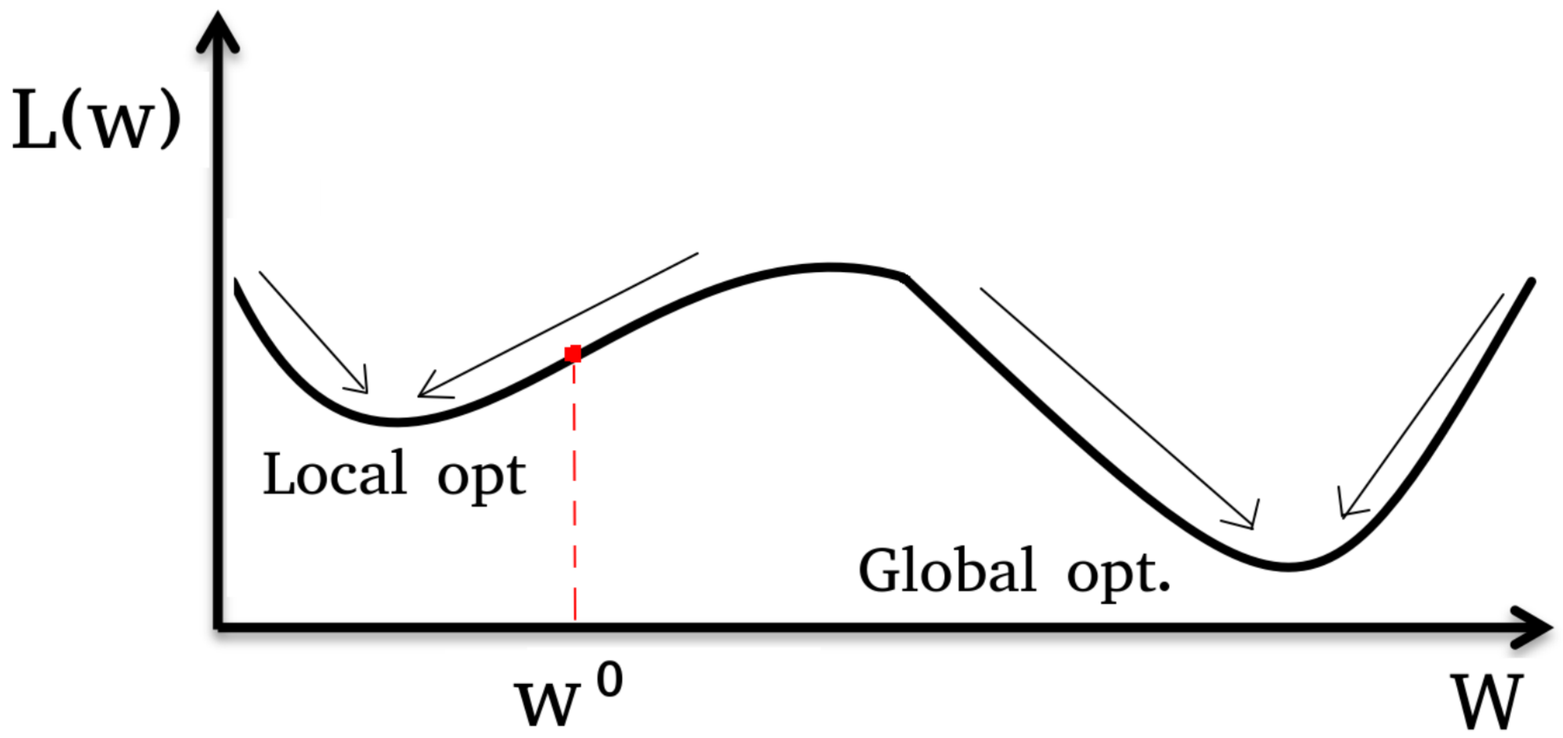


Stochastic Gradient Descent SGD

In SGD, learning is performed sample by sample:

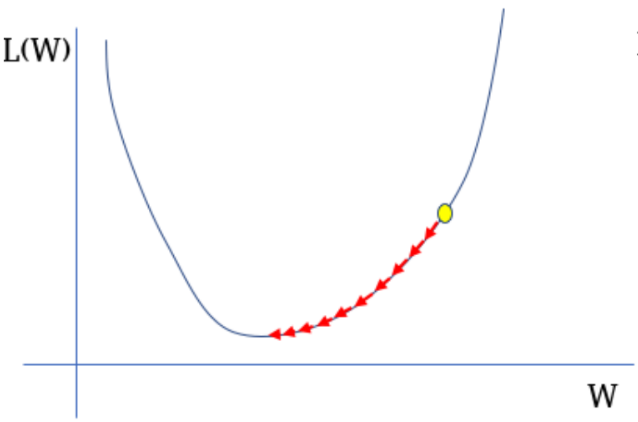
1. Shuffle the training set.
2. Compute the perceptron's outputy_i for the i-th sample.
3. Update the weights using $W^{t+1} = W^t + \eta(Y_i - y_i)X_i$.
4. Repeat steps 2 to 3 for all training samples.
5. Jump to step 1 if the totall loss $L = \sum_{(X_i,Y_i)\in D}(Y_i - y_i)^2$ is below a certain value or the maximum number of iteration is reached

Initial weights matter



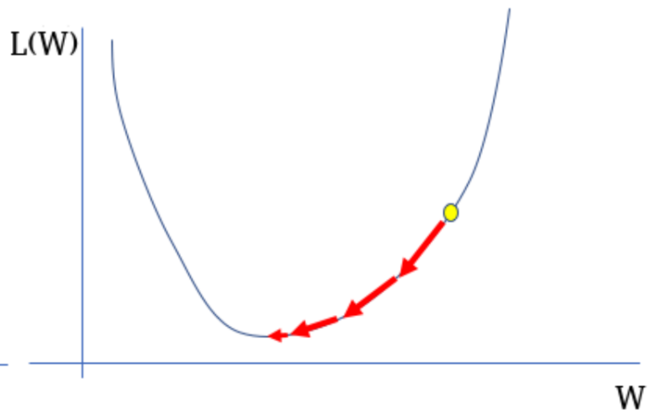
Learning rate matters

Too low



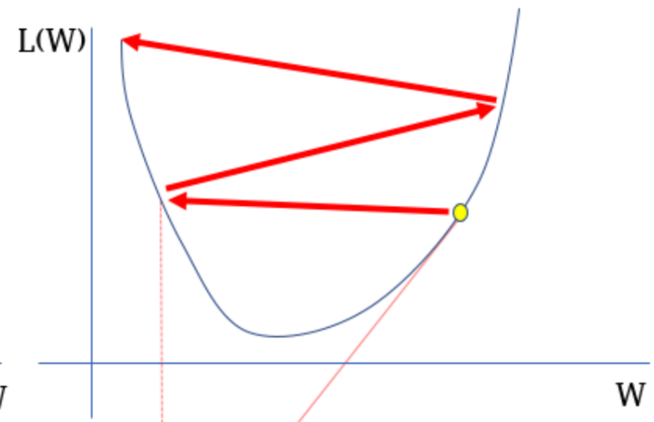
A small learning rate requires many updates before reaching the minimum point

Just right



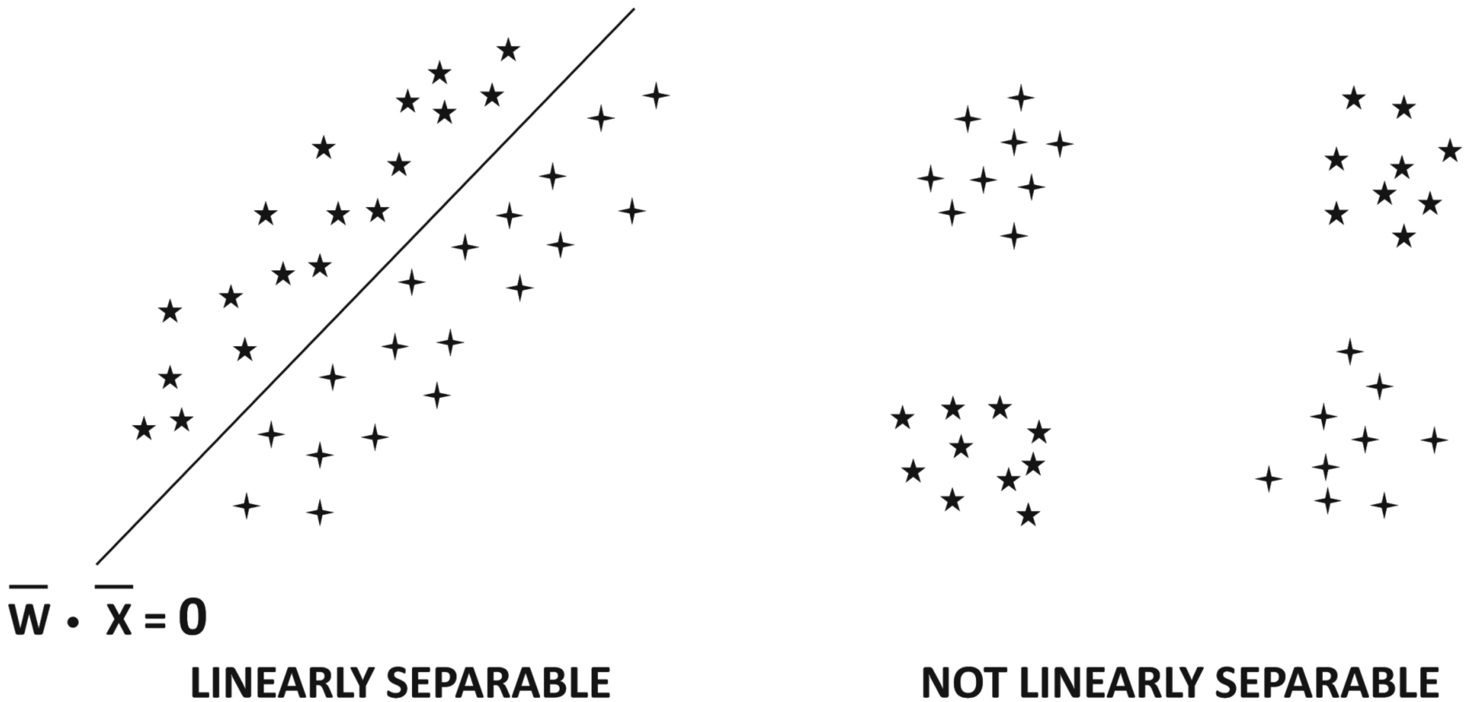
learning rate swiftly reaches the minimum point

Too high



Too large of a learning rate causes drastic updates which lead to divergent behaviors

Perceptorn is a linear model

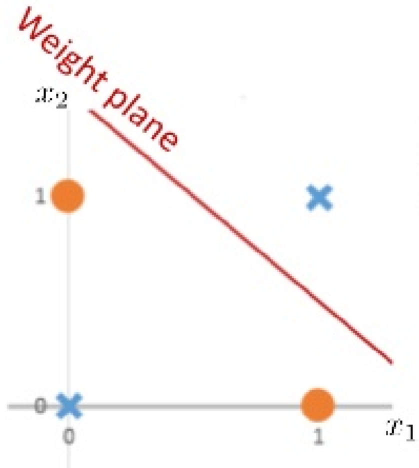


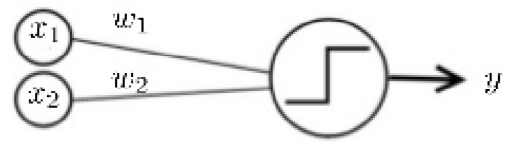
The XOR problem

XOR Problem

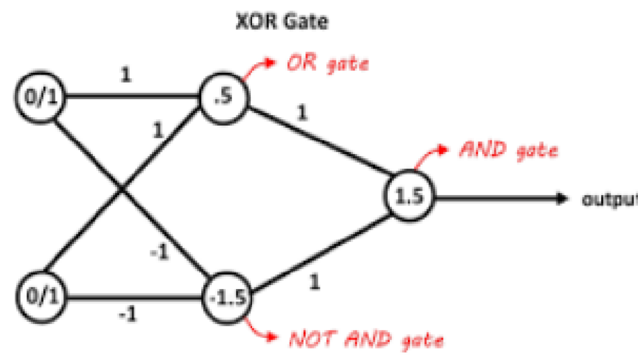
Training Data

x_1	x_2	t
0	0	0
1	0	1
0	1	1
1	1	0



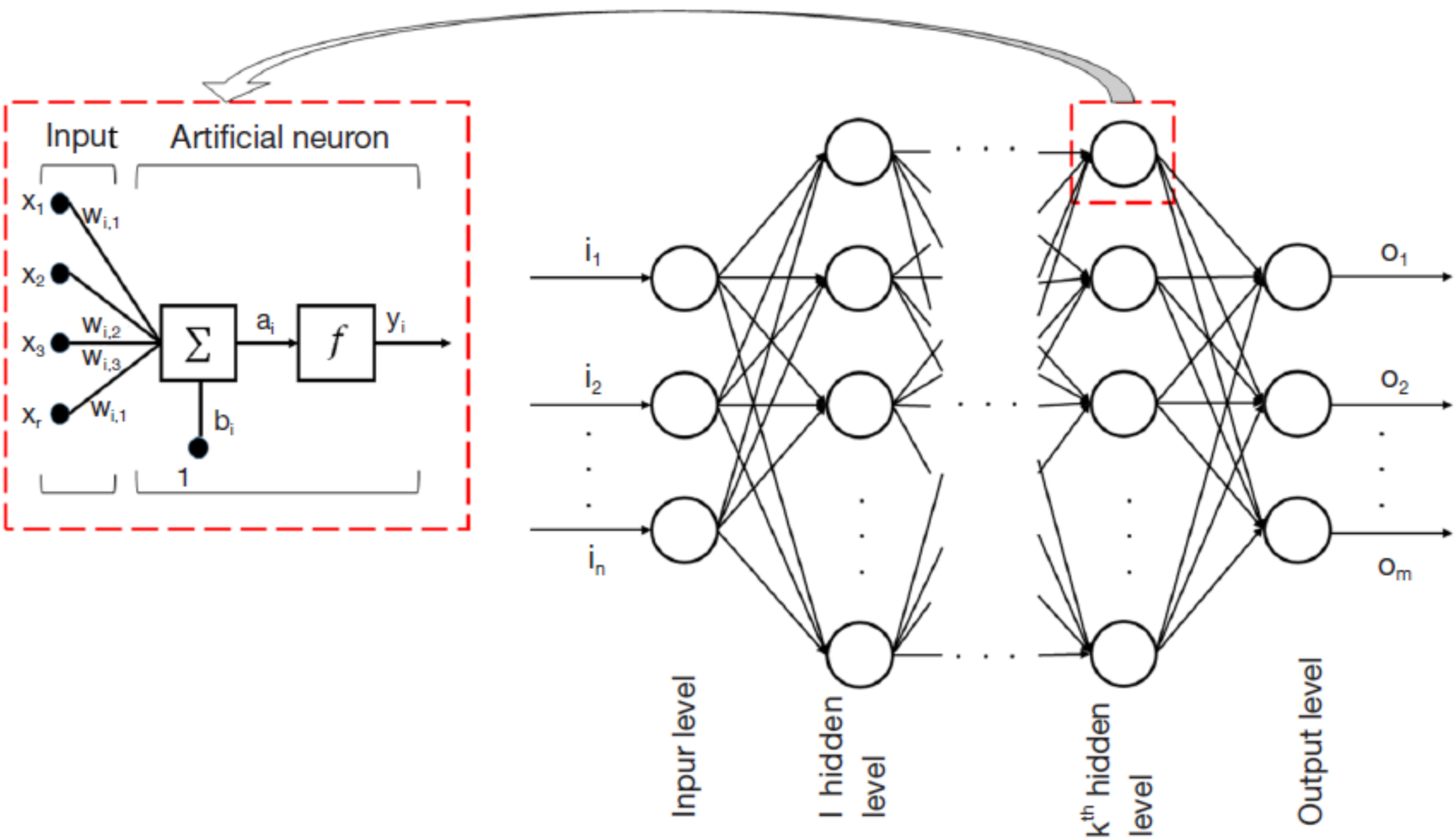


A single perceptron can only solve linear problems.

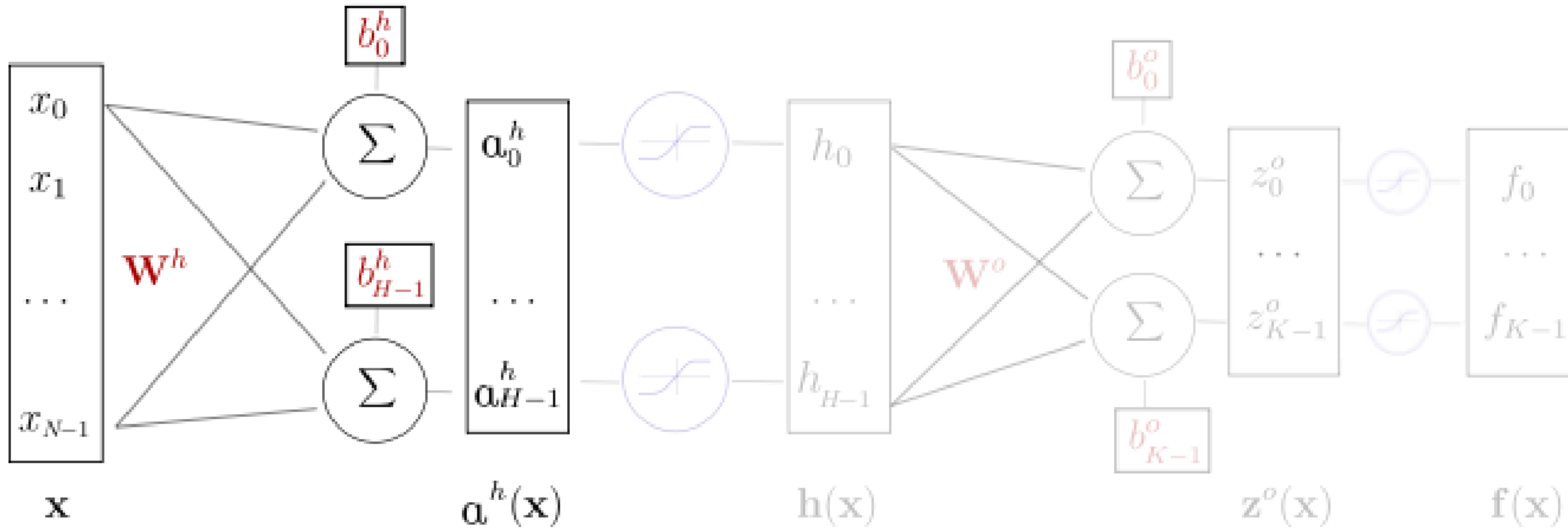


Multi-layer perceptron can solve non-linearly separable problems.

Multi-layer Perceptron (MLP)

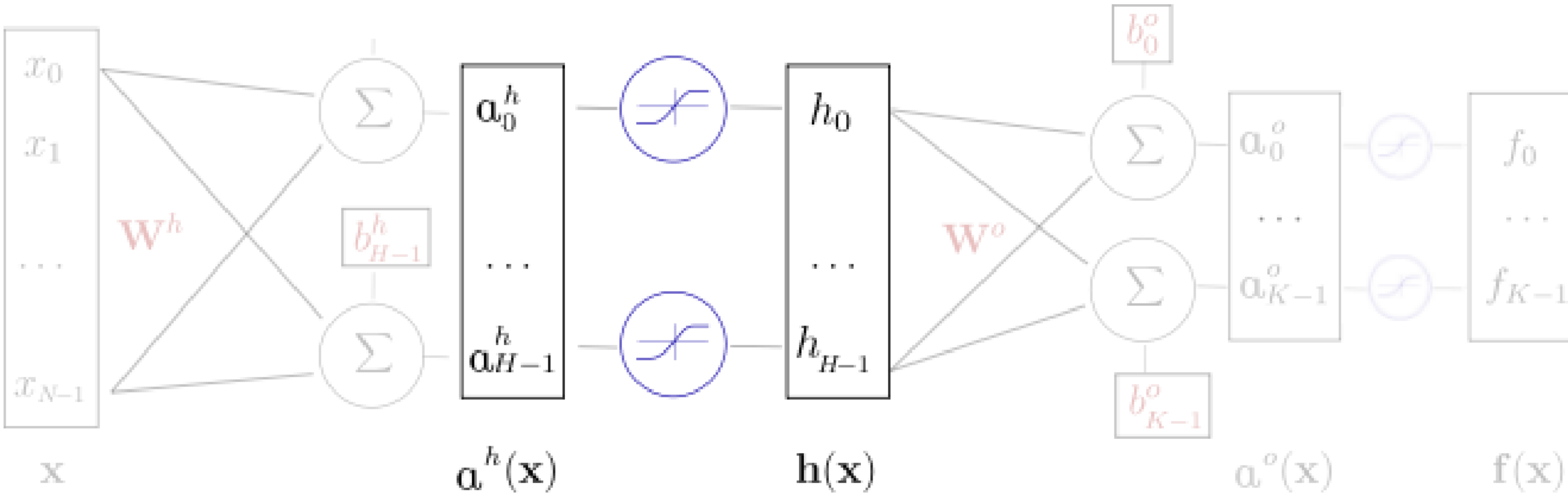


Forward Propagation



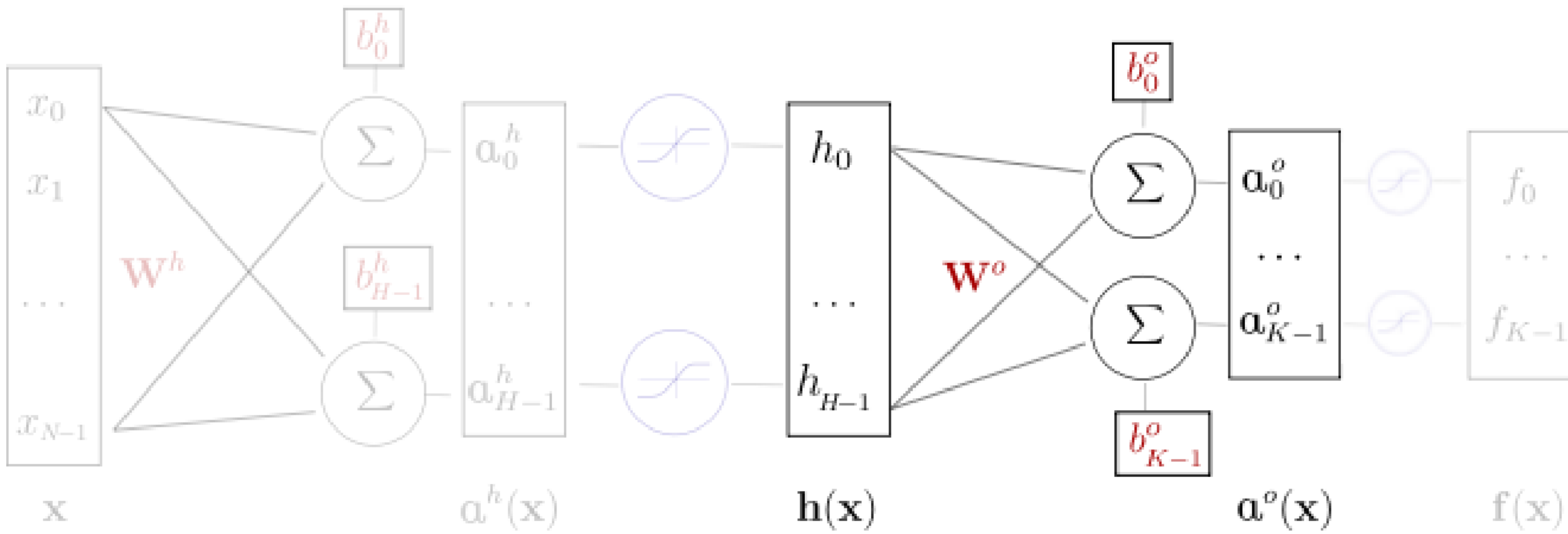
- $a^h(x) = W^h.x + b^h$

Forward Propagation



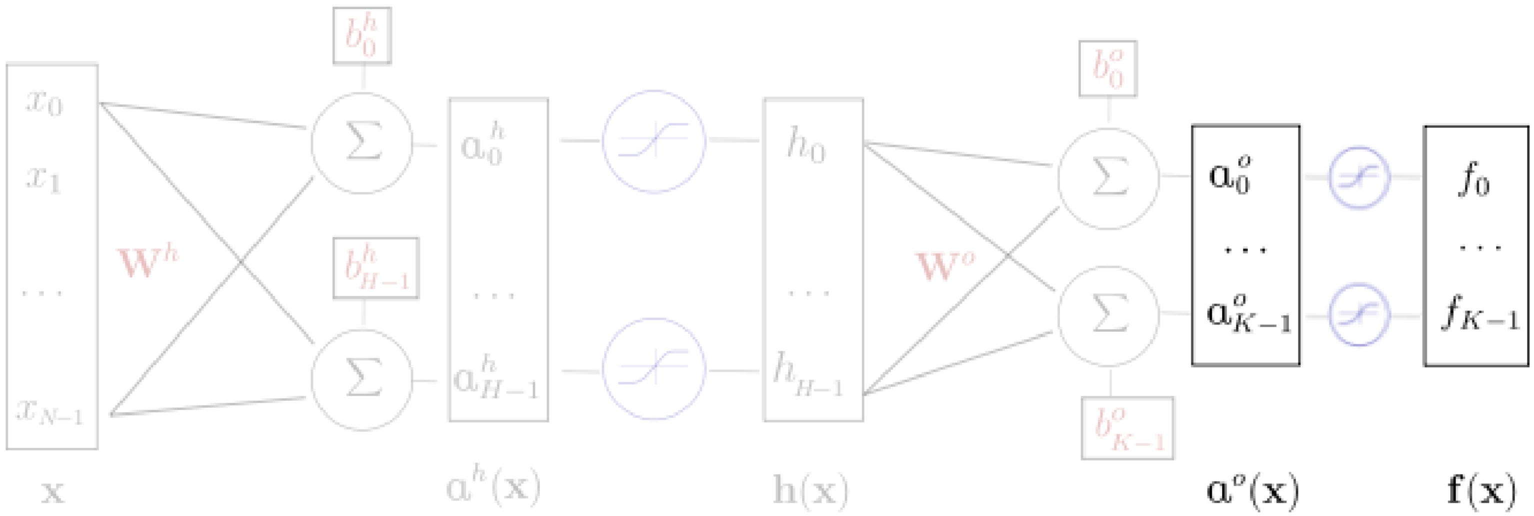
- $a^h(x) = W^h.x + b^h$
- $h(x) = \Phi(a^h(x)) = \Phi(W^h.x + b^h)$

Forward Propagation



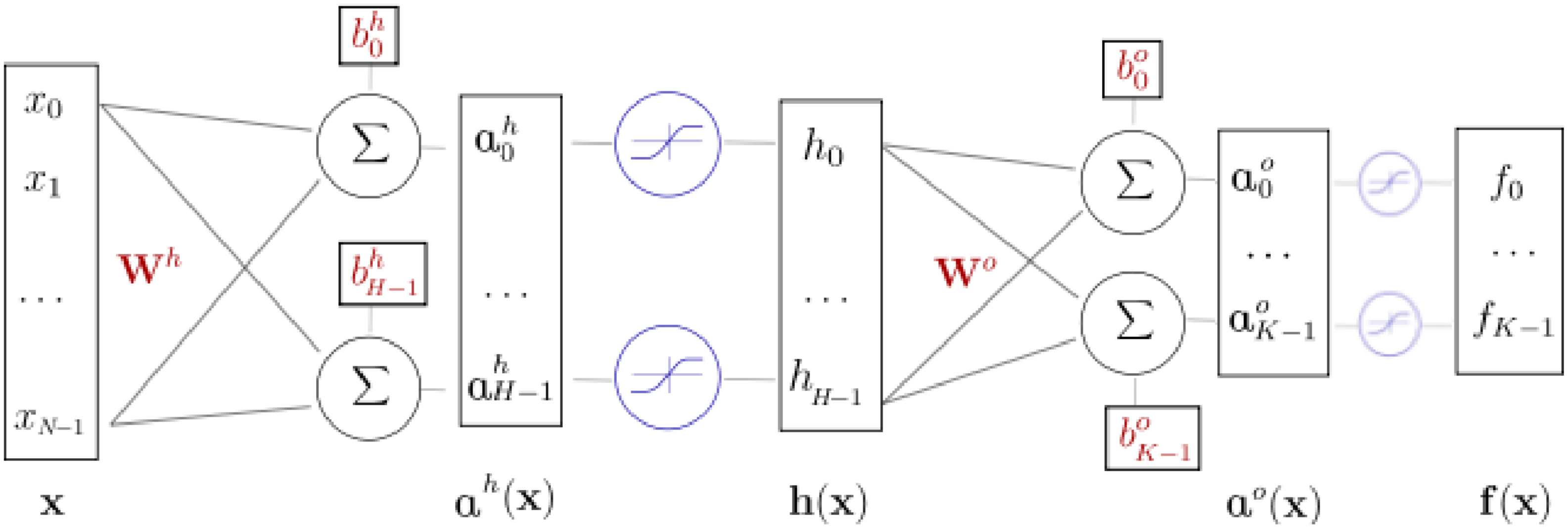
- $a^h(x) = W^h.x + b^h$
- $h(x) = \Phi(a^h(x)) = \Phi(W^h.x + b^h)$
- $a^o(x) = W^o.h(x) + b^o$

Forward Propagation



- $a^h(x) = W^h.x + b^h$
- $h(x) = \Phi(a^h(x)) = \Phi(W^h.x + b^h)$
- $a^o(x) = W^o.h(x) + b^o$
- $f(x) = \Phi(a^o(x)) = \Phi(W^o.h(x) + b^o)$

Forward Propagation



- $a^h(x) = W^h.x + b^h$
- $h(x) = \Phi(a^h(x)) = \Phi(W^h.x + b^h)$
- $a^o(x) = W^o.h(x) + b^o$
- $f(x) = \Phi(a^o(x)) = \Phi(W^o.h(x) + b^o)$

Activation Functions

Sign function: $\Phi(a) = sign(a)$

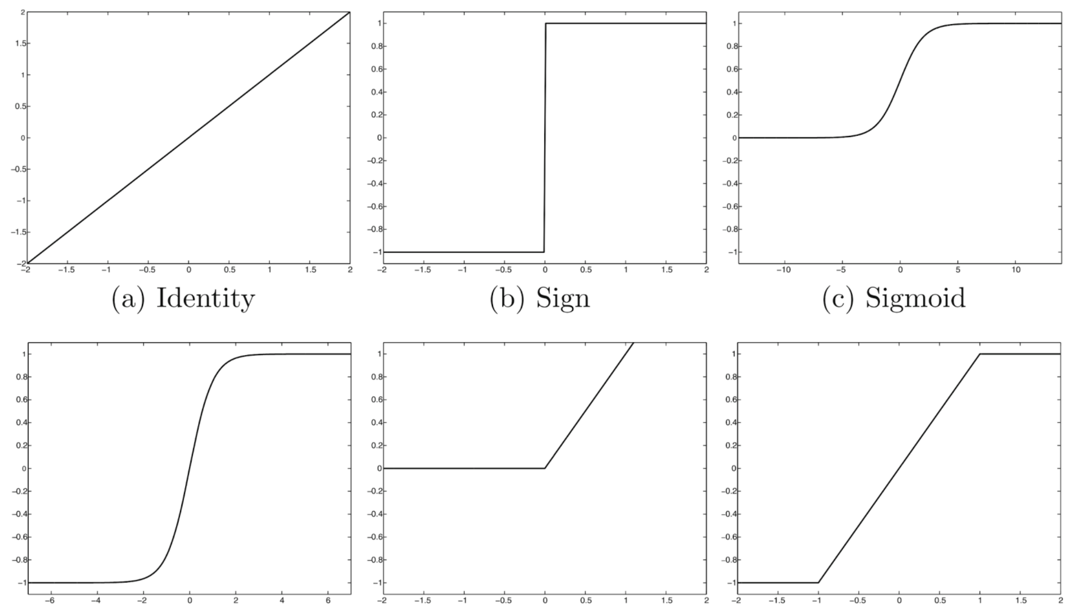
Sigmoid function: $\Phi(a) = \frac{1}{1+e^{-a}}$

Tangh function: $\Phi(a) = \frac{e^{2a}-1}{e^{2a}+1}$

ReLU: $\Phi(a) = max\{a, 0\}$

Hard Tangh:

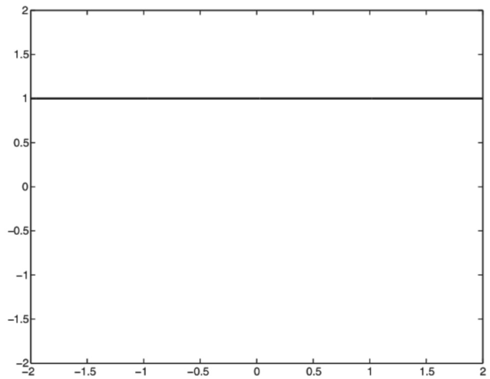
$\Phi(a) = max\{min[v, 1], -1\}$



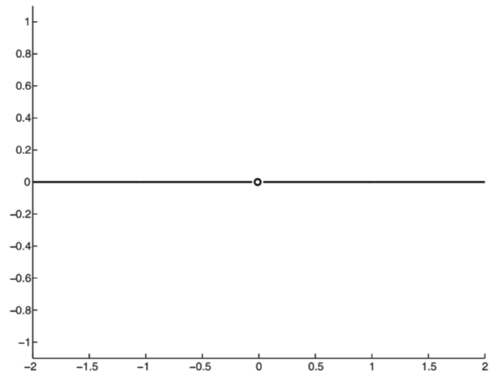
(a) Identity (b) Sign (c) Sigmoid

(d) Tanh (e) ReLU (f) Hard Tanh

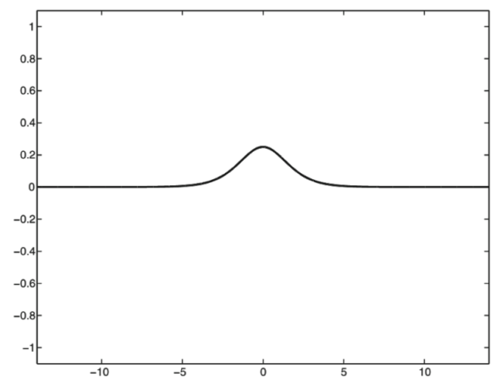
Derivations of Activations



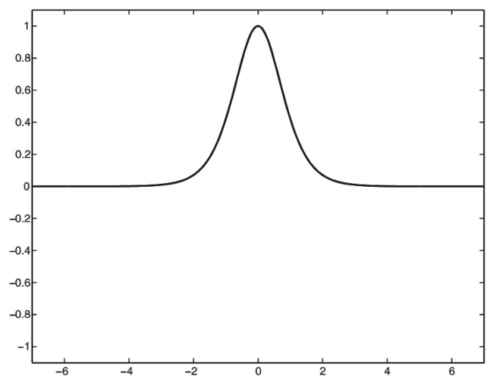
(a) Identity



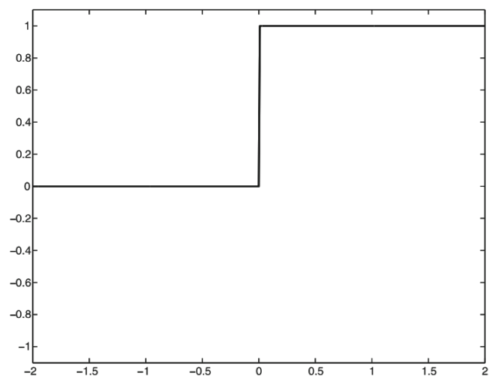
(b) Sign



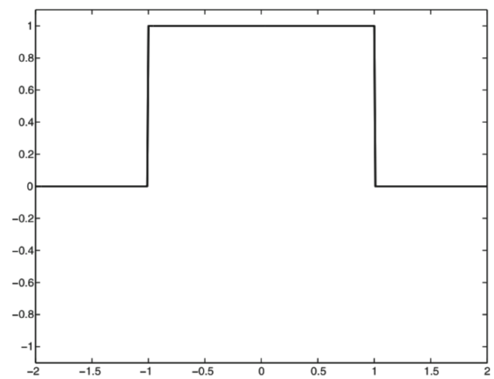
(c) Sigmoid



(d) Tanh



(e) ReLU

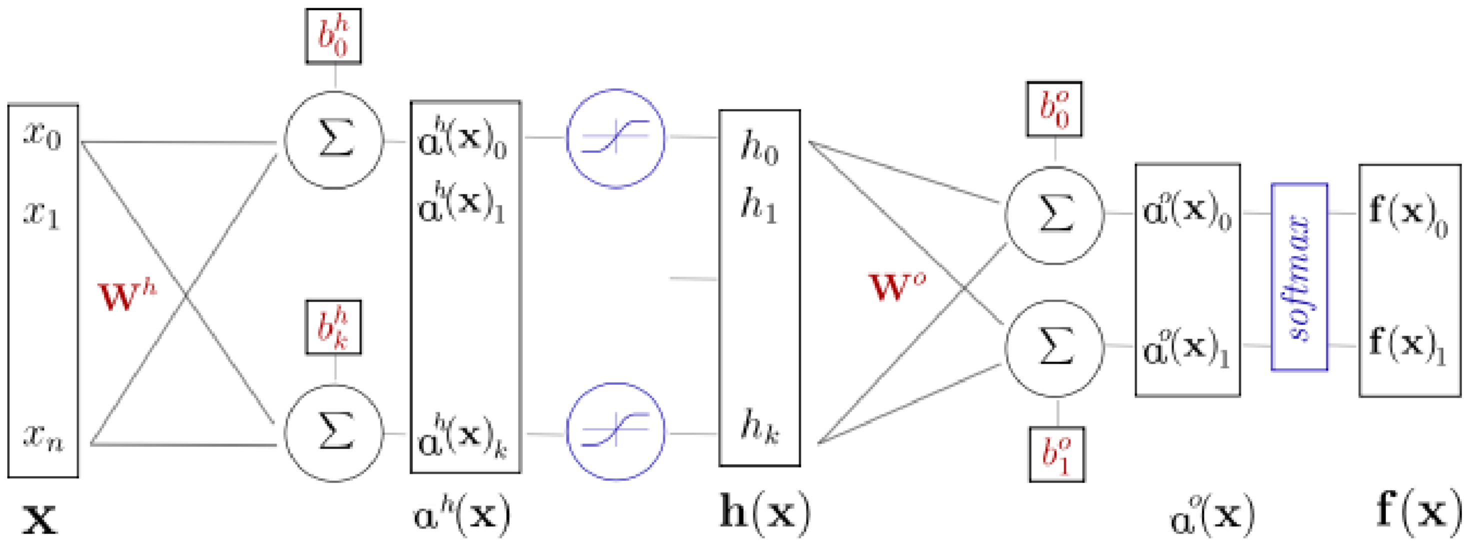


(f) Hard Tanh

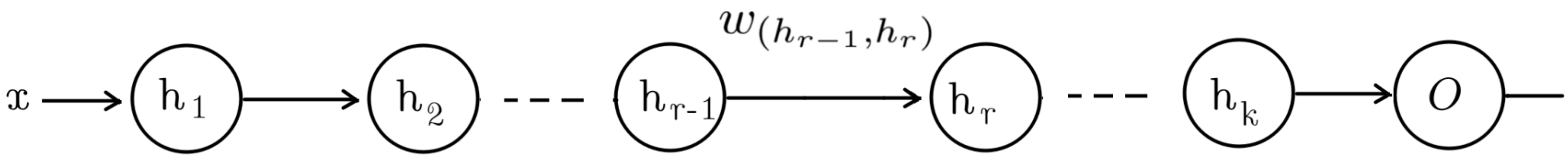
Softmax

$$softmax(x) = \frac{1}{\sum_{i=1}^{n} e^{x_i}} \cdot \begin{bmatrix} e^{x_1} \\ \vdots \\ e^{x_n} \end{bmatrix}$$

- Softmax activation for each neuron is in range [0,1] .
- The summation of neurons' activation is 1.
- It is ususally used in the output layer.

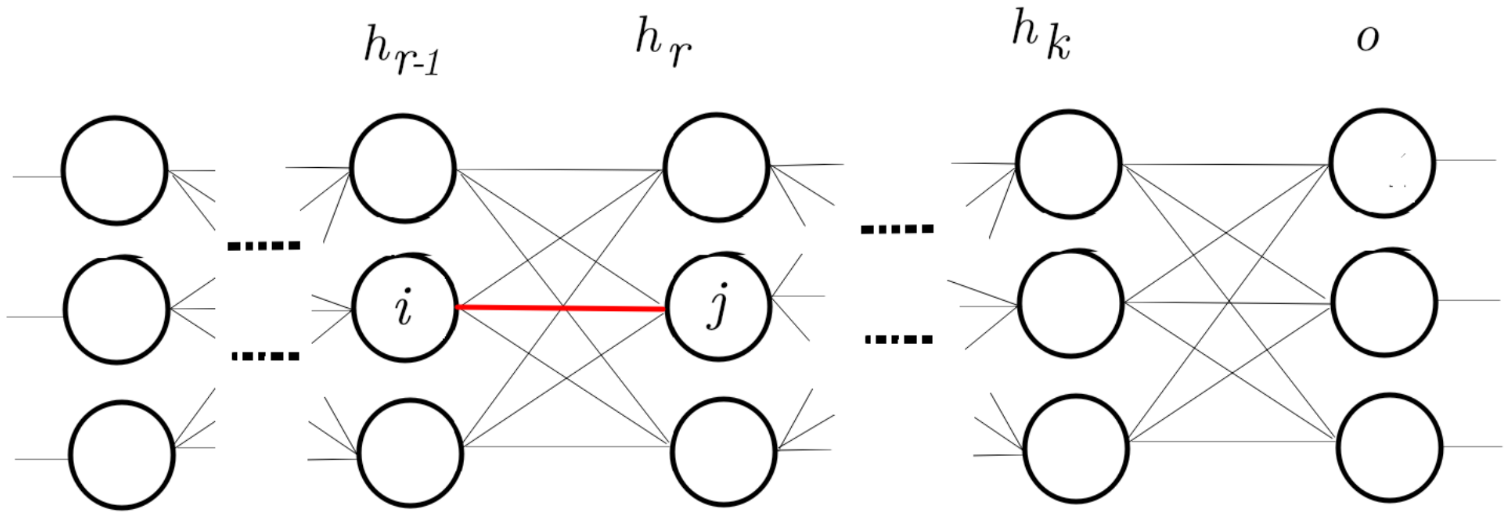


Error Backpropagation



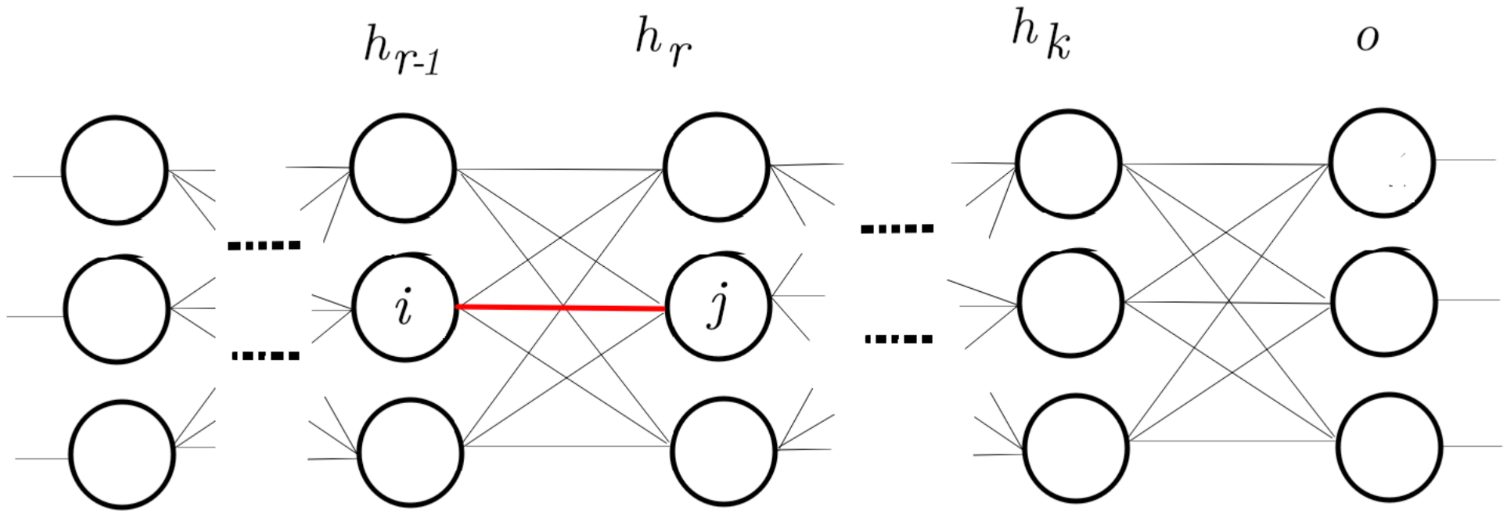
$$\frac{\partial L}{\partial w_{(h_{r-1},h_r)}} = \frac{\partial L}{\partial o} \cdot \left[\frac{\partial o}{\partial h_k} \prod_{i=r}^{k-1} \frac{\partial h_{i+1}}{\partial h_i} \right] \frac{\partial h_r}{\partial w_{(h_{r-1},h_r)}}$$

Error Backpropagation



$$\frac{\partial L}{\partial w_{(h^i_{r-1}, h^j_r)}} = \underbrace{\left[\sum_{[h^j_r, h_{r+1}, \ldots h_k, o] \in \mathcal{P}} \frac{\partial L}{\partial o} \frac{\partial o}{\partial h_k} \prod_{i=r}^{k-1} \frac{\partial h_{i+1}}{\partial h_i} \right]}_{\delta(h^j_r, o) = \frac{\partial L}{\partial h^j_r}} \frac{\partial h^j_r}{\partial w_{(h^i_{r-1}, h^j_r)}}$$

Error Backpropagation



$$\frac{\partial L}{\partial w_{(h_{r-1}^{i}, h_r^{j})}} = \delta(h_r^i, o) \cdot h_{r-1}^{i} \cdot \Phi'(a_{h_r^i})$$

SGD using Backpropagation

For each training sample:

- Compute the forward path.
- Compute $\Delta(o,o)$ for each output neuron.
- Update each connecting weight of the output layer as

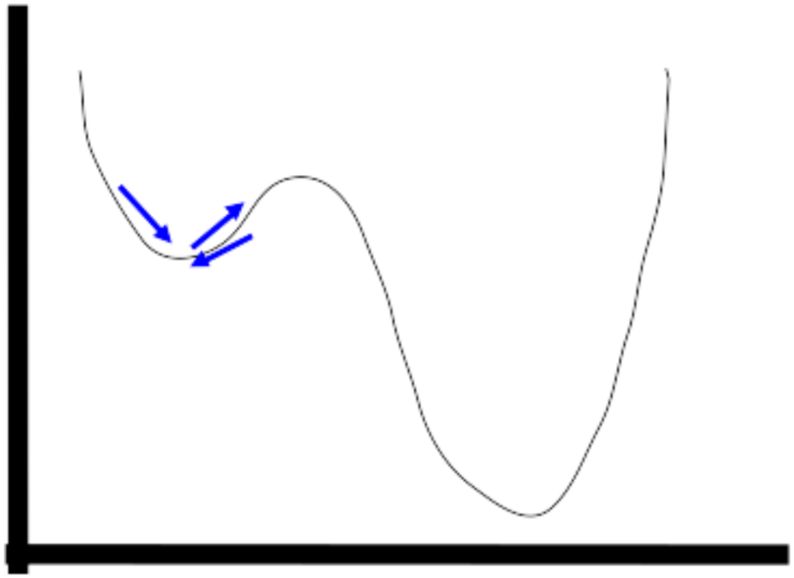
$$w_{(h_k,o)} = w_{(h_k,o)} - \eta \cdot \delta(o,o) \cdot h_k \cdot \Phi'(a_o)$$

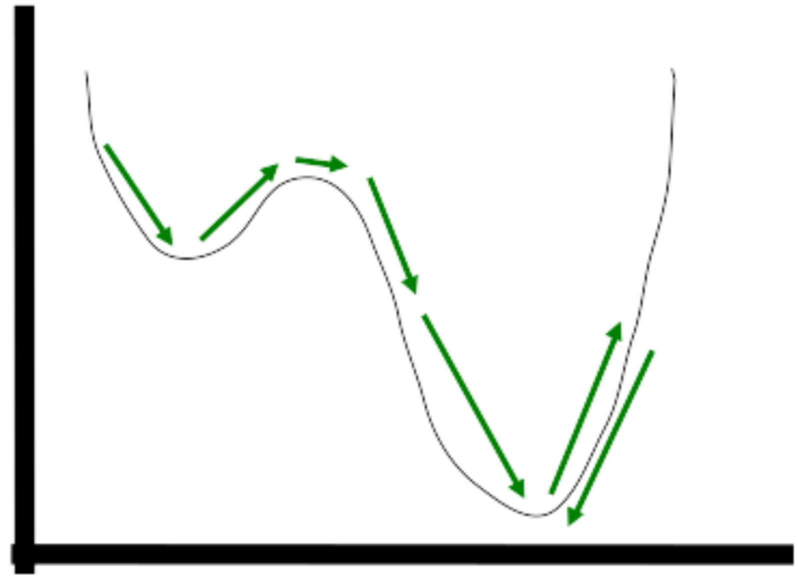
- For $r = k, k-1, ..., 1$:
  - Compute $\Delta(h_r^i, o)$ for the i-th neuron at the r-th hidden layer.
  - Update each connecting weight of the i-th neuron at the r-th hidden layer as:

$$w_{(h_{r-1},h_r)} = w_{(h_{r-1},h_r)} - \eta \cdot \delta(h_r, o) \cdot h_{r-1} \cdot \Phi'(a_{h_r})$$

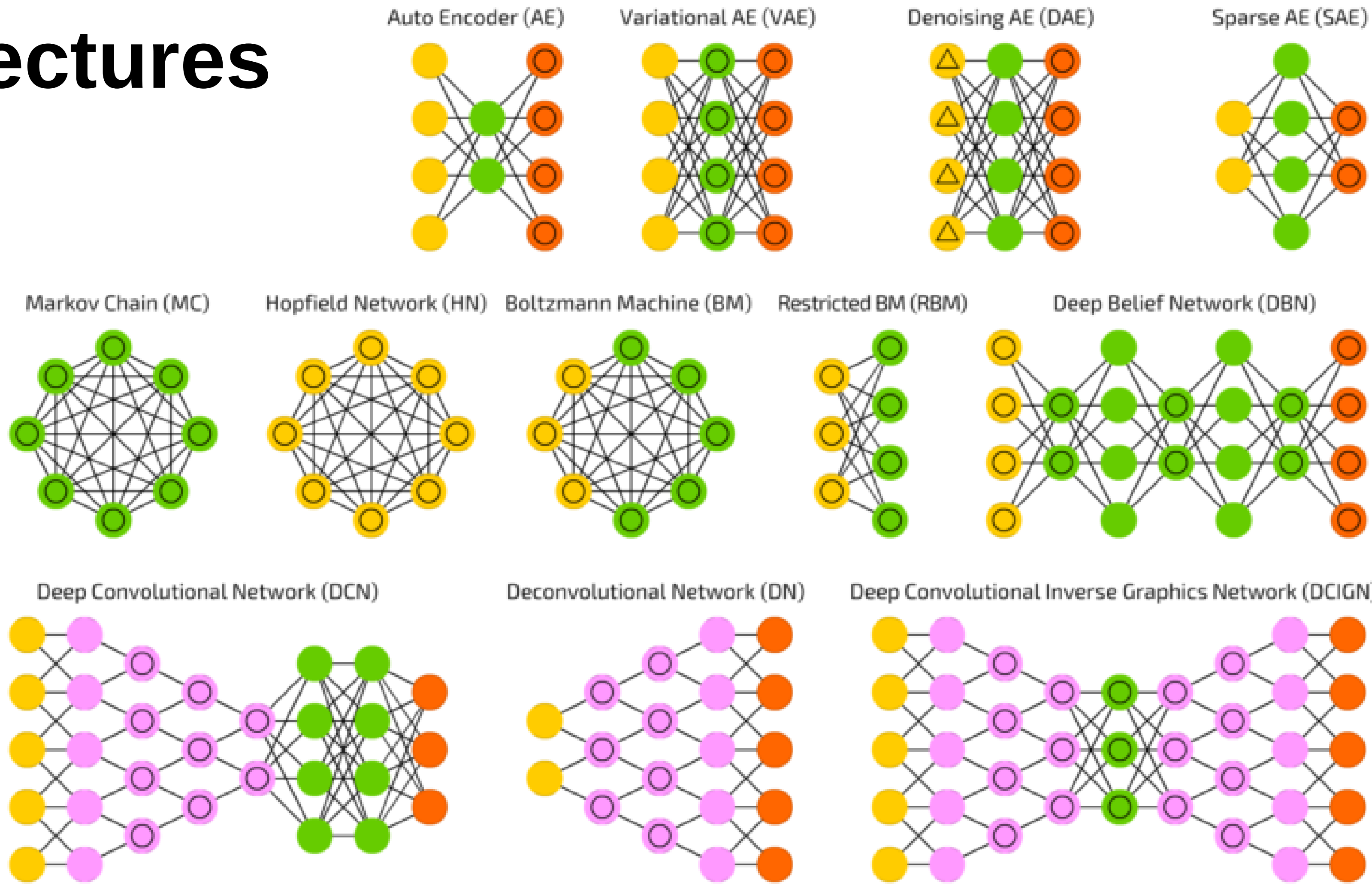
SGD with Momentum

$$\Delta w_{ij} = \eta . \delta_i . x_j . \Phi'(a_i) + \alpha . \Delta w_{ij}$$

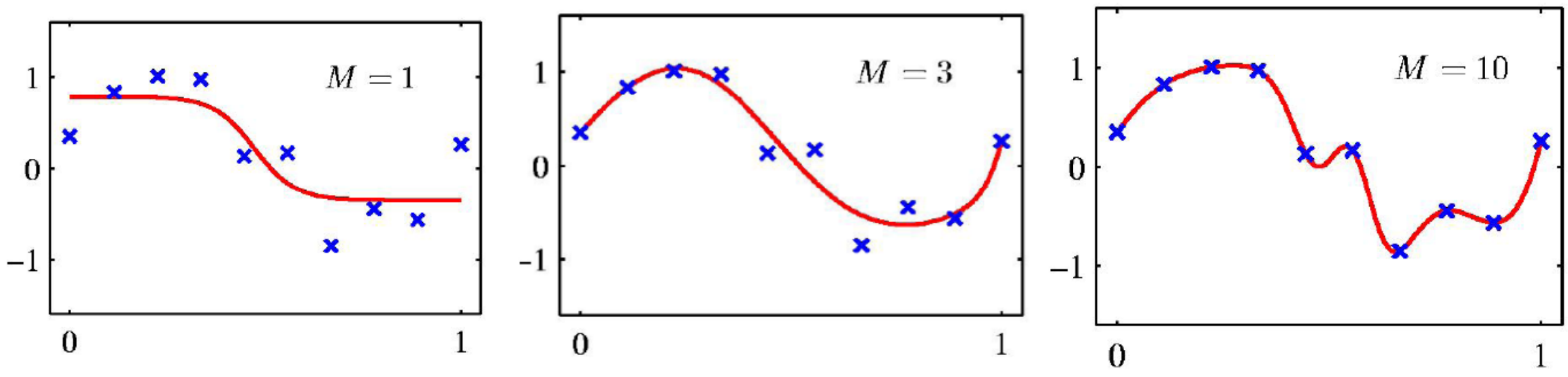




Architectures



Overfitting



- **Generalization**: To establish a balance between correct responses for the training patterns and unseen new patterns.
- **Memorization**: When the model momorizes training samples instead of learning the descriptive common patterns.
- **Overfitting**: Weak generalization. It happens when the network complexity is more than the problem complexity.

How to avoid overfitting

- One possible approach is to reduce the size of the network.
  - However, large networks have the potential to be more powerful than small networks.
- Provide more training samples (not always possible).
- Stop learning before overfitting happens.
- Use regularization terms to dynamically adjust network complexity.
- Use ensemble methods.
- use random dropout technique for hidden neurons.

Regularization

- Since a larger number of parameters causes overfitting, a natural approach is to constrain the model to use fewer non-zero parameters.
- The most applied regularization is adding the penalty $\lambda||W||$ to the loss function .

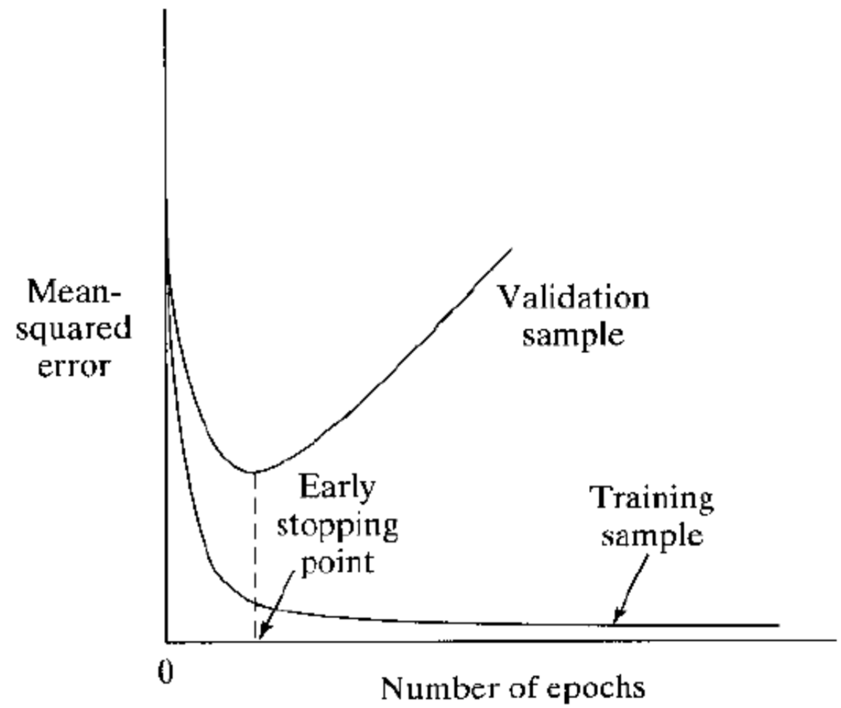
$$L = \frac{1}{2}(Y - y)^2 + \lambda||W||$$

- Therefore the learnin rule is re-written as:

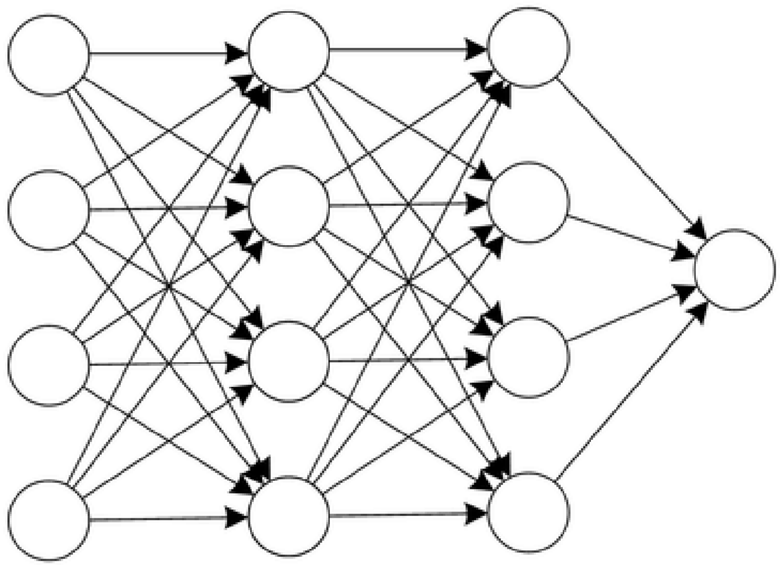
$$\Delta w_{ij} = \eta.\delta_i.x_j.\Phi'(a_i) - \eta.\lambda.w_{ij}$$

Early stopping

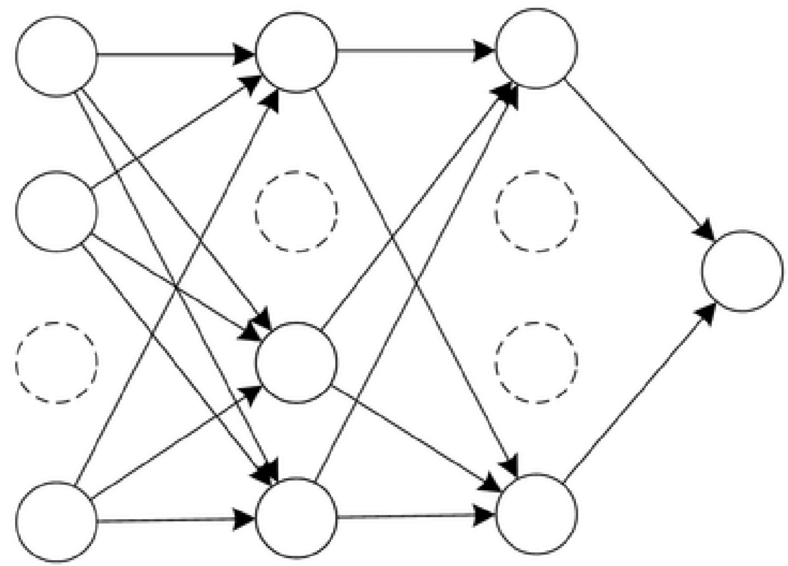
- Split training samples into a training set (80%) and a validation set (20%).



Dropout



(a) Standard Neural Network



(b) Network after Dropout

Thank you