Multi Layer Perceptron

Ludwig Krippahl

Multi Layer Perceptron

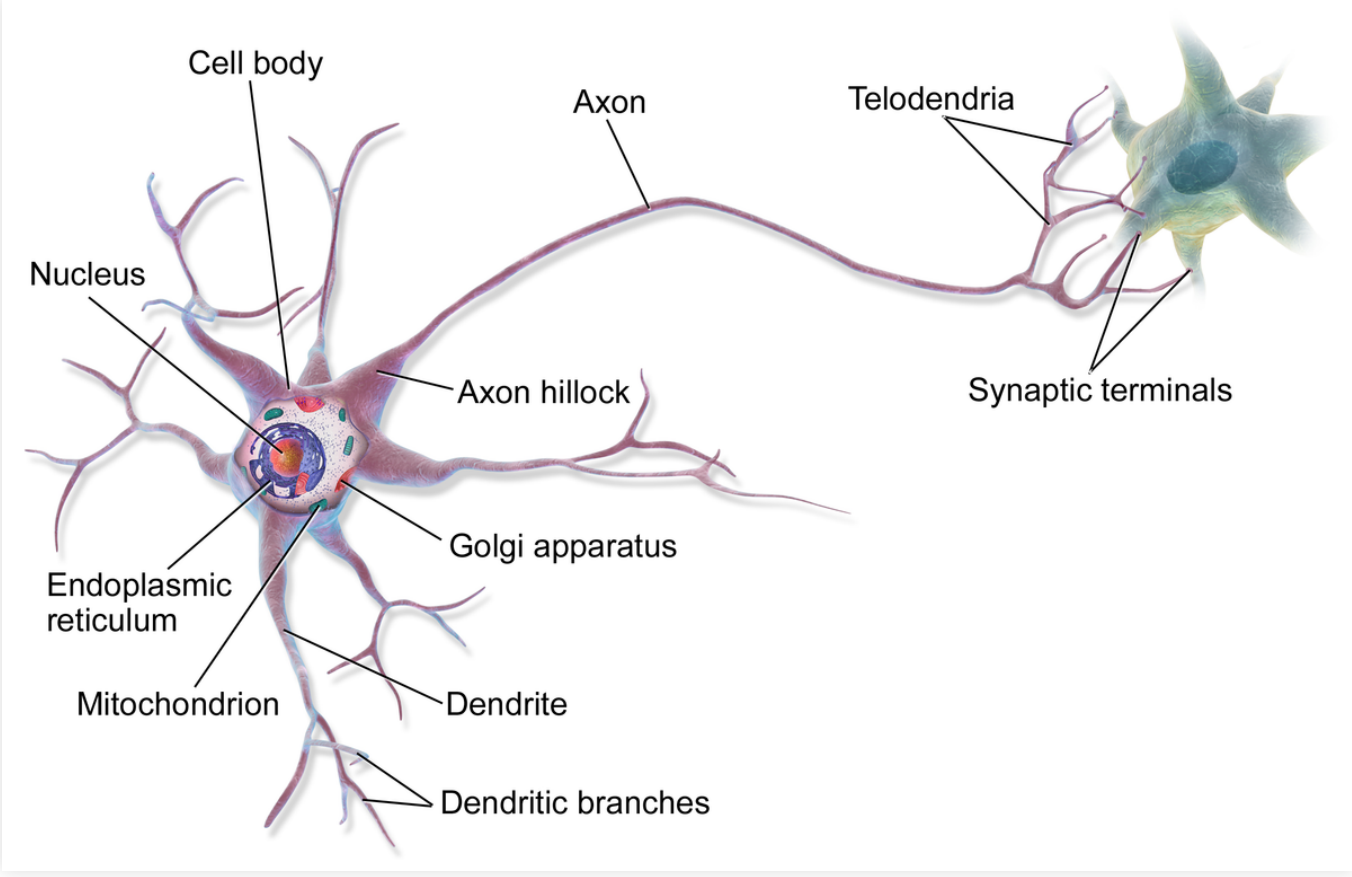
Summary

- Perceptron and linear discrimination
- Multilayer Perceptron, nonlinear discrimination
- Backpropagation and training MLP
- Regularization in MLP

Perceptron

Perceptron

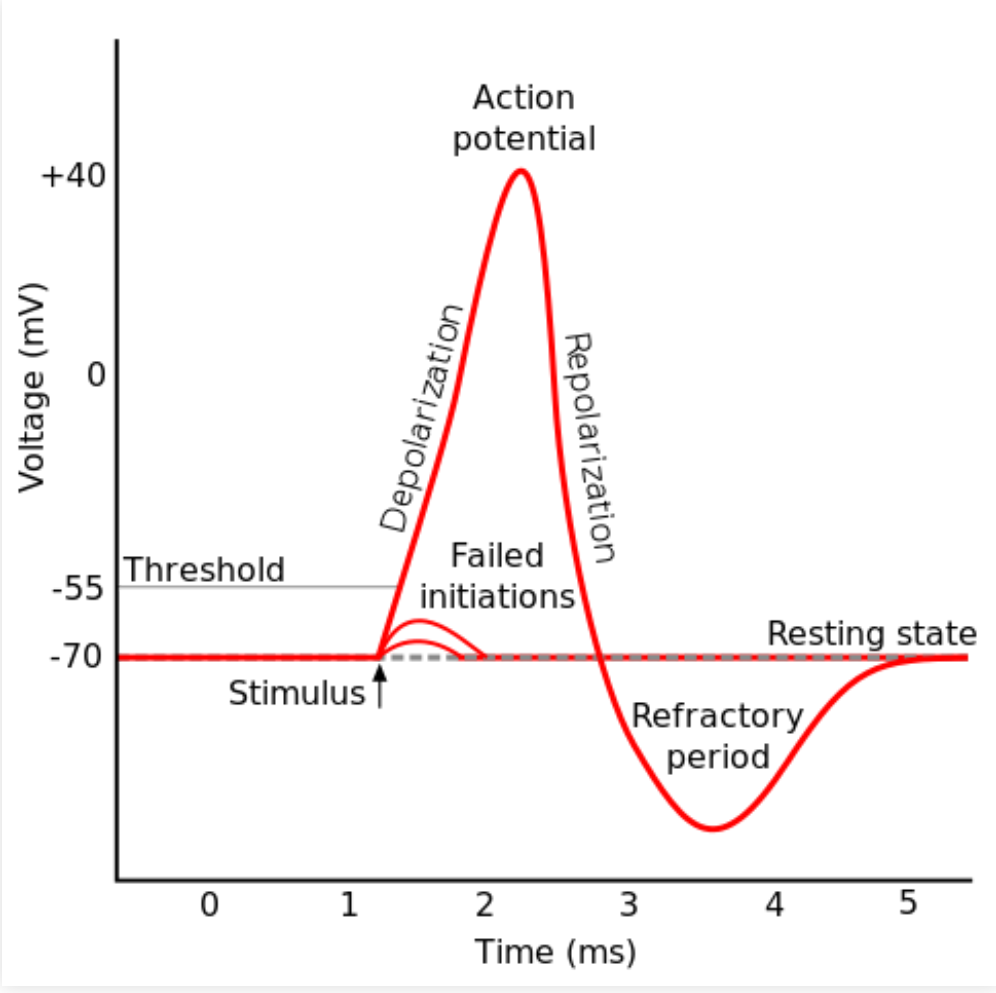
Inspired on the neuron:



BruceBlaus, CC-BY, source Wikipedia

Perceptron

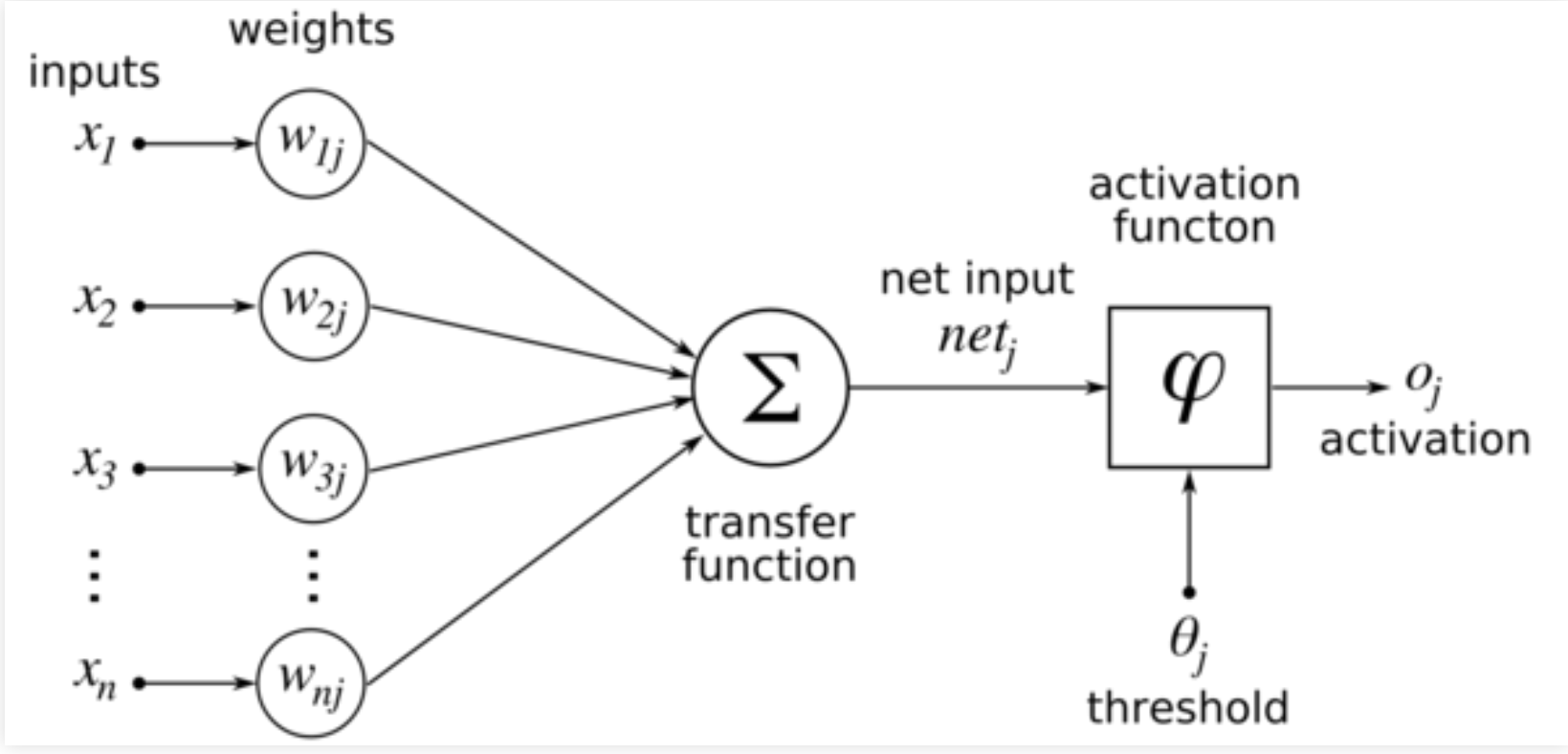
Action potential is "all or nothing"



Chris 73, CC-BY, source Wikipedia

Perceptron

- Model neuron as linear combination of inputs and non-linear function



Chrislb, CC-BY, source Wikipedia

Perceptron

- Orginal perceptron combined a linear combination of the inputs and a threshold function:

$$y = \sum_{j=1}^{d} w_j x_j + w_0 \qquad s(y) = \begin{cases} 1, & y > 0 \\ 0, & y \leq 0 \end{cases}$$

- Note: we can include w_0 in $\vec{w}$ as before.

Training rule for the original perceptron:

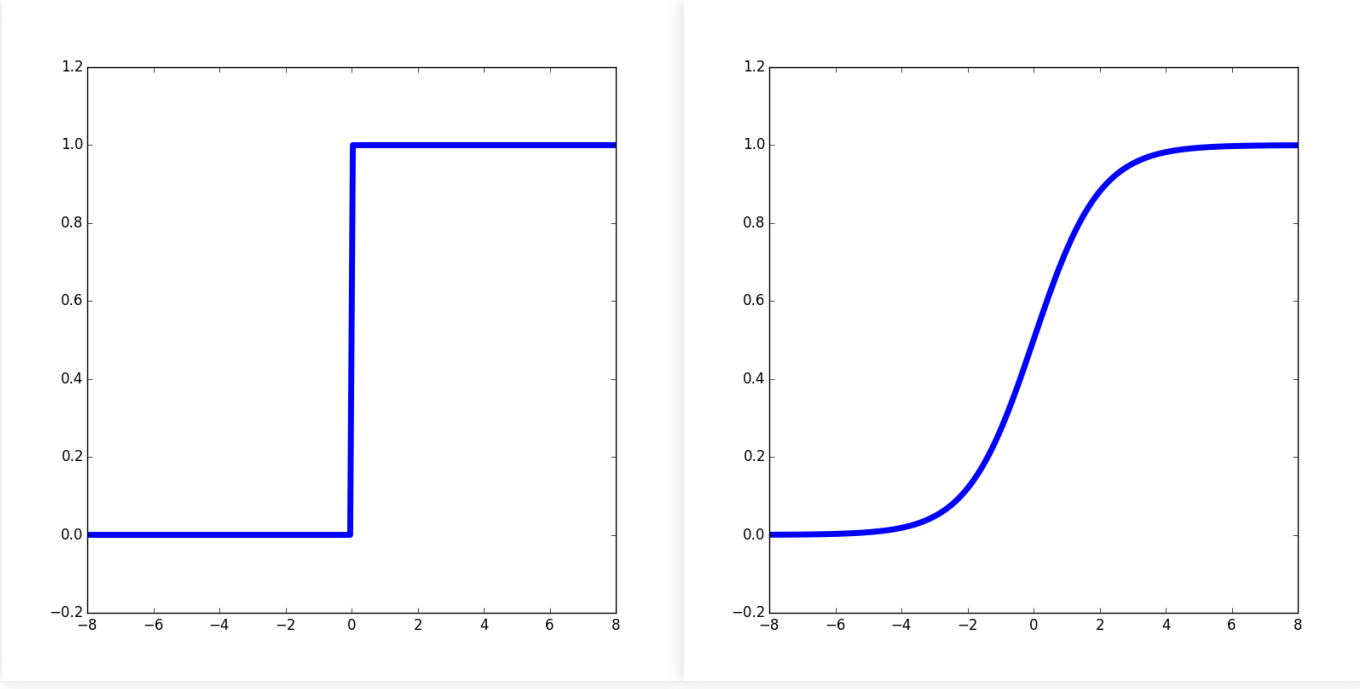
$$w_i = w_i + \Delta w_i \qquad \Delta w_i = \eta(t - o)x_i$$

- Adjust weights slightly to correct misclassified examples.
- Greater adjustment to those with larger inputs.
- Problem: not differentiable. But there are alternatives.

Perceptron

- To use in multiple layers, need continuous derivative (E.g. sigmoid)

$$s(y) = \frac{1}{1 + e^{-y}} = \frac{1}{1 + e^{-\vec{w}^T \vec{x}}}$$



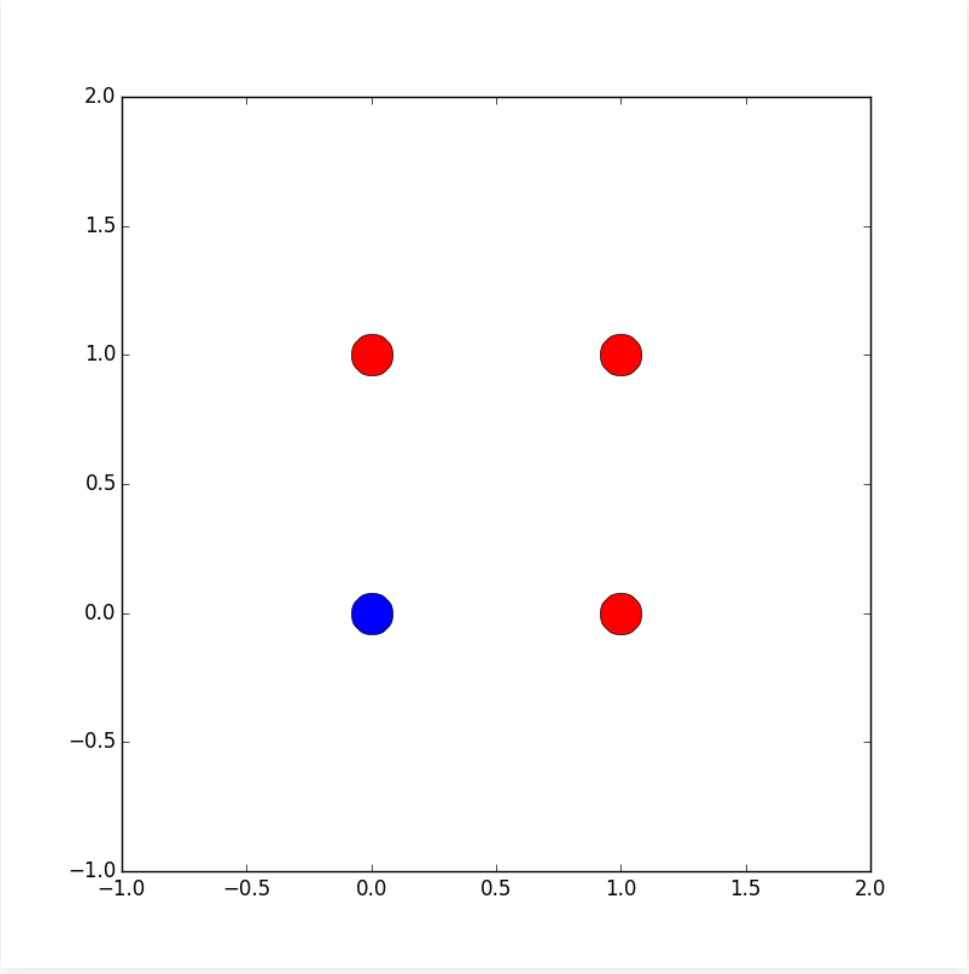
Perceptron

- Strictly speaking, perceptron is the original, single layer, network model by Frank Rosenblatt (1957), with threshold activation.
- However, the term is often also used for neurons with continuous threshold functions, such as in the the multilayer perceptron
- In this course, we will not worry about the name...

One Neuron

Single Neuron

- Can classify linearly separable sets, such as the OR function



Training a single neuron

- Minimize quadratic error between class and output

$$E = \frac{1}{2}\sum_{j=1}^{N}(t^j - s^j)^2$$

- Like perceptron, present each example and adjust weights.

$$E^t = \frac{1}{2}(t^j - s^j)^2$$

Training a single neuron

- Gradient of the error wrt w: $-\frac{\delta E^j}{\delta w_i} = -\frac{\delta E^j}{\delta s^j}\frac{\delta s^j}{\delta net^j}\frac{\delta net^j}{\delta w_i}$

$$E^t = \frac{1}{2}(t^j - s^j)^2 \qquad \frac{\delta E^j}{\delta s^j} = -(t^j - s^j)$$

$$s^j = \frac{1}{1 + e^{-net^j}} \qquad \frac{\delta s^j}{\delta net^j} = s^j(1 - s^j)$$

$$net^j = w_0 + \sum_{i=1}^{M} w_i x_i \qquad \frac{\delta net^j}{\delta w_i} = x_i$$

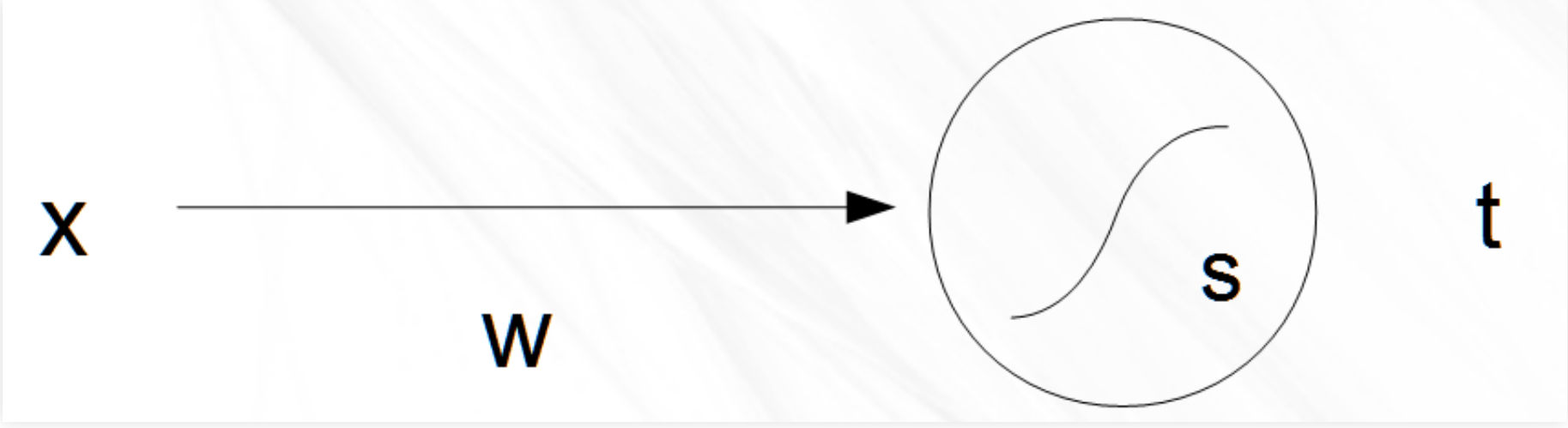
- Update rule (η generally between 0.1 and 0.5):

$$\Delta w_i^j = -\eta \frac{\delta E^j}{\delta w_i} = \eta(t^j - s^j)s^j(1 - s^j)x_i^j$$

Single Neuron

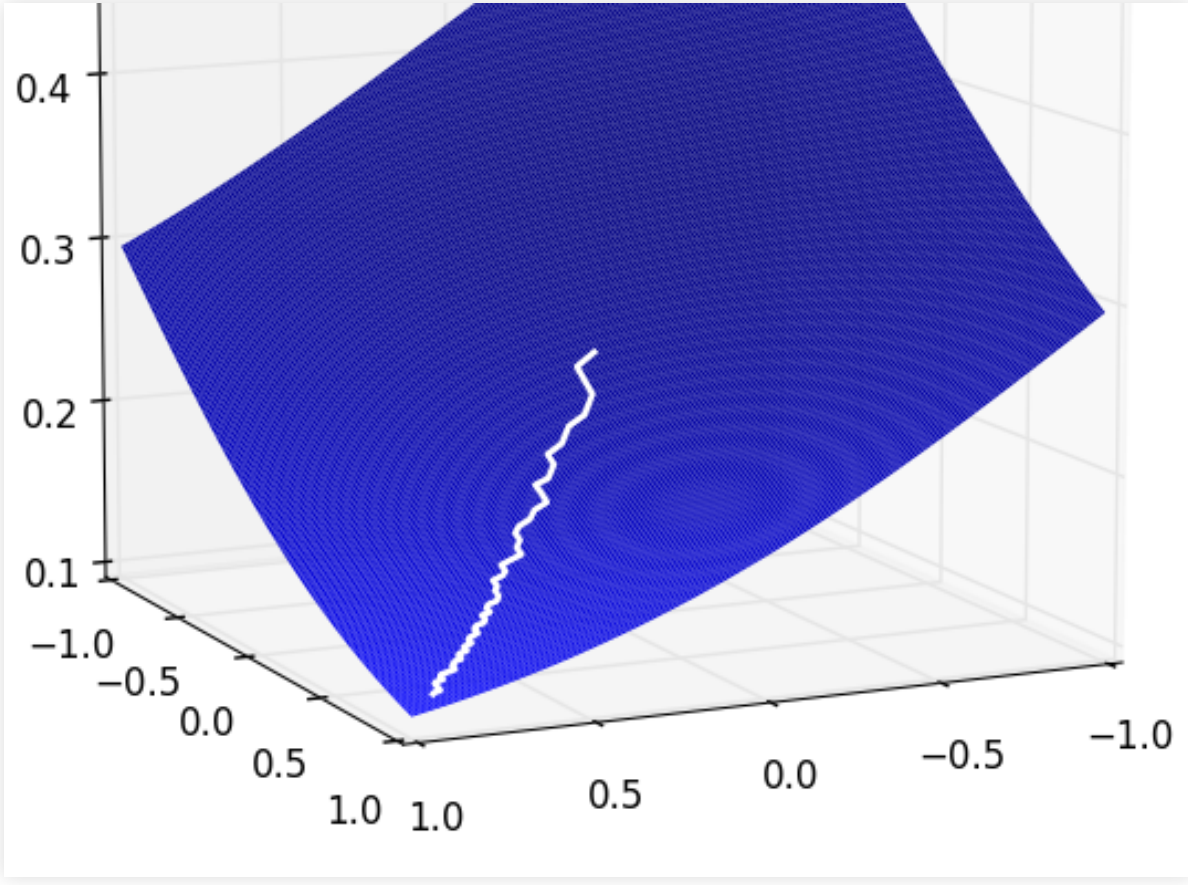
Intuitive explanation:

$$\Delta w_i^j = -\eta \frac{\delta E^j}{\delta w_i} = \eta(t^j - s^j)s^j(1 - s^j)x_i^j$$



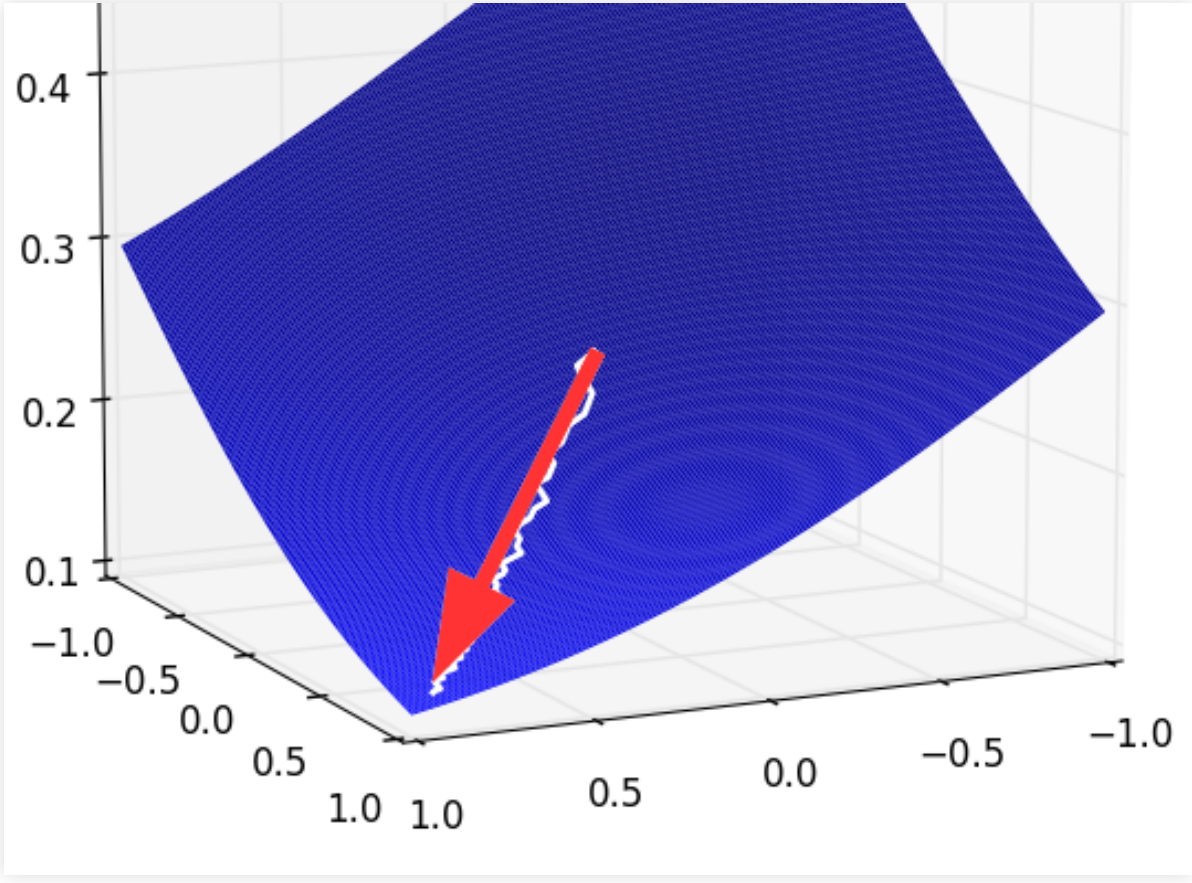
Stochastic Gradient Descent

- Online learning: one step per example, in random order



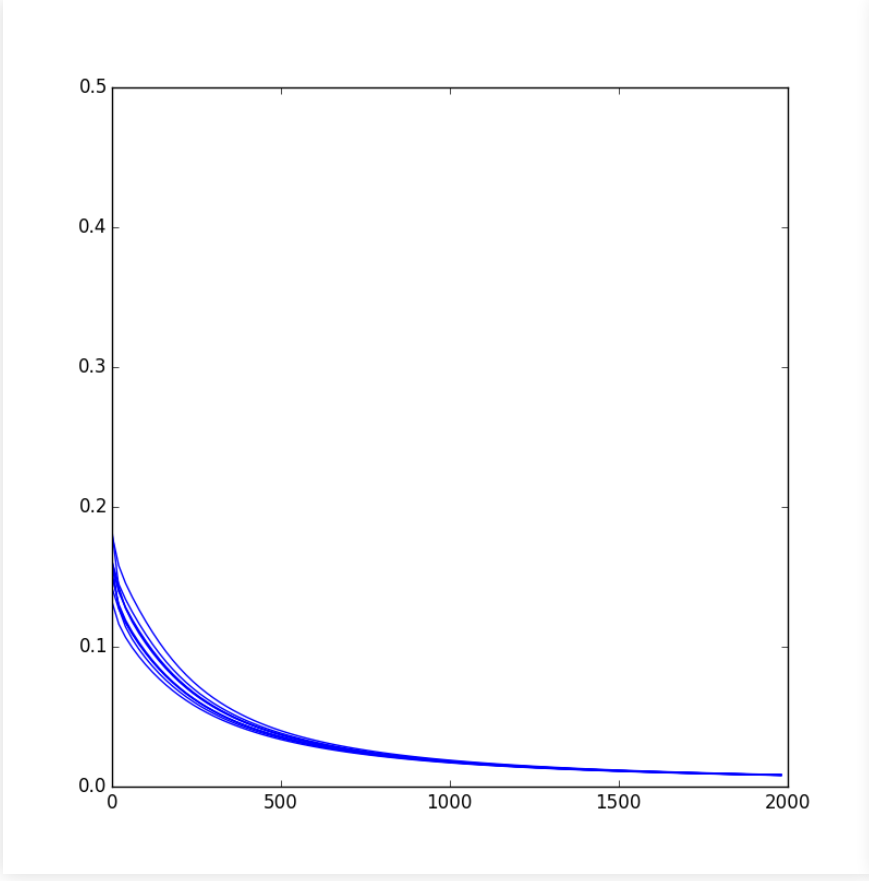
Stochastic Gradient Descent

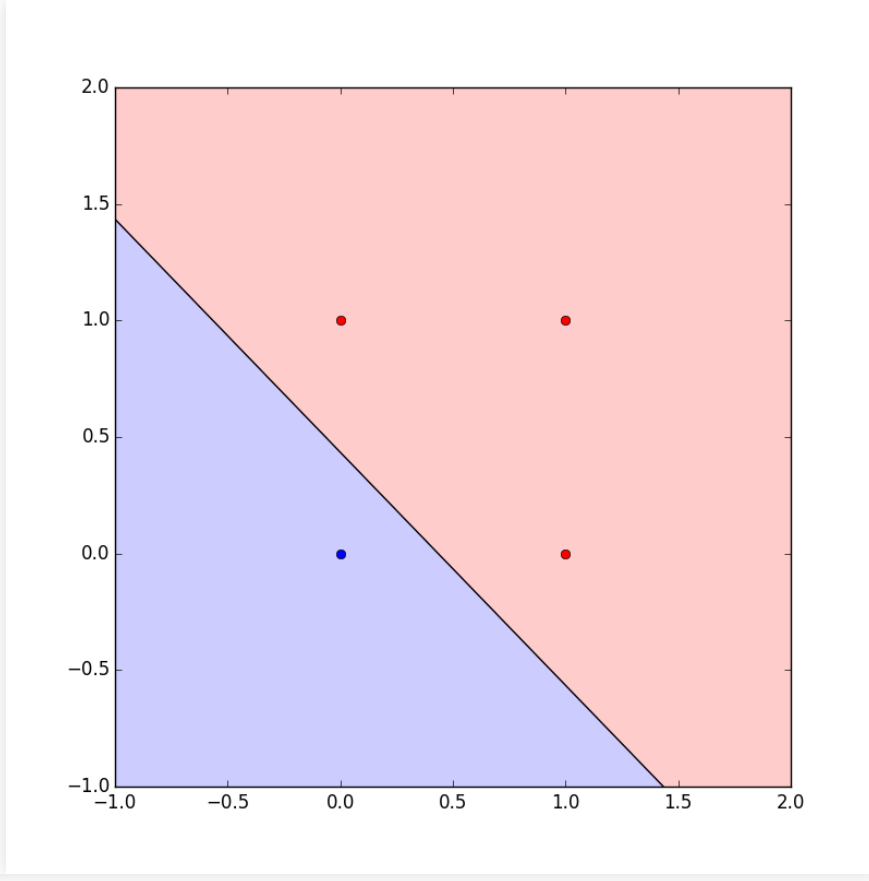
- Batch training: add Δw_i^j for batch, then update.



Single Neuron

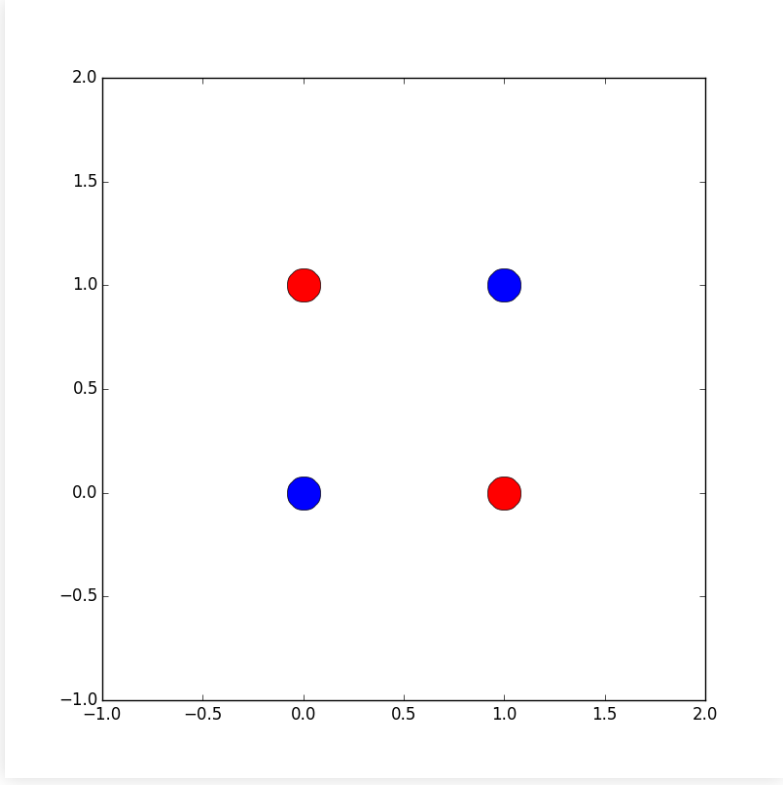
- OR: error per epoch, classifier.





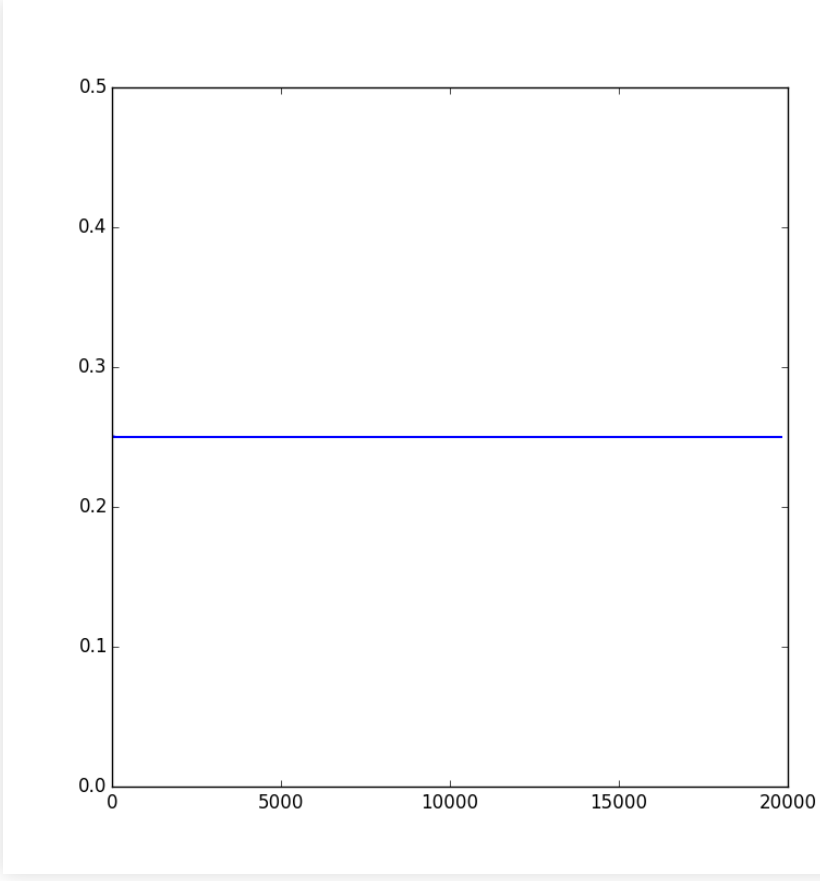
Single Neuron

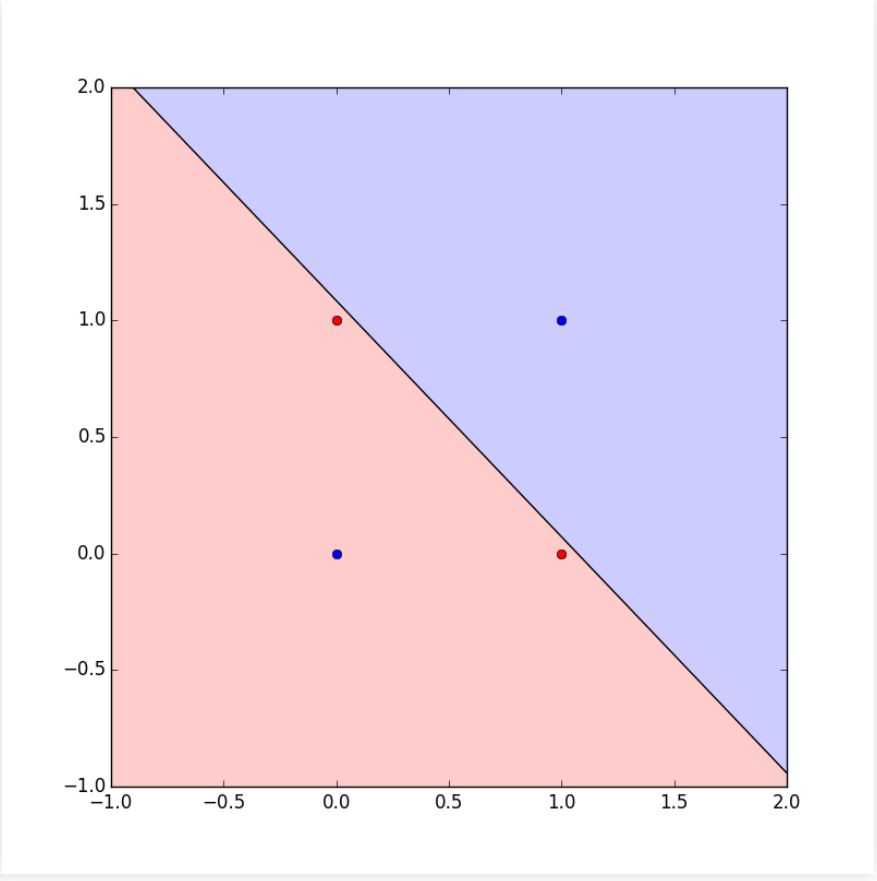
- Cannot classify non-linearly separable sets (e.g. XOR function)



Single Neuron

- XOR: error per epoch, classifier.

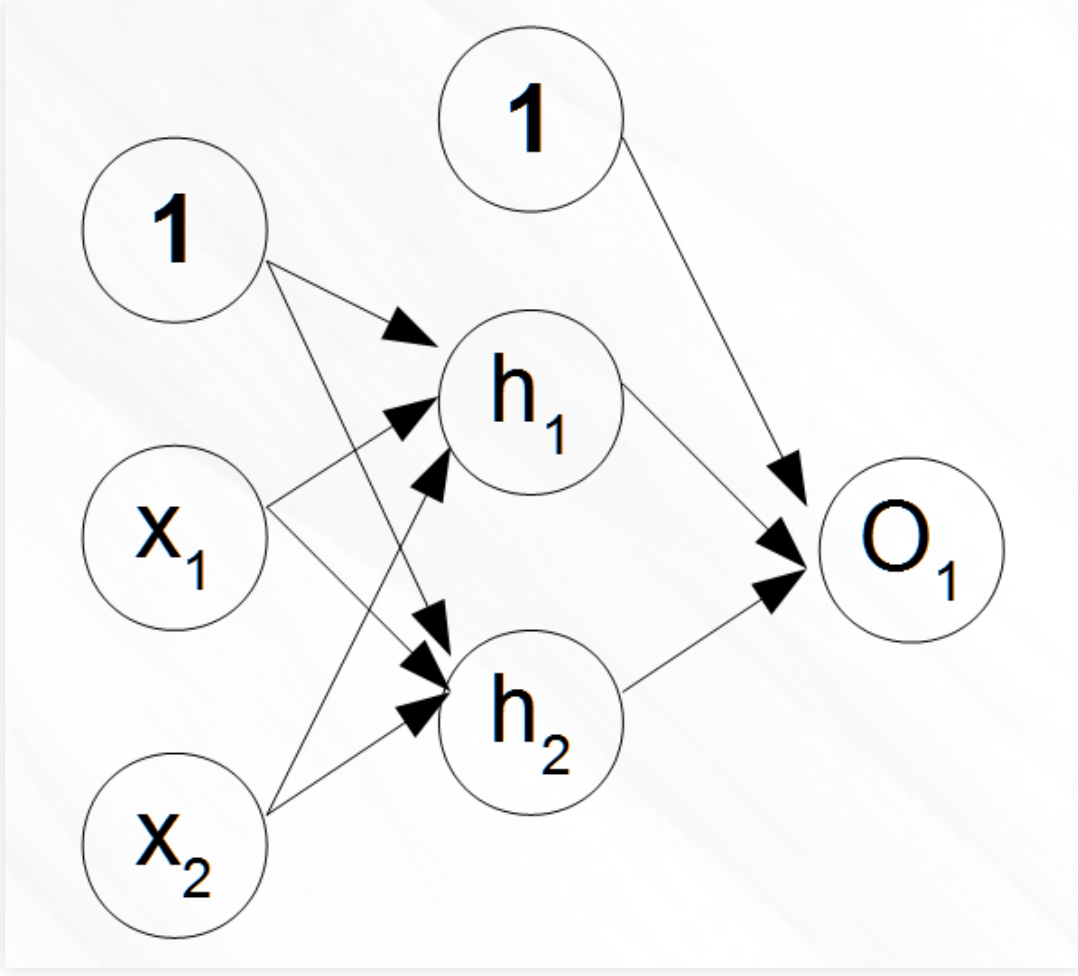




Multilayer Perceptron

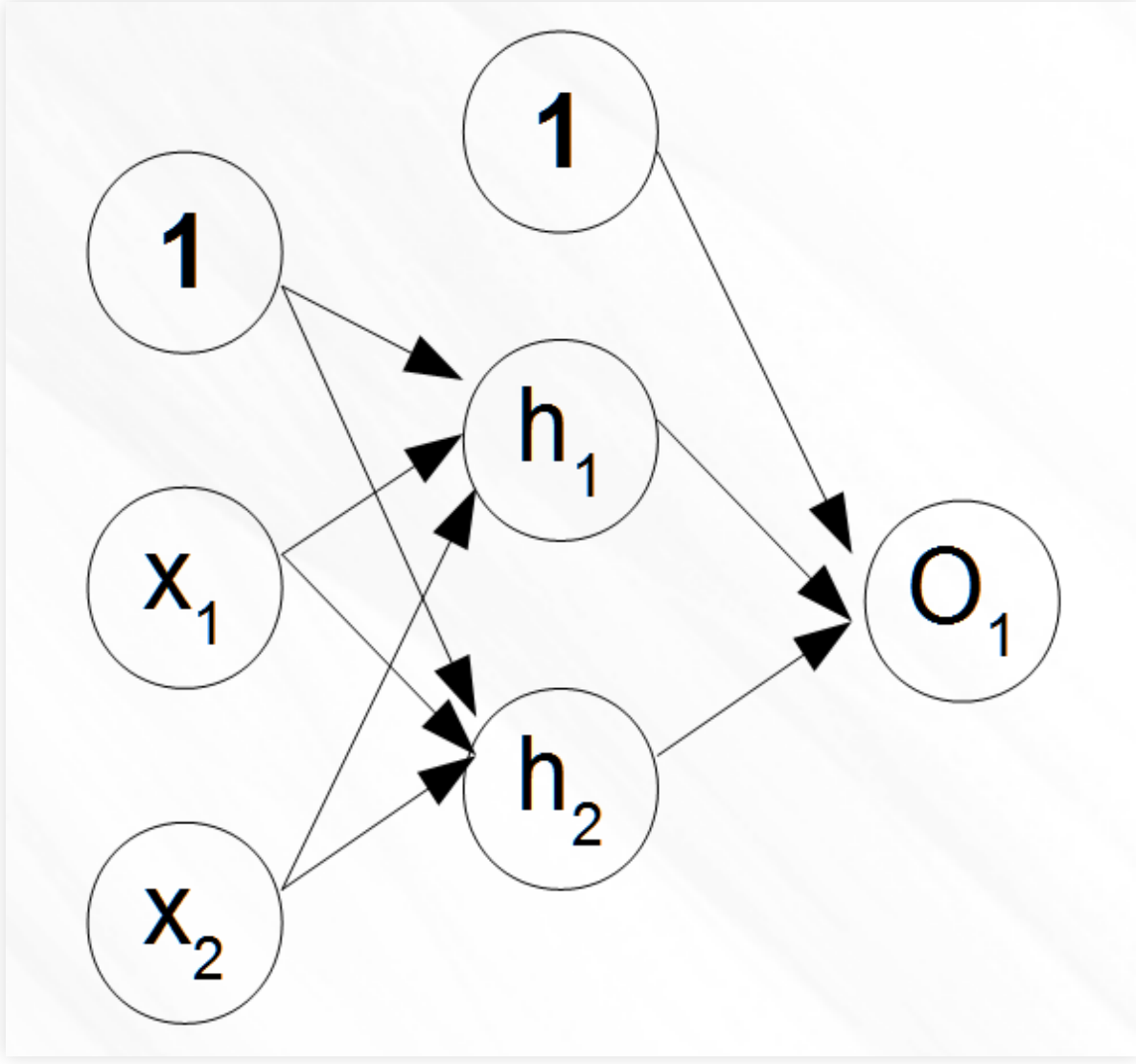
Multilayer Perceptron

- We can solve the XOR problem with a hidden layer



Multilayer Perceptron

Fully connected
Feed-forward
Input layer: x_1, x_2
Hidden layer: H_1, H_2
Output layer: O_1
(all sigmoidal)



Training a Multilayer Perceptron

- Output neuron n of layer k receives input from m from layer i through weight j
- Same as single neuron but using output of previous instead of x

$$\begin{aligned}\Delta w^j_{mkn} &= -\eta \frac{\delta E^j_{kn}}{\delta s^j_{kn}} \frac{\delta s^j_{kn}}{\delta net^j_{kn}} \frac{\delta net^j_{kn}}{\delta w_{mkn}} \\ &= \eta(t^j - s^j_{kn}) s^j_{kn}(1 - s^j_{kn}) s^j_{im} = \eta \delta_{kn} s^j_{im}\end{aligned}$$

- Compute δ for each neuron

$$\delta = (t^j - s^j_{kn}) s^j_{kn}(1 - s^j_{kn})$$

Multilayer Perceptron

- For a weight m on hidden layer i, we must propagate the output error backwards from all neurons ahead
- Gradient of error w.r.t. weight of output neuron:

$$\frac{\delta E_{kn}^{j}}{\delta s_{kn}^{j}} \frac{\delta s_{kn}^{j}}{\delta net_{kn}^{j}} \frac{\delta net_{kn}^{j}}{\delta w_{mkn}}$$

- Propagate back the errors of all forward neurons (and compute δ):

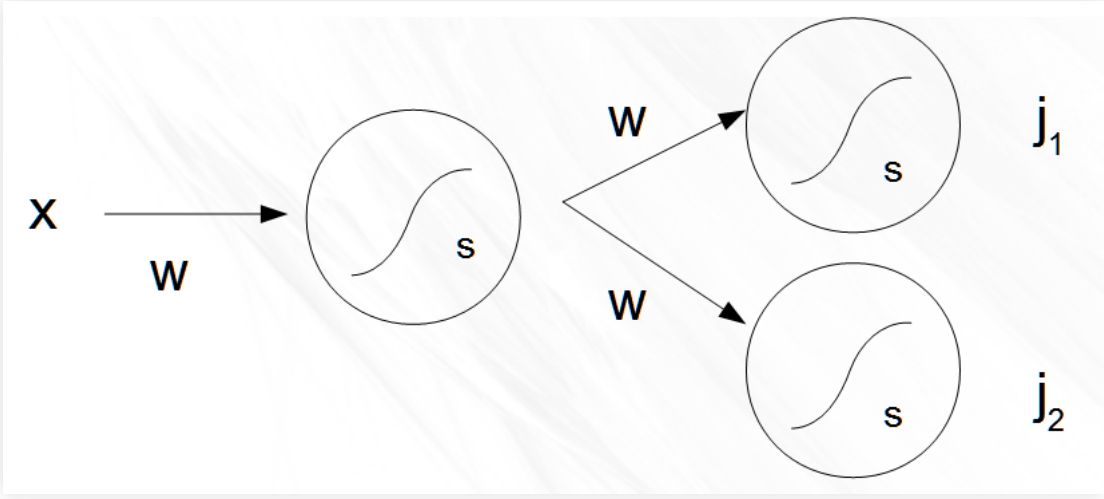
$$\Delta w_{min}^{j} = -\eta \left(\sum_{p} \frac{\delta E_{kp}^{j}}{\delta s_{kp}^{j}} \frac{\delta s_{kp}^{j}}{\delta net_{kp}^{j}} \frac{\delta net_{kp}^{j}}{\delta s_{in}^{j}} \right) \frac{\delta s_{in}^{j}}{\delta net_{in}^{j}} \frac{\delta net_{in}^{j}}{\delta w_{min}}$$

$$= \eta (\sum_{p} \delta_{kp} w_{mkp}) s_{in}^{j} (1 - s_{in}^{j}) x_{i}^{j} = \eta \delta_{in} x_{i}^{j}$$

Intuitive explanation:

$$\Delta w^j_{min} = -\eta \left(\sum_p \frac{\delta E^j_{kp}}{\delta s^j_{kp}} \frac{\delta s^j_{kp}}{\delta net^j_{kp}} \frac{\delta net^j_{kp}}{\delta s^j_{in}} \right) \frac{\delta s^j_{in}}{\delta net^j_{in}} \frac{\delta net^j_{in}}{\delta w_{min}}$$

$$= \eta(\sum_p \delta_{kp} w_{mkp}) s^j_{in}(1 - s^j_{in}) x^j_i = \eta \delta_{in} x^j_i$$



Backpropagation Algorithm

- Propagate the input forward through all layers

$$s(\vec{x}) = \frac{1}{1 + e^{-(\vec{w}^T \vec{x})}}$$

- For output neurons compute

$$\delta_k = s_k(1 - s_k)(t - s_k)$$

- Backpropagate errors to back layers to compute all δ

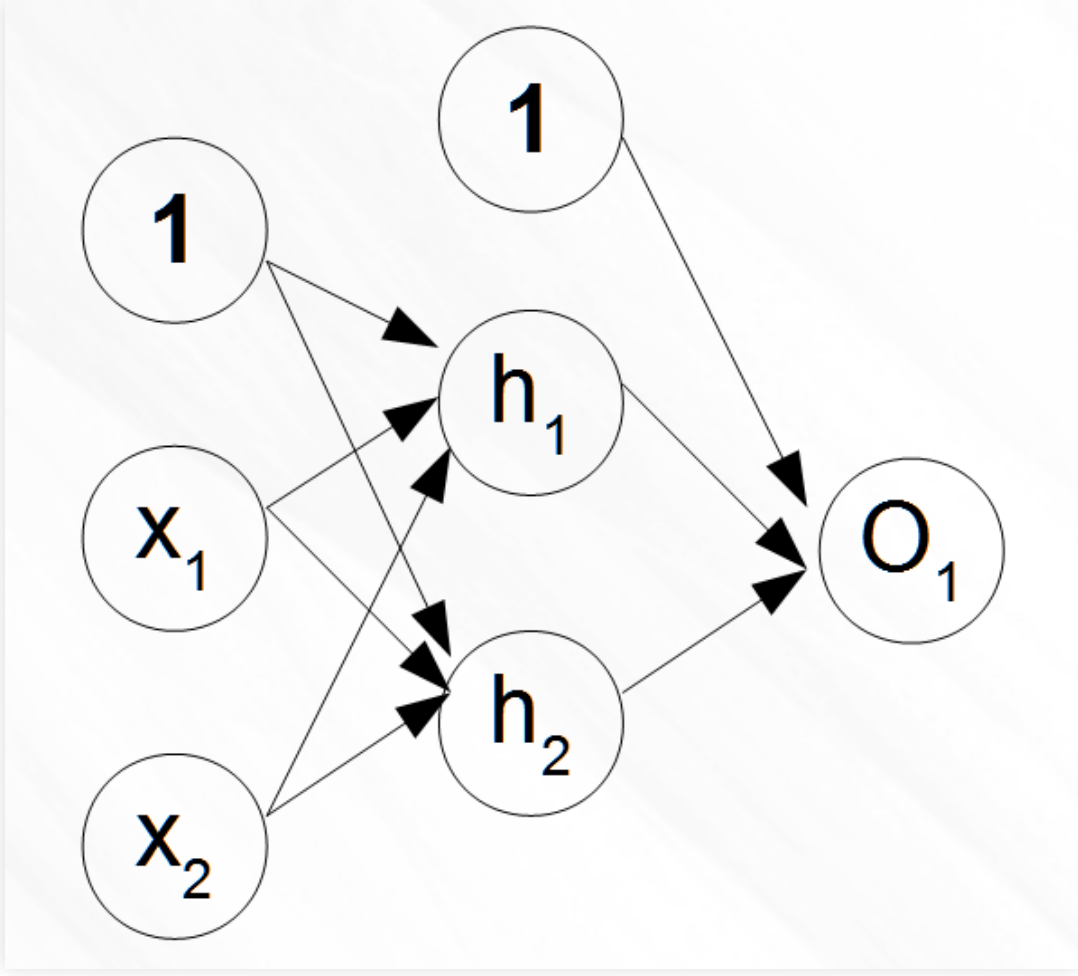
$$\delta_i = s_i(1 - s_i) \sum_p \delta_p w_{pk}$$

- Note: w_{pk} are weights of "front" neurons connecting to neuron i
- Update weights (for forward layers, x is s of back layer)

$$\Delta w_{ki} = \eta \delta_i x_{ki}$$

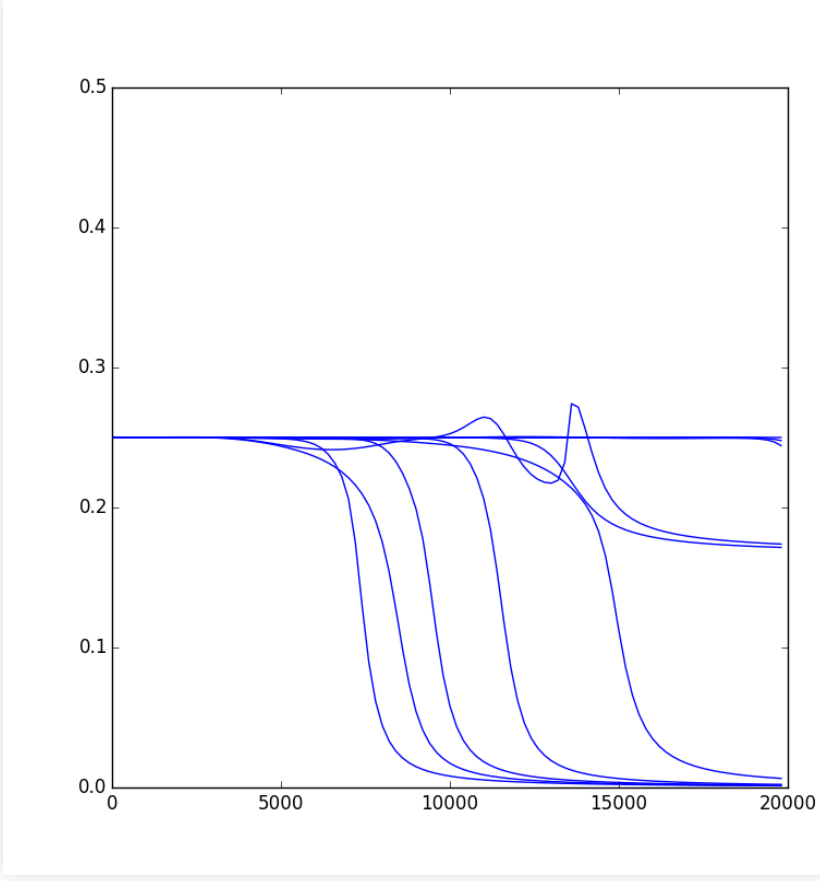
Multilayer Perceptron

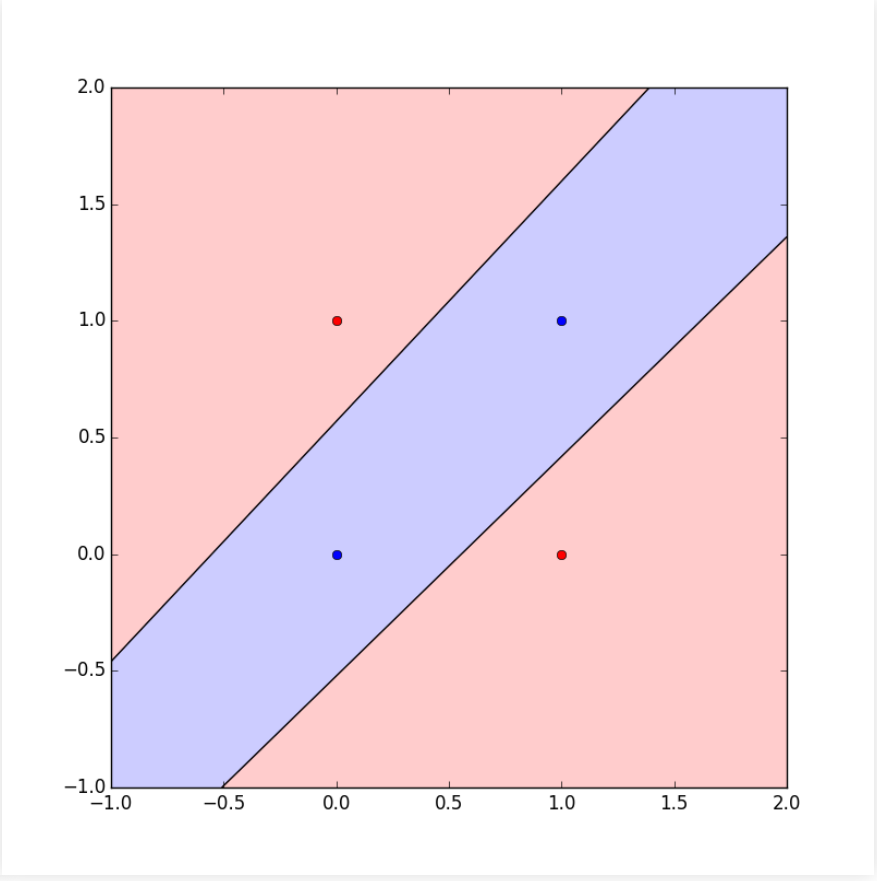
- XOR problem, one hidden layer



Multilayer Perceptron

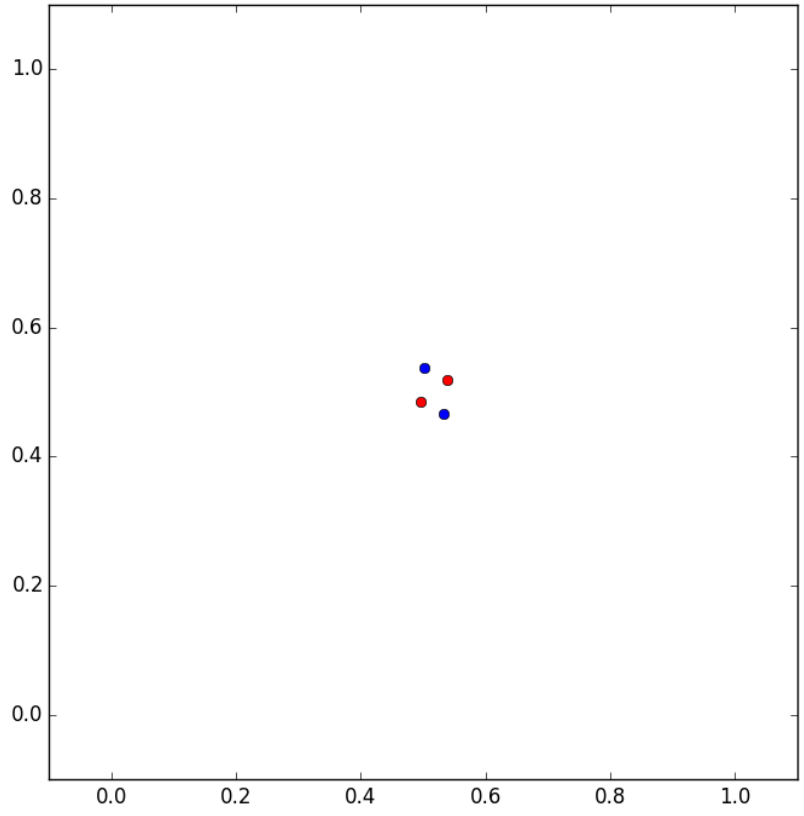
- XOR: error per epoch, classifier.





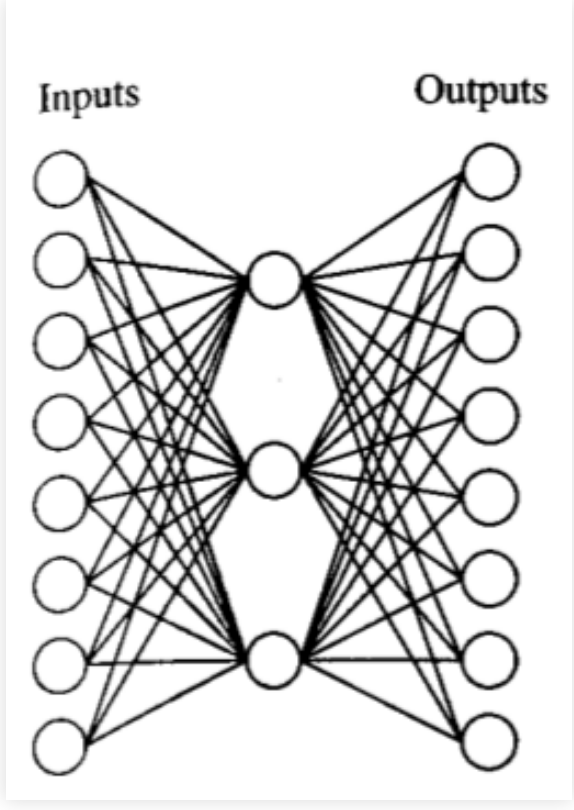
Multilayer Perceptron

- Hidden layer maps inputs to a linearly separable representation.



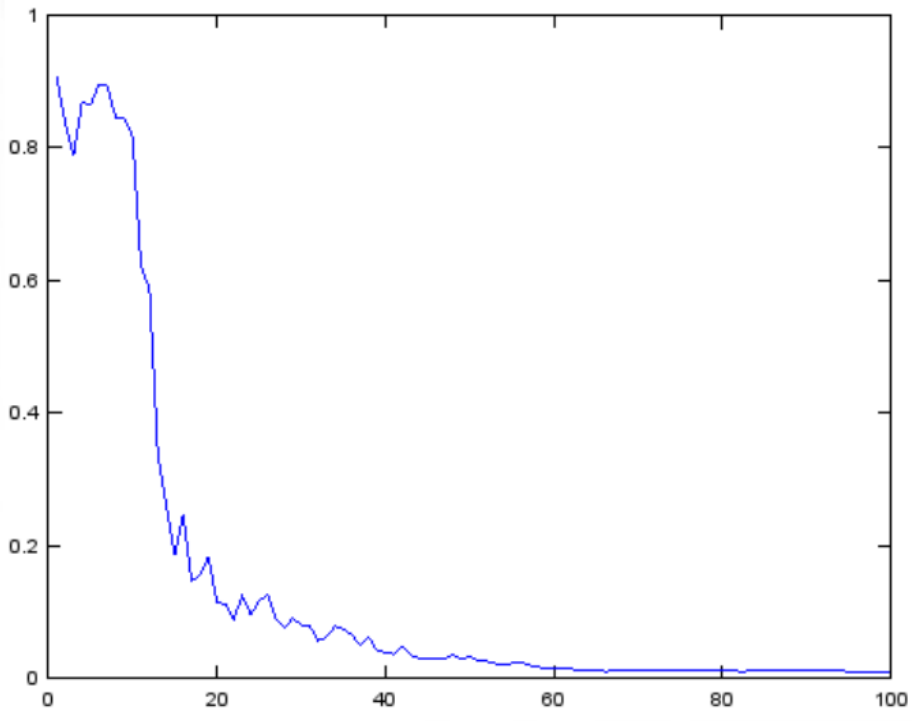
Autoassociator (autoencoder)

- We can use the "reencoding" of the hidden layers for reducing the dimensions.
  - Mitchell's binary encoder
    - 8 inputs
    - 3 hidden neurons
    - 8 output neurons
    - 10000000 ... 00000001
  - Training:
    - Backpropagation, forcing output to be the same as input



Mitchell 1999

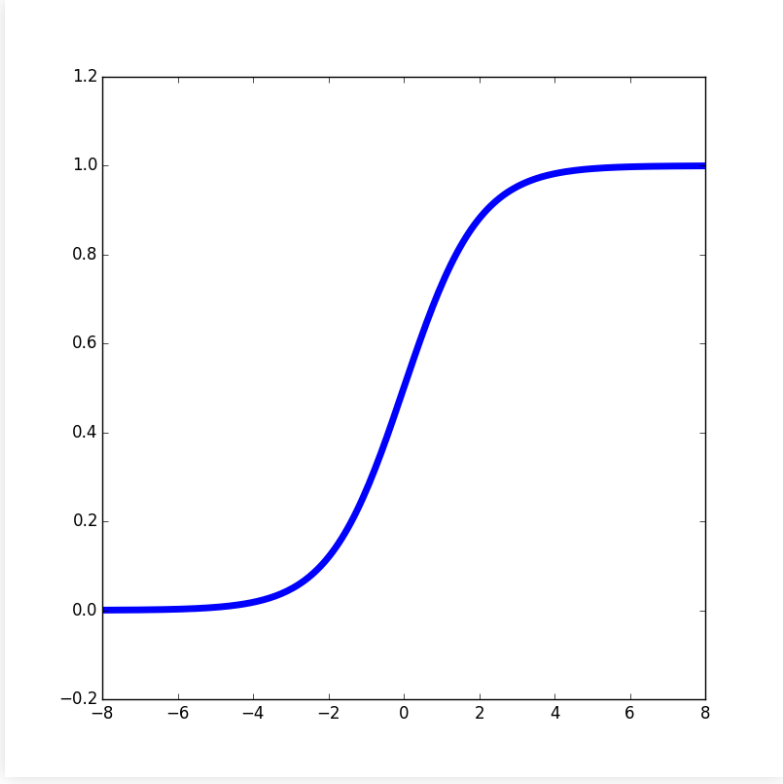
Autoassociator (autoencoder)




```
10000000:      1 0 1
01000000:      0 1 0
00100000:      1 1 1
00010000:      1 0 0
00001000:      0 1 1
00000100:      0 0 0
00000010:      0 0 1
00000001:      1 1 0
```

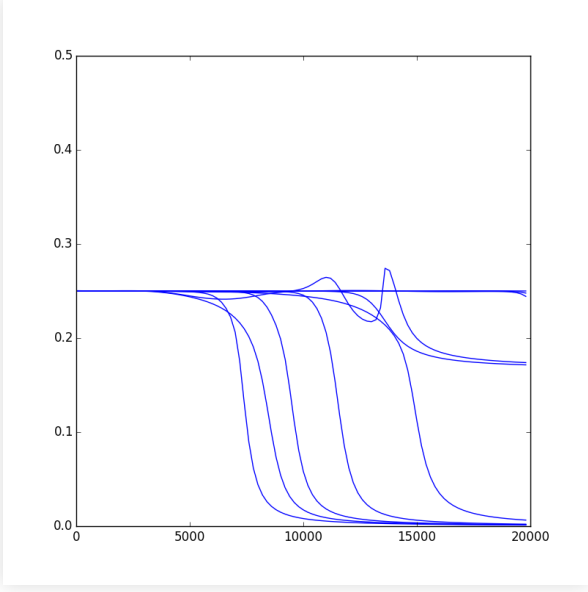
MLP, in practice

- Start with random values close to zero (sigmoidal function saturates rapidly away from zero)



MLP, in practice

- Start with random values close to zero (sigmoidal function saturates rapidly away from zero)
- Since weight initialization and order of examples is random, expect different runs to converge at different epochs



MLP, in practice

- Start with random values close to zero (sigmoidal function saturates rapidly away from zero)
- Since weight initialization and order of examples is random, expect different runs to converge at different epochs
- Standardize or normalize the inputs

- (keep scaling factors for classification)

- Since there is only one η for all dimensions, all values should be on same scale
- Also to avoid saturating the sigmoidal function
- The logistic function ouputs in [0,1], so use these class values
- For multiple classes, use multiple output neurons

MLP, in practice

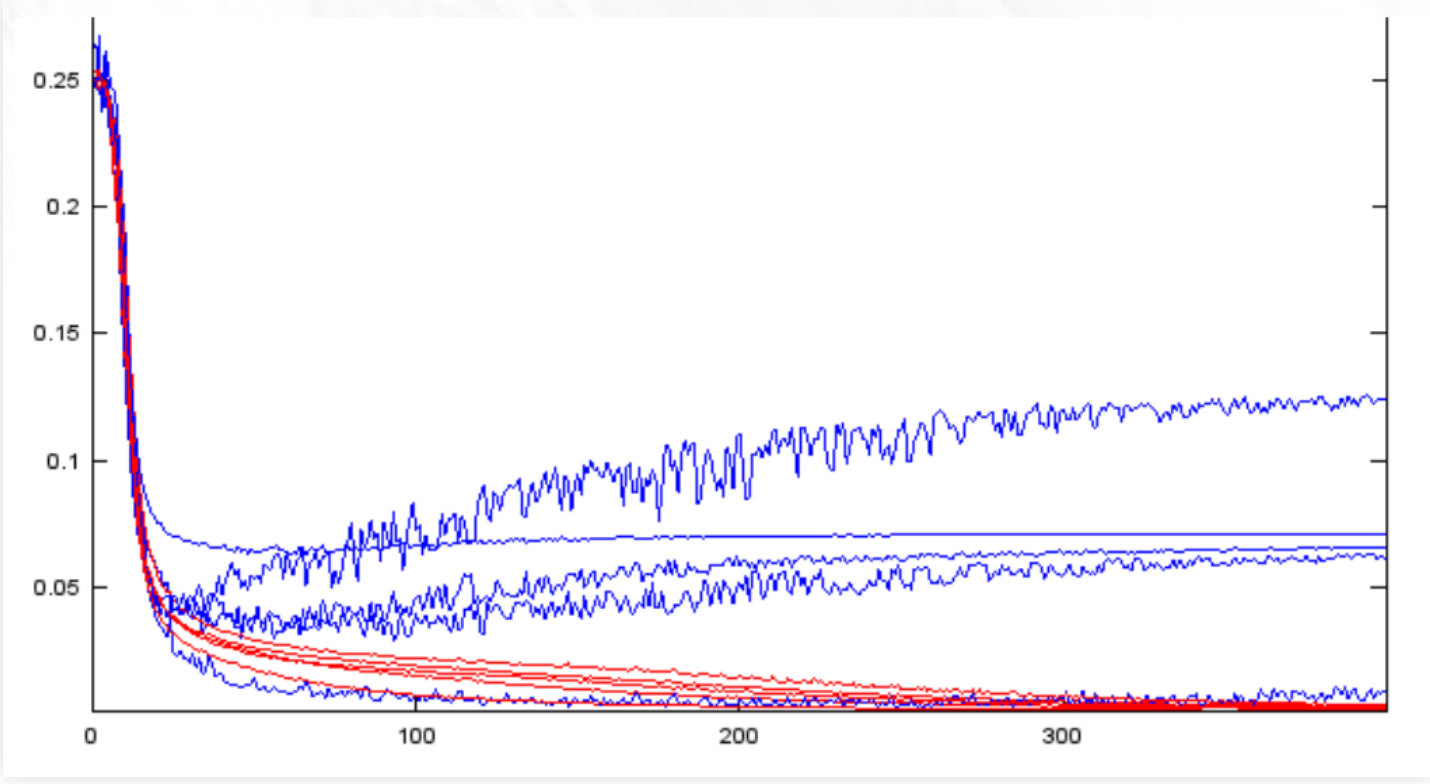
- Stochastic Gradient Descent

- Present training examples in random orderr

- One pass through all examples is one epoch
- Evaluate the error, $(t - s)^2$, for training or cross-validation
- Repeat until convergence or before overfitting (cross-validation)

Regularization with early stop

- E.g. 5-fold cross-validation, stop around 40 epochs



Regularization by weight decay

- Decrease weight values by a small fraction

$$\Delta w_j = -\eta \frac{\delta E}{\delta w_j} - \lambda w_j$$

- Useless parameters will tend to zero

Summary

Multilayer Perceptron

Summary

- Perceptron, classical and continuous output function
- Stochastic gradient descent (squared error)
- One layer vs several layers (multilayer perceptron)
- Training with backpropagation
- Regularization: early stop and weight decay

Further reading

- Mitchell, Chapter 4
- Alpaydin, Chapter 11 (note: linear output)
- Marsland, Chapter 3
- (Bishop, Chapter 5)