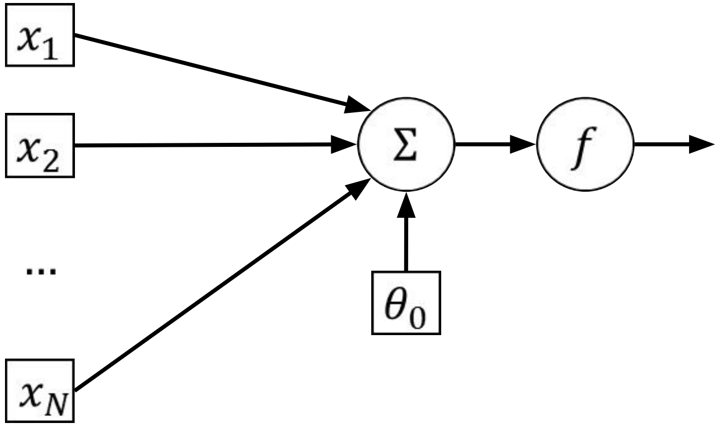
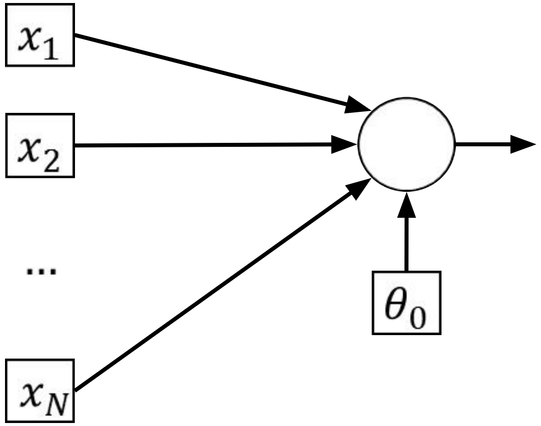
B.1 Basic definitions: Perceptron, MLP

Perceptron (McCulloch-Pitts neuron)

Perceptron has very limited learning capacity. For example, cannot learn XOR.





$$\hat{y} = f(\theta^T \mathbf{x})$$

$$f(x) = \begin{cases} 1 \; if \; x > 0 \\ 0 \; if \; x < 0 \end{cases}$$ (Heaviside function)

Problem of linearly separable two-class classification task

$\theta^T \mathbf{x} > 0$, if $\mathbf{x}$ is of class 0

$\theta^T \mathbf{x} < 0$, if $\mathbf{x}$ is of class 1

Training procedure (perceptron rule)

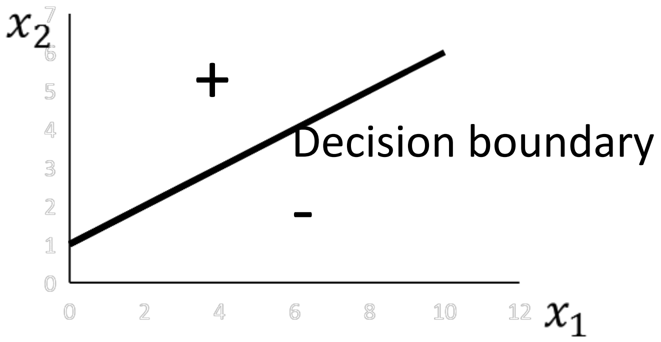
$$\theta^{(i)} = \theta^{(i-1)} + \alpha_i \sum_{\mathbf{x}_n \in M} y_n \mathbf{x}_n$$

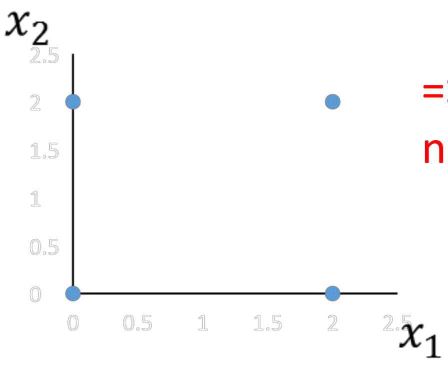
M – misclassified examples

B.1 Basic definitions: Perceptron, MLP

Perceptron (McCulloch-Pitts neuron)

$$\hat{y} = f(\theta^T \mathbf{x})$$
$$\dim(\mathbf{x}) = 2$$





=> Problem: How to deal with non linearly separable problems?

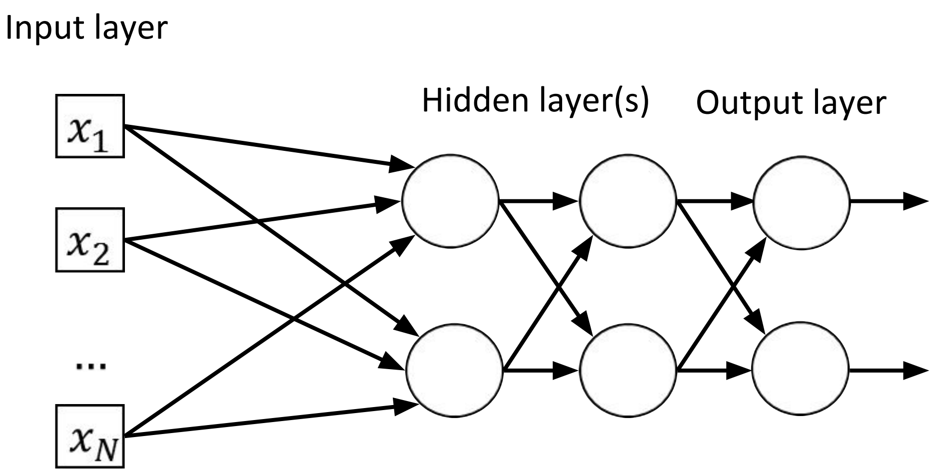
Let's make better representation of input data:

1) Convert to another feature space.

2) Perform linear classification.

B.1 Basic definitions: Perceptron, MLP

Feed-forward multilayer neural network (MLP)

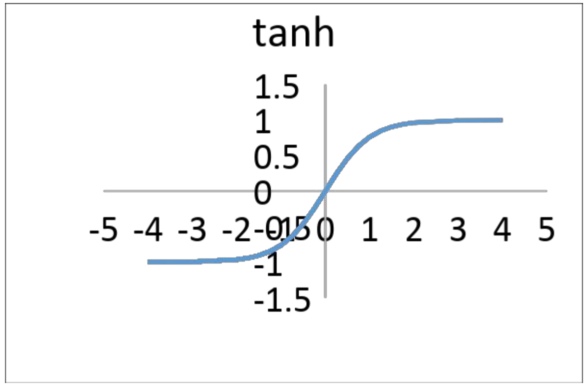


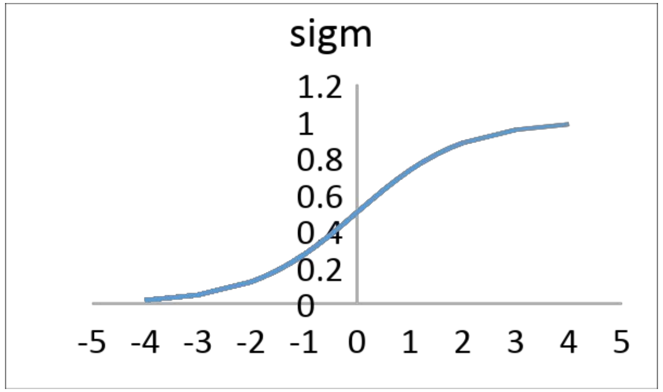
Logistic regression models stacked on top of each other with the final layer being regression / classification model.

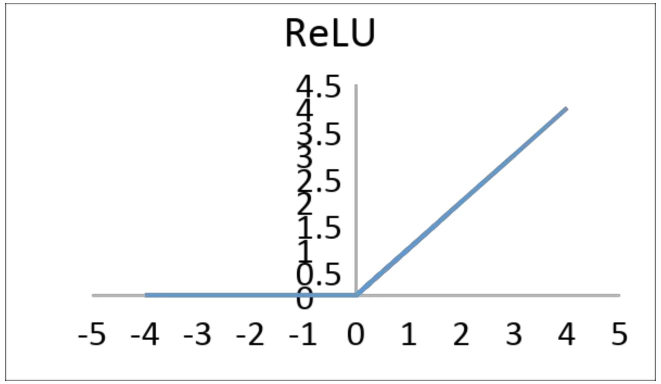
$$\tanh(x) = \frac{\exp(2x) - 1}{\exp(2x) + 1} \qquad \mathrm{sigm}(x) = \frac{1}{1 + \exp(-x)}$$

$$\mathrm{ReLU}(x) = \max(0, x)$$

faster than exp!

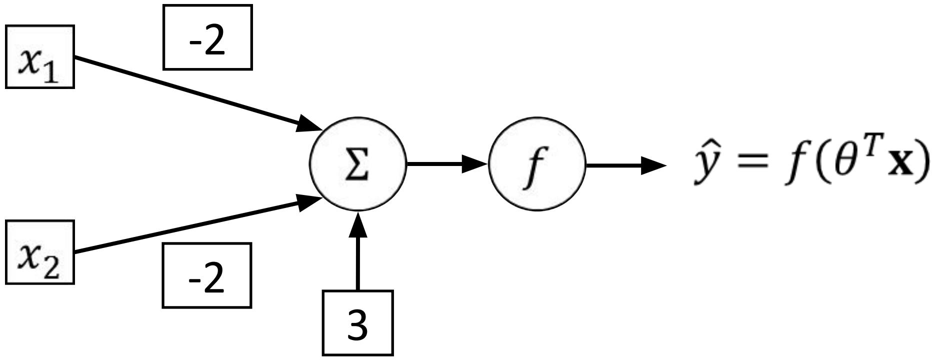






B.2 Basic definitions: Universality of NNs

Perceptron making NAND gate



X	Y	X|Y
0	0	1
0	1	1
1	0	1
1	1	0

Provides basis for the rest of boolean functions of two variables (NAND gate is universal).

=> We can use combination of perceptrons to calculate any boolean function.

B.2 Basic definitions: Universality of NNs

- **MLP can approximate any function**

Consider NN with one hidden layer (K neurons), single output neuron and activation f:

$$\mathrm{OUT}(\mathbf{x}) = \sum_{k=1}^{K} c_k f(\theta^T \mathbf{x}) + c_0 \quad (1)$$

$f(z)$ is non-constant, bounded and monotonically-increasing function.

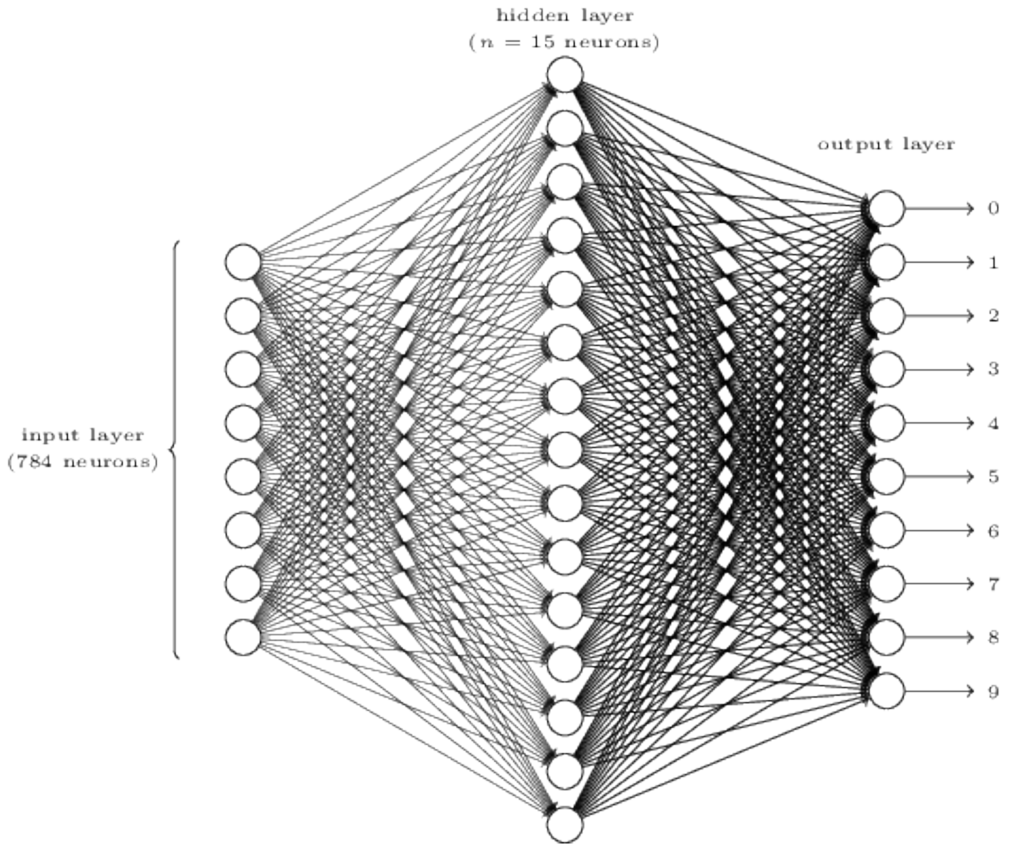
Theorem:

Let $g(\mathbf{x})$ be a continuous function defined in a compact subset $\mathbf{S} \subset \mathbf{R}^n$ and any $\varepsilon > 0$. Then there is a two layer network with $\mathrm{K}(\varepsilon)$ hidden nodes of the form (1), so that:

$$|g(\mathbf{x}) - \mathrm{OUT}(\mathbf{x})| < \varepsilon \; \forall \, \mathbf{x} \in \mathbf{S} \quad (2)$$

B.3 Basic definitions: NN structure for MNIST

Using MLP for solving MNIST task



Why 10 neurons in output layer?

Viable options:

a) 1 neuron (regression)
b) 4 neurons (binary code)
c) 10 neurons (for each numeral)

Softmax activation function for output layer:

$$\hat{y}_{nk} = \frac{\exp(z_{nk})}{\sum_{k=1}^{d} \exp(z_{nk})}$$

* Picture from http://neuralnetworksanddeeplearning.com/chap1.html

D.1 Training NNs: Overview and SGD

Gradient descent:

$$\theta^{(i)} = \theta^{(i-1)} - \alpha_i \nabla_\theta L\big(Y, f(\mathbf{X}, \theta)\big)$$

Stochastic gradient descent algorithm (SGD):

SGD updates parameters after each example (online learning):

1. Initialize θ
2. For each training example $(\mathrm{y}_n, \mathbf{x}_n)$ do (= one epoch):

$$\theta \leftarrow \theta - \alpha \nabla_\theta L\big(y_n, f(\mathbf{x}_n, \theta)\big)$$

3. Repeat until stop criterion is met

D.2 Training NNs: Backpropagation algorithm

- **Naïve approach: let's find derivatives numerically!**

Much much slower.. Full forward pass for each parameter is needed.

$\frac{\partial L}{\partial \omega_i} \approx \frac{L(\omega_i+\varepsilon)-L(\omega_i)}{\varepsilon}$ and same for each parameter (millions of them).

Backpropagation algorithm (BP):

Idea: let's use chain rule in order to calculate derivatives (θ is a vector!):

$$\frac{\partial}{\partial \theta} L\big(y_n, f(\mathbf{x}_{\boldsymbol{n}}, \theta)\big) = \frac{\partial L(y_n, f)}{\partial f} \frac{\partial f(\mathbf{x}_{\boldsymbol{n}}, \theta)}{\partial \theta}$$

EASY! HARD..

D.2 Training NNs: Backpropagation algorithm

- **Backpropagation algorithm (BP) – notations**:

ω_{jk}^{l} - weight for the connection from the k^{th} neuron in the $(l-1)^{th}$ layer to the j^{th} neuron in the l^{th} layer.

b_j^l - bias for j^{th} neuron in the l^{th} layer.

a_j^l - activation of the j^{th} neuron in the l^{th} layer.

$$a_j^l = \sigma\left(\sum_k \omega_{jk}^{l} a_k^{l-1} + b_j^l\right)$$

*image from neuralnetworksanddeeplearning.com/chap2.html

D.2 Training NNs: Backpropagation algorithm

- **Backpropagation algorithm (BP) – matrix notations**:

Same in matrix form:

$\omega_{jk}^{l} \rightarrow \omega^{l}$ - weight matrix for the l^{th} layer

$b_{j}^{l} \rightarrow b^{l}$ - bias vector for the l^{th} layer.

$a_{j}^{l} \rightarrow a^{l}$ - activation vector for the l^{th} layer.

$z^{l} = \omega^{l} \, a^{l-1} + b^{l}$ (weighted input)

$a^{l} = \sigma(z^{l})$ (element-wise)

=> Use ext. libraries for fast matrix calculation.

*image from neuralnetworksanddeeplearning.com/chap2.html

D.2 Training NNs: Backpropagation algorithm

- **Backpropagation algorithm (BP) – easy part**:

$a^L(\mathbf{x}_n) = f(\mathbf{x}_n, \theta)$ – activation of neurons of L^{th} layer.

And let's find derivative of loss function first:

$$\frac{\partial L(y_n, f)}{\partial f} = \frac{1}{2N}\frac{\partial}{\partial f}\sum_{n=1}^{N}(f(\mathbf{x}_n, \theta) - y_n)^2 = \frac{1}{2N}\frac{\partial}{\partial a^L}\sum_{n=1}^{N}\left(a^L(\mathbf{x}_n) - y_n\right)^2 =$$
$$= \frac{1}{N}\sum_{n=1}^{N}\left(a^L(\mathbf{x}_n) - y_n\right)$$

Definition

Hadamar product of matrices:
$(A \circ B)_{ij} = A_{ij}B_{ij}$

D.2 Training NNs: Backpropagation algorithm

- **Backpropagation algorithm (BP) – formulas**:

More complex task: Calculating $\frac{\partial a^L}{\partial \theta}$, where θ are weights ω_{jk}^l and biases b_j^l.

Let's introduce auxiliary quantities:

$\delta_j^l \equiv \frac{\partial L}{\partial z_j^l}$ (error on l^{th} layer)

$$\sigma(x) = \frac{1}{1+\exp(-x)}$$

$$\sigma'(x) = \sigma(x)(1-\sigma(x))$$

Equation #1:

$\delta_j^L = \frac{\partial L}{\partial a_j^L}\sigma'(z_j^L)$ or in matrix notations: $\delta^L = \frac{\partial L}{\partial a^L} \circ \sigma'(z^L)$

May cause slow learning if neuron is "saturated" (activation is close to 0 or 1)

Proof: just apply chain rule.

D.2 Training NNs: Backpropagation algorithm

- **Backpropagation algorithm (BP) – formulas**:

Equation #2:

$\delta_j^l = \sum_k \omega_{kj}^{l+1} \delta_k^{l+1} \, \sigma'\left(z_j^l\right)$ or in matrix notations: $\delta^l = \left(\left(\omega^{l+1}\right)^T \delta^{l+1}\right) \circ \sigma'\left(z^l\right)$

Proof:

$$\delta_j^l \equiv \frac{\partial L}{\partial z_j^l} = \sum_k \frac{\partial L}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l} = \sum_k \frac{\partial z_k^{l+1}}{\partial z_j^l} \delta_k^{l+1} = \sum_k \omega_{kj}^{l+1} \sigma'(z_j^l) \delta_k^{l+1}$$

Use definition of z_j^l to make the last transition:

$z^l = \omega^l \, a^{l-1} + b^l$

D.2 Training NNs: Backpropagation algorithm

- **Backpropagation algorithm (BP) – formulas**:

Equation #3:

$\frac{\partial L}{\partial b_j^l} = \delta_j^l$ or in matrix notations: $\frac{\partial L}{\partial b^l} = \delta^l$

Proof:

$$\frac{\partial L}{\partial b_j^l} = \frac{\partial L}{\partial z_j^l}\frac{\partial z_j^l}{\partial b_j^l} = \delta_j^l * 1 = \delta_j^l$$

Equation #4:

$\frac{\partial L}{\partial \omega_{jk}^l} = a_k^{l-1}\delta_j^l$ or in matrix notations: $\frac{\partial L}{\partial \omega^l} = a^{l-1}\delta^l$

Proof:

$$\frac{\partial L}{\partial \omega_{jk}^l} = \frac{\partial L}{\partial z_j^l}\frac{\partial z_j^l}{\partial \omega_{jk}^l} = a_k^{l-1}\delta_j^l$$

D.2 Training NNs: Backpropagation algorithm

- **Backpropagation algorithm (BP) – pseudocode 1**:

Given: input $\mathbf{x}$

Forward pass:

$z^1 \; = \omega^1 \, \mathbf{x} + b^1$ for $l = 1$

$a^1 \; = \sigma(z^1)$ for $l = 1$

$z^l \; = \omega^l \; a^{l-1} + b^l$ for $l > 1$

$a^l \; = \sigma(z^l)$ for $l > 1$ till $l = L$

D.2 Training NNs: Backpropagation algorithm

- **Backpropagation algorithm (BP) – pseudocode 2**:

Given: input $\mathbf{x}$

Backward pass:

$\delta^L = \frac{\partial L}{\partial a^L} \circ \sigma'(z^L)$ for $l = L$

$\delta^l = \left((\omega^{l+1})^T \delta^{l+1} \right) \circ \sigma'(z^l)$ for $l < L$

Using derivatives calculated above:

$\frac{\partial L}{\partial b^l} = \delta^l$ biases

$\frac{\partial L}{\partial \omega^l} = a^{l-1} \delta^l$ weights