Многослойные нейронные сети

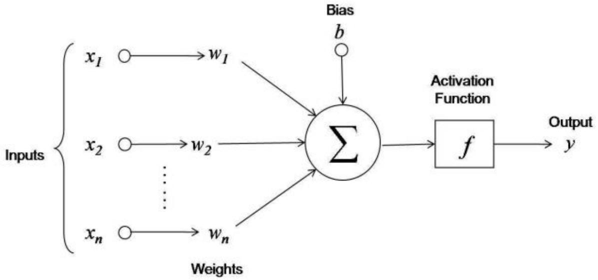
А.Г. Трофимов
к.т.н., доцент, НИЯУ МИФИ

lab@neuroinfo.ru
http://datalearning.ru

Курс "Нейронные сети"

Март 2018

Artificial Neuron Model



Inputs: $x_1, ..., x_n$

Parameters: weights (synaptic coefficients) $w_1, ..., w_n$ and bias b

Activation function: f (transfer function)

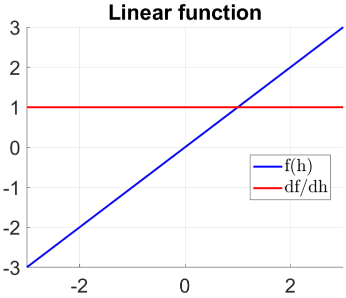
Activation: $h = \sum_{j=1}^{n} w_j x_j - b$

Output: $y = f(h)$

Types of Activation Functions

- Linear function

$$f(h) = h, \quad \frac{df}{dh} = 1$$



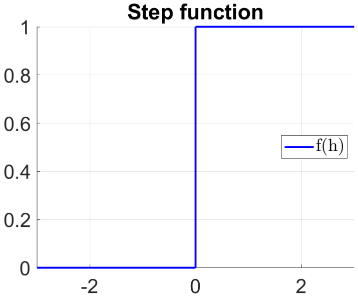
Neuron's output is a linear combination of its inputs:

$$y = w_1 x_1 + \ldots + w_n x_n - b$$

Types of Activation Functions

- Step function

$$f(h) = \begin{cases} 1, & h > 0 \\ 0, & otherwise \end{cases}$$



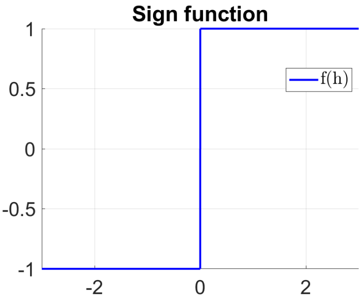
The derivative is 0 for all $h \neq 0$

Neuron's output is binary: all possible values are 0 or 1

Types of Activation Functions

- Sign function

$$f(h) = \begin{cases} 1, & h > 0 \\ -1, & otherwise \end{cases}$$



The derivative is 0 for all $h \neq 0$

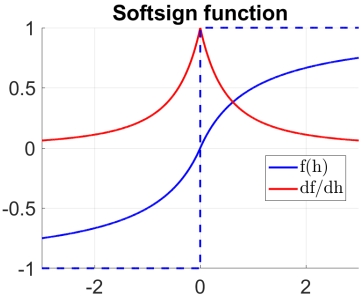
Neuron's output is bipolar: all possible values are −1 or 1

Both step function and sign function are threshold functions

Types of Activation Functions

- Softsign function

$$f(h) = \frac{h}{1 + |h|}, \quad \frac{df}{dh} = \frac{1}{(1 + |h|)^2}$$

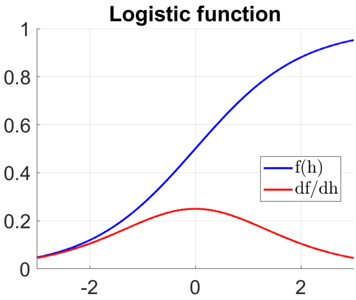


Softsign is a smooth approximation of hard sign function

Types of Activation Functions

- Logistic function

$$f(h) = \frac{1}{1+e^{-h}}, \quad \frac{df}{dh} = f(h)(1-f(h))$$

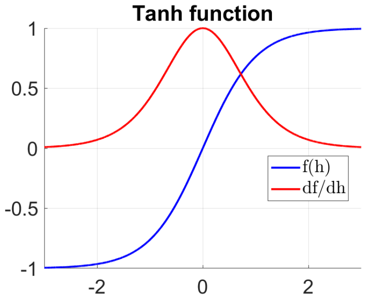


Neuron's output is in range from 0 to 1

Types of Activation Functions

- Tanh function

$$f(h) = \frac{e^h - e^{-h}}{e^h + e^{-h}}, \quad \frac{df}{dh} = 1 - f^2(h)$$



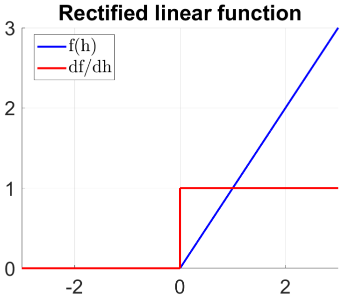
Tanh is a hyperbolic tangent function — shifted and scaled version of logistic function

Softsign, logistic and tanh function are sigmoid functions

Types of Activation Functions

- Rectified linear function

$$f(h) = \begin{cases} h, & h > 0 \\ 0, & otherwise \end{cases} \qquad \frac{df}{dh} = \begin{cases} 1, & h > 0 \\ 0, & otherwise \end{cases}$$



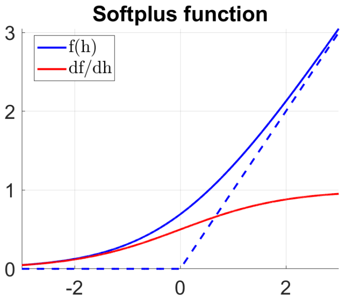
The derivative is a step function

Neuron's output is non-negative

Types of Activation Functions

- Softplus function

$$f(h) = \ln(1 + e^h), \quad \frac{df}{dh} = \frac{1}{1 + e^{-h}}$$

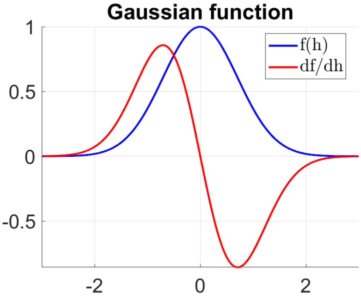


Softplus is a smooth approximation of rectified linear function
The derivative is a logistic function
Neuron's output is non-negative

Types of Activation Functions

- Gaussian function

$$f(h) = e^{-h^2}, \quad \frac{df}{dh} = -2he^{-h^2}$$



Gaussian is a symmetric function with maximum at zero activation

Neuron's output is non-negative

Multilayer Perceptron

In neural networks the outputs of some neurons are transferred to the inputs of other neurons

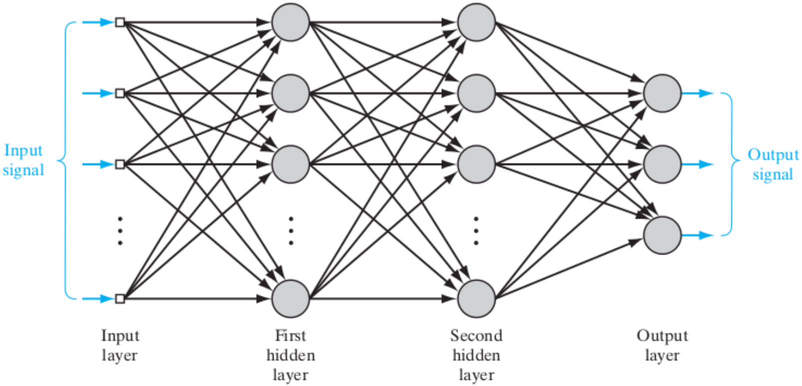
Networks without cycles (feedback loops) are called Feed-forward neural networks (FFNN)

The input signal in FFNN propagates in one direction — from input neurons to output neurons

Multilayer perceptron (MLP) is a special case of FFNN architecture

The input signal in MLP propagates from input neurons to output neurons in layer-by-layer mode

Multilayer Perceptron. Illustration



The architecture consists of: input layer, hidden layers, output layer

Mathematical Model

Definitions:

M is a number of network's inputs

L is a number of layers

K is a number of network's outputs

N_l is a number of neurons in l-th layer, $l = 0, ..., L$

$h^l = (h^l_1, ..., h^l_{N_l})^T$ is the l-th layer's activation, $l = 1, ..., L$

$y^l = (y^l_1, ..., y^l_{N_l})^T$ is the l-th layer's output, $l = 0, ..., L$

f_l is an activation function of neurons in l-th layer, $l = 1, ..., L$

Usually all neurons within layer have the same activation function

Inputs and outputs:

network's inputs is equal to the input layer size: $M \equiv N_0$

network's outputs is equal to the output layer size: $K \equiv N_L$

$x = y^0 = (x_1, ..., x_M)^T$ is network's input

$y = y^K = (y_1, ..., y_K)^T$ is network's output

Mathematical Model

Network's parameters:

Synaptic matrix of l-th layer, $l = 1, ..., L$:

$$W^l = \begin{pmatrix} w_{11}^l & ... & w_{1,N_{l-1}}^l \\ ... & ... & ... \\ w_{N_l,1}^l & ... & w_{N_l,N_{l-1}}^l \end{pmatrix}$$

Vector of biases of l-th layer, $l = 1, ..., L$:

$$b^l = (b_1^l, ..., b_{N_l}^l)^T$$

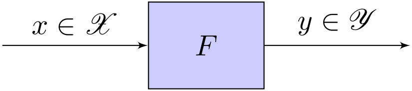
Mathematical model:

$$\begin{cases} h^l = W^l y^{l-1} - b^l \\ y^l = f_l(h^l) \end{cases} \qquad l = 1, ..., L$$

MLP as a Mapping

The MLP can be viewed as a mapping from input domain $\mathscr{X}$ to output domain $\mathscr{Y}$:

$$F : \mathscr{X} \to \mathscr{Y}$$



This mapping is characterized by MLP's parameters: synaptic coefficients and biases of MLP's neurons

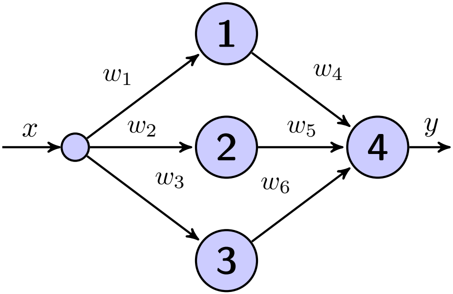
Let's vectorize all parameters and denote them as w:

$$w = vec\left(W^1, ..., W^L, b^1, ..., b^L\right)$$

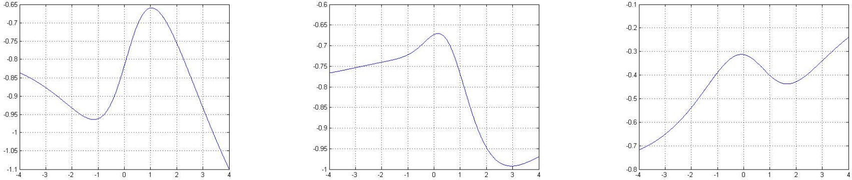
The network's output:

$$y = F(x; w)$$

Example. Two-Layered Network



Plots $y(x)$ at different parameters w:



Universal Approximation Theorem (UAT)

Theorem (Cybenko, 1989)

Let $\varphi(\cdot)$ be a nonconstant, bounded, and monotonically increasing continuous function. Then, given any function $f(x_1, ..., x_M)$, continuous on the M-dimensional unit hypercube $[0, 1]^M$, there exist an integer N and sets of real constants α_i, b_i and w_{ij}, $i = 1, ..., N$, $j = 1, ..., M$, such that we may define

$$F(x_1, ..., x_M) = \sum_{i=1}^{N} \alpha_i \varphi \left(\sum_{j=1}^{M} w_{ij} x_j + b_i \right)$$

as approximate realization of the function $f(x_1, ..., x_M)$:

$$|F(x_1, ..., x_M) - f(x_1, ..., x_M)| < \varepsilon$$

for all $\varepsilon > 0$ and for all $x_1, ..., x_M$ from unit hypercube $[0, 1]^M$

Neural Interpretation of UAT

The universal approximation theorem is directly applicable to multilayer perceptrons

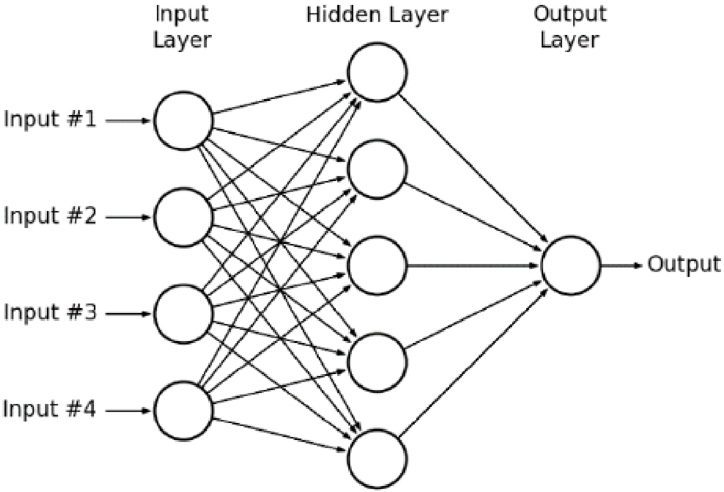
Suppose:

$x_1, ..., x_M$ — the MLP's inputs
N — the number of neurons in MLP's hidden layer
w_{ij} — the weights of hidden layer, $i = 1, ..., N$, $j = 1, ..., M$
b_i — the biases of hidden layer, $i = 1, ..., N$
α_i — the weights of output neuron, $i = 1, ..., N$
$\varphi(\cdot)$ — activation function of hidden layer (e.g. sigmiod)
Activation function of output neuron is linear

Then:

$F(x_1, ..., x_M)$ represents the output of a multilayer perceptron with one hidden layer

Neural Interpretation of UAT. Illustration



UAT and MLP

The universal approximation theorem states that a single hidden layer is sufficient for a multilayer perceptron to compute an approximation to a function represented by a set of observations:

$$\left(x_1^{(1)}, ..., x_M^{(1)}; \sigma^{(1)}\right) \qquad \sigma^{(1)} = f(x^{(1)}), \quad x^{(1)} = (x_1^{(1)}, ..., x_M^{(1)})$$

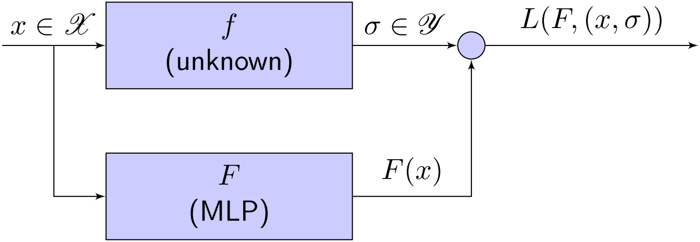
$$...$$

$$\left(x_1^{(n)}, ..., x_M^{(n)}; \sigma^{(n)}\right) \qquad \sigma^{(n)} = f(x^{(n)}), \quad x^{(n)} = (x_1^{(n)}, ..., x_M^{(n)})$$

But the theorem does not say that a single hidden layer is an optimal approximation (in the sense of searching the unknown parameters, ease of implementation, etc.)

The UAT is an existence theorem and almost useless in practise

How to Build MLP?

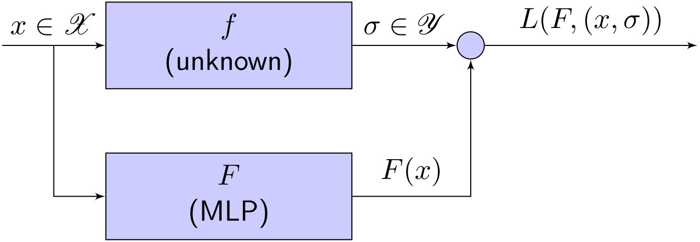


How to choose number of hidden layers, number of neurons, activation functions and the parameters of neurons (synaptic coefficients and biases)?

Neural models are build in a data-driven manner

If we want to build a model we need a some accuracy measure. Accuracy measure must represent the concordance between the model and the object (or process) under modelling

Loss Function



Definition

Loss function (cost function) $L(F, (x, \sigma)) \in \mathbb{R}^+$ is some measure of predictive inaccuracy of model F at $(x, \sigma) \in \mathscr{X} \times \mathscr{Y}$

When comparing the same type of loss among many models, lower loss indicates a better model

The best value: $L(F, (x, \sigma)) = 0$ (means no error on $x \in \mathscr{X}$)

MLP Training Problem

Given:

$\mathscr{D} = \{(x^{(1)}, \sigma^{(1)}), \ldots, (x^{(n)}, \sigma^{(n)})\}$ — available data sample

$x^{(i)} = \left(x_1^{(i)}, \ldots, x_M^{(i)}\right)^T$ — i-th input vector, $i = 1, \ldots, n$

$\sigma^{(i)} = \left(\sigma_1^{(i)}, \ldots, \sigma_K^{(i)}\right)^T$ — i-th target vector, $i = 1, \ldots, n$

Problem:

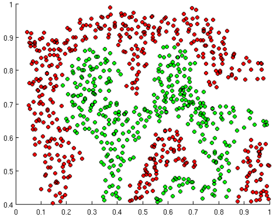
The training of MLP is the minimization of mean loss over data sample $\mathscr{D}$:

$$E(w) = \frac{1}{n} \sum_{i=1}^{n} L\left(F, (x^{(i)}, \sigma^{(i)})\right) \to \min_w$$

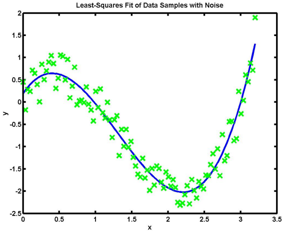
The training of MLP is a kind of optimization problem with objective function $E(w)$. To resolve it optimization techniques are used

Types of Problems

- Classification
  $f(x)$ is discrete
  $\mathscr{D} = \{(x^{(1)}, \sigma^{(1)}), ..., (x^{(n)}, \sigma^{(n)})\}$
  $x^{(i)} \in \mathscr{X}$ is i-th sample, $i = \overline{1, n}$
  $\sigma^{(i)} \in \mathscr{Y}$ is label of $x^{(i)}$
  $\mathscr{Y} = \{1, ..., K\}$ is a set of class labels



- Regression
  $f(x)$ is continuous
  $\mathscr{D} = \{(x^{(1)}, \sigma^{(1)}), ..., (x^{(n)}, \sigma^{(n)})\}$
  $x^{(i)} \in \mathscr{X}$ is i-th sample, $i = \overline{1, n}$
  $\sigma^{(i)} \in \mathscr{Y}$ is response for $x^{(i)}$
  $\mathscr{Y} = \mathbb{R}^K$ is a set of responses



Different tasks impose different loss functions

Types of Loss Functions

- Quadratic loss

$$L(F,(x,\sigma)) = ||F(x)-\sigma||^2$$

Commonly used for regression tasks (σ is continuous)

- Binary cross-entropy loss

$$L(F,(x,\sigma)) = -\left(\sigma \ln F(x) + (1-\sigma)\ln(1-F(x))\right)$$

Commonly used for binary classification tasks ($\sigma \in \{0,1\}$)

- Multinomial (categorical) cross-entropy loss

$$L(F,(x,\sigma)) = -\sum_{k=1}^{K} \sigma_k \ln F_k(x)$$

where $\sigma = (\sigma_1, ..., \sigma_K)^T$, $F(x) = (F_1(x), ..., F_K(x))^T$
Commonly used for multiclass classification tasks (σ is one-hot encoded class label)

Quadratic Loss Function

Given:
$\mathscr{D} = \{(x^{(1)}, \sigma^{(1)}), ..., (x^{(n)}, \sigma^{(n)})\}$ — available data sample

Quadratic loss function:

$$L(F, (x, \sigma)) = ||F(x) - \sigma||^2$$

Mean loss over data sample $\mathscr{D}$:

$$E(w) = \frac{1}{n} \sum_{i=1}^{n} L\left(F, (x^{(i)}, \sigma^{(i)})\right) = \frac{1}{n} \sum_{i=1}^{n} \left|\left|y^{(i)} - \sigma^{(i)}\right|\right|^2$$

The mean loss E for quadratic loss function is called mean squared error (MSE)

The vector of parameters w can be estimated using well-known least squares method (LSM)

Binary Classification. Statistical Model of Classes

Given:

$\mathscr{D} = \{(x^{(1)}, \sigma^{(1)}), ..., (x^{(n)}, \sigma^{(n)})\}$ — available data sample

$\sigma^{(i)} \in \{0, 1\}$ — class labels, $i = 1, ..., n$

Statistical model:

Assume that $\sigma^{(i)}$ is drawn from Bernoulli distribution:

$S_i \sim B(1, p(x^{(i)}, w))$, where $p(x^{(i)}, w) = P(S_i = 1|x^{(i)}, w)$

$P(S_i = 0|x^{(i)}, w) = 1 - p(x^{(i)}, w)$

$$P(S_i = k|x^{(i)}, w) = p(x^{(i)}, w)^k (1 - p(x^{(i)}, w))^{1-k}, \quad k \in \{0, 1\}$$

The statistical model is characterized by unknown vector of parameters w

Given the data sample $\mathscr{D}$ the vector w can be estimated using the well-known statistical maximum likelihood method (MLE)

Maximum Likelihood Estimation

The sample likelihood:

$$\mathscr{L}(\sigma^{(1)}, ..., \sigma^{(n)}, w) = \prod_{i=1}^{n} p(x^{(i)}, w)^{\sigma^{(i)}} (1 - p(x^{(i)}, w))^{1-\sigma^{(i)}} \to \max_{w}$$

Negative log-likelihood:

$$E(w) = -\sum_{i=1}^{n} \left(\sigma^{(i)} \ln p(x^{(i)}, w) + (1 - \sigma^{(i)}) \ln(1 - p(x^{(i)}, w)) \right) \to \min_{w}$$

$$E(w) = \sum_{i=1}^{n} H\left(\sigma^{(i)}, p(x^{(i)}, w)\right) \to \min_{w}$$

where $H\left(\sigma^{(i)}, p(x^{(i)}, w)\right)$ is a cross-entropy between distributions $B(1, \sigma^{(i)})$ and $B(1, p(x^{(i)}, w))$ (binary cross-entropy)

Cross-Entropy

Definition

Cross-entropy between distributions p and q is defined as follows:

$$H(p, q) = H(p) + D_{KL}(p||q)$$

where $H(p)$ is the entropy of p, $D_{KL}(p||q)$ is the Kullback–Leibler divergence of q from p (the relative entropy of p with respect to q)

For discrete case:

$$H(p) = -\sum p_j \log p_j, \qquad D_{KL}(p||q) = -\sum p_j \log \frac{q_j}{p_j}$$

$$H(p, q) = -\sum p_j \log q_j$$

The sum is over all possible values of distributions p and q

MLE Problem and MLP Training

Let's the MLP's output $y(x^{(i)}, w)$ to be in range from 0 to 1 (it can be achieved by using the logistic activation function for the output neuron)

Then the MLP's output $y(x^{(i)}, w)$ can be interpreted as a probability $p(x^{(i)}, w)$ of the class label 1 for input vector $x^{(i)}$:

$$y(x^{(i)}, w) = p(x^{(i)}, w) = P(S_i = 1 | x^{(i)}, w)$$

Then, maximum likelihood estimation problem

$$E(w) = -\sum_{i=1}^{n} \Big(\sigma^{(i)} \ln y(x^{(i)}, w) + (1 - \sigma^{(i)}) \ln(1 - y(x^{(i)}, w)) \Big) \to \min_{w}$$

and the MLP training problem with binary cross-entropy loss function for binary classification are identical problems

Multiclass Classification. Statistical Model of Classes

Given:

$\mathscr{D} = \{(x^{(1)}, \sigma^{(1)}), ..., (x^{(n)}, \sigma^{(n)})\}$ — available data sample

$\sigma^{(i)} \in \{1, ..., K\}$ — class labels, $i = 1, ..., n$

Statistical model:

Assume that $\sigma^{(i)}$ is drawn from multinomial distribution:

$$S_i \sim Mult(1, p_1(x^{(i)}, w), ..., p_K(x^{(i)}, w))$$

where $p_k(x^{(i)}, w) = P(S_i = k | x^{(i)}, w), \quad k \in \{1, ..., K\}$

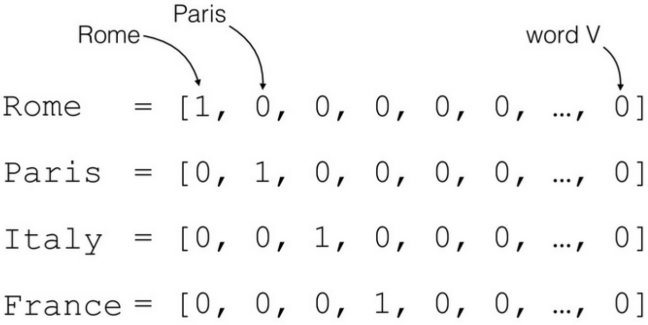
Let's re-label: $\sigma^{(i)} := \left(\sigma_1^{(i)}, ..., \sigma_K^{(i)}\right), \quad \sigma_k^{(i)} = \begin{cases} 1, & \sigma^{(i)} = k \\ 0, & otherwise \end{cases}$

So $\sigma^{(i)}$ is a binary vector that contains one 1 at k-th position, other elements are 0, $i = 1, ..., n$ (one-hot encoded vector)

One-Hot Encoding. Illustration

One-hot encoding is a process by which categorical variable x with K variants is converted into a binary vector y that contains one 1 and other elements are 0, such that

$$x = k \Leftrightarrow y_k = 1, y_i = 0 \ \forall i \neq k, \qquad k = 1, ..., K$$



Multiclass Classification. Statistical Model of Classes

Probabilities:

$$P(S_i = \sigma^{(i)} | x^{(i)}, w) = \prod_{k=1}^{K} \Big(p_k(x^{(i)}, w) \Big)^{\sigma_k^{(i)}}$$

Given the data sample $\mathscr{D}$ the vector of unknown parameters w can be estimated using the maximum likelihood method (MLE)

The sample likelihood:

$$\mathscr{L}(\sigma^{(1)}, ..., \sigma^{(n)}, w) = \prod_{i=1}^{n} \prod_{k=1}^{K} \Big(p_k(x^{(i)}, w) \Big)^{\sigma_k^{(i)}} \to \max_w$$

Negative log-likelihood:

$$E(w) = -\sum_{i=1}^{n} \sum_{k=1}^{K} \sigma_k^{(i)} \ln p_k(x^{(i)}, w) \to \min_w$$

MLE Problem and MLP Training

Let's the MLP's outputs $y_1(x^{(i)}, w), ..., y_K(x^{(i)}, w)$ all to be in range from 0 to 1 and $\sum_{k=1}^{K} y_k(x^{(i)}, w) = 1$ for all $i = 1, ..., n$ (it can be achieved by using the softmax activation function for the output layer)

Then the MLP's output $y_k(x^{(i)}, w)$ can be interpreted as a probability $p_k(x^{(i)}, w)$ of the class label k for input vector $x^{(i)}$:

$$y_k(x^{(i)}, w) = p_k(x^{(i)}, w) = P(S_i = k | x^{(i)}, w), \quad k = 1, ..., K$$

Maximum likelihood estimation problem

$$E(w) = -\sum_{i=1}^{n} \sum_{k=1}^{K} \sigma_k^{(i)} \ln y_k(x^{(i)}, w) \to \min_w$$

and the MLP training problem with multinomial cross-entropy loss function for multiclass classification are identical problems

MLP Training as an Optimization Problem

The MLP training is an optimization problem with the given objective function $E(w)$

How to resolve this problem?

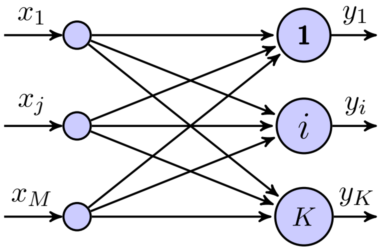
The most popular optimization technique is gradient descent:

$$w(\tau+1) = w(\tau) - \alpha \nabla E(w(\tau))$$

where $\nabla E(w) = \left(\frac{\partial E(w)}{\partial w_1}, ..., \frac{\partial E(w)}{\partial w_m} \right)^T$ is a gradient of objective $E(w)$ at $w = (w_1, ..., w_m)^T$, τ is the iteration

To apply the gradient descent we need to know how to calculate partial derivatives of $E(w)$ with respect to all MLP's adjustable parameters from vector w

Training of Single-Layer Perceptron



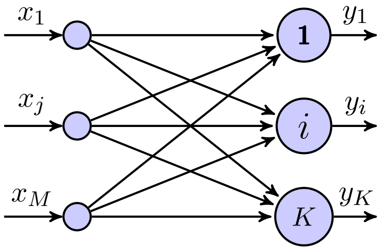
$$\frac{\partial E^{(p)}}{\partial w_{ij}} = \sum_{k=1}^{K} \frac{\partial E^{(p)}}{\partial y_k^{(p)}} \frac{\partial y_k^{(p)}}{\partial w_{ij}},$$

Objective:

$$E(w) = \frac{1}{n} \sum_{p=1}^{n} E^{(p)}(w) \to \min_{w}$$

where $E^{(p)}(w)$ is a loss on sample $(x^{(p)}, \sigma^{(p)})$, $p = 1, ..., n$

Training of Single-Layer Perceptron



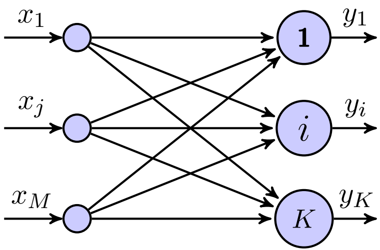
Objective:

$$E(w) = \frac{1}{n}\sum_{p=1}^{n} E^{(p)}(w) \to \min_{w}$$

where $E^{(p)}(w)$ is a loss on sample $(x^{(p)}, \sigma^{(p)})$, $p = 1, \ldots, n$

$$\frac{\partial E^{(p)}}{\partial w_{ij}} = \sum_{k=1}^{K} \frac{\partial E^{(p)}}{\partial y_k^{(p)}} \frac{\partial y_k^{(p)}}{\partial w_{ij}}, \qquad \frac{\partial y_k^{(p)}}{\partial w_{ij}} = f'(h_k^{(p)}) \frac{\partial h_k^{(p)}}{\partial w_{ij}} =$$

Training of Single-Layer Perceptron



Objective:

$$E(w) = \frac{1}{n}\sum_{p=1}^{n} E^{(p)}(w) \to \min_w$$

where $E^{(p)}(w)$ is a loss on sample $(x^{(p)}, \sigma^{(p)})$, $p = 1, ..., n$

$$\frac{\partial E^{(p)}}{\partial w_{ij}} = \sum_{k=1}^{K} \frac{\partial E^{(p)}}{\partial y_k^{(p)}} \frac{\partial y_k^{(p)}}{\partial w_{ij}}, \qquad \frac{\partial y_k^{(p)}}{\partial w_{ij}} = f'(h_k^{(p)}) \frac{\partial h_k^{(p)}}{\partial w_{ij}} = f'(h_k^{(p)}) x_j^{(p)} \delta_{ik}$$

$$\frac{\partial E^{(p)}}{\partial w_{ij}} = \frac{\partial E^{(p)}}{\partial y_i^{(p)}} f'(h_i^{(p)}) x_j^{(p)} = \frac{\partial E^{(p)}}{\partial h_i^{(p)}} x_j^{(p)} = \Delta_i^{(p)} x_j^{(p)}$$

where $\Delta_i^{(p)} = \frac{\partial E^{(p)}}{\partial h_i^{(p)}} = \frac{\partial E^{(p)}}{\partial y_i^{(p)}} f'(h_i^{(p)})$ is called 'dual' activation of i-th neuron, $i = 1, ..., K$

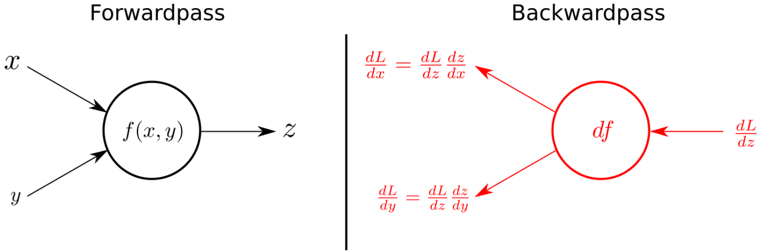
Chain Rule

Chain rule is a formula for computing the derivative of the composition of two or more functions:

if $z = f(y)$ and $y = g(x)$, then

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx} = f'(y)g'(x) = f'(g(x))g'(x)$$

if $z = f(y_1, ..., y_n)$, $y_i = g_i(x_1, ..., x_m)$, then $\displaystyle\frac{\partial z}{\partial x_j} = \sum_{i=1}^{n} \frac{\partial z}{\partial y_i}\frac{\partial y_i}{\partial x_j}$



Backpropagation Equations

Backpropagation (BP) equations are closed-form expressions for partial derivatives of loss function $E^{(p)}(w)$ on the sample $(x^{(p)}, \sigma^{(p)})$, $p = 1, ..., n$, with respect to any synaptic coefficient or bias of MLP neurons (Rumelhart, 1986):

$$\frac{\partial E^{(p)}}{\partial w_{ij}^{l}} = \Delta_i^{(p)l} y_j^{(p),l-1}, \quad l = \overline{1, L}, \quad y^{(p)0} \equiv x^{(p)}$$

$$\Delta_i^{(p)L} = \frac{\partial E^{(p)}}{\partial y_i^{(p)}} f_L'(h_i^{(p)L}), \quad i = \overline{1, K}$$

$$\Delta_i^{(p)l} = \left(\sum_{j=1}^{N_{l+1}} \Delta_j^{(p),l+1} w_{ji}^{l+1} \right) f_l'(h_i^{(p)l}), \quad l = \overline{1, L-1}, \ i = \overline{1, N_l}$$

Backpropagation Algorithm

Step 1. Apply the input vector $x^{(p)}$ from the training set to the network and forward propagate it to obtain the output vector $y^{(p)}$

Step 2. Using the target vector $\sigma^{(p)}$ compute the loss $E^{(p)}$

Step 3. Evaluate 'dual' activations $\Delta_1^{(p)L}, ..., \Delta_K^{(p)L}$ for each output neuron

Step 4. Evaluate 'dual' activations for each hidden neuron using backward propagation

Step 5. Evaluate derivatives the loss $E^{(p)}$ with respect to each adjustable synaptic coefficient and bias

Step 6. Repeat steps 1–6 for each pattern $(x^{(p)}, \sigma^{(p)})$ from the training set

Derivatives of Loss Functions

The backward propagation initiates by the partial derivatives of loss function $\frac{\partial E^{(p)}}{\partial y_1^{(p)}}, \ldots, \frac{\partial E^{(p)}}{\partial y_K^{(p)}}$ with respect to MLP outputs

The derivatives of commonly used loss functions are quite simple

- Quadratic loss

$$E^{(p)} = ||y^{(p)} - \sigma^{(p)}||^2 = \sum_{k=1}^{K} \left(y_k^{(p)} - \sigma_k^{(p)} \right)^2$$

$$\frac{\partial E^{(p)}}{\partial y_k^{(p)}} = 2 \left(y_k^{(p)} - \sigma_k^{(p)} \right)$$

where $\sigma_k^{(p)} \in \mathbb{R}$, $y_k^{(p)} \in \mathbb{R}$, $k = 1, \ldots, K$

Derivatives of Loss Functions

- Binary cross-entropy loss ($K = 1$)

$$E^{(p)} = -(\sigma^{(p)} \ln y^{(p)} + (1 - \sigma^{(p)}) \ln(1 - y^{(p)}))$$

$$\frac{\partial E^{(p)}}{\partial y^{(p)}} = -\frac{\sigma^{(p)}}{y^{(p)}} + \frac{1 - \sigma^{(p)}}{1 - y^{(p)}}$$

where $\sigma^{(p)} \in \{0, 1\}$, $0 < y^{(p)} < 1$

- Multinomial cross-entropy loss

$$E^{(p)} = -\sum_{k=1}^{K} \sigma_k^{(p)} \ln y_k^{(p)}$$

$$\frac{\partial E^{(p)}}{\partial y_k^{(p)}} = -\frac{\sigma_k^{(p)}}{y_k^{(p)}}$$

where $\sigma^{(p)} = (\sigma_1^{(p)}, ..., \sigma_K^{(p)})^T$ is one-hot encoded class label, $0 < y_k^{(p)} < 1$, $k = 1, ..., K$, $\sum_{k=1}^{K} y_k^{(p)} = 1$

Quadratic vs Cross-Entropy Loss

The derivatives for a neuron with sigmoid activation function $f(h)$

For quadratic loss:

$$\frac{\partial E^{(p)}}{\partial w_j} = \frac{\partial E^{(p)}}{\partial y^{(p)}} \frac{\partial y^{(p)}}{\partial h^{(p)}} \frac{\partial h^{(p)}}{\partial w_j} = 2\left(y^{(p)} - \sigma^{(p)}\right) f'(h^{(p)}) x_j^{(p)}$$

For cross-entropy loss:

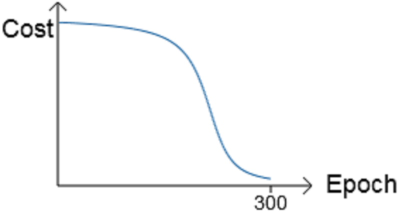
$$\frac{\partial E^{(p)}}{\partial w_j} = \frac{\partial E^{(p)}}{\partial y^{(p)}} \frac{\partial y^{(p)}}{\partial h^{(p)}} \frac{\partial h^{(p)}}{\partial w_j} = \left(-\frac{\sigma^{(p)}}{y^{(p)}} + \frac{1-\sigma^{(p)}}{1-y^{(p)}}\right) f'(h^{(p)}) x_j^{(p)}$$

$$= \frac{y^{(p)} - \sigma^{(p)}}{y^{(p)}(1-y^{(p)})} \left(y^{(p)}(1-y^{(p)})\right) x_j^{(p)} = \left(y^{(p)} - \sigma^{(p)}\right) x_j^{(p)}$$

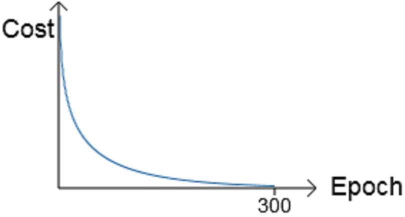
For quadratic loss the partial derivative $\frac{\partial E^{(p)}}{\partial w_j}$ contains $f'(h^{(p)})$ that is close to 0 if neuron's output $y^{(p)}$ is close to targets 0 or 1

Quadratic vs Cross-Entropy Loss. Illustration

Training process
for quadratic loss



Training process
for cross-entropy loss



For cross-entropy loss the partial derivatives $\frac{\partial E^{(p)}}{\partial w_j}$ depend linearly on neuron's error $\left(y^{(p)} - \sigma^{(p)}\right)$: the larger the error, the faster the neuron will learn