SVILUPPO RISORSE PER L'AZIENDA
CISITA
PARMA

Unione Parmense degli Industriali

GIA Gruppo Imprese
Gruppo Imprese Artigiane

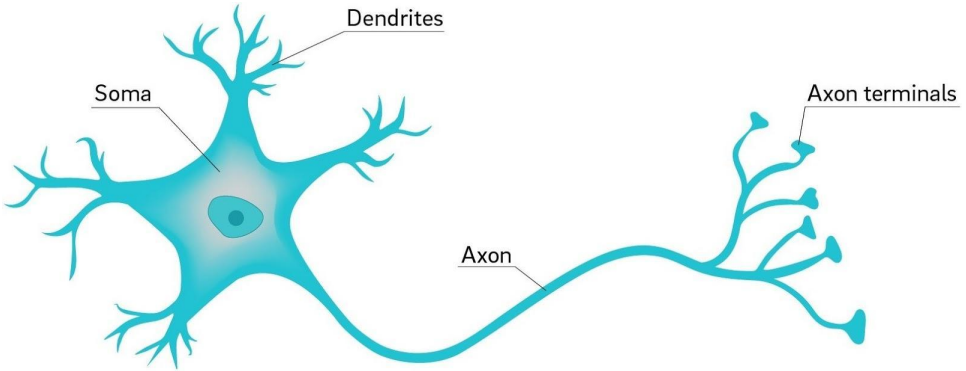
Neural networks

Artificial Intelligence and Machine Learning

Gianfranco Lombardo MEng, Ph.D Candidate in ICT
gianfranco.lombardo@unipr.it

Biological neural networks

- Network of neurons (about 10^{11} in humans)
- Each neuron receives impulses from dendrites
- Soma is excited from these impulses and it propagates a new electric signal through the axon to other neurons



Artificial Neural Network (ANN)

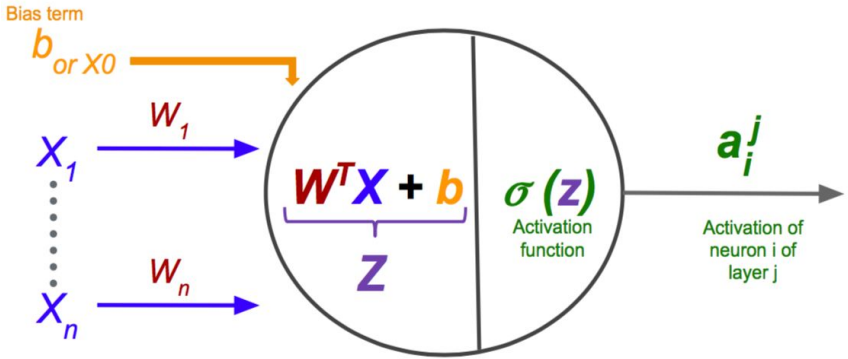
- Artificial Neural Network(ANN): Computational paradigm inspired by a mathematical model of the neuron (McCulloch & Pitts 1943) devised to study the computational abilities of biological neurons and neural networks.

- It takes inspiration from the architecture of human brain for building an intelligent machine.

- Network of nodes (artificial neurons)

Artificial Neuron

- An artificial neuron is a function that maps an *input vector* $\{x_1, \ldots, x_k\}$ to a *scalar output* y via a *weight vector* $\{w_1, \ldots, w_k\}$ and a *function* f (typically non-linear).
- Where the input vector represent the dendrites
- The output scalar value represents the activation of the neuron and the signal propagated over the axon
- Neuron receives all the stimuli (vector X), it computes a weighted sum and then apply an activation function that define a threshold to define the output value



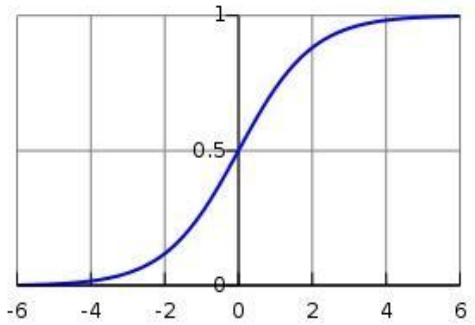
$$a_i^j = f\left(\sum_{i=0}^{K} w_i x_i\right) = f(\mathbf{w}^{\mathbf{T}}\mathbf{x})$$

Activation function

- The *function f* is called the **activation function** and generates a non-linear input/output relationship.
- A common choice for the activation function is the **Logistic function** (or **Sigmoid**).

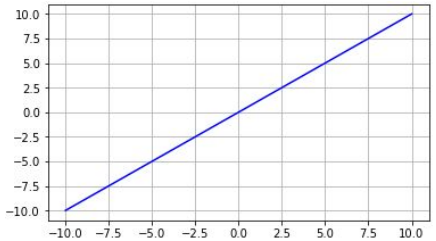
Sigmoid

$$f(\mathbf{w}^{\mathbf{T}}\mathbf{x}) \rightarrow y = \frac{1}{1 + e^{-\mathbf{w}^{\mathbf{T}}\mathbf{x}}}$$



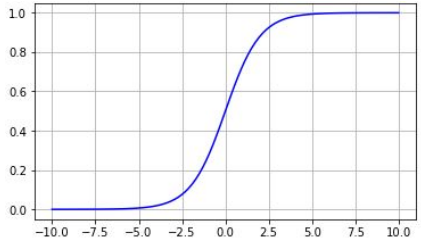
Activation functions in Neural networks

Linear Activation



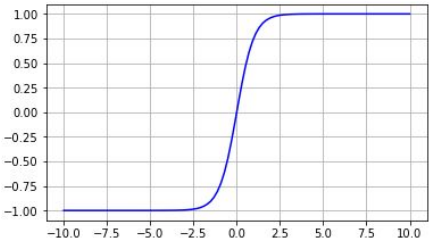
f(u) = u

Sigmoid Activation



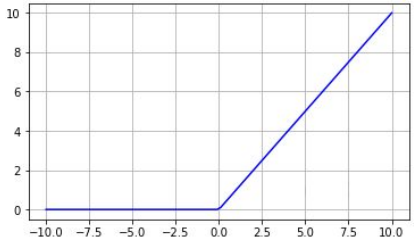
$f(u) = \frac{1}{1+e^{-u}}$

Tanh Activation



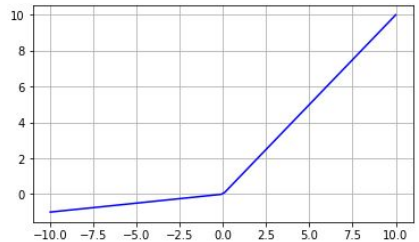
$f(u) = \frac{e^{u}-e^{-u}}{e^{u}+e^{-u}}$

ReLU Activation



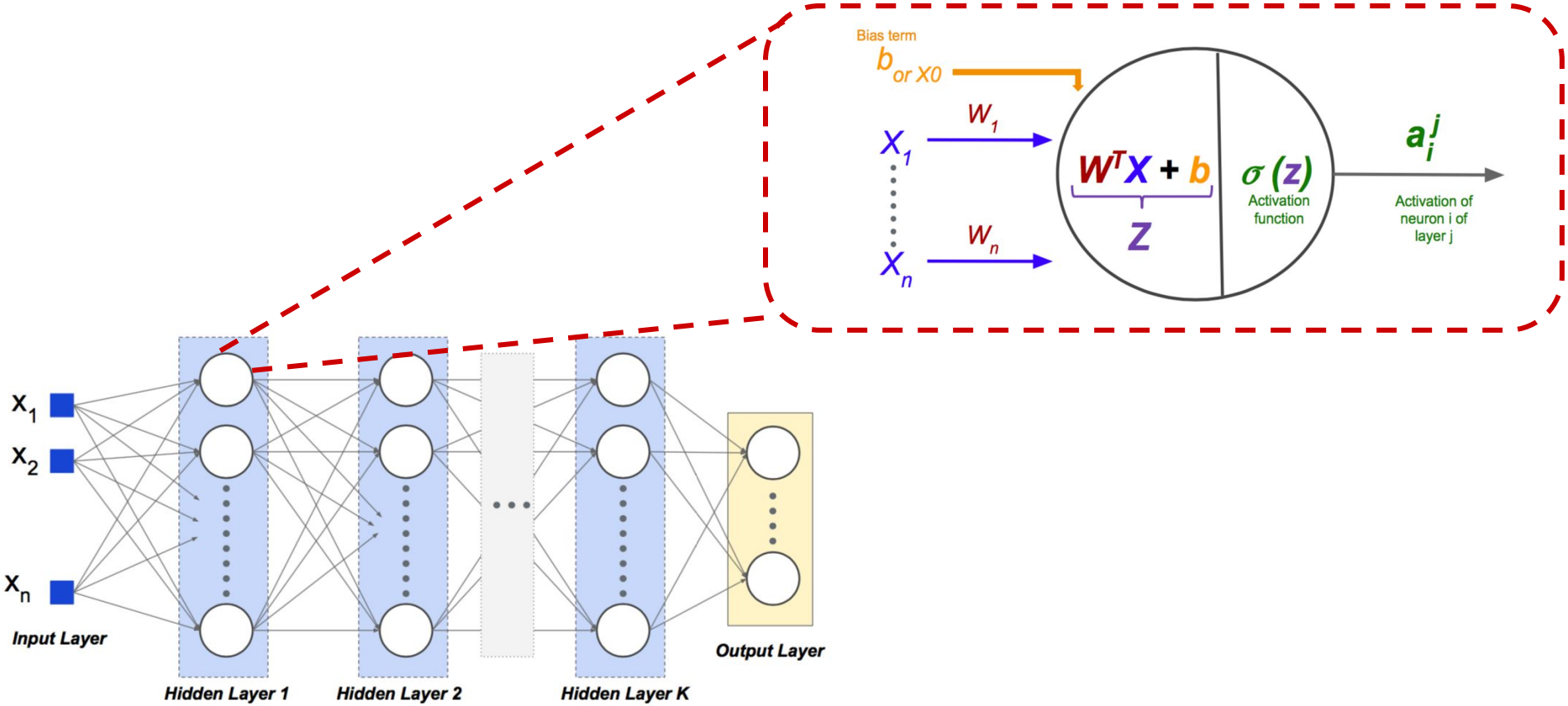
f(u) = max(0,u)

Leaky ReLU Activation

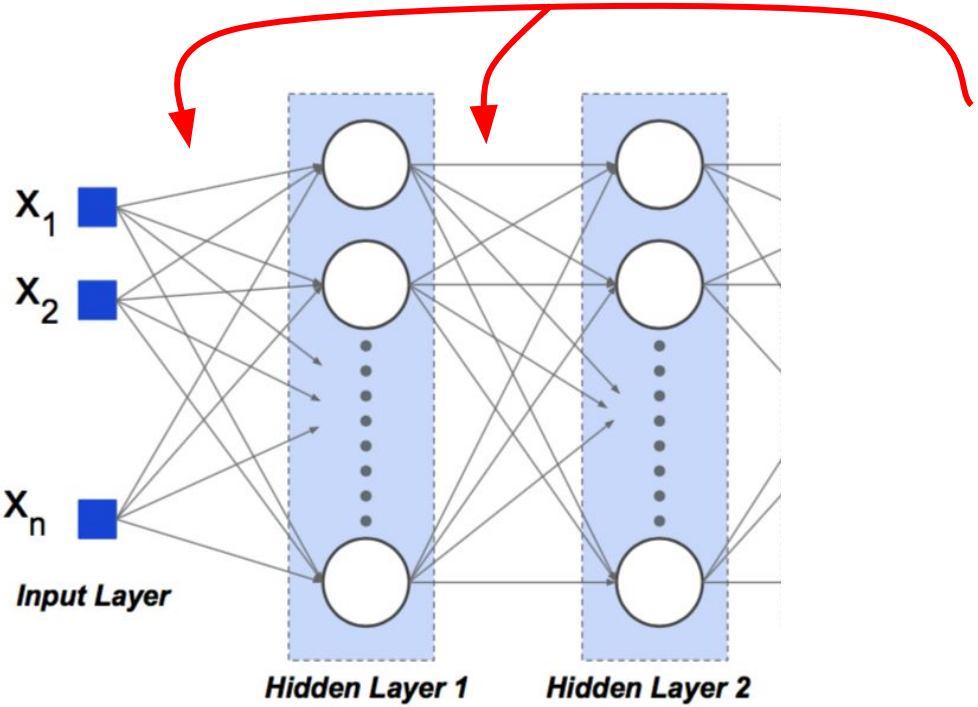


f(u) = max(0.1*u,u)

Artificial Neural Network (ANN)



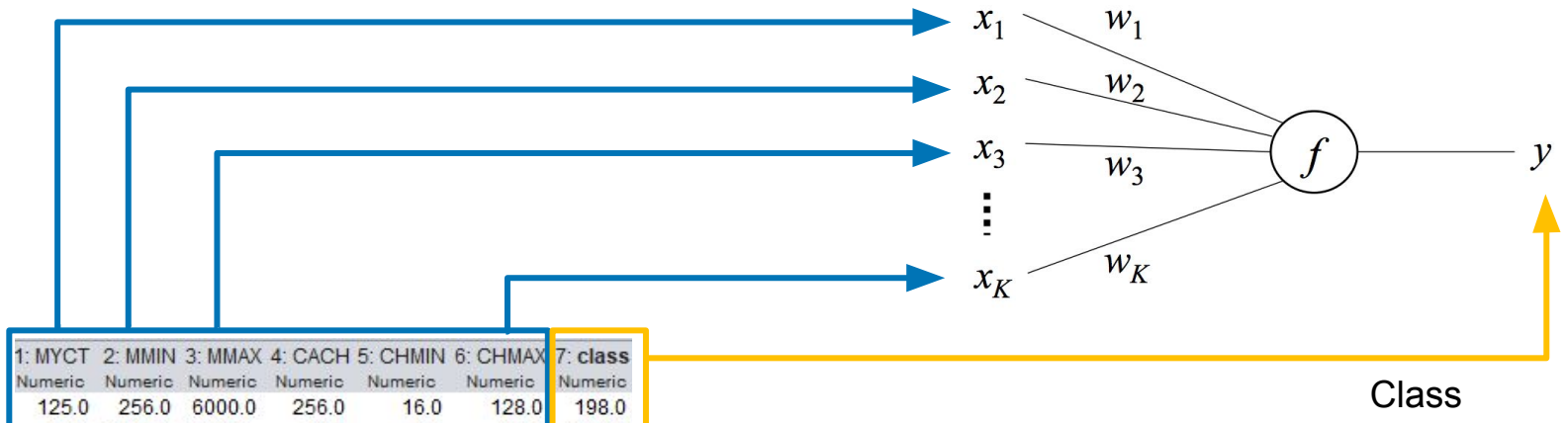
Artificial Neural Network (ANN)



- Each connection between input and neurons, and between neurons and neurons has an associated weight **w**
- Weights are randomly initialized
- ***Our goal during the training step is to learn these weights***
- All weights between two layers are organized in a matrix ***W***

Neural networks: High level learning

Attributes



1: MYCT	2: MMIN	3: MMAX	4: CACH	5: CHMIN	6: CHMAX	7: class
Numeric	Numeric	Numeric	Numeric	Numeric	Numeric	Numeric
125.0	256.0	6000.0	256.0	16.0	128.0	198.0
29.0	8000.0	3200...	32.0	8.0	32.0	269.0
29.0	8000.0	3200...	32.0	8.0	32.0	220.0
29.0	8000.0	3200...	32.0	8.0	32.0	172.0
29.0	8000.0	1600...	32.0	8.0	16.0	132.0
26.0	8000.0	3200...	64.0	8.0	32.0	318.0
23.0	1600...	3200...	64.0	16.0	32.0	367.0
23.0	1600...	3200...	64.0	16.0	32.0	489.0
23.0	1600...	6400...	64.0	16.0	32.0	636.0
23.0	3200...	6400...	128.0	32.0	64.0	1144.0

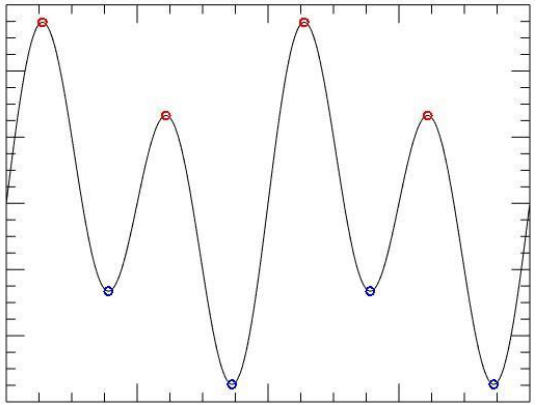
Goal □ Estimate a function $y = f(W, X)$ by learning weights W

Neural networks: Optimization problem

- Our goal is to **minimize** an **objective function**, which measures the difference between the actual output t and the predicted output y.
    - In this case we will consider as the objective function the **squared loss function**.

Squared loss function

$$E = \frac{1}{2}(t - y)^2 = \frac{1}{2}(t - f(\mathbf{w}^\mathbf{T}\mathbf{x}))^2$$



Loss functions

Squared loss function

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

Mean squared error

$$E = \frac{1}{n}\sum_{i=1}^{n}(t_i - y_i)^2$$

Mean absolute error

$$E = \frac{\sum_{i=1}^{n} | t_i - y_i |}{n}$$

Cross entropy

$$E = -\sum_{i=1}^{n} t_i \log(y_i)$$

Kullback Leibler divergence

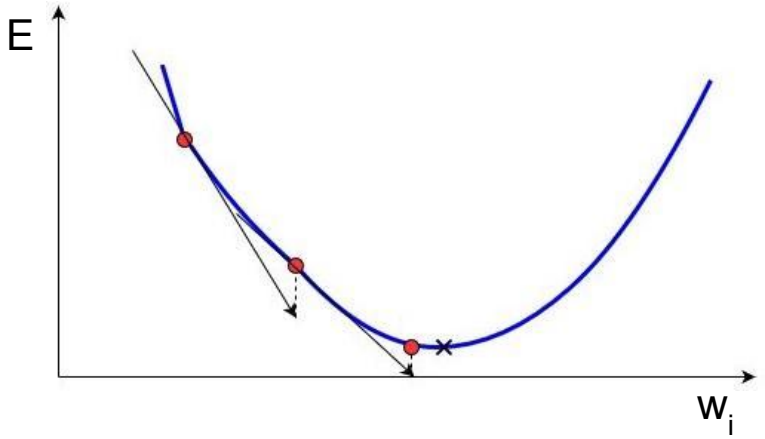
$$E = \sum_{i=1}^{n} t_i \log\left(\frac{t_i}{y_i}\right)$$

Cosine proximity

$$E = \frac{t \ y}{||t|| \ ||y||}$$

Gradient descent

- We want to find the **weights** $\{w_1,...,w_k\}$ such that the objective function is minimized.
- We do this with **Gradient Descent** (**GD**):
  - Iterative optimization algorithm used in machine learning to find the best results (minima of a curve).
  - Compute the gradient of the objective function with respect to an element w_i of the vector $\{w_1,...,w_k\}$.



$$w_i^{new} = w_i^{old} - \eta \frac{\partial E}{\partial w_i}$$

Gradient descent

$$w_i^{new} = w_i^{old} - \eta \frac{\partial E}{\partial w_i}$$

$$\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial u} \cdot \frac{\partial u}{\partial w_i}$$

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

Squared Loss

$$y = \frac{1}{1 + e^{-u}}$$

Sigmoid activation

$$u = w_i \, x_i$$

Gradient descent

$$w_i^{new} = w_i^{old} - \eta \frac{\partial E}{\partial w_i}$$

$$\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial u} \cdot \frac{\partial u}{\partial w_i}$$

$$= (y - t) \cdot y(1 - y) \cdot x_i$$

$$E = \frac{1}{2}(t - y)^2 \longrightarrow \frac{\partial E}{\partial y} = \frac{1}{2}\, 2\,(t - y)(-1) = y - t$$

$$y = \frac{1}{1 + e^{-u}} \longrightarrow \frac{\partial y}{\partial u} = \frac{(-1)(e^{-u})(-1)}{(1 + e^{-u})^2} = \frac{e^{-u}}{(1 + e^{-u})^2} =$$

$$\frac{e^{-u} + 1 - 1}{(1 + e^{-u})^2} = \frac{1 + e^{-u}}{(1 + e^{-u})^2} + \frac{(-1)}{(1 + e^{-u})^2} =$$

$$\frac{1}{(1 + e^{-u})} - \left(\frac{1}{(1 + e^{-u})}\right)^2 = y - y^2 =$$

$$y\,(1 - y)$$

$$u = w_i\, x_i \longrightarrow \frac{\partial u}{\partial w_i} = x_i$$

Gradient descent

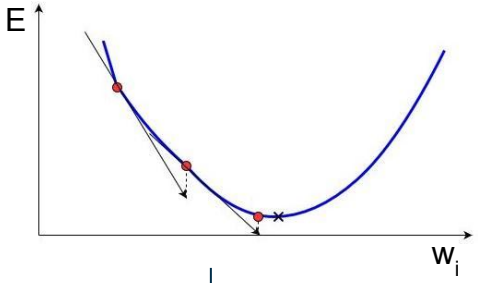
Gradient descent

- Let's update the weights using the gradient descent update equation (in vector notation)

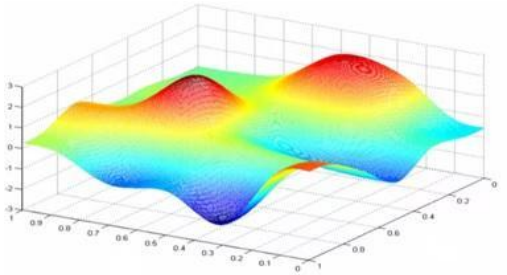
$$w^{new} = w^{old} - \eta \frac{\partial E}{\partial w}$$

$$w^{new} = w^{old} - \eta \, (y - t) \, y \, (1 - y) \, x$$

- η > 0 is the step size □ Learning Rate

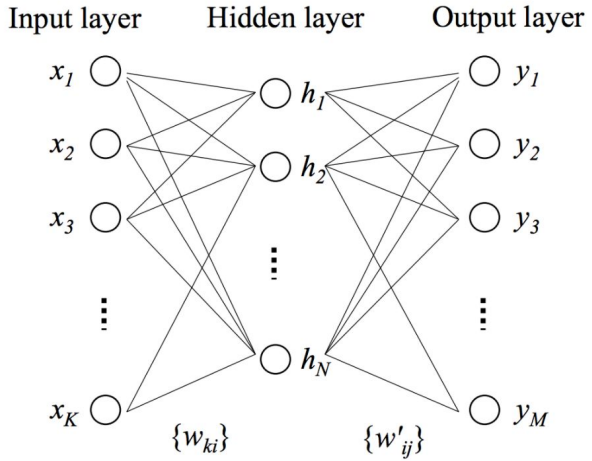


For all the weights



Backpropagation

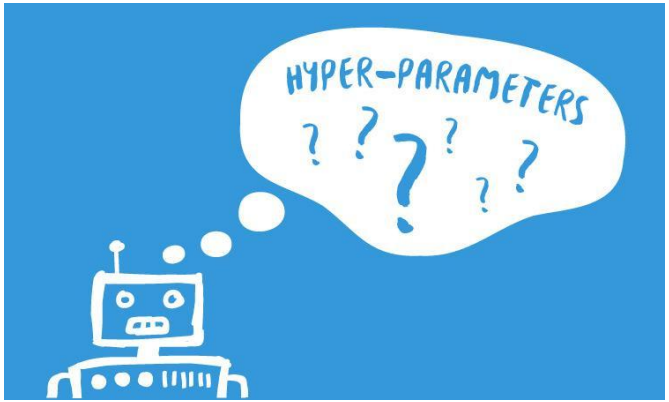
- We start by computing the error (y_j - t_j) for output neuron j. The error gets propagated backwards throughout the network's layers in order to update the weights.
- The gradient of the error with respect to the weights connecting a hidden layer with the next one depend (only) on the gradients of the neurons that are closer to the output layer than it is, which can be computed starting from the output layer and going backwards.



$$w_{ki}^{new} = w_{ki}^{old} - \eta \cdot \sum_{j=1}^{M} [(y_j - t_j) \cdot y_j(1 - y_j) \cdot w'_{ij}] \cdot h_i(1 - h_i) \cdot x_k$$

Hyperparameters

- Hyperparameters are the parameters which determine the **network structure** (e.g. Number of Hidden Units) and the parameters which determine **how** the **network** is **trained** (e.g. Learning Rate)
  - Number of neurons
  - Number of layers
  - Learning rate
  - Batch size
  - Number of epochs
  - others

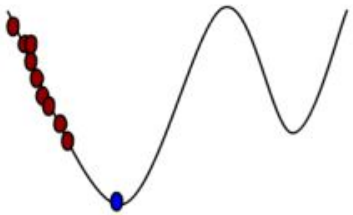


Learning rate

- Training parameter that controls the size of weight changes in the learning phase of the training algorithm.
- The learning rate determines how much an updating step influences the current value of the weights.

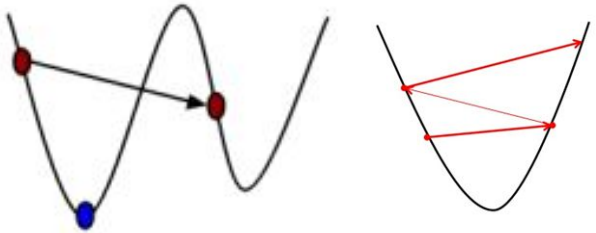
$$w_i^{new} = w_i^{old} - \eta \frac{\partial E}{\partial w_i}$$

Very small learning rate



Many updates required before reaching the minimum.

Too big learning rate



Drastic updates can lead to divergent behaviors, missing the minimum.

Hyperparameters

Number of epochs

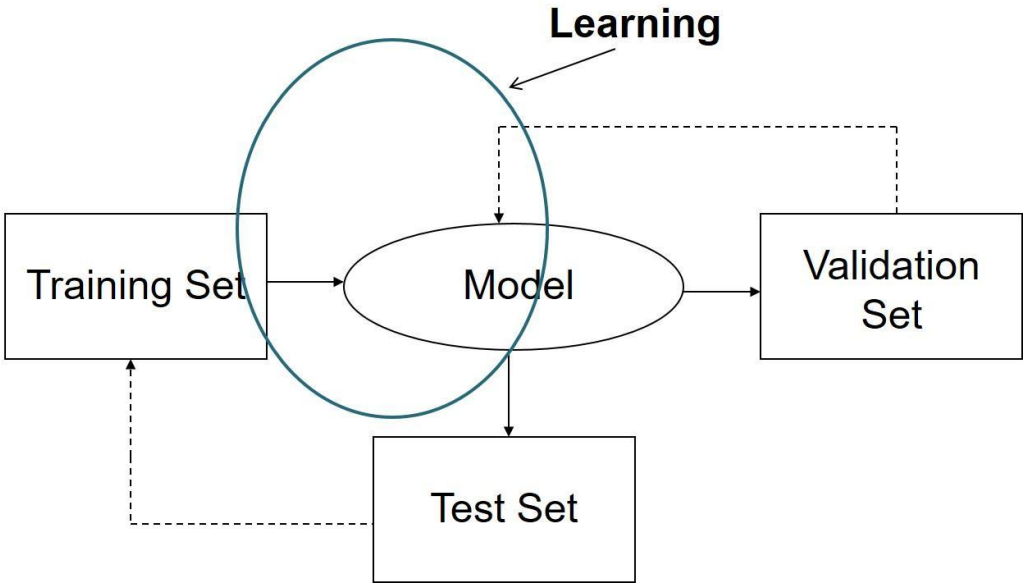
- The number of epochs is the number of times the whole training data is shown to the network while training.

Batch size

- The number of samples shown to the network before the gradient computation and the parameter update.

Validation set

- Data set with the ‘same’ goal of the test set (verifying the quality of the model which has been learnt), but not as a final evaluation, but as a way to fine-tune the model.
- Its aim is to provide a feedback which allows one to find the best settings for the learning algorithm (parameter tuning).



Keras

- Keras is an open-source library that provides tools to develop artificial neural networks
- Keras acts as an interface for the TensorFlow library
- First install TensorFlow: pip install tensorflow
- Then pip install Keras

BREAK

Example: breast cancer classification

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import numpy as np
import matplotlib.pyplot as plt

from keras import models
from keras import layers
```

Example: breast cancer classification

```
X, y = load_breast_cancer(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.20)

print("Training set dimensions (train_data):")
print(X_train.shape)
```

Example: breast cancer classification

```
model = models.Sequential()
#The first layer that you define is the input layer. This
layer needs to know the input dimensions of your data.
# Dense = fully connected layer (each neuron is fully
connected to all neurons in the previous layer)
model.add(layers.Dense(64, activation='relu',
input_shape=(X_train.shape[1],)))
# Add one hidden layer (after the first layer, you don't need
to specify the size of the input anymore)
model.add(layers.Dense(64, activation='relu'))
# If you don't specify anything, no activation is applied (ie.
"linear" activation: a(x) = x)
model.add(layers.Dense(1,activation='sigmoid'))
```

Example: breast cancer classification

```
model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])

# Fit the model to the training data and record events into a
History object.
history = model.fit(X_train, y_train, epochs=10, batch_size=1,
validation_split=0.2, verbose=1)

# Model evaluation
test_loss,test_acc = model.evaluate(X_test,y_test)
print(test_acc)
```

Breast cancer: Plot Loss VS Epochs

```
# Plot loss (y axis) and epochs (x axis) for training set and
validation set
plt.figure()
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.plot(history.epoch,
np.array(history.history['loss']),label='Train loss')
plt.plot(history.epoch,
np.array(history.history['val_loss']),label = 'Val loss')
plt.legend()
plt.show()
```

Setting learning rate and optimizer

```
opt = keras.optimizers.Adam(lr=0.01)
model.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
```

Available optimizers:
https://keras.io/api/optimizers/

Example: Boston regression

```
from sklearn.datasets import load_boston
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
import numpy as np
import matplotlib.pyplot as plt

from keras import models
from keras import layers
```

Example: Boston regression

```
X, y = load_boston(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.20)

print("Training set dimensions (train_data):")
print(X_train.shape)
```

Example: Boston regression

```
model = models.Sequential()
model.add(layers.Dense(64,
activation='relu',input_shape=(X_train.shape[1],)))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(1,activation='relu'))
model.compile(optimizer='rmsprop', loss='mse', metrics=['mse'])

history = model.fit(X_train, y_train, epochs=10, batch_size=1,
validation_split=0.2, verbose=1)
test_loss_score, test_mse_score = model.evaluate(test_data,
test_targets)

# MSE
print(test_mse_score)
```

Exercise

- Starting from the breast cancer example try to report how the accuracy changes when:
  - Increasing number of epochs
  - Increasing the batch size
  - Modify the network architecture: try to add more hidden layers and to build a funnel structure
    - First hidden layer should have 75% of input layer
    - Second: 50%
    - Third: 25%
    - Fourth: 12.5%
    - Five: 6% and so on

- Let’s make a challenge: Who is able to get the best accuracy ?

BREAK

Cars sale prediction

- **Cars dataset**: the task is to predict the value of a potential car sale (i.e. how much a particular person will spend on buying a car) for a customer on the basis of the following attributes: age, gender, average miles driven per day, personal debt, monthly income
- Create a sequential model (loss='mse', optimizer='rmsprop', metrics=['mse'], epochs=150, batch_size=50) adding:
  - An input (“Dense”) layer composed of 12 nodes (input_dim=5, activation=‘relu’)
  - A hidden (“Dense”) layer composed of 8 nodes (activation=‘relu’)
  - An output (“Dense”) layer composed of 1 node
- Plot the mean squared error over the epochs for the training set and for the validation set (validation_spilt=0.2)
- Compute the root mean squared error on the test set

Sonar classification

- **Sonar dataset**: the task is to train a network to discriminate between sonar signals (60 features) bounced off a metal cylinder (class M) and those bounced off a roughly cylindrical rock (class R).
- Create a sequential model (loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'], epochs=100, batch_size=5) adding:
  - An input (“Dense”) layer composed of 60 nodes (input_dim=60, activation=‘relu’)
  - An output (“Dense”) layer composed of 1 node (activation='sigmoid‘)
- Make predictions for the labels of the test set and evaluate the model (accuracy and confusion matrix)