ERHVERVSAKADEMI AARHUS

Machine Learning – Week 5

Sila, Sept 23th 2025.

Welcome - Todays Program.

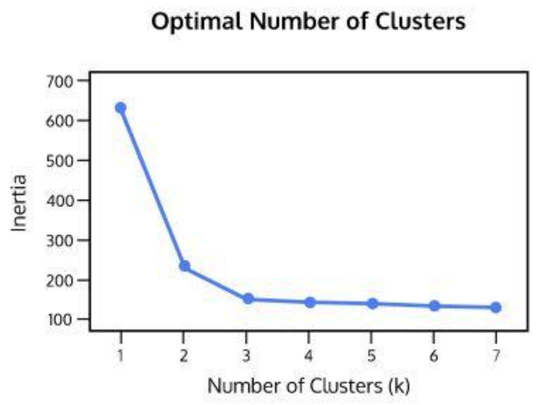
- Comments about last weeks exercises.
- Introduction and History
- Neural Networks & Training
- Applications - some examples from the world of ML
- Exercises

Exercises – Kmeans. Note.

K-Means.

Inertia measures how well a dataset was clustered by K-Means.
It is calculated by measuring the distance between each data point and its centroid, squaring this distance, and summing these squares across one cluster.
A good model is one with low inertia
AND a low number of clusters

To find the optimal K for a dataset, use the Elbow method; find the point where the decrease in inertia begins to slow. K=3 is the "elbow" of this graph.



Inertia measures how well a dataset was clustered by K-Means. It is calculated by measuring the distance between each data point and its centroid, squaring this distance, and summing these squares across one cluster.

Exercises – Last week.

In groups of 2-3.
Discuss:

Decision trees & Random forest.

Did you get it to work?
Discuss: Problems, observations…

- Your solution for the Comedians exercise?
- What was your solution (code) for the circles dataset ?

// Exercise 5 - classification on other data.
Try exercise 3 but on the Circles dataset instead
(must be labeled data of course) from the
article in exercise 1. //

Exercises last week.
Hard and soft voting classifiers.

Discuss in groups.

What is the difference between hard and soft voting classifiers?

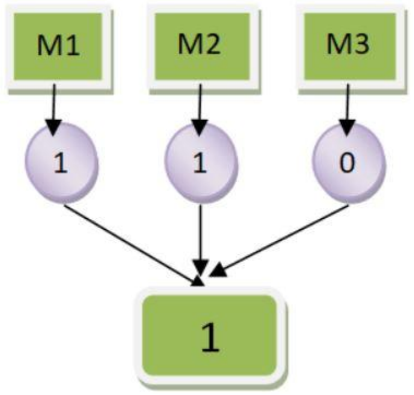
Look it up – If you didn’t get around to it (last week in exercises).

Exercises last week. Hard and soft voting classifiers.

Discuss in groups.

What is the difference between
hard and soft voting classifiers?

In classification problems,
there are two types of voting:
Hard voting and soft voting.
Hard voting entails picking the prediction with
the highest number of votes, whereas soft voting
entails combining the probabilities of each
Prediction in each model and picking the
prediction with the highest total probability.



Hard Voting

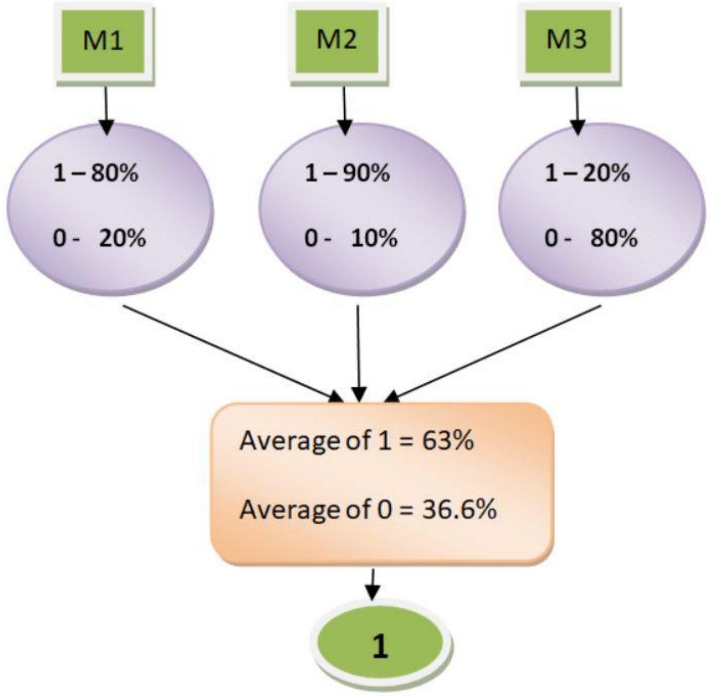
Exercises last week.
Hard and soft voting classifiers.

Discuss in groups.

What is the difference between
hard and soft voting classifiers?

In classification problems,
there are two types of voting:
Hard voting and soft voting.
Hard voting entails picking the prediction with
the highest number of votes, whereas soft voting
entails combining the probabilities of each
Prediction in each model and picking the
prediction with the highest total probability.

Soft:



Dataset – Diabetes. Exercise.

Exercise Diabetes dataset.

See the file
Diabetes_DataSet_Exercise.pdf
(on Canvas).

Machine Learning

Intro to Neural Networks

ERHVERVSAKADEMI AARHUS

Machine Learning – NN everywhere...

An AI generated movie script actually turned into a short movie….

https://youtu.be/LY7x2Ihqjmc

Trained using a special kind of Neural Networks - an article tries to explain the idea behind this:

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

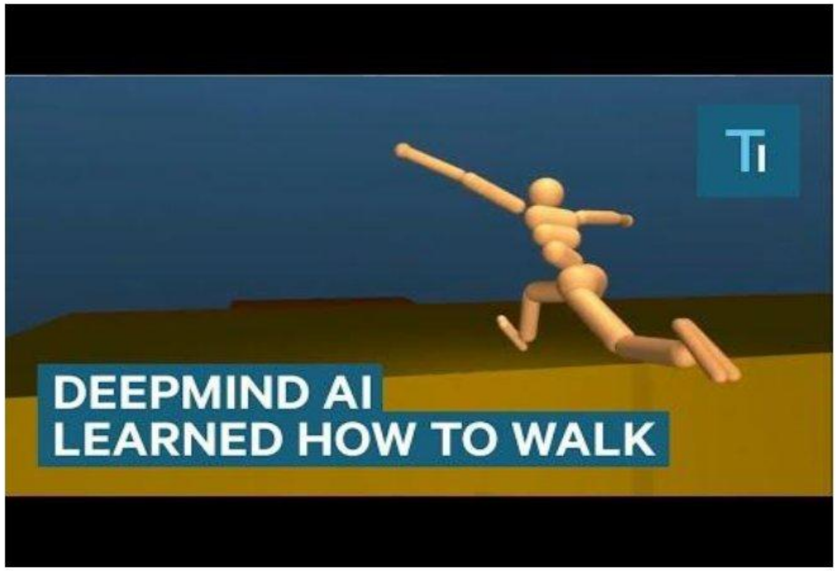
Watching a play written by an AI and reading the subtitles on Youtube generated by yet another one….

Gen Ai - Inspirational video from Veed (March 2024). https://drive.google.com/file/d/16UN6VNPbwtMgQ8JRxWQzAU5Ew8zYgx6N/view?usp=sharing

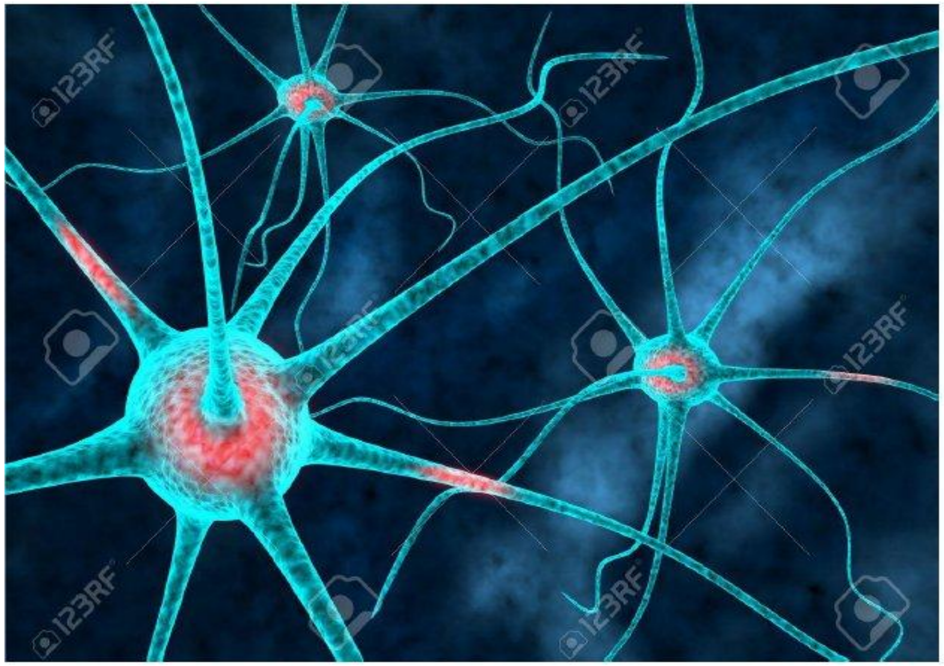
NN Introduction and history

Machine Learning – Intro

ANN Example: Learning to walk

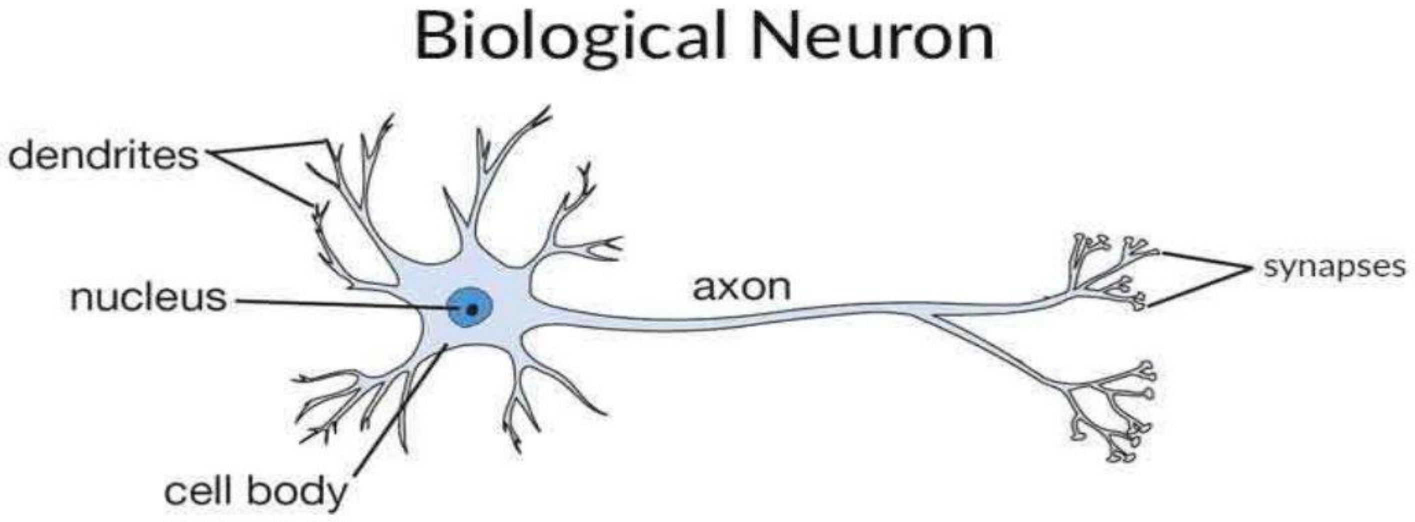


Introduction - inspiration from the brain



Artistic impression of
biological neurons and
their networks.

Neuron - biological model



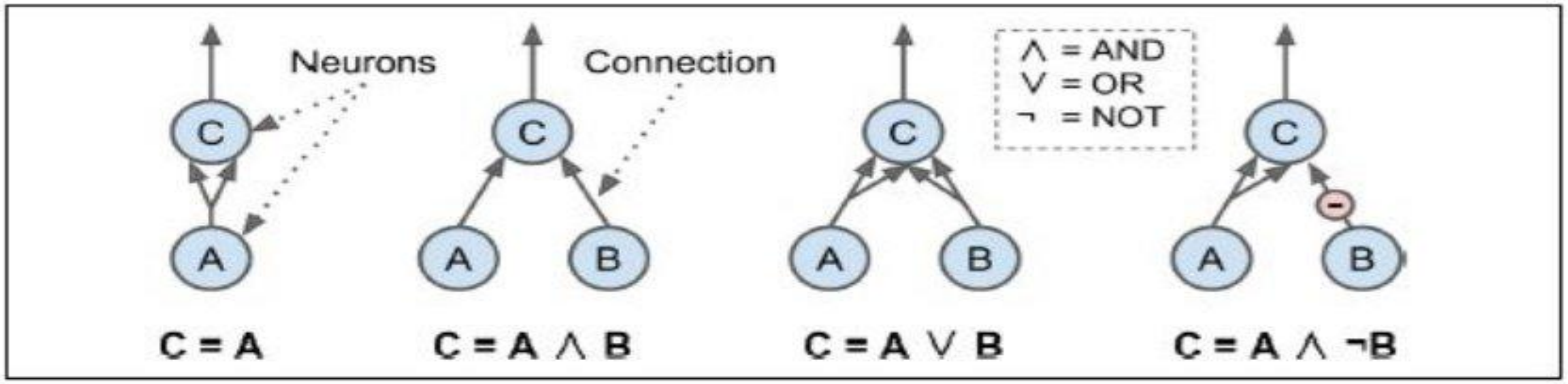
Machine Learning – Intro

History of ANN (Artificial Neural Networks).

- Introduced in 1943! (by Warren McCulloch)
- Had a revival in the 1980's with better training algorithms.
- Then other methods gained traction.
- In the last 10 years ANN's have had a huge revival, we have faster computers, and simply more data to train them on now.
- Recent results such as AlphaGo and AlphaZero all uses some form of Neural Networks. https://deepmind.com/research/case-studies/alphago-the-story-so-far

Machine Learning – Intro

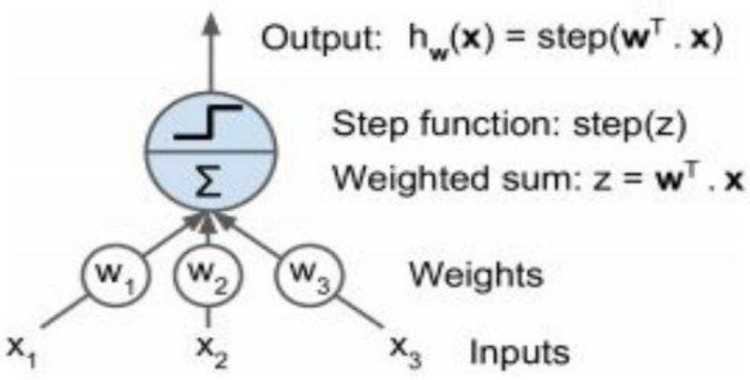
Neurons can simulate OR, AND & NOT gates



In this case, the C neuron is set to fire when input evaluates to true.
So in the AND case both A and B have to be active, for C to be activated.
In the OR case, it is enough if just one of them is activated
In the last case, imagine that B can prevent the neuron from firing. The constructions here are called Perceptrons.

Machine Learning – Intro

Output/Input of a Neuron


Output: $h_w(\mathbf{x}) = \text{step}(\mathbf{w}^T . \mathbf{x})$
Step function: step(z)
Weighted sum: $z = \mathbf{w}^T . \mathbf{x}$
Σ
w_1
w_2
w_3
Weights
x_1
x_2
x_3
Inputs

$$\text{heaviside}(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases} \qquad \text{sgn}(z) = \begin{cases} -1 & \text{if } z < 0 \\ 0 & \text{if } z = 0 \\ +1 & \text{if } z > 0 \end{cases}$$

Example:
input : (2,3,5)
weights = (1,2,-1)
Sum :
z = 2*1 + 3*2 + 5*-1 = 3
Output:
h(z) = h(3) = 1.
1 is then the input to the
next neuron

Step function - or Activation function

- Many different functions can be used here as the
- Sigmoid function (also known as the logistic function) used to be very popular - more or less biological version.

$$S(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}.$$

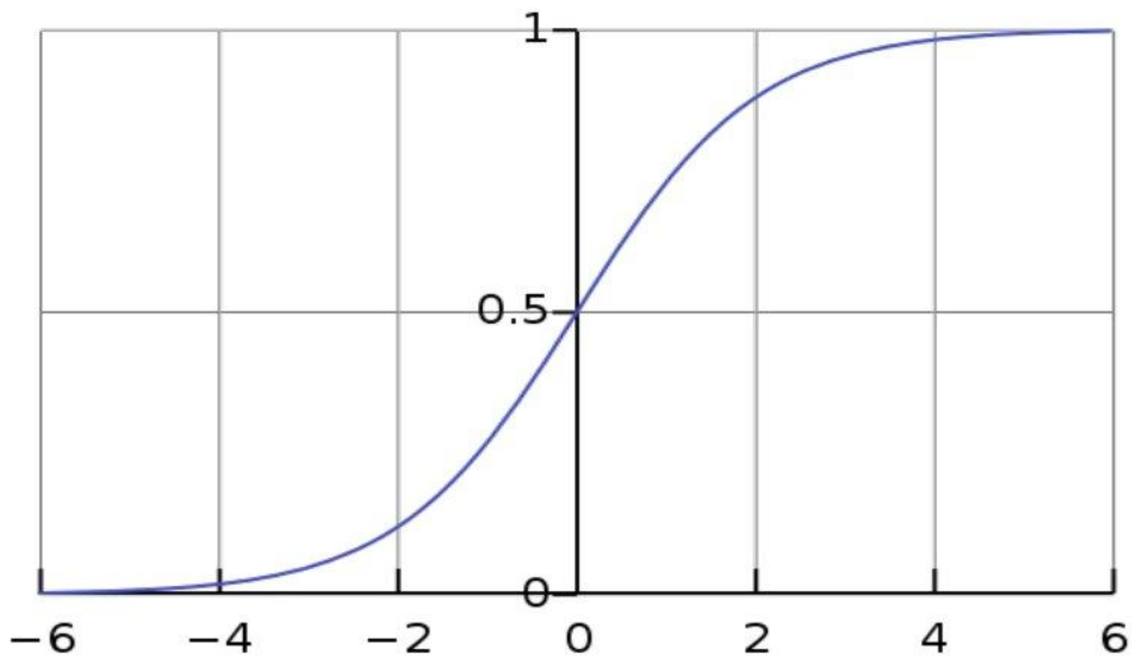
- But ReLu function is better for ANN (paper by Glorot & Bengio, 2010):

f(x) = max(0,x) (Relu = Rectified Linear Unit)

This was a major breakthrough! We will have more on this in the next lecture. (Notice the year 2010 by the way…..)

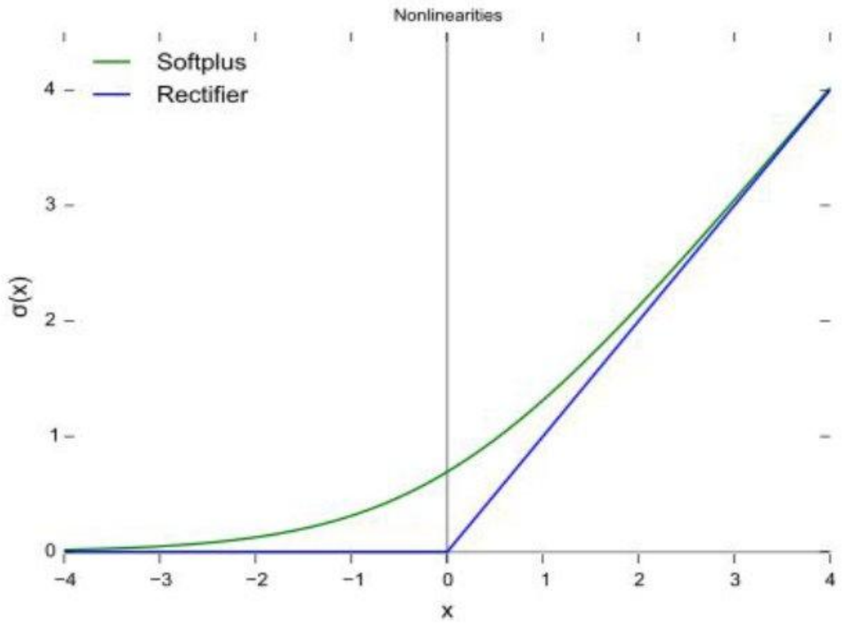
Machine Learning – Intro

Input/Output - activation functions - sigmoid



ERHVERVSAKADEMI AARHUS

Activation Functions - Relu - fast to compute



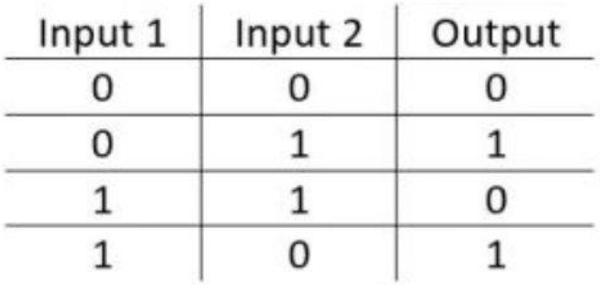
The green line is another version, called Softplus. It is differentiabel in all points as opposed to ReLu.

Softplus:
f(x) = log(1 + exp(x))

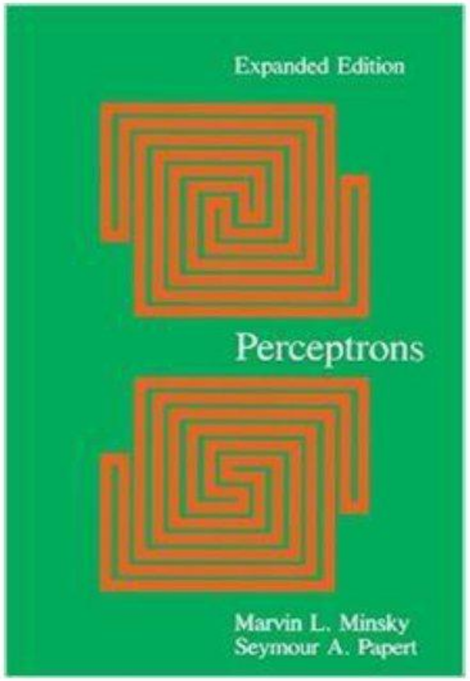
Machine Learning – Intro

Limitations of Perceptrons

Minsky and Papert showed in 1969 that the simple Perceptrons have some limitations. For instance, they cannot simulate an XOR gate.



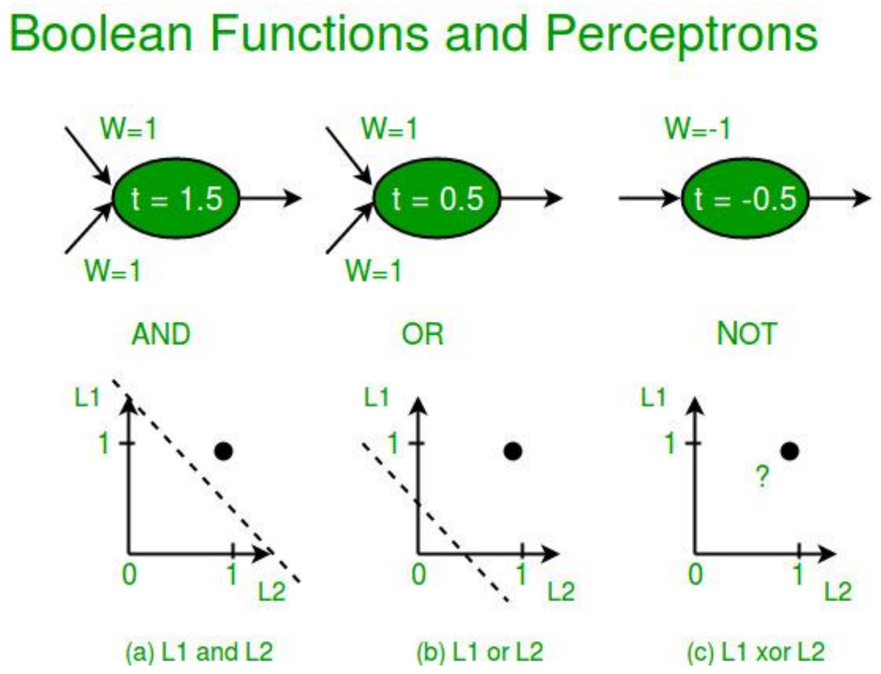
Input 1	Input 2	Output
0	0	0
0	1	1
1	1	0
1	0	1



ERHVERVSAKADEMI
AARHUS

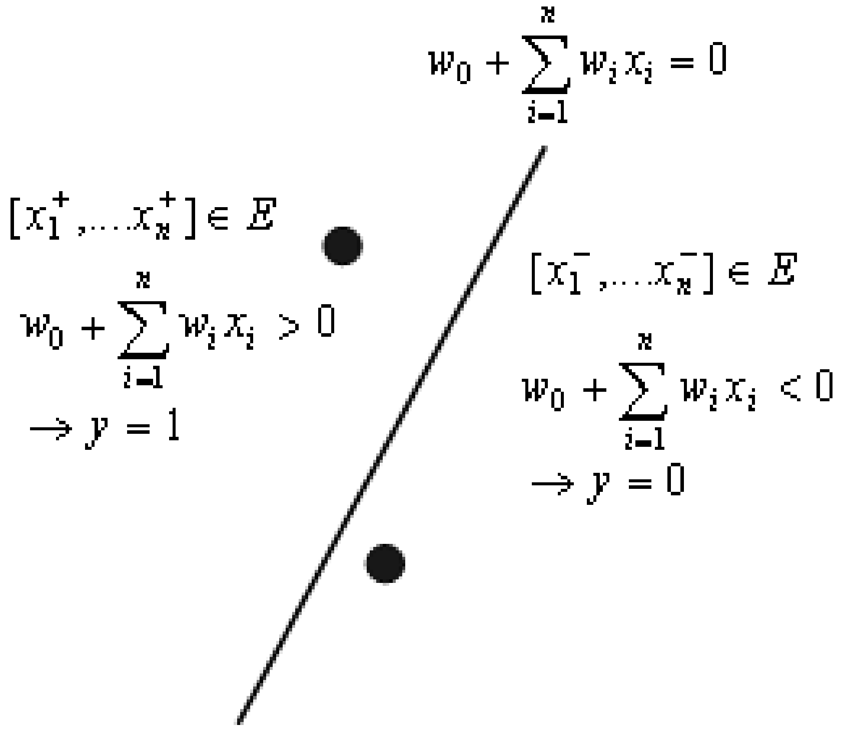
Machine Learning – Intro

Neurons can simulate OR, AND & NOT gates. But not Xor.



Machine Learning – Intro

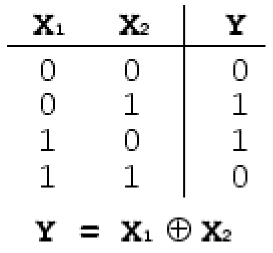
Neurons can simulate OR, AND & NOT gates. But not Xor.


$w_0 + \sum_{i=1}^{n} w_i x_i = 0$
$[x_1^+, \ldots x_n^+] \in E$
$w_0 + \sum_{i=1}^{n} w_i x_i > 0$
$\rightarrow y = 1$
$[x_1^-, \ldots x_n^-] \in E$
$w_0 + \sum_{i=1}^{n} w_i x_i < 0$
$\rightarrow y = 0$

ERHVERVSAKADEMI
AARHUS

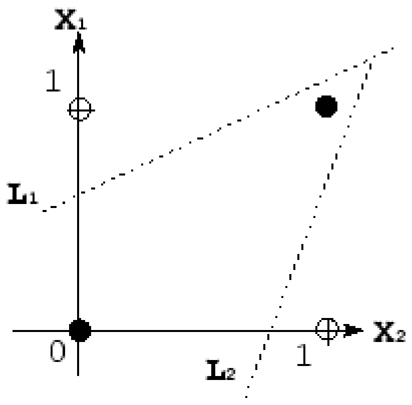
Machine Learning – Intro

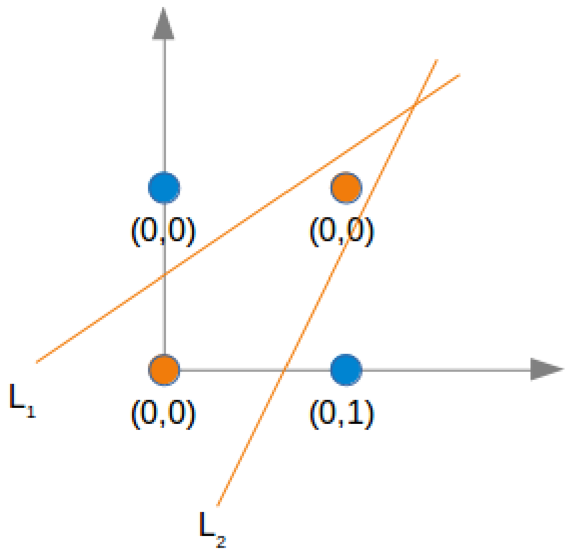
Neurons can simulate OR, AND & NOT gates. But not Xor.



X_1	X_2	Y
0	0	0
0	1	1
1	0	1
1	1	0

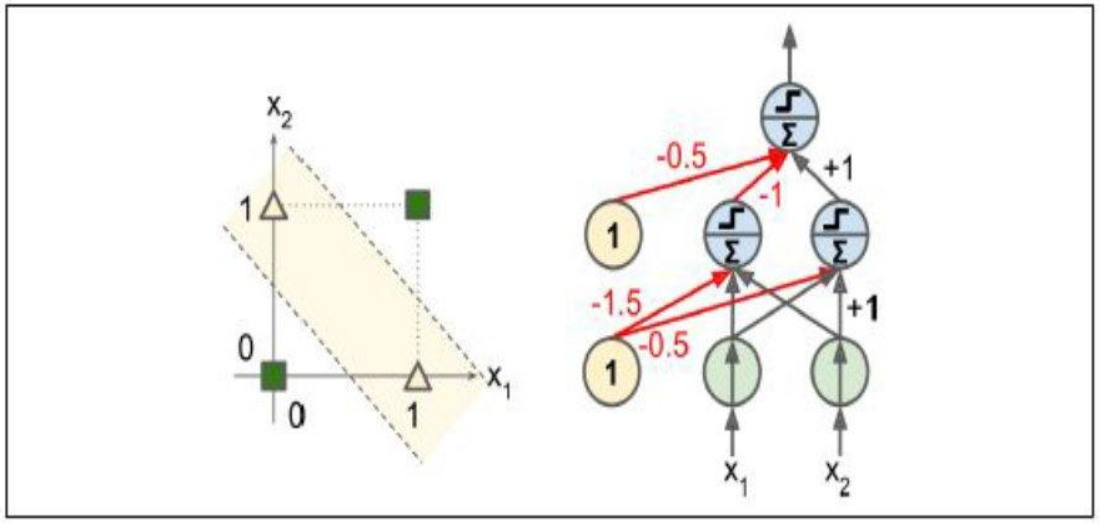
$Y = X_1 \oplus X_2$





Solution - combine several gates to produce XOR

This network can in fact compute the XOR function. You can try to check yourself! This is called a Multi Layer Perceptron.

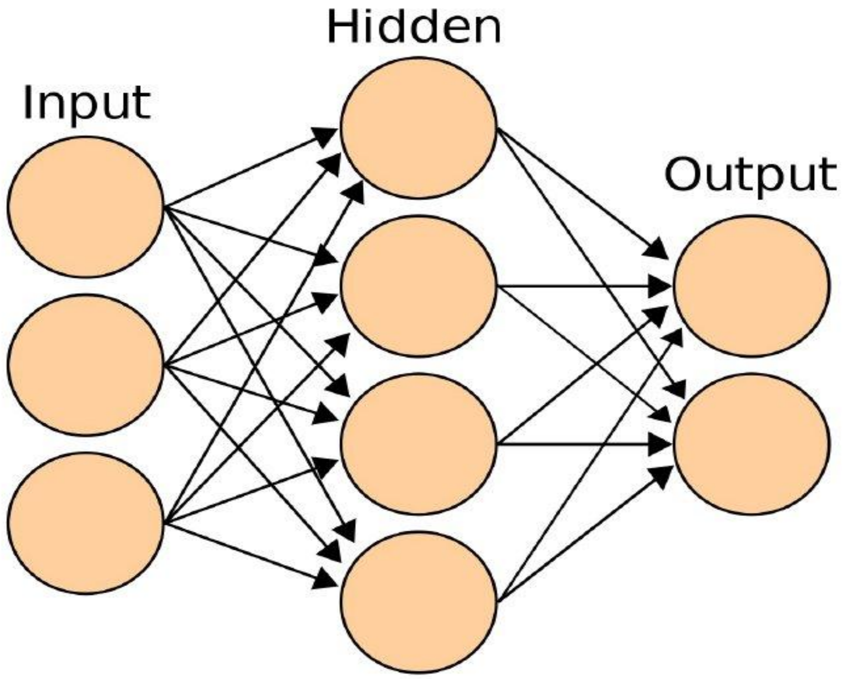


Machine Learning – NN Training.

Neural Networks and training

Machine Learning – NN Training.

What is a Neural Network



Note: all nodes in the previous layer is **fully connected** to the next layer.
There can be *any* number of hidden layers. Also *any* number
of inputs and outputs.
If there is more than 1-hidden layer, then this is called **Deep Learning,** because the Network is **Deep.**

Machine Learning – Training.

How many layers should I use and #neurons?

In general for the vast majority of problems 1 hidden layer is enough.

But there is a tendency that 2 or 3 hidden layers sometimes converge faster on a solution in many cases.
The number of neurons in the input layer should in general correspond to the number of features in the dataset that you want to train on.
The number of neurons in the output layer should correspond to the training set. For a binary classification problem the output layer is just 1 neuron.
As a rule of thumb the number of neurons in the hidden layers should be between the number of neurons in the output layer and in the input layer. But there is no hard and fast rule. So experiment.

Machine Learning – Training.

Training NNs

- The purpose of training is to adjust the weights (on the edges) in the network, so that the “correct” output for the training input is produced in as many cases as possible.
- This means that we need to have **labelled data for training** - supervised learning.
- There are **many ways** to train the ANN.
- Currently, the most common form of training is using the **back-propagation algorithm** or some variant of it.
- Almost all training algorithms work in an iterative way, gradually improving the weights.
- In general, training is NP-hard, meaning no guarantee to find optimal weights, unless we try them all, which will take forever

https://authors.library.caltech.edu/26705/1/88-20.pdf

https://people.csail.mit.edu/rivest/pubs/BR93.pdf

Machine Learning – Training

Training ANNs - back propagation algorithm

● For many years, it was difficult to train NNs.

● But in 1985, Rumelhart et Al published a breakthrough article about the back propagation algorithm. It's actually a mutation of the Gradient Descent algorithm

● We will not go into all the details of how this algorithm work - just a high-level overview

● Backpropagation is an iterative algorithm, it works with labelled data (so for supervised learning). It does not guarantee optimal training - but training is NP-Hard problem anyway, so no algorithm does this (without trying all combinations)

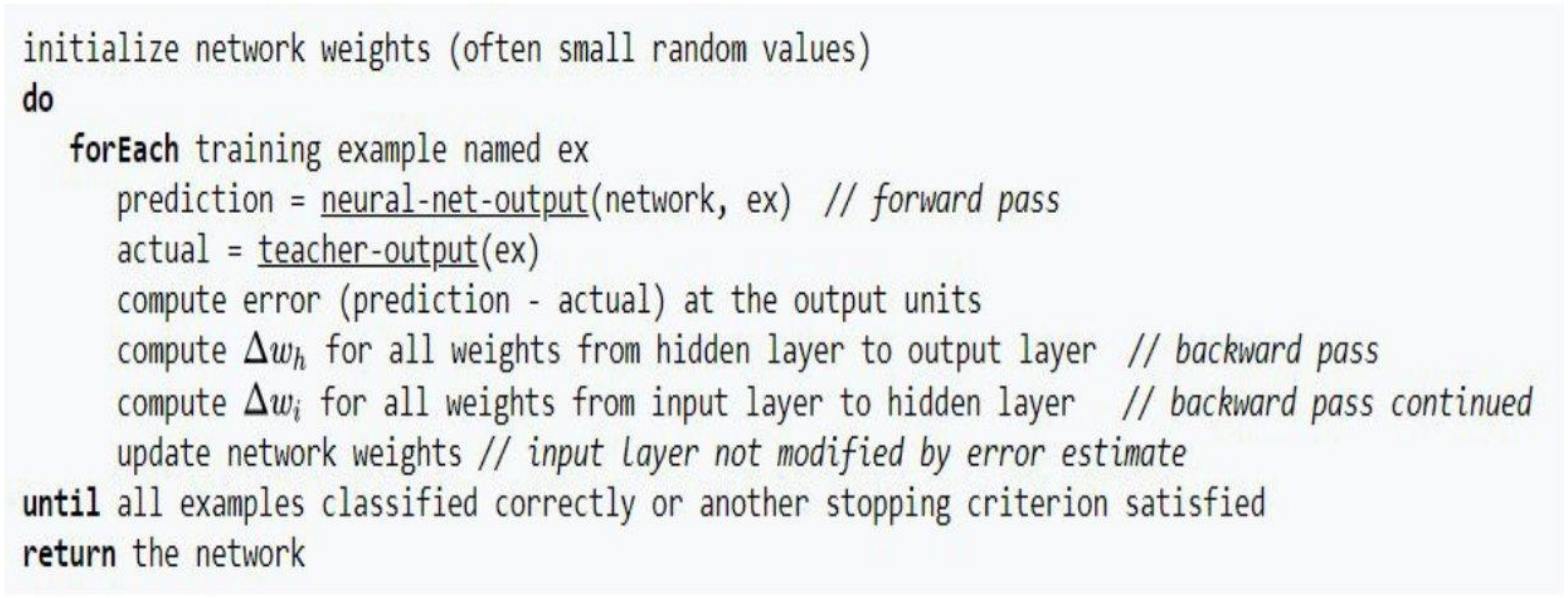
https://apps.dtic.mil/dtic/tr/fulltext/u2/a164453.pdf

Back propagation in words

For each training instance the algorithm will run the training input through the neural network **(forward pass)** to make a prediction
Then the algorithm looks at the **output value and compares that to the real output value** (which we know, because we have labelled training data).
It then looks at the error (real-actual) and then goes back layer-by-layer and **changes each of the weights on the connections**, to make the error smaller (the gradient descent step).
This is then repeated over and over **until the weights converge or you decide to stop the algorithm.**
Backpropagation learning does not require normalization of input vectors.

Machine Learning – Training.

Back propagation in pseudo code (1 hidden layer)



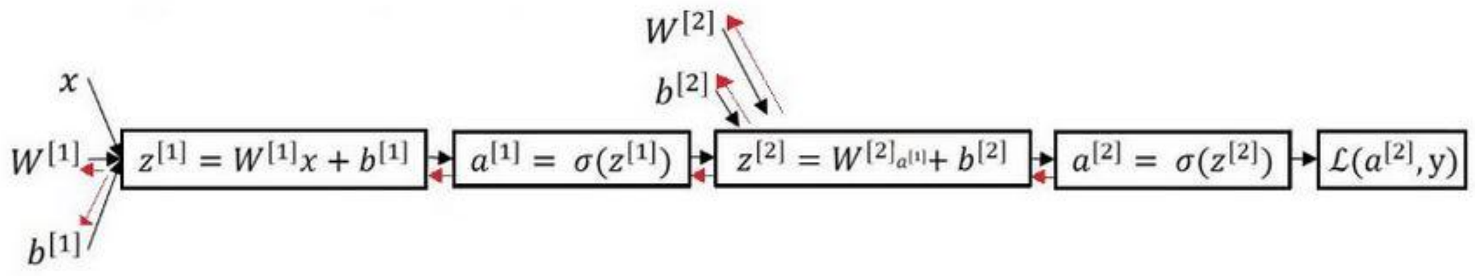
```
initialize network weights (often small random values)
do
    forEach training example named ex
        prediction = neural-net-output(network, ex)  // forward pass
        actual = teacher-output(ex)
        compute error (prediction - actual) at the output units
        compute $\Delta w_h$ for all weights from hidden layer to output layer  // backward pass
        compute $\Delta w_i$ for all weights from input layer to hidden layer   // backward pass continued
        update network weights // input layer not modified by error estimate
until all examples classified correctly or another stopping criterion satisfied
return the network
```

Source: Wikipedia

ERHVERVSAKADEMI
AARHUS

Machine Learning – Training.

Back propagation:

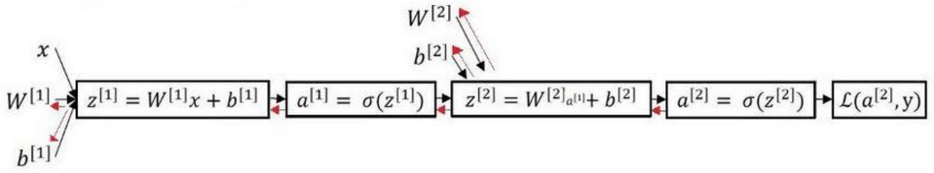


x
$W^{[1]}$
$b^{[1]}$
$z^{[1]} = W^{[1]}x + b^{[1]}$
$a^{[1]} = \sigma(z^{[1]})$
$W^{[2]}$
$b^{[2]}$
$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$
$a^{[2]} = \sigma(z^{[2]})$
$\mathcal{L}(a^{[2]}, y)$

In the above figure, the forward propagation (indicated by black lines) is used to compute the output for a given input X. The backward propagation (indicated by red lines) is used to update the Weight Matrices W[1], W[2] and biases b[1], b[2].

Machine Learning – Training.

Back propagation:



x
$W^{[1]}$
$b^{[1]}$
$z^{[1]} = W^{[1]}x + b^{[1]}$
$a^{[1]} = \sigma(z^{[1]})$
$W^{[2]}$
$b^{[2]}$
$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$
$a^{[2]} = \sigma(z^{[2]})$
$\mathcal{L}(a^{[2]}, y)$

```
compute $\Delta w_h$ for all weights from hidden layer to output layer  // backward pass
compute $\Delta w_i$ for all weights from input layer to hidden layer   // backward pass continued
update network weights // input layer not modified by error estimate
```

$$dA^{[2]} = \frac{\delta L(A^{[2]},Y)}{\delta A^{[2]}} = \frac{-Y}{A^{[2]}} + \frac{1-Y}{1-A^{[2]}}$$

$$dZ^{[2]} = \frac{\delta L(A^{[2]},y)}{\delta Z^{[2]}} = \frac{\delta L(A^{[2]},y)}{\delta A^{[2]}} * \frac{\delta A^{[2]}}{\delta Z^{[2]}} = A^{[2]} - Y$$

$$dW^{[2]} = \frac{\delta L(A^{[2]},y)}{\delta W^{[2]}} = \frac{\delta L(A^{[2]},y)}{\delta Z^{[2]}} * \frac{\delta Z^{[2]}}{\delta W^{[2]}} = dZ^{[2]} A^{[1]T}$$

$$db^{[2]} = \frac{\delta L(A^{[2]},y)}{\delta b^{[2]}} = \frac{\delta L(A^{[2]},y)}{\delta Z^{[2]}} * \frac{\delta Z^{[2]}}{\delta b^{[2]}} = dZ^{[2]}$$

$$dA^{[1]} = \frac{\delta L(A^{[2]},Y)}{\delta A^{[1]}} = \frac{\delta L(A^{[2]},Y)}{\delta Z^{[2]}} * \frac{\delta Z^{[2]}}{\delta A^{[1]}} = dZ^{[2]} W^{[2]}$$

$$dZ^{[1]} = \frac{\delta L(A^{[2]},y)}{\delta Z^{[1]}} = \frac{\delta L(A^{[2]},y)}{\delta A^{[1]}} * \frac{\delta A^{[1]}}{\delta Z^{[1]}} = W^{[2]T} dZ^{[2]} * \sigma'\left(Z^{[1]}\right)$$

$$dW^{[1]} = \frac{\delta L(A^{[2]},y)}{\delta W^{[1]}} = \frac{\delta L(A^{[2]},y)}{\delta Z^{[1]}} * \frac{\delta Z^{[1]}}{\delta W^{[1]}} = dZ^{[1]} X^T$$

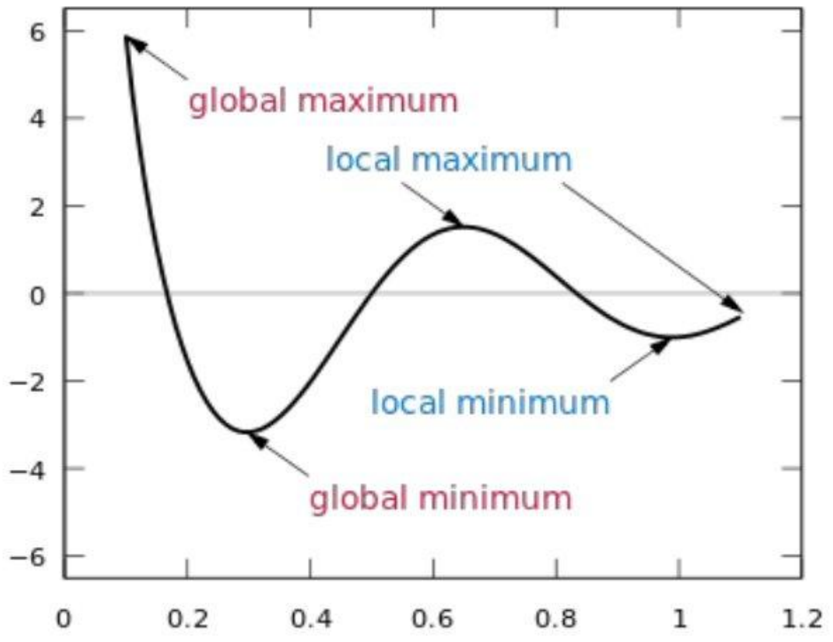
$$db^{[1]} = \frac{\delta L(A^{[2]},y)}{\delta b^{[1]}} = \frac{\delta L(A^{[2]},y)}{\delta Z^{[1]}} * \frac{\delta Z^{[1]}}{\delta b^{[1]}} = dZ^{[1]}$$

$$L(\hat{y}, y) = -\left[y \log \hat{y} + (1-y)\log(1-\hat{y})\right]$$

Using the above equation for loss L and using the sigmoid function as the activation function for the hidden and output layer, with the help of chain rule of derivatives, we get delta w….

Machine Learning – Training.

Backpropagation performance



As with all Gradient Descent algorithms, back propagation is not guaranteed to find an optimum solution, that minimizes
the training error.
But in practice the algorithm performs very well.
It can be run in **batch-mode** - i.e. only updating weights after a number of training samples - faster.

Machine Learning – Training.

Neural Network simulator

A great visualization tool to play around with can be found at:

http://playground.tensorflow.org/

Neural Network simulator

A great visualization tool to play around with can be found at:

http://playground.tensorflow.org/

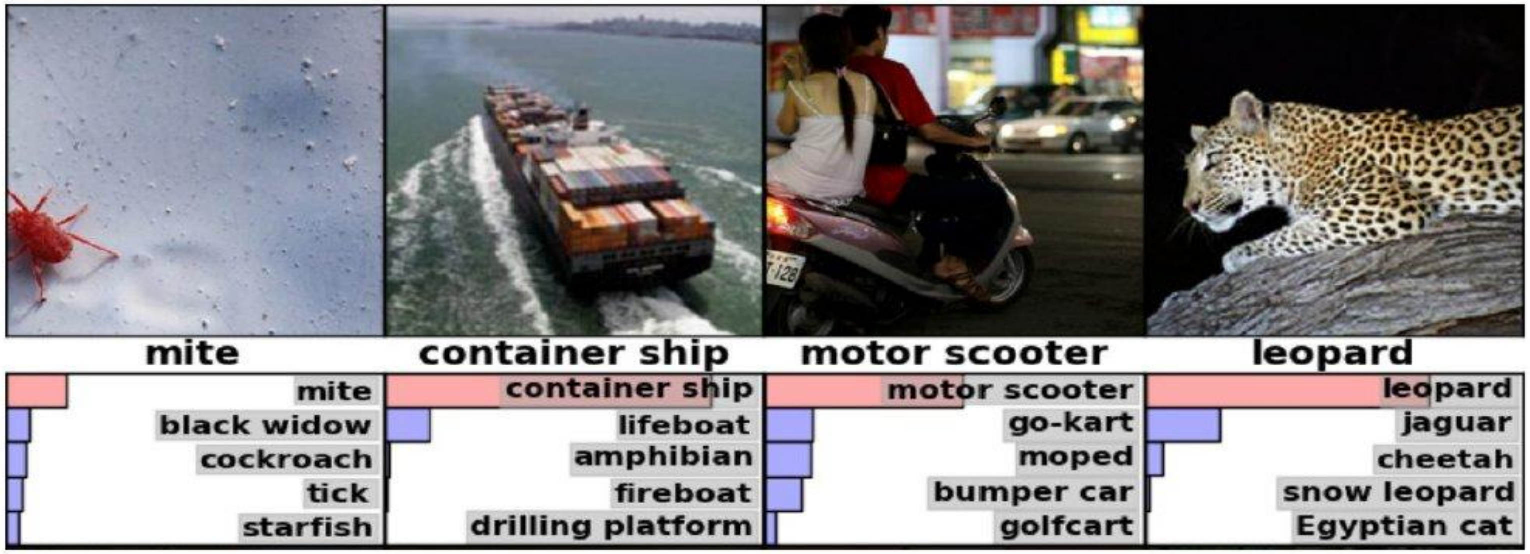
Exercises 1 -3 in Playground TensorFlow Exercises.

ANN Applications - some examples.

Machine Learning – Applications.

ANN Example: Image recognition.

https://www.tensorflow.org/tutorials



Machine Learning – Applications.

ANN Example: AlphaGo

Machine Learning – Applications.

An Example: AlphaZero (reinforcement learning)

According to Bing:
AlphaGo uses a neural network to evaluate board positions and select moves. The neural network is trained using supervised learning on a dataset of expert human moves and then fine-tuned using reinforcement learning through self-play 1. During self-play, AlphaGo plays against itself and learns from its mistakes. It uses Monte Carlo tree search (MCTS) to explore the game tree and select moves that maximize its chances of winning.

Machine Learning – Frameworks.

ML Frameworks for deep learning

Machine Learning – Frameworks.

The case for using frameworks.

- Implementing ANNs and the training algorithm from scratch **takes a LOT of time, also in debugging everything.**
- By now, some **very good frameworks** have emerged.
- E.g. **Tensorflow**
- Many other frameworks exists - in fact SciKit learn also has limited support for Neural Networks (from version 0.18)
- Also Keras, which is a higher level framework, is quite popular.

Tensorflow overview

- Made by Google
- Can be run on both desktop computers, in the cloud and on mobile (Tensorflow Lite).
- Currently in version 2
- It is open source!
- Is very general purpose - can be used for many different kind of computations.

Machine Learning – Frameworks.

Who is using Tensorflow? Some examples.

airbnb
NVIDIA
UBER
SAP
kakao
DeepMind
Dropbox
ebay
Google
intel
Coca-Cola
mi
ZTE
QUALCOMM

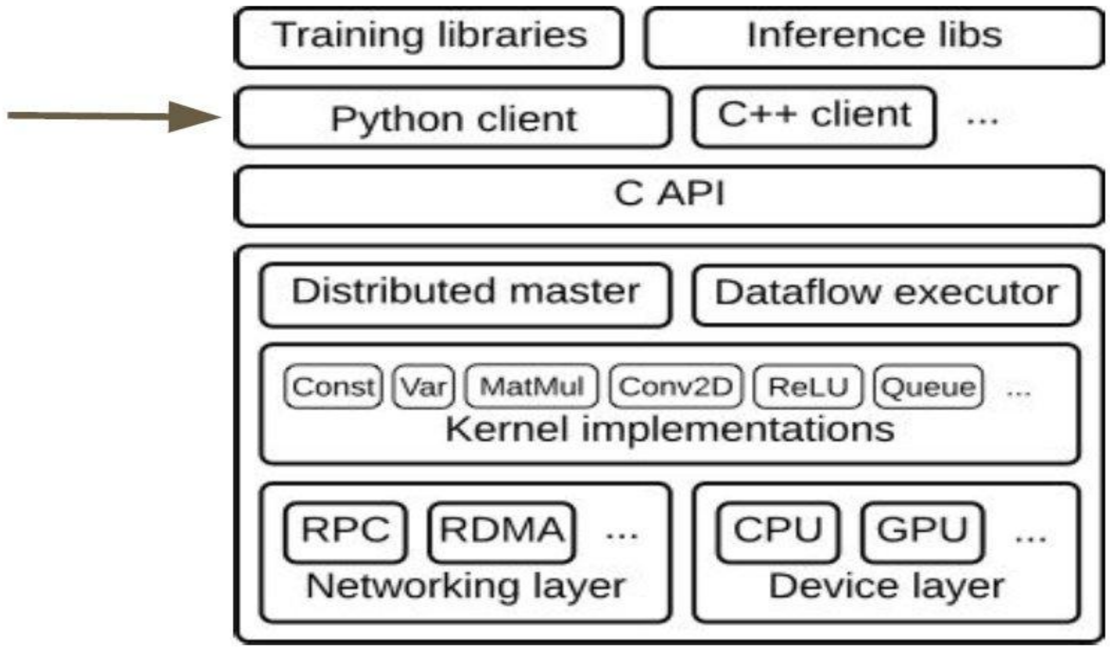
Machine Learning – Frameworks.

(In case you want to try it out) Installation:

Tensorflow installation:
https://www.tensorflow.org/install/

Machine Learning – Frameworks.

Tensorflow - architecture



Machine Learning – Frameworks.

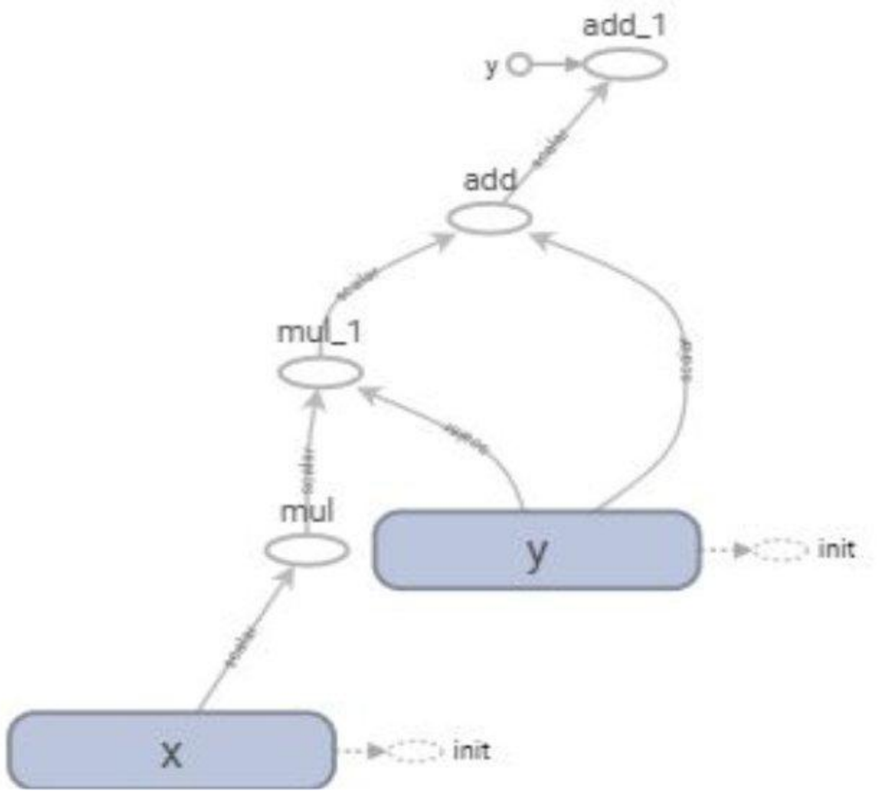
Tensorflow

- There are usually two phases:

1. The construction phase - where you construct the TF graph.
2. The execution phase - where you run your data through the graph

Machine Learning – Frameworks.

```
import tensorflow as tf
#print the version and make sure everything works.

print ("Tensorflow version "+str(tf.__version__))

#define two variables and initialize them to 3 and 4.
x = tf.Variable(3,name="x")
y = tf.Variable(4,name="y")

#define a function
f = x*x*y+y+2

init = tf.global_variables_initializer()

session = tf.InteractiveSession()
init.run()
result = f.eval()

print("result: "+str(result)) # should give 42 - the answer to
everything
```

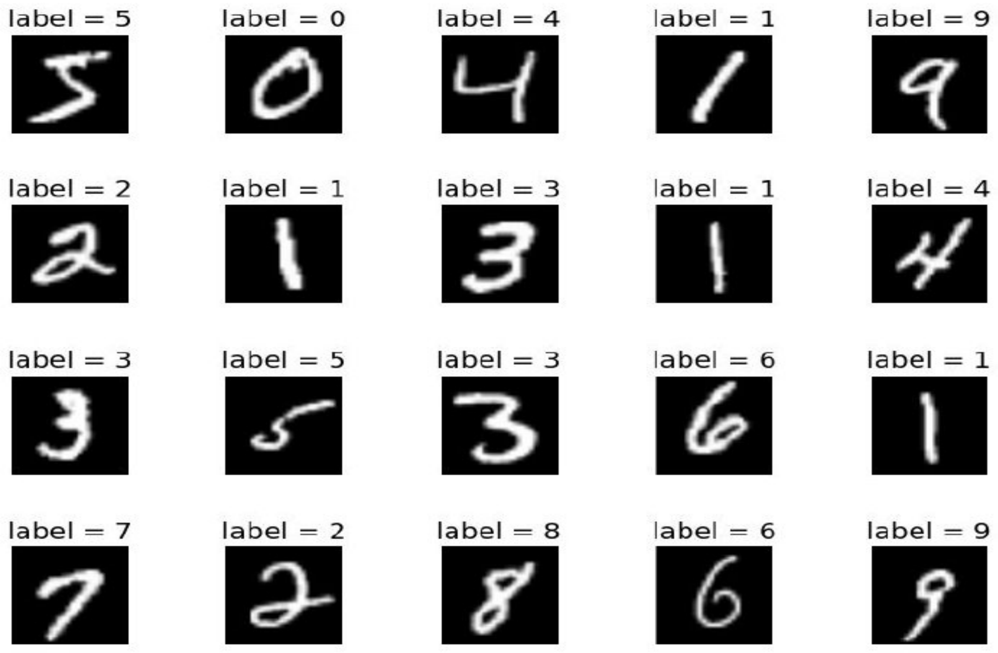


Tensorflow workflow in general

1. load/prepare data that you want to use
2. Define your model with TF
3. Train the model
4. Evaluate the model
5. Prediction of new values with the model

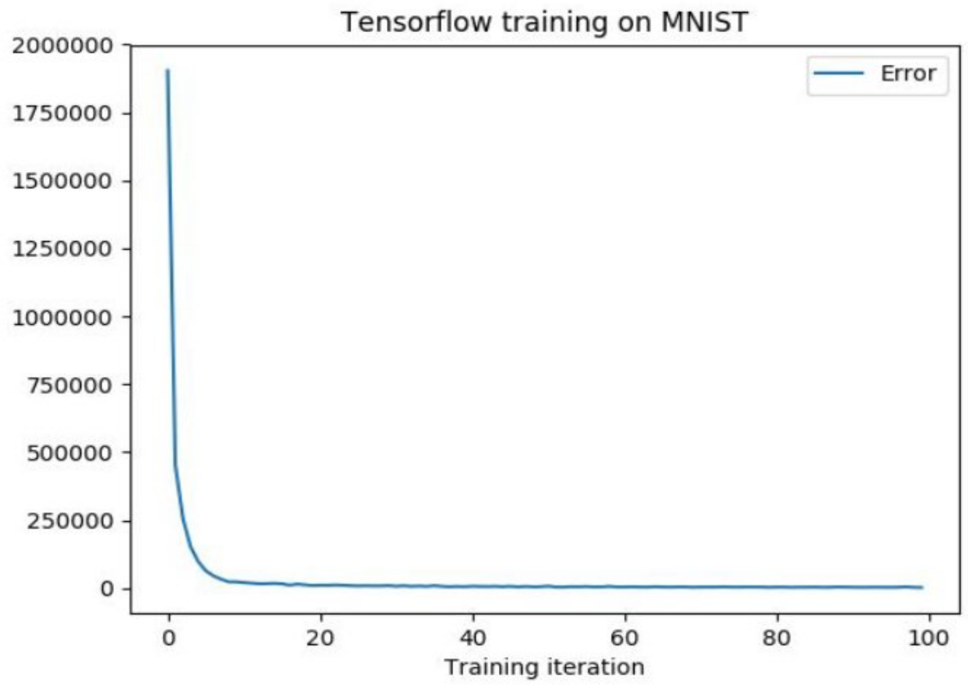
Machine Learning – Framworks.

TensorFlow - Example - MNIST and Tensorflow



Machine Learning – Frameworks.

Tensorflow - results on MNIST



100 training iterations.

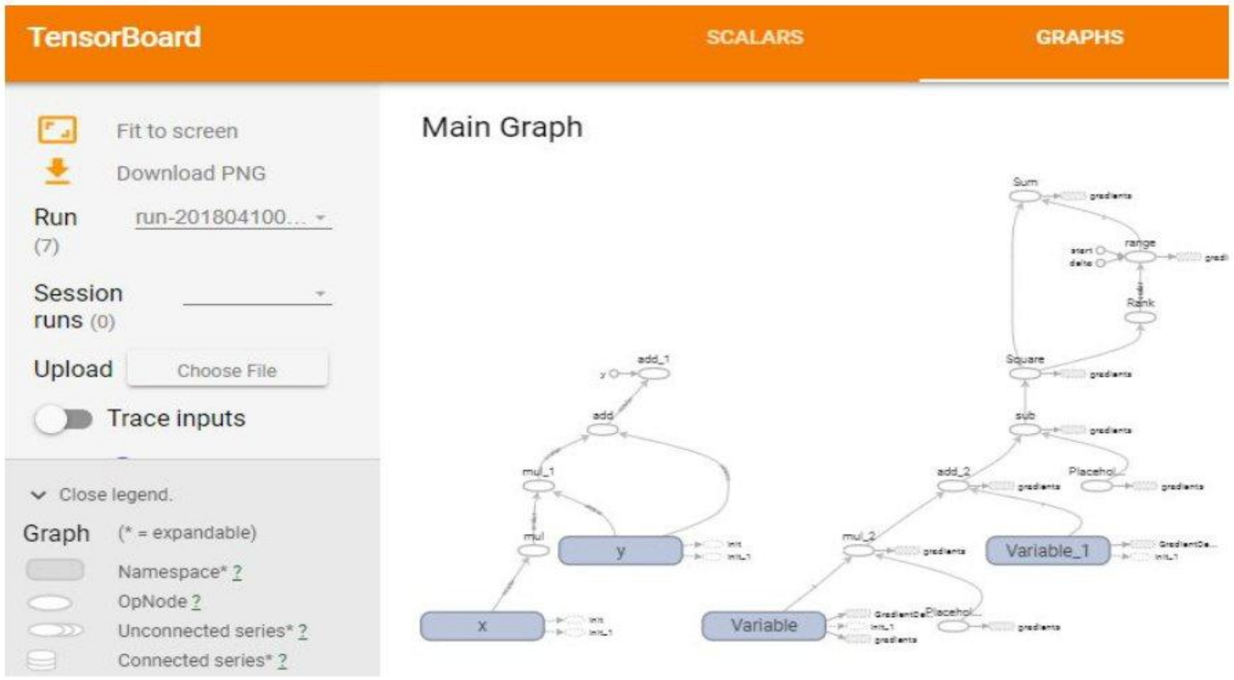
Error approaching, but never reaching zero.

Total accuracy:
0.9745 = 97,45 %

Training Time: approx 20 minutes on a good laptop

Machine Learning – Frameworks.

Tensorboard - visualizing your graphs



Exercises 2 - 5.

Exercise 2,3, 4 & 5.

In groups (2-3).

(Carefully) Discuss, and agree on an answer
(-Write the groups answer down.
&
-Draw the activation functions yourself (3) as part of the discussion – in order to get a feel for the functions).

We take it in class afterwards.

Exercise 6 (and then afterwards exercise 7…)

Exercise 6.

(tutorial)
Multilayer Perceptron exercise.

-We start with this exercise here in class-
(In order to make sure – that everyone has tried it before going home today).

Questions:

A) What happens if you change the testsize ? Make it smaller, and larger. 5%, 90 %
B) Remove the scaler (StandardScaler). Can the classifier work without the scaler?
C) Change the number of iterations, epochs, max_iter, to 100. Was that a good idea?
D) Reset max_iter to 1000. Experiment with the number of hidden layers. Start with 1 hidden layer. How many hidden layers with how many nodes gives the best results?
   Will mlp = MLPClassifier(hidden_layer_sizes=(2), max_iter=1000) work? Why not?

Exercise 7 (when you are done with exercise 6…)

Exercise 7.

(tutorial)
Multilayer Perceptron exercise.

-We start with this exercise here in class-
(In order to make sure – that everyone has tried it before going home today).

Homework

Week 5.
Topics : Neural Nets.

Exercises.
All of the exercises for this week.

Litterature:

[Hands on Machine Learning] chapter 10. 299- 317, 344-353 v.3.

You don't need to understand all the details as some of
it can be a bit technical, but try to read through it,
and try to get a general idea of what this is all about.
(Optional: Chapter 11 in the book is for those who want more in-depth with Tensorflow and NNs, so an optional extra chapter).