Lecture 1

Introduction Artificial Neural Networks (ANN) and Deep learning

Deep Learning is a subset of machine learning and it uses similar general techniques. We remember that ML methods can be divided into 2 big groups

- Supervised ML
- Unsupervised ML

For Supervised ML generally one dataset is transformed into another.

First ML algorithm is trained on a selected set of data when the correct answer is also given. Then we can predict the result also for new dataset unseen by ML algorithm during training

Unsupervised ML algorithm is trying to do the same job - transform the given dataset into prognosis of the result.

Still during training process there is No right answer. An unsupervised algorithm tries to find patterns in this data and informs us about them.

Deep learning technique is based on the main assumption the prediction is done by applying the framework of neural network which mimics our brains.

The aim of this course is to study main steps in DL algorithm, compare various modifications of artificial NN and construct efficient, robust and accurate algorithms for predictions given by developed ANN ~~alg~~ frame-works.

We start with the very simple neural network. It is based only on forward propagation of information.

Our aim is predict the result from the given input data.

Input data - in our case it is a number recorded somewhere

Prediction result - is what the ANN tells us.

Questions: • is this production always right? No.

• How does the network learn?

Our first NN

- multiplies the input by a weight (a parameter of NN)
- it scales the input number and tells a prediction.

prediction = input * weight

See python version of the code

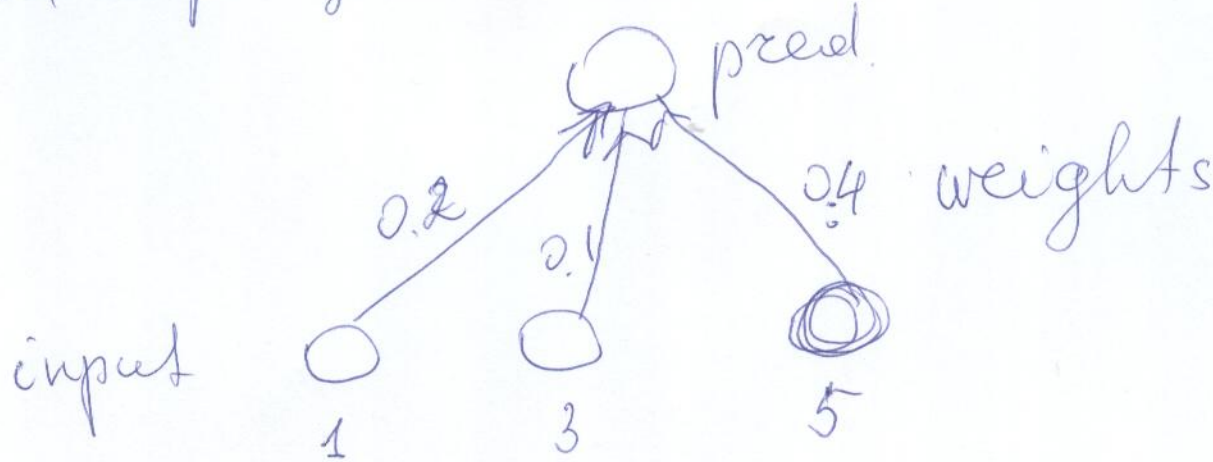
```
weight = 0.23
def neural_network(input, weight):
    prediction = input * weight
    return prediction
```

Example: (python3 nn.py)

```
input = 7.2
pred = neural_network(input, weight)
print(pred)
```

If the accuracy of prediction is not good (or sufficient) we consider a more complex ANN, with multiple inputs.

Example of NN.



input = [1, 3, 5]

weights = [0.2, 0.1, 0.4]

pred = 1×0.2 + 3×0.1 + 5×0.4

We compute a prediction as a weighted sum of input data.

NumPy code

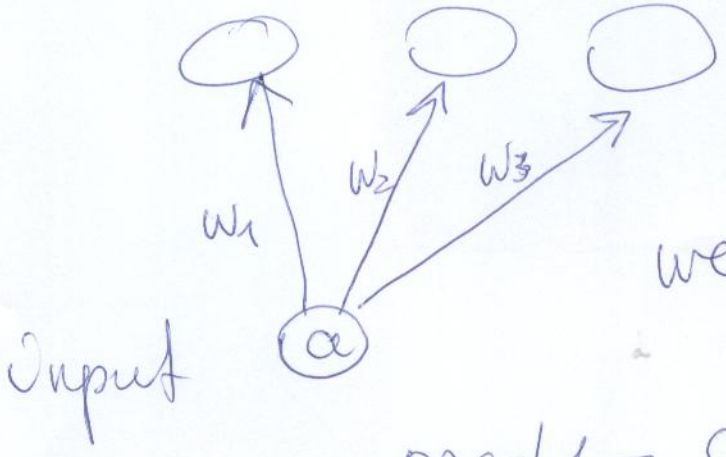
```
import numpy as np
weights = np.array([0.2, 0.1, 0.4])
def neural_network(input, weights):
    pred = input.dot(weights)
    return pred

input = np.array([1., 3., 5.])
pred = neeeral_network(input, weights)
print("pred=", pred)
```

How to find weights? (parameters)

This task is solved during training stage and we will solve it ~~then~~ later.

Making a prediction with multiple outputs

We start with the case when the input is equal to a number



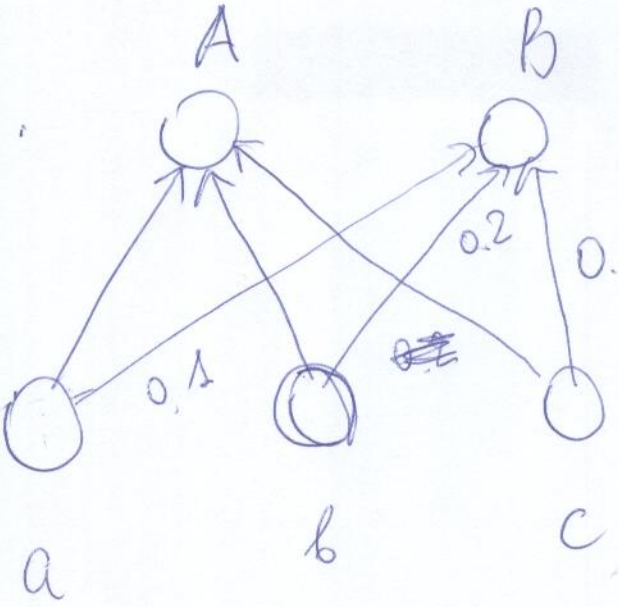
weights = [0.3, 0.1, 0.4]

$pred1 = a \times 0.3$
$pred2 = a \times 0.1$
$pred3 = a \times 0.4$

Python code for elementwise multiplication

```
def ele_mul(number, vector):
    output = [0, 0, 0]
    assert(len(output) == len(vector))
    for i in range(len vector)):
        output[i] = number * vector[i]
    return output

def neural_network(input, weights):
    pred = ele_mul(input, weights)
    return pred

pred = neural_network(inputs, weights 1)
print(pred)
```

ANN with multiple inputs and outputs



inputs = [a, b, c]

weights = [[0.1, 0.1, -0.3], # A?

[0.1, 0.2, 0.0]] # B?

$A = a \times 0.1 + b \times 0.1 + c \times (-0.3)$

$B = a \times 0.1 + b \times 0.2 + c \times 0$

Python code

```
def vect_mat_mul(vect, matrix):
    assert(len(vect) == len(matrix))
    output = [0, 0, 0]
    for i in range(len(vect)):
        output[i] = vect.dot(matrix[i])
    return output
```

Example.

```
def neural_network(input, weights):
    pred = vect_mat_mul(input, weights
    return pred
input1 = [8.5, 0.65, 1.2]
pred1 = neural_network(input1, weights
```