Creating a simple neural network in Python The logic XOR gate

- We have two binary entries (0 or 1) and the output will be 1 only when just one of the entries is 1 and the other is 0.
- It means that from the four possible combinations only two will have 1 as output.

Inputs		Outputs
X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	0

The network

This is the code for this study case, later on, we'll analyze the code step by step.

```
import numpy as np

from keras.models import Sequential
from keras.layers.core import Dense

# menyiapkan data training
training_data = np.array([[0,0],[0,1],[1,0],[1,1]], "float32")
target_data = np.array([[0],[1],[1],[0]], "float32")

#  siapkan arsitektur ANN
model = Sequential()
model.add(Dense(16, input_dim=2, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

# setting parameter
model.compile(loss='mean_squared_error',
optimizer='adam',
metrics=['binary_accuracy'])

# lakukan training
model.fit(training_data, target_data, epochs=1000)
scores = model.evaluate(training_data, target_data)

# tampilkan hasil training
print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
print (model.predict(training_data).round())
```

```
1/1 [==============================] - 0s 12ms/step - loss: 0.1783 - binary_accuracy
```

Keras and Tensorflow

Keras is a high level library. It makes easier the job of describing the layers of our network and the engine that will run and train the network is Google's Tensorflow which is the best implementation we have available nowadays.

Analyzing the code

First we need to import the classes we will use.

- Numpy will be used for the management of arrays.
- From Keras we import the Sequential model and the Dense layer type which is the usual one.

```
import numpy as np
from keras.models import Sequential
from keras.layers.core import Dense
```

After this let's create the input and output arrays

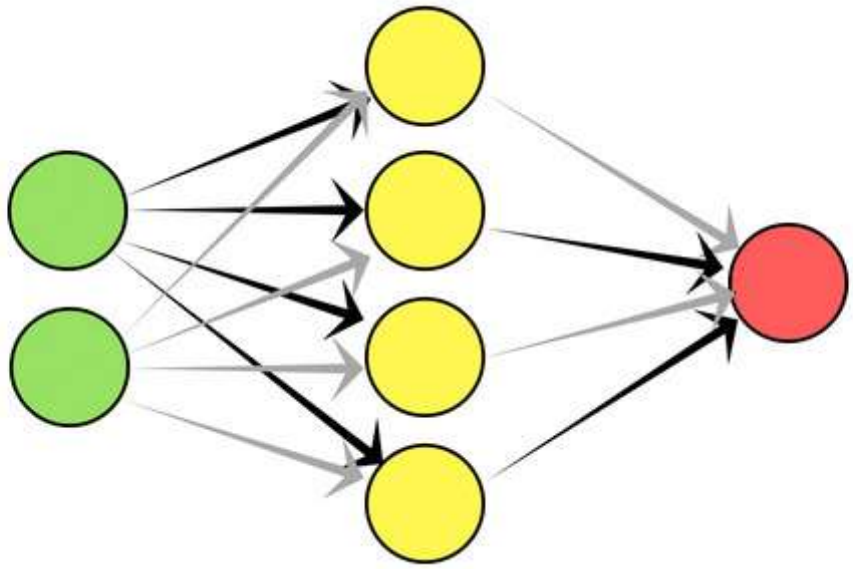
These are the combinations of the XOR and the expected results in the same order

```
training_data = np.array([[0,0],[0,1],[1,0],[1,1]], "float32")
target_data = np.array([[0],[1],[1],[0]], "float32")
```

Now is time to create the architecture of our neural network.

Pada gambar, neuron yang berwarna kuning seharusnya ada 16 buah.

Visual representation of the created network



```
model = Sequential()
model.add(Dense(16, input_dim=2, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

- Here we create an empty model of type Sequential, it means our layers will be one after the other.
- Then we add two Dense layers with the model's add method.
- It is actually three layers we are adding because by specifying input_dim = 2 it means we defined the input layer with two neurons, for our XOR input values.
- Our first hidden layer has 16 neurons. As activation function we use "relu" because we know it gives good results.
- You can set up your network as you want, this is just an example, there are off course, well defined architectures but you can be creative for one time and explore by changing:

1. the number of layers
2. the activation function
3. the number of neurons.

- Finally we add the output layer with one node that will hold the result value with a sigmoid activation function.

Let's train it

Before training the network we have to make some adjustments.

Here we define:

1. the loss type we'll use,
2. the weight's optimizer for the neuron's connections and
3. the metrics we need.

```
model.compile(loss='mean_squared_error',optimizer='adam',metrics=['binary_accuracy'])
```

Now is time to train the net.

Using the **fit** method we indicate the inputs, outputs and the amount of iterations for the training process.

This is just a simple example but remember that for bigger and more complex models you'll need more iterations and the training process will be slower.

```
model.fit(training_data, target_data, epochs=1000)
```

Training results

- Let's check the training outputs

Epoch 1/10004/4 [==============================] - 0s 43ms/step - loss: 0.2634 - binary_accuracy: 0.5000

Epoch 2/10004/4 [==============================] - 0s 457us/step - loss: 0.2630 - binary_accuracy: 0.2500

- We can see it was kind of luck the firsts iterations and accurate for the half of the outputs, but after the second it only provides a correct result of one quarter of the iterations.
- Then, in the 24th epoch recovers 50% of accurate results, and this time is not a coincidence, is because it correctly adjusted the network's weights.

Epoch 24/1000 4/4 [==============================] - 0s 482us/step - loss: 0.2549 - binary_accuracy: 0.5000

Epoch 107/1000 4/4 [==============================] - 0s 621us/step - loss: 0.2319 - binary_accuracy: 0.7500

Epoch 169/1000 4/4 [==============================] - 0s 1ms/step - loss: 0.2142 - binary_accuracy: 1.0000

- And, in this case, in the iteration number 107 increases the accuracy rate to 75%, 3 out of 4 and in the iteration number 169 it produces almost 100% correct results and it keeps like that 'till the end.

- As it starts with random weights the iterations in your computer would probably be slightly different but at the end you'll achieve the binary precision, which is 0 or 1.

Evaluation and prediction

- First, we evaluate the model.
- Then we see we achieved a 100% accuracy, keeping in mind the simplicity of the example.

```
scores = model.evaluate(training_data, target_data)
print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

```
1/1 [==============================] - 0s 16ms/step - loss: 0.0238 - binary_accuracy: 1.

binary_accuracy: 100.00%
```

And we make the 4 possible predictions for the XOR passing the entries.

```
print (model.predict(training_data).round())
```

```
[[0.]
 [1.]
 [1.]
 [0.]]
```

Tuning the parameters of the network

This is a very simple example with only 4 possible entries but we can have a really complex network and it'll be necessary to adjust the network's parameters like

1. Amount of layers
2. Amount of neurons per layer
3. Activation function per layer
4. When compiling the model, the loss functions, the optimizer and metrics
5. Number of training iterations

- I recommend you to play with the parameters to see how many iterations it needs to achieve the 100% accuracy rate.
- There are many combinations of the parameters settings so is really up to your experience and the classic examples you can find in "must read" books.

Reference: https://broutonlab.com/blog/tutorial-create-simple-neural-network