Single Neuron Model

In this session, we will use one neuron to model the logical "AND" function.

1. Assuming value 1 for True and value 0 for False, the data file AND.txt contains all four cases of AND operation.
    - Each line represents a case
    - The first two numbers in the line represents the two inputs
    - The third number represents the targeted output
    - Note that values in a line are separated by comma (",")

2. Create a new Colab notebook for Python 3 code.

3. Place the and.txt file in the appropriate Google Drive folder (called "directory" in Unix/Linux). Add code to the notebook to mount the drive and change directory to the target folder. Refer to the last-week worksheet for handling any struggling with this step

4. In order to read from text input, it is convenient to use a feature of "numpy" library. Import the "numpy" library with the following statement.

    ```
    import numpy as np
    ```

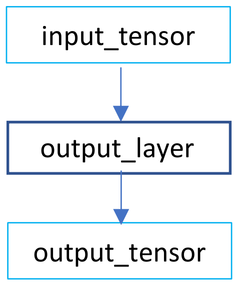
    The following code will load the text data file into a numpy array (a resemblance of "list") . Print the dataset to check the loaded value.

    ```
    dataset = np.loadtxt("and.txt", delimiter=",")
    ```

5. Extract inputs, trainX, and targeted output, trainY, from the dataset.
    - trainX contains the first two columns of all rows of the dataset.
    - trainY contains the last column of all rows of the dataset.
    - Print both arrays to verify whether the extraction works as expected.

    ```
    trainX = dataset[:,0:2]
    trainY = dataset[:,2]
    ```

6. The model of the neuron will be defined with a neural network layer in the following form:



7. Import necessary modules for constructing keras layers

```
from tensorflow.keras.layers import Input, Dense
```

8. Construct input tensor to have a shape of two sequential items

```
input_tensor = Input(shape=(2,))
```

NOTE: Make sure to understand the meaning of **shape** clearly (ref: numpy array)

9. Construct output layer as a “Dense” neuron layer, containing only one neuron.
    - The activation of the neuron is given by the logistic sigmoid function.

$$Out = \frac{1}{(1 + e^{-netIn})}$$

    - The value of $netIn$ is the sum of *weighted* input into the neuron, including the value of “*bias*” (an essential feature to be used in our neuron model)

```
output_layer = Dense(1, activation='sigmoid', use_bias=True)
```

NOTE:
- Pay attention to the geometrical analogy of the neuron activation and bias.
- There are many candidates for activation function.

10. Construct output tensor from the created output layer.

```
output_tensor = output_layer(input_tensor)
```

11. The neural network, which will learn to resemble the logical “AND” operation, is created as a “Model”, which requires the input and output tensors to be specified.

```
from tensorflow.keras.models import Model

model = Model(input_tensor, output_tensor)
```

12. For the model to learn, an optimizer will be defined for adapting neural network's parameters so that the neural network's function matches the target operation.
    - The optimizer chosen for the start is SGD (Stochastic Gradiant Descent).
    - The model must be compiled before running. The compilation requires loss function and the optimizer to be defined.
    - The learning rate of the SGD optimizer can be defined through the variable "learning_rate".

```
from tensorflow.keras.optimizers import SGD

model.compile(loss='mean_squared_error', \
              optimizer=SGD(learning_rate=0.1))
```

NOTE:

- Numerical optimization is a large enough field to be made as its own subject.
- There are many options for optimizer and loss function.
- The key point, for now, is to grasp the concept of how optimization is carried out here.

13. The following code starts the learning process.

```
history = model.fit(trainX, trainY, epochs=500, batch_size=1)
```

- "epochs" is the number of learning iterations. Each iteration is one round of feeding every input once.
- "batch_size" is the number of inputs that will be fed into the neural network, each learned independently of one another, before the neural network adapts itself once.
- The return value of "model.fit" is the recorded trace of epoch-based learning report

As the learning converges, Is the loss always decreased?

14. Once the neural network finishes learning, we can see the neural network's output for each input by running the following code.

```
output = model.predict(trainX)
print(np.concatenate((trainX, output), axis=1))
```

15. The learning progress, recorded as variable "history", can be later viewed graphical using Pandas and Matplotlib as the following code.

```
import pandas as pd
import matplotlib.pyplot as plt

pd.DataFrame(history.history).plot(figsize=(8, 5))
plt.grid(True)
plt.gca().set_ylim(0, 1)
plt.show()
```