

# Lab 6

AIM: To study and implement Neural Network

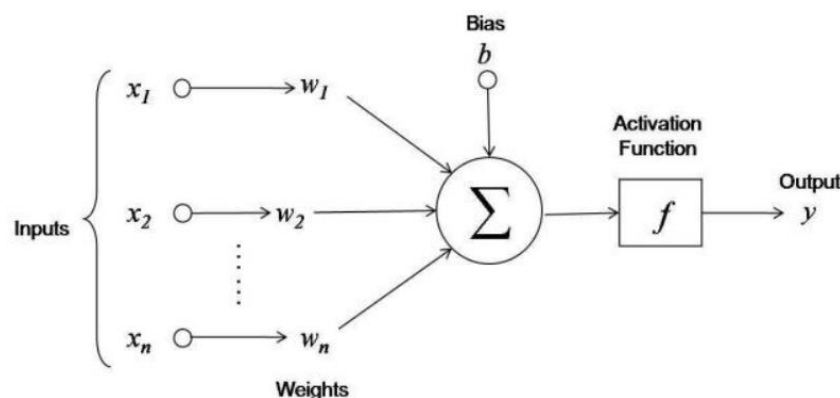
- a) Creating simple neural network from scratch with Python
- b) Neural Network with Backpropagation in Python

Module used: Tensorflow (Python)

Theory:

Neural Networks are algorithms that try to mimic the brain. They were actually widely used in the 80s & 90s. Recently their varied applications have been widely recognised.

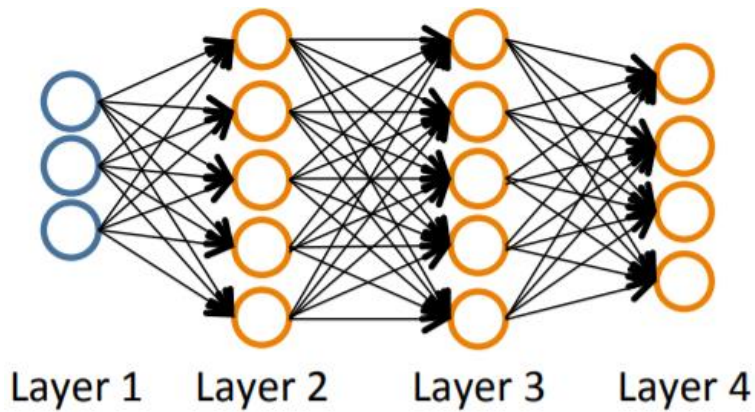
The first layer in a neural network is always the input layer where the data is fed. This data is later convoluted or spooled before reaching the final layer of output. The intermediate layers are known as hidden layers which is responsible for the efficiency of the model. Almost every layer has a +1 element which is responsible for reducing the bias.



This flow chart shows the basic functioning of a neuron cell.

The neurons then apply an activation function like Sigmoid or ReLu. In a sigmoid activation function, the whole cost function calculated value between 0 and 1. This keeps on repeating in every hidden layer until it reaches the output layer.

The neuron keeps on learning with every iteration and we can't get the best output on the first try itself. Forward propagation keeps on teaching the neuron the relationship between the dataset while back propagation is used to modify the weights which in turn helps in reducing the loss generated. Learning rate is a parameter that can be tweaked to control the rate at which the network learns. It cannot be too small or too large.



The above given neural network is used for multiclass classification of 4 elements.

General mathematic expressions used:

1. Cost function:

**Neural network:**

$$h_{\Theta}(x) \in \mathbb{R}^K \quad (h_{\Theta}(x))_i = i^{th} \text{ output}$$

$$J(\Theta) = -\frac{1}{m} \left[ \sum_{i=1}^m \sum_{k=1}^K y_k^{(i)} \log(h_{\Theta}(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - (h_{\Theta}(x^{(i)}))_k) \right]$$

$$+ \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} (\Theta_{ji}^{(l)})^2$$

where,

$$\mathcal{D} = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$$

$L$  = total no. of layers in network

$s_l$  = no. of units (not counting bias unit) in layer  $l$

Backpropagation helps in determining how well the network is doing on an example  $l$  from the dataset. It helps in adjusting the weights associated while training and reduce the loss generated with every epoch.

Code:

```
import tensorflow as tf
import os
import zipfile
```

```
DESIRED_ACCURACY = 0.999
```

```

!wget --no-check-certificate \
    "https://storage.googleapis.com/laurencemoroney-
blog.appspot.com/happy-or-sad.zip" \
    -O "/tmp/happy-or-sad.zip"

zip_ref = zipfile.ZipFile("/tmp/happy-or-sad.zip", 'r')
zip_ref.extractall("/tmp/h-or-s")
zip_ref.close()

class myCallback(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs={}):
        if(logs.get('acc') is not None and logs.get('acc')>DESIRED_
ACCURACY):
            print("\nReached 99.9% accuracy so cancelling training!
")
            self.model.stop_training = True

callbacks = myCallback()

model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(16, (3,3), activation='relu', input_shap
e=(150, 150, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),

    tf.keras.layers.Conv2D(32, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),

    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

from tensorflow.keras.optimizers import RMSprop

model.compile(loss='binary_crossentropy',
              optimizer=RMSprop(lr=0.001),
              metrics=['accuracy'])

from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale=1/255)

train_generator = train_datagen.flow_from_directory(
    '/tmp/h-or-s',

```

```

        target_size=(150, 150),
        batch_size=10,
        class_mode='binary')

validation_datagen = ImageDataGenerator(rescale=1/255)

val_generator = validation_datagen.flow_from_directory(
    '/tmp/h-or-s_test',
    target_size=(150, 150),
    batch_size=10,
    class_mode='binary')

history = model.fit_generator(
    train_generator,
    steps_per_epoch = 8,
    epochs = 15,
    verbose = 2)

%matplotlib inline
import matplotlib.image as mpimg
import matplotlib.pyplot as plt

acc = history.history['accuracy']
loss = history.history['loss']
val_acc = history.history['val_accuracy']
val_loss = history.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'b', "Training Accuracy")
plt.plot(epochs, loss, 'r', "Training Loss")
plt.title('Training parameters')
plt.figure()

plt.plot(epochs, val_acc, 'b', "Validation Accuracy")
plt.plot(epochs, val_loss, 'r', "Validation Loss")
plt.title('Validation parameters')
plt.figure()

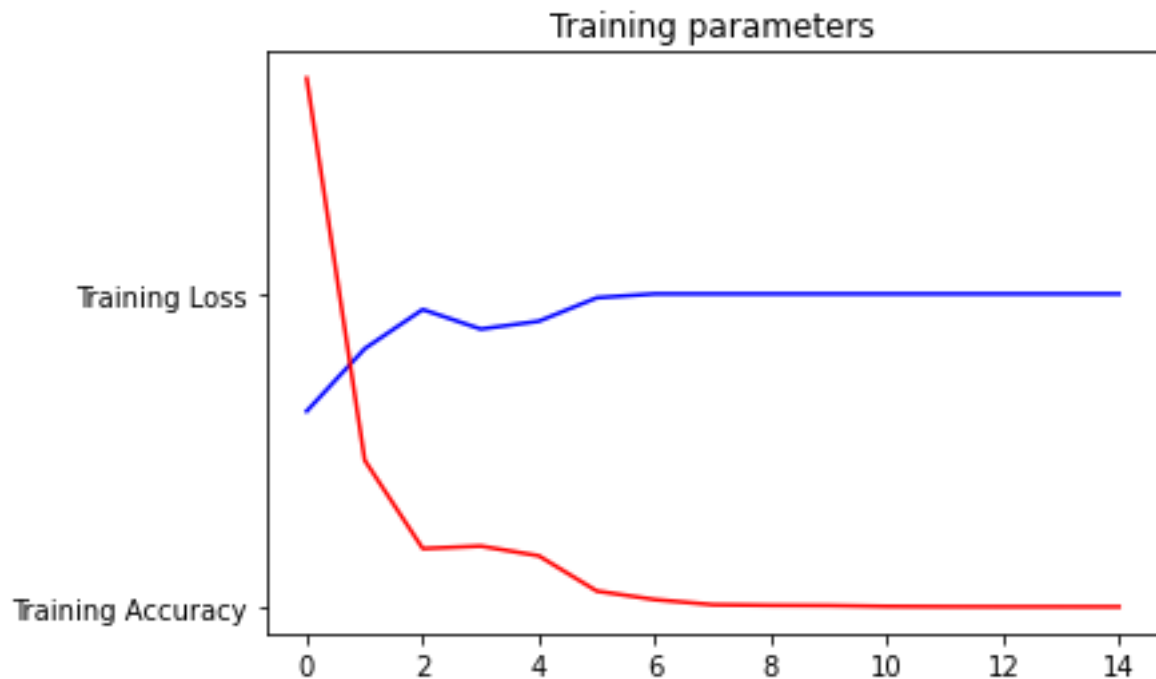
```

Output:

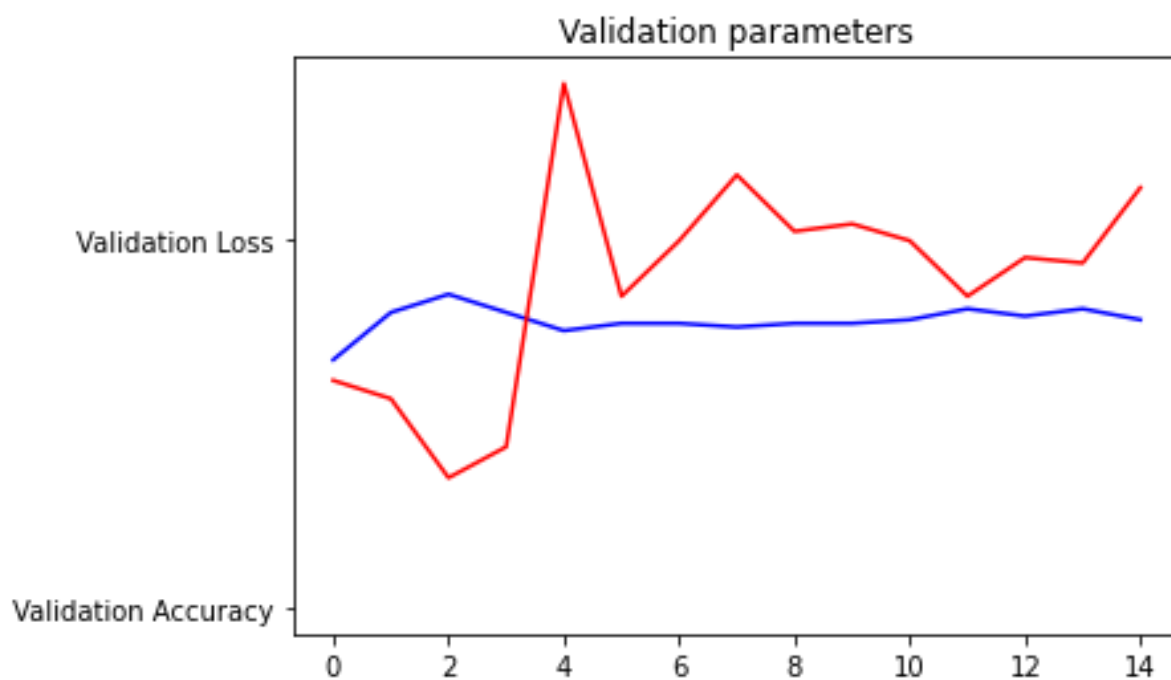
```
epochs = 15,  
verbose = 2,  
validation_data = val_generator)
```

/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/training.py:1844: UserWarning: `Model.fit\_generator` is deprecated and

```
Epoch 1/15  
8/8 - 1s - loss: 1.6889 - accuracy: 0.6250 - val_loss: 0.6174 - val_accuracy: 0.6733  
Epoch 2/15  
8/8 - 0s - loss: 0.4683 - accuracy: 0.8250 - val_loss: 0.5676 - val_accuracy: 0.8020  
Epoch 3/15  
8/8 - 0s - loss: 0.1861 - accuracy: 0.9500 - val_loss: 0.3528 - val_accuracy: 0.8515  
Epoch 4/15  
8/8 - 0s - loss: 0.1941 - accuracy: 0.8875 - val_loss: 0.4369 - val_accuracy: 0.8020  
Epoch 5/15  
8/8 - 0s - loss: 0.1630 - accuracy: 0.9125 - val_loss: 1.4240 - val_accuracy: 0.7525  
Epoch 6/15  
8/8 - 0s - loss: 0.0503 - accuracy: 0.9875 - val_loss: 0.8459 - val_accuracy: 0.7723  
Epoch 7/15  
8/8 - 0s - loss: 0.0232 - accuracy: 1.0000 - val_loss: 0.9967 - val_accuracy: 0.7723  
Epoch 8/15  
8/8 - 0s - loss: 0.0067 - accuracy: 1.0000 - val_loss: 1.1757 - val_accuracy: 0.7624  
Epoch 9/15  
8/8 - 0s - loss: 0.0048 - accuracy: 1.0000 - val_loss: 1.0224 - val_accuracy: 0.7723  
Epoch 10/15  
8/8 - 0s - loss: 0.0040 - accuracy: 1.0000 - val_loss: 1.0424 - val_accuracy: 0.7723  
Epoch 11/15  
8/8 - 0s - loss: 0.0013 - accuracy: 1.0000 - val_loss: 0.9969 - val_accuracy: 0.7822  
Epoch 12/15  
8/8 - 0s - loss: 5.6091e-04 - accuracy: 1.0000 - val_loss: 0.8460 - val_accuracy: 0.8119  
Epoch 13/15  
8/8 - 0s - loss: 2.7438e-04 - accuracy: 1.0000 - val_loss: 0.9512 - val_accuracy: 0.7921  
Epoch 14/15  
8/8 - 0s - loss: 2.6119e-04 - accuracy: 1.0000 - val_loss: 0.9364 - val_accuracy: 0.8119  
Epoch 15/15  
8/8 - 0s - loss: 1.2751e-04 - accuracy: 1.0000 - val_loss: 1.1411 - val_accuracy: 0.7822
```



Training Accuracy, Loss vs Epochs Graph



Validation Accuracy, Loss vs Epochs Graph

```
[ ] model.summary()
```

Model: "sequential"

| Layer (type)                  | Output Shape         | Param # |
|-------------------------------|----------------------|---------|
| =====                         |                      |         |
| conv2d (Conv2D)               | (None, 148, 148, 16) | 448     |
| max_pooling2d (MaxPooling2D)  | (None, 74, 74, 16)   | 0       |
| conv2d_1 (Conv2D)             | (None, 72, 72, 32)   | 4640    |
| max_pooling2d_1 (MaxPooling2) | (None, 36, 36, 32)   | 0       |
| conv2d_2 (Conv2D)             | (None, 34, 34, 64)   | 18496   |
| max_pooling2d_2 (MaxPooling2) | (None, 17, 17, 64)   | 0       |
| flatten (Flatten)             | (None, 18496)        | 0       |
| dense (Dense)                 | (None, 512)          | 9470464 |
| dense_1 (Dense)               | (None, 1)            | 513     |
| =====                         |                      |         |
| Total params: 9,494,561       |                      |         |
| Trainable params: 9,494,561   |                      |         |
| Non-trainable params: 0       |                      |         |

Conclusion: In this experiment, we have successfully created a neural network using the Tensorflow module. The dataset opted is a small chunk of a huge one thus, issues of overfitting are prominent. The Neural Network used here consists of 1 input layer, 1 hidden layers and 1 output layer. The optimizer responsible for the backpropagation algorithm is the binary\_crossentropy which has a learning rate set to 0.001. Finally, we trained the neural network and plotted its Accuracy-Loss Curves.

References :

- Part A: [First neural network for beginners explained \(with code\) | by Arthur Arnx | Towards Data Science](#)
- Part B: <https://machinelearningmastery.com/implement-backpropagation-algorithm-scratch-python/>
- Machine Learning by Andrew Ng