Predicting Ethernet network configuration correctness using feedforward Neural Networks

Louis James Weber

019087433A

University of Luxembourg

November 2021

Introduction

The purpose of the Neural Network is to predict whether a network configuration will be feasible or not, based on the given dataset. And our goal is to obtain the maximal accuracy on the testing set while minimizing over/under-fitting.

Design decisions

I made no changes for the structure of the Neural Network, I kept it in its original form.
As for the hyperparameters we have the learning rate, epochs, batch size and the training set size.
I didn't implement a dropout layer, as I found out, through the experiments, that by removing it or setting the dropout rate to 0 I get overall better results.
The values I used for the hyperparameters are:

- Learning rate = 0.01
- Epochs = 20
- Batch size = 128
- Training set size = 7000

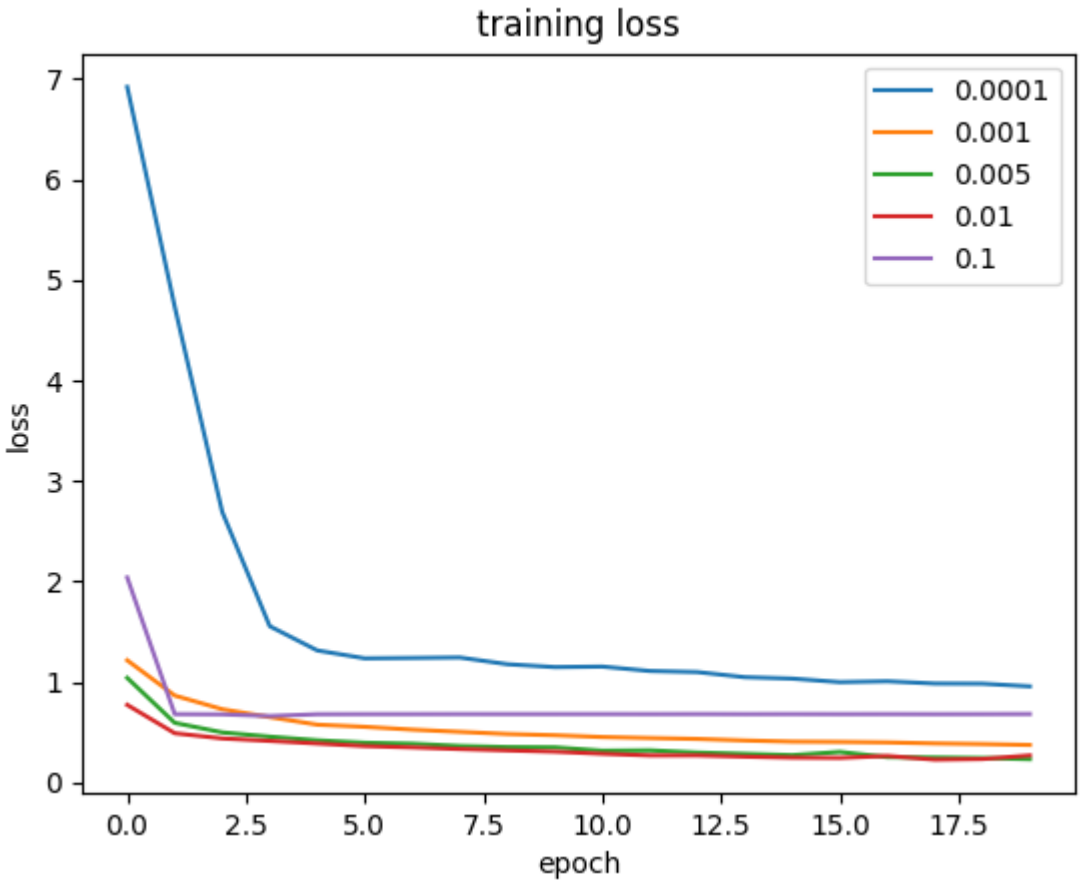
The reason I used these values is because they were the optimal result from the experiments I conducted, which you will see in the **Experiments** chapter in the report.

Experiments

Finding the optimal learning rate:

```
from matplotlib import pyplot as plt

all_lr = [0.0001, 0.001, 0.005, 0.01, 0.1]
all_loss = []
all_acc = []

for lr in all_lr:
    tf.keras.backend.clear_session()
    model = models.Sequential() # linear sequence of layers
    model.add(layers.Dense(100, activation='relu', input_shape=(6,)))
    model.add(layers.Dense(1, activation='sigmoid')) # output probability
    opt = tf.keras.optimizers.Adam(learning_rate=lr)
    model.compile(optimizer=opt, loss='binary_crossentropy',
metrics=['accuracy'])

    history = model.fit(np.array(dataset[0]), np.array(dataset[1]), epochs=20,
batch_size=128)
    all_loss.append(history.history['loss'])


for loss in all_loss:
    plt.plot(loss)

plt.title('training loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(all_lr, loc='upper right')
plt.show()

return model
```

By implementing the code above I got the following results:



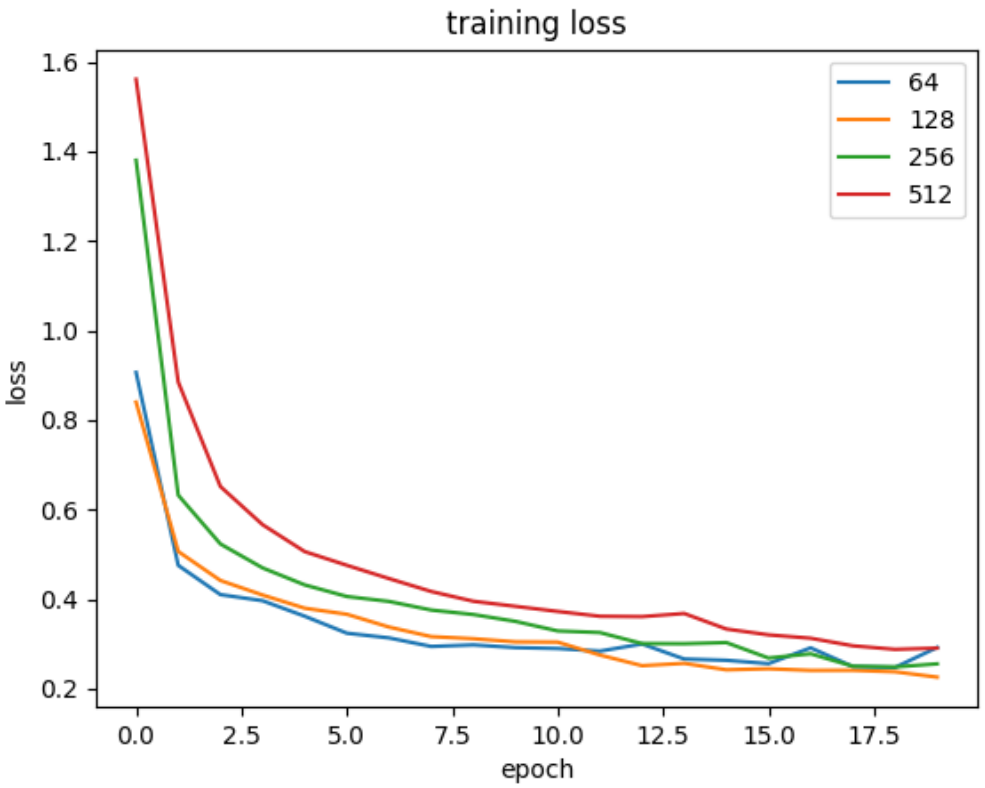
In the diagram you can see that 0.01(red line) has overall the lowest loss. Therefore, the optimal learning rate is 0.01.

Finding the optimal batch size:

```
all_batch = [64, 128, 256, 512]
all_loss = []

for batch in all_batch:
    tf.keras.backend.clear_session()
    model = models.Sequential() # linear sequence of layers
    model.add(layers.Dense(100, activation='relu', input_shape=(6,)))
    model.add(layers.Dense(1, activation='sigmoid')) # output
probability
    opt = tf.keras.optimizers.Adam(learning_rate=0.01)
    model.compile(optimizer=opt, loss='binary_crossentropy',
metrics=['accuracy'])

    history = model.fit(np.array(dataset[0]), np.array(dataset[1]),
epochs=20, batch_size=batch)
    all_loss.append(history.history['loss'])
```

```
from matplotlib import pyplot as plt
for loss in all_loss:
    plt.plot(loss)

plt.title('training loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(all_batch, loc='upper right')
plt.show()

return model
```

By implementing the code above I got the following results:



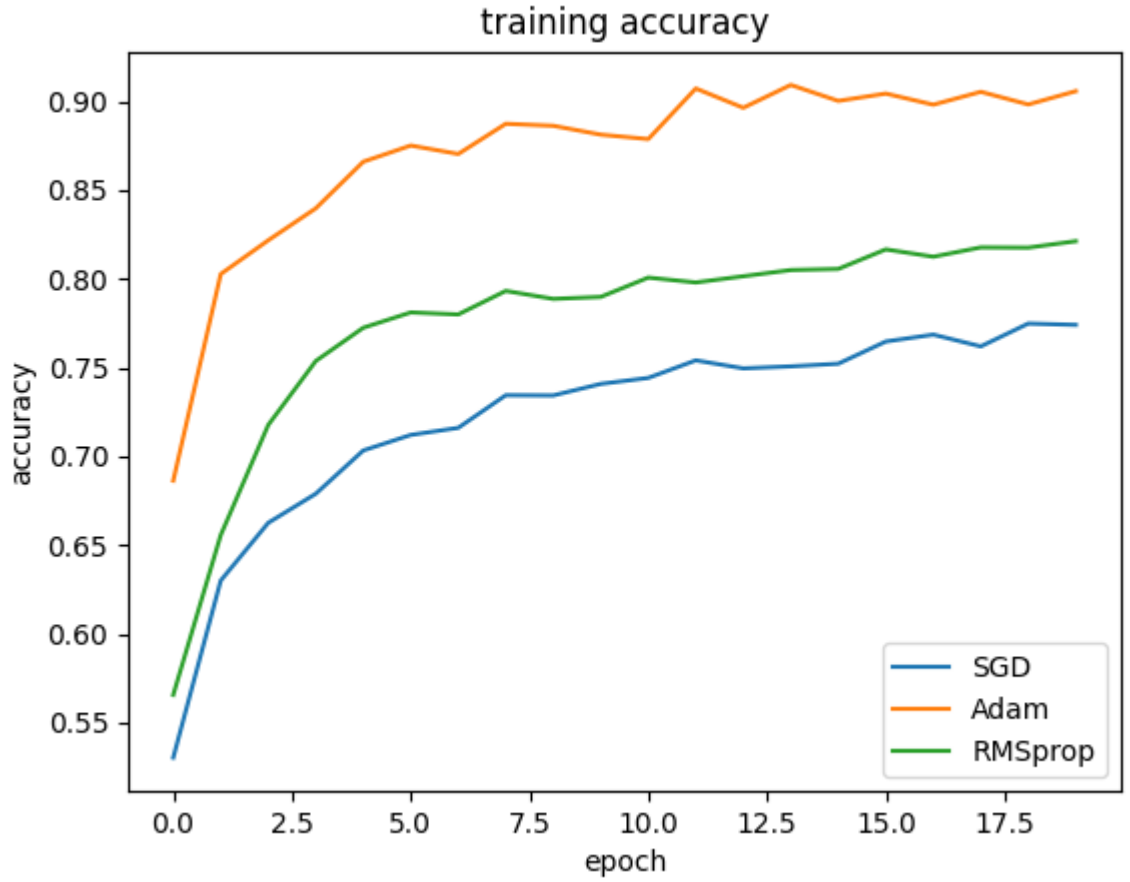
In the diagram you can see that 128(orange line) has overall the lowest loss. Therefore, the optimal batch size is 128.

Finding the optimal optimizer:

```
lr = 0.01
all_opt = [tf.keras.optimizers.SGD(learning_rate=lr),
tf.keras.optimizers.Adam(learning_rate=lr),
tf.keras.optimizers.RMSprop(learning_rate=lr)]
all_history = []
```

```
all_acc = []

for opt in all_opt:
    tf.keras.backend.clear_session()
    model = models.Sequential() # linear sequence of layers
    model.add(layers.Dense(100, activation='relu', input_shape=(6,)))
    model.add(layers.Dense(1, activation='sigmoid')) # output
probability
    model.compile(optimizer=opt, loss='binary_crossentropy',
metrics=['accuracy'])

    history = model.fit(np.array(dataset[0]), np.array(dataset[1]),
epochs=20, batch_size=128)
    all_history.append(history.history)

train_acc = []
from matplotlib import pyplot as plt

for history in all_history:
    plt.plot(history['accuracy'])

plt.title('training accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(["SGD", "Adam", "RMSprop"], loc='lower right')
plt.show()

return model
```

By implementing the code above I got the following results:



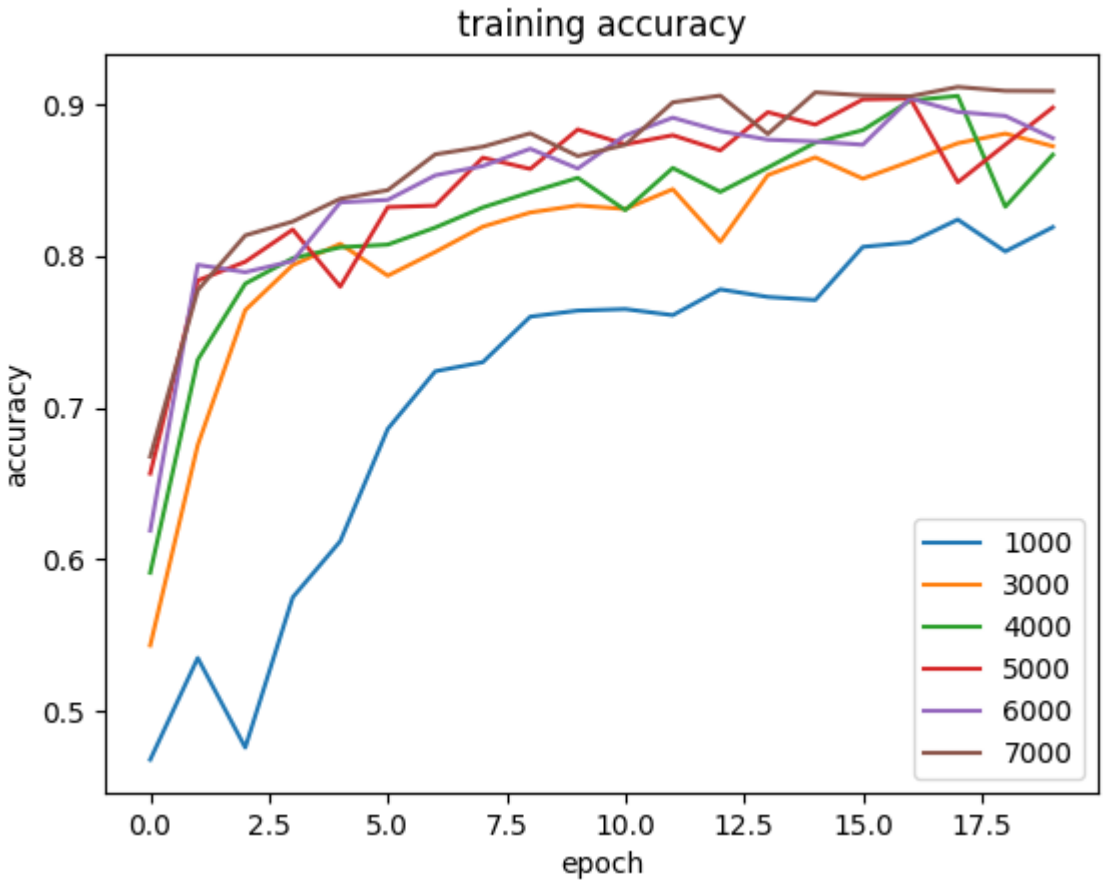
In the diagram you can see that the Adam optimizer(orange line) has overall the highest accuracy. Therefore, the optimal optimizer is the Adam optimizer.

Finding the optimal number of training set:

```
all_training_set_size = [1000, 3000, 4000, 5000, 6000, 7000]
all_history = []

for training_set_size in all_training_set_size:
    tf.keras.backend.clear_session()
    model = models.Sequential() # linear sequence of layers
    model.add(layers.Dense(100, activation='relu', input_shape=(6,)))
    model.add(layers.Dense(1, activation='sigmoid')) # output
probability
    opt = tf.keras.optimizers.Adam(learning_rate=0.01)
    model.compile(optimizer=opt, loss='binary_crossentropy',
metrics=['accuracy'])

    history = model.fit(dataset[0][:training_set_size],
dataset[1][:training_set_size], epochs=20, batch_size=128)
    all_history.append(history.history)
```

```
from matplotlib import pyplot as plt
train_acc = []
for history in all_history:
    plt.plot(history['accuracy'])
plt.title('training accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(all_training_set_size, loc='lower right')
plt.show()

return model
```

By implementing the code above I got the following results:

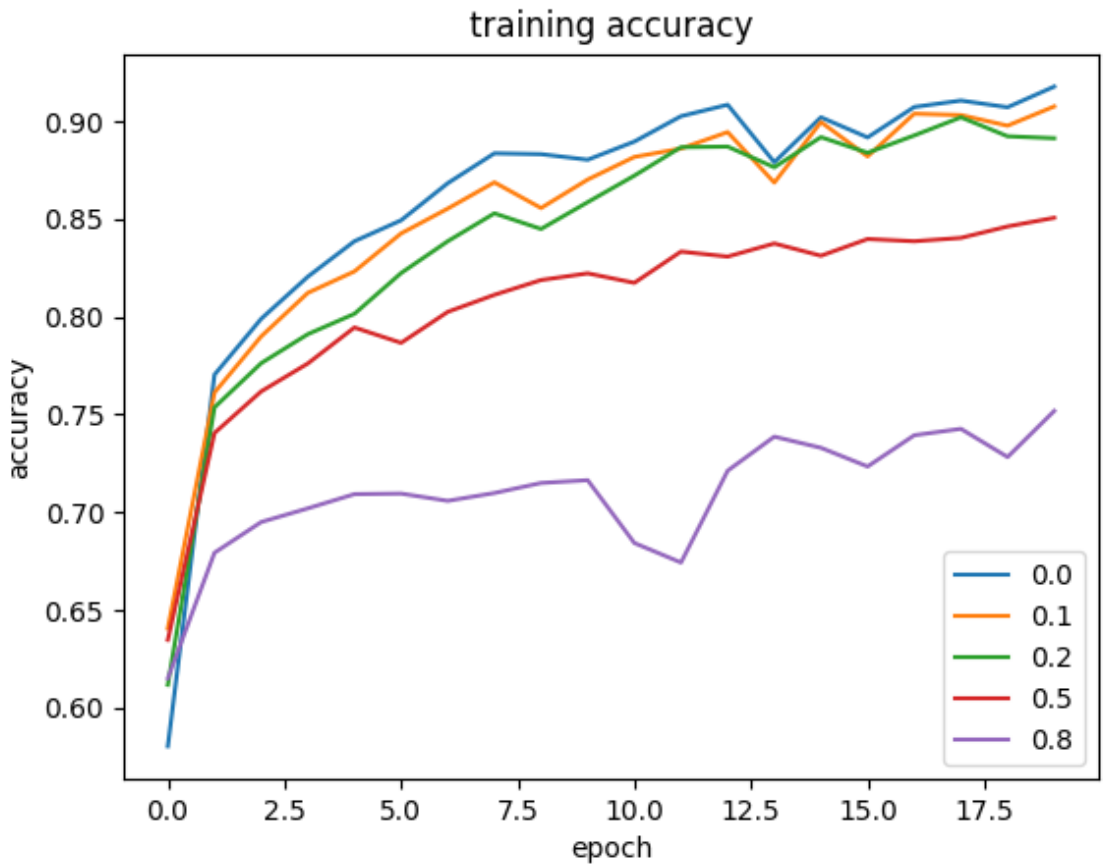


In the diagram you can see that the 7000(brown line)has overall the highest accuracy. Therefore, the optimal training set size is 7000.

Finding the optimal dropout rate:

```
all_dropout_rate = [0.0, 0.1, 0.2, 0.5, 0.8]
all_history = []

for dropout_rate in all_dropout_rate:
    tf.keras.backend.clear_session()
    model = models.Sequential() # linear sequence of layers
    model.add(layers.Dense(100, activation='relu', input_shape=(6,)))
    model.add(layers.Dropout(dropout_rate))
    model.add(layers.Dense(1, activation='sigmoid')) # output
probability
    opt = tf.keras.optimizers.Adam(learning_rate=0.01)
    model.compile(optimizer=opt, loss='binary_crossentropy',
metrics=['accuracy'])

    history = model.fit(dataset[0][:7000], dataset[1][:7000], epochs=20,
batch_size=128)
    all_history.append(history.history)


from matplotlib import pyplot as plt
train_acc = []
for history in all_history:
    plt.plot(history['accuracy'])
plt.title('training accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(all_dropout_rate, loc='lower right')
plt.show()

return model
```

By implementing the code above I got the following results:

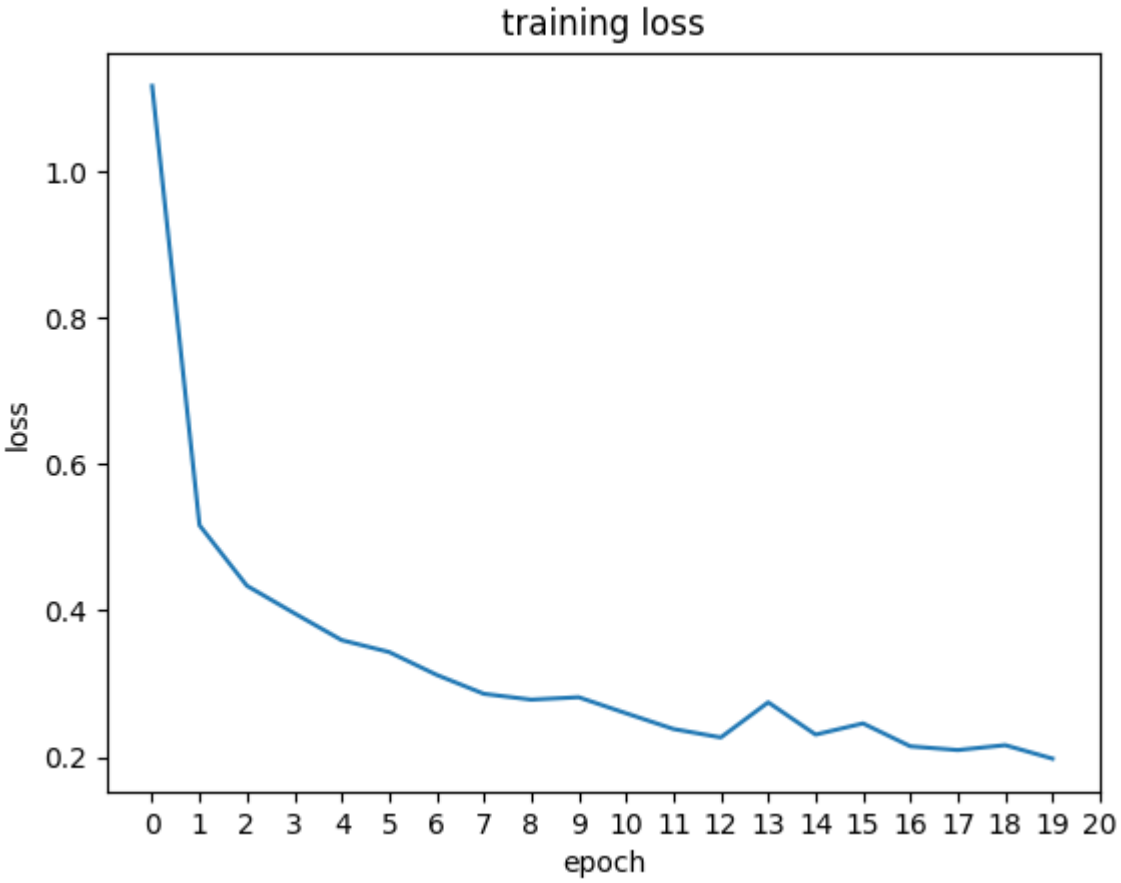


In the diagram you can see that the 0.0(blue line)has overall the highest accuracy. Since having a dropout rate of 0 is the same as having no dropout layer I decided to remove the dropout layer.

Final Solution

1) The following code allows us to plot the loss score per epochs on the training set.

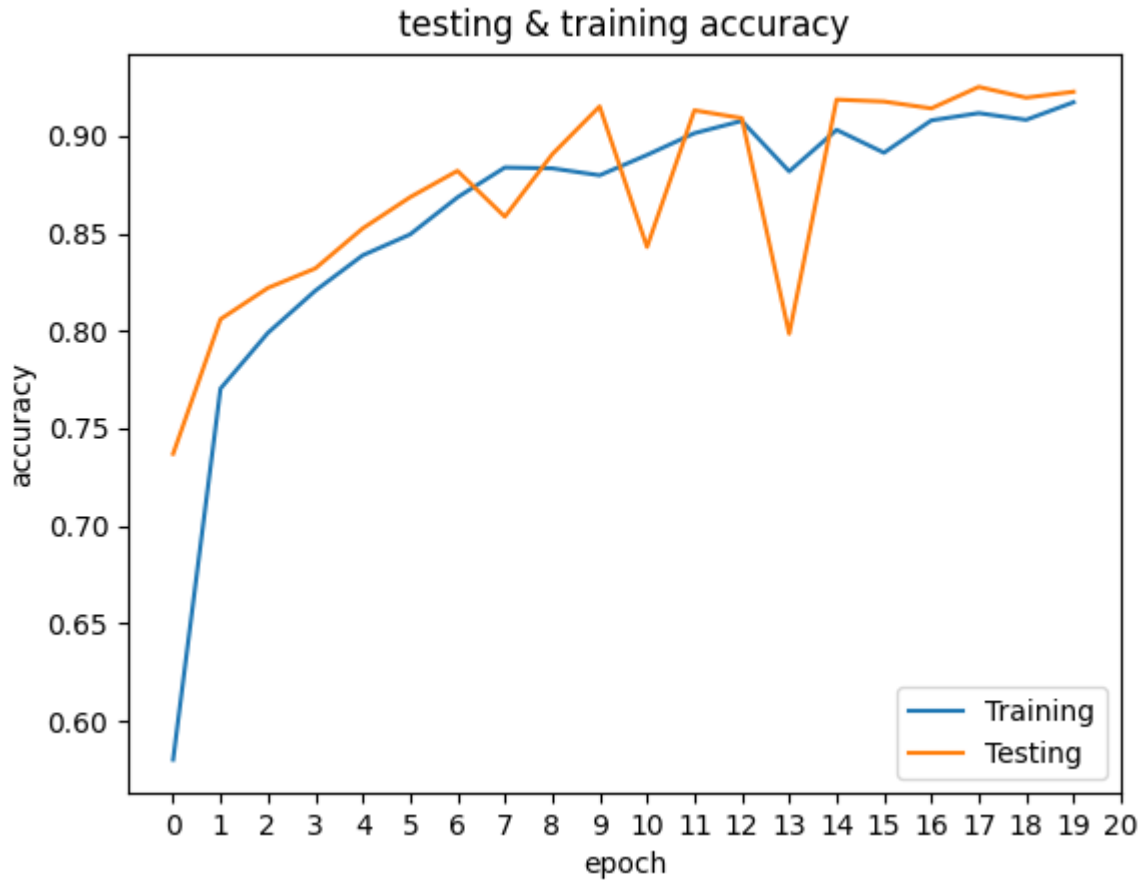
```
    all_loss = []

    tf.keras.backend.clear_session()
    model = models.Sequential() # linear sequence of layers
    model.add(layers.Dense(100, activation='relu', input_shape=(6,)))
    model.add(layers.Dense(1, activation='sigmoid')) # output
probability
    opt = tf.keras.optimizers.Adam(learning_rate=0.01)
    model.compile(optimizer=opt, loss='binary_crossentropy',
metrics=['accuracy'])

    history = model.fit(dataset[0][:7000], dataset[1][:7000], epochs=20,
batch_size=128)
    all_loss.append(history.history['loss'])

    from matplotlib import pyplot as plt
    epochs = list(range(0, 20+1))
    for loss in all_loss:
        plt.plot(loss)
    plt.title('training loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.xticks(epochs)
    plt.show()

    return model
```



2) Next we use the code below to create a plot for predicting the accuracy per epoch on the training and testing set.

```
    all_history = []

    tf.keras.backend.clear_session()
    model = models.Sequential() # linear sequence of layers
    model.add(layers.Dense(100, activation='relu', input_shape=(6,)))
    model.add(layers.Dense(1, activation='sigmoid')) # output
probability
    opt = tf.keras.optimizers.Adam(learning_rate=0.01)
    model.compile(optimizer=opt, loss='binary_crossentropy',
metrics=['accuracy'])

    history = model.fit(dataset[0][:7000], dataset[1][:7000], epochs=20,
batch_size=128, validation_data=(dataset[2], dataset[3]))
    all_history.append(history.history)

    from matplotlib import pyplot as plt
    epochs = list(range(0, 20+1))
    for hist in all_history:
        plt.plot(hist['accuracy'])
```

```
        plt.plot(hist['val_accuracy'])
    plt.title('testing & training accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['Training', 'Testing'], loc='lower right')
    plt.xticks(epochs)
    plt.show()

    return model
```



Conclusion

By using the optimal values found with the experiments I was able to achieve a testing accuracy of 0.9240000247955322 and an overfitting penalty of 0.0.

```
Accuracy  -  test: 0.9240000247955322; training: 0.9202101230621338; overfitting: 0.0037899017333984375
Grade: 20.47059115241555 (/20.0 + 2.0)  (overfitting penalty: 0.0)
Done 0:00:02.377762
```