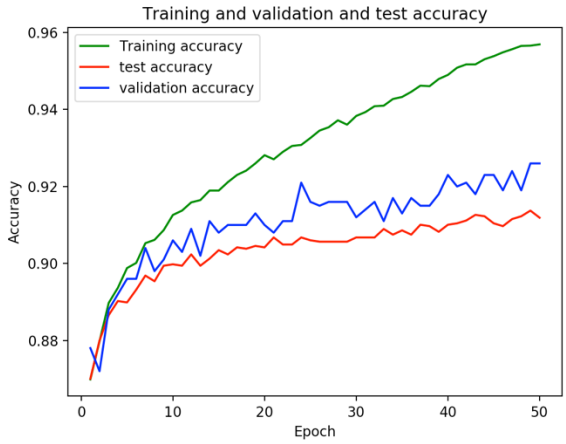
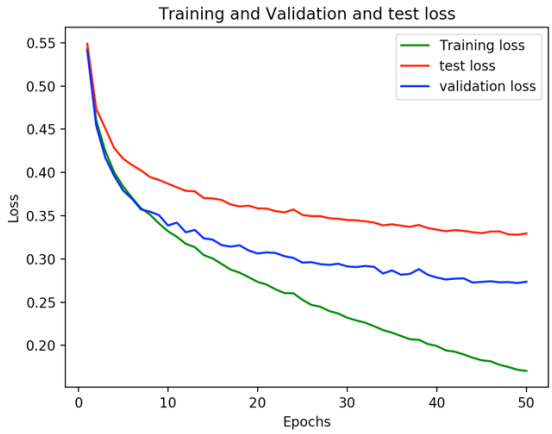
1. layer-wise building block: The function should accept two arguments, the input tensor and the number of the hidden units

   my function takes an extra parameter which is the target data. This could also be omitted

```
def my_function(X,y,units):

    class myCallback(tf.keras.callbacks.Callback):
        def on_epoch_end(self, epoch, logs={}):
            losstr, accuracy = model.evaluate(trainData, trainTarget)
            train_loss.append(losstr)
            train_acc.append(accuracy)

            losste, accuracy = model.evaluate(testData, testTarget)
            test_loss.append(losste)
            test_acc.append(accuracy)

            lossv, accuracy = model.evaluate(validData, validTarget)
            val_loss.append(lossv)
            val_acc.append(accuracy)


    callbacks = myCallback()

    model = Sequential()
    model.add(Dense(units,input_shape=(784,),kernel_initializer=glorot_normal(seed=None),
                bias_initializer='zeros', activation='relu'))
    model.add(Dense(10, activation='softmax'))


    sgd = SGD(lr=0.01)



    model.compile(optimizer=sgd, loss='sparse_categorical_crossentropy',
metrics=['accuracy'])







# now we need to train our model
# and check the validation data as well

    history = model.fit(X,y,validation_data=(validData,validTarget),
batch_size=batch_size, epochs = num_epoch , shuffle=False,callbacks=[callbacks])



#evaluate the keras model

    _, accuracy = model.evaluate(X, y)
    print('Accuracy: %.2f' % (accuracy*100))
```

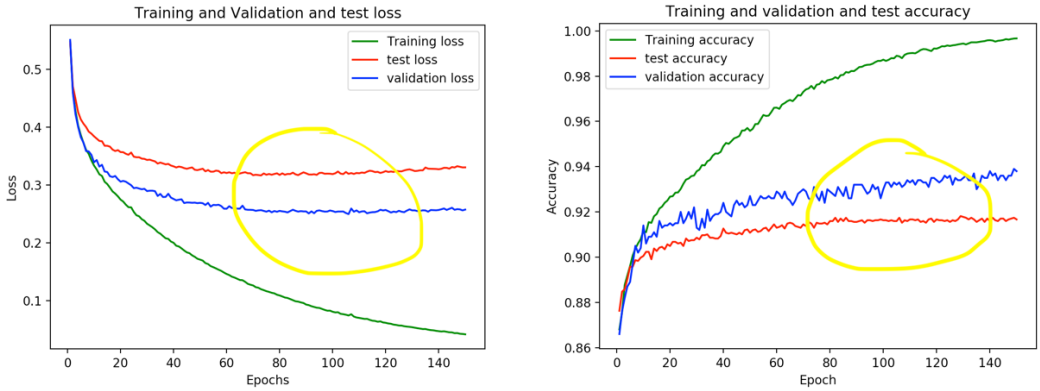
2. Learning. plot the training, validation, and test classification error vs. the number of epochs. Make a second plot for the cross-entropy loss vs. the number of epochs. Comment on your observations





Based on the graph show lower loss is better as long as there is no over fitting. Our loss function has been computed based on the training data as well as the validation. Our main goal here was to minizine the loss function with the use of hyperparameters and changing the weights via optimization methods. The loss value here shows how good or bad a model has worked on each step of optimization. Ideally, we expect reduction in loss over time. Accuracy, unlike the loss is determine once the model parameters have been learned. Once the learning stops the tests samples are check against the model to see how accurate their prediction will be. Hence a loss and accuracy for classification of test data will be generated. Based on the graph we can say that no over fitting has occurred. that means the model has learn as oppose to just memorizing the result.

3. Early stopping: Are the early stopping points the same on the two plots? Why or why not? Which plot should be used for early stopping, and why?



Early stopping point is when the model stops improving. in other words, the validation loss will not improve after each iteration.

Looking at the above graph the early stopping points are not the same on the two plots.

Focusing on the loss function is a better way of determining the early stopping point. because the loss shows how good or bad a model is for prediction. As shown above after 70^{th} epochs, the loss function merely improved and even got worst by the end of 150^{th} epoch. On the other hand, we cannot see the same effect in the Accuracy plot.

4. Number of hidden units:

[100, 500, 1000] hidden units

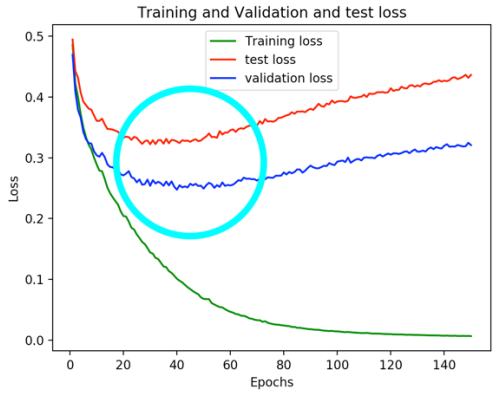
100 hidden units : validation error : 0.2766

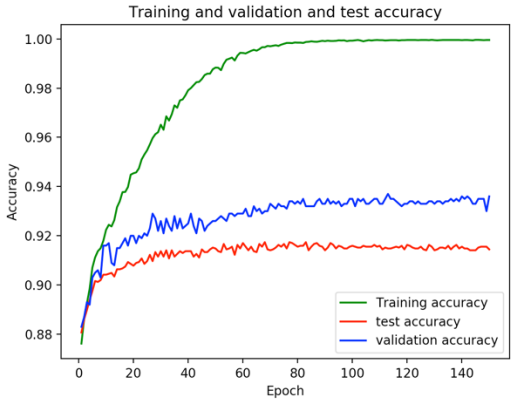
500 hidden units : validation error : 0.2572

1000 hidden units : validation error : 0.2614

Having a medium size approximately 2/3 size input data is a good number of hidden nodes based in order to get a good result. With 500 hidden units → Test error : 0.3387

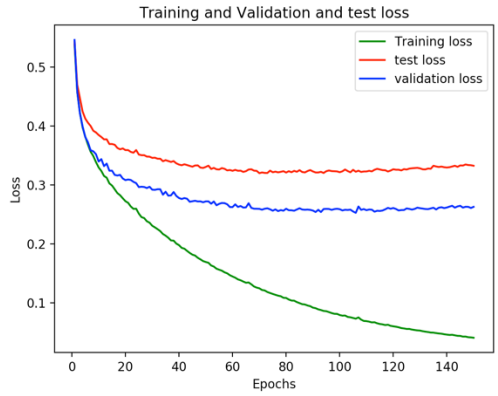
5. train a neural network with two hidden layers of 500 hidden units each. Using the test set, compare this architecture with the one-layer case.

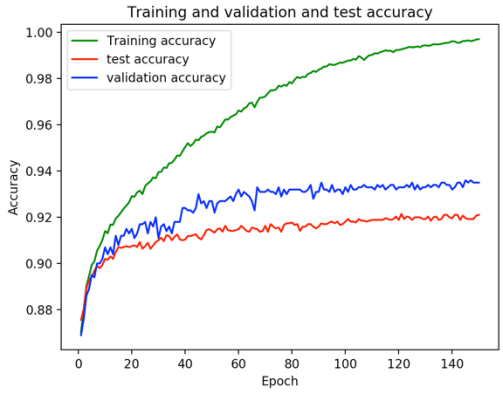




*The light blue shows the early stopping point

Test error : 0.4361 (2 x 500 units)
Valid error : 0.3207 (2 x 500 units)



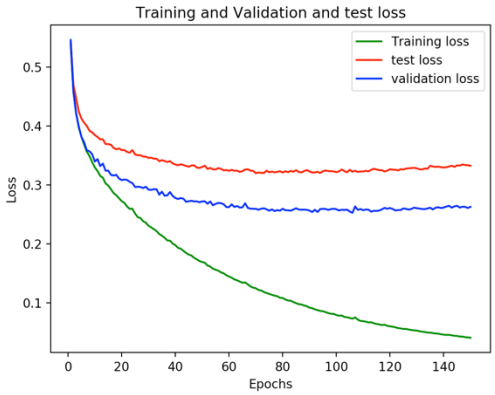


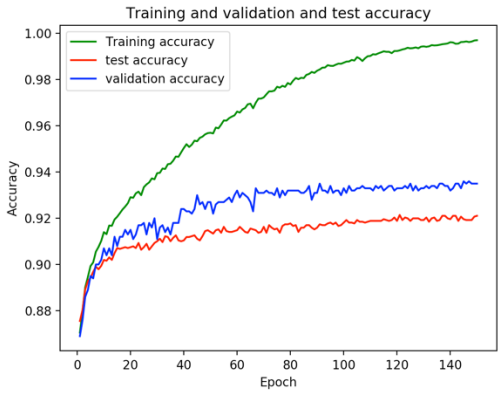
Test error : 0.3326 (1000 units)
Valid error : 0.2629 (1000 units)

According to the result, using two layers of each 500 units we obtained a test error of 0.4361 compare to error of 1 layer of 1000 units with test error of 0.3326
Therefore using 1 layer of 1000 units would be more beneficial compare to 2 layers of 500 each in order to obtain a lower test error.
It is also important to note that the early stopping happen earlier in the case of using 2 layers of 500 units.
Training based on more and more layers brings up the vanishing gradient problem as well. By using less neurons and more layers we risk having more information loss. However, such technique (using less neurons and more layers) adds more complexity to the model and make a more clever network.

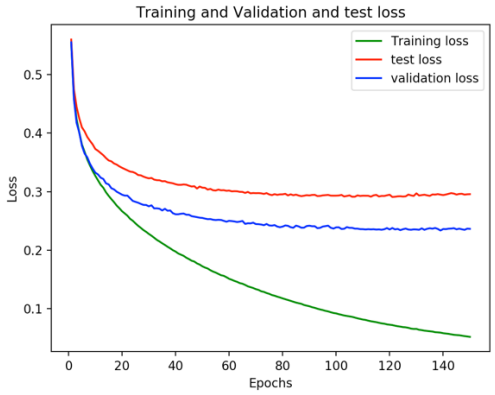
6. Dropout: Compare the results with the case that we do not have any dropout. Summarize the observed effect of dropout.

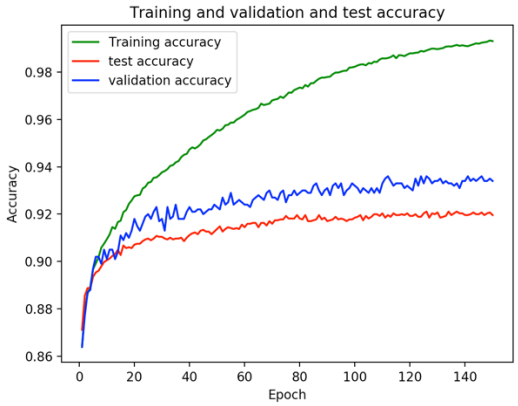
WITHOUT DROPOUT





Test error : 0.3326 (1000 units)
Valid error : 0.2629 (1000 units)





Test error : 0.2956 (1000 units)
Valid error : 0.2363 (1000 units)

As seen in the first pair of plot of loss and accuracy, after about the 80th epoch, there was no more improvement in the loss in test and validation data, however, the loss for training data got improved more and more and got closer and close to 0. This effect can be explained by the overfitting phenomena in which the model will get extremely tuned with the training data but not the tests data (specific data vs general data). Using a dropout technique, we can make sure on each iteration a portion of the training data are omitted so that it will be less likely for the model to memorize the desired output based on the training data. Hence the result will be more likely to be generalized.