EE2211 Tutorial 12

Dr Feng LIN

Q1

The convolutional neural network is particularly useful for applications related to image and text processing due to its dense connections.

a) True

b) False

Q1

The convolutional neural network is particularly useful for applications related to image and text processing due to its dense connections.

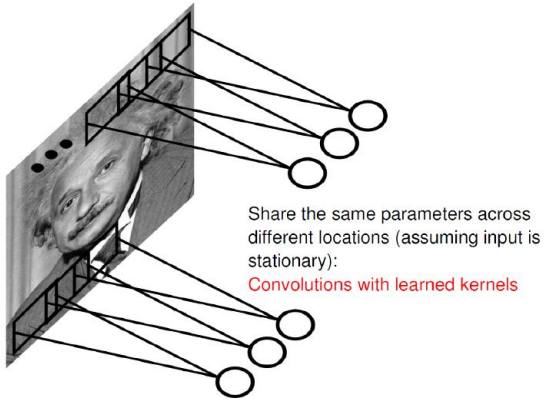
a) True

b) False

Ans: b).

Convolutional Neural Network (CNN)

- Hence, we introduce CNN to reduce the number of parameters.
- Works in **a sliding-window manner**!



Q2

In neural networks, nonlinear activation functions such as sigmoid, and ReLU

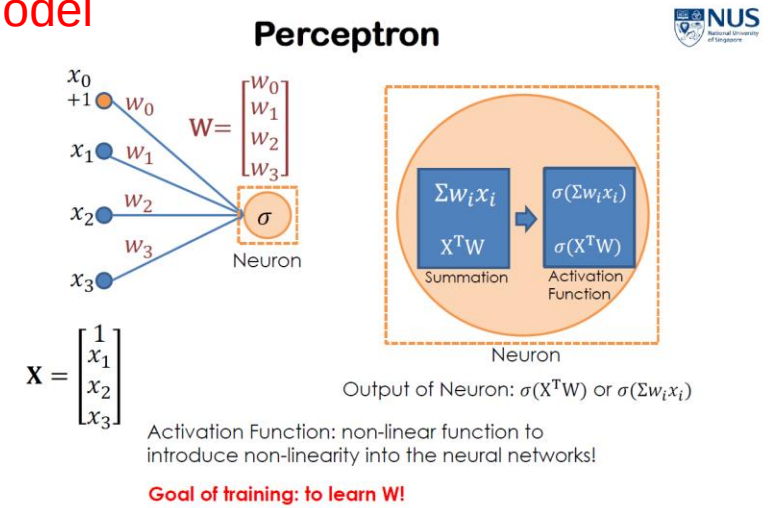
a) speed up the gradient calculation in backpropagation, as compared to linear units
b) are applied only to the output units
c) help to introduce non-linearity into the model
d) always output values between 0 and 1

Q2

In neural networks, nonlinear activation functions such as sigmoid, and ReLU

a) speed up the gradient calculation in backpropagation, as compared to linear units
b) are applied only to the output units
c) help to introduce non-linearity into the model
d) always output values between 0 and 1

Ans: c.



Q3

Question 3: A fully connected network of 2 layers has been constructed as

$$\boldsymbol{F}_{\mathbf{w}}(\mathbf{X}) = f(f(\mathbf{X}\mathbf{W}_1)\mathbf{W}_2)$$

where $\mathbf{X} = \begin{bmatrix} 1 & 1 & 3.0 \\ 1 & 2 & 2.5 \end{bmatrix}$, $\mathbf{W}_1 = \mathbf{W}_2 = \begin{bmatrix} -1 & 0 & 1 \\ 0 & -1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$.

Suppose the Rectified Linear Unit (ReLU) has been used as the activation function (f) for all the nodes. Compute the network output matrix $\boldsymbol{F}_{\mathbf{w}}(\mathbf{X})$ (up to 1 decimal place for each entry) based on the given network weights and data.

$$\boldsymbol{F}_{\mathbf{w}}(\mathbf{X}) = \begin{bmatrix} blank1 & blank2 & blank3 \\ blank4 & blank5 & blank6 \end{bmatrix}$$

Q3

$$f(\mathbf{X}\mathbf{W}_1) = f\left(\begin{bmatrix}1 & 1 & 3.0\\ 1 & 2 & 2.5\end{bmatrix}\begin{bmatrix}-1 & 0 & 1\\ 0 & -1 & 0\\ 1 & 0 & 1\end{bmatrix}\right)$$

$$= f\left(\begin{bmatrix}2 & -1 & 4.0\\ 1.5 & -2 & 3.5\end{bmatrix}\right)$$

$$= \begin{bmatrix}2 & 0 & 4.0\\ 1.5 & 0 & 3.5\end{bmatrix}$$

$$f(f(\mathbf{X}\mathbf{W}_1)\mathbf{W}_2) = f\left(\begin{bmatrix}2 & 0 & 4.0\\ 1.5 & 0 & 3.5\end{bmatrix}\begin{bmatrix}-1 & 0 & 1\\ 0 & -1 & 0\\ 1 & 0 & 1\end{bmatrix}\right)$$

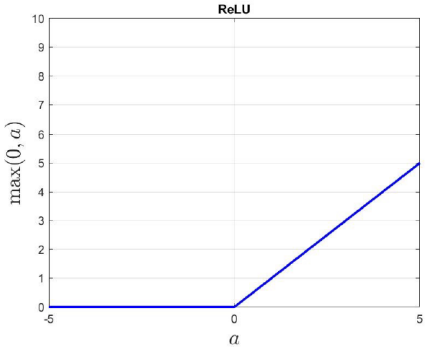
$$= f\left(\begin{bmatrix}2 & 0 & 6\\ 2 & 0 & 5\end{bmatrix}\right)$$

$$= \begin{bmatrix}2 & 0 & 6\\ 2 & 0 & 5\end{bmatrix}$$

ReLU Activation Function

$\sigma(a) = \max(0, a)$

Rectified Linear Unit (ReLU)



Q3

```
import numpy as np
X = np.array([[1,1,3],[1,2,2.5]])
W = np.array([[-1,0,1],[0,-1,0],[1,0,1]])

Ftemp = X@W
F1 = np.zeros((2,3))
for i in range(0,2,1):
    for j in range(0,3,1):
        F1[i,j] = max(0.0,Ftemp[i,j])

Ftemp = F1@W
F2 = np.zeros((2,3))
for i in range(0,2,1):
    for j in range(0,3,1):
        F2[i,j] = max(0.0,Ftemp[i,j])

print(F1)
print(F2)
```

Q4

Question 4: A fully connected network of 3 layers has been constructed as

$$\boldsymbol{F}_{\mathbf{w}}(\mathbf{X}) = f([\mathbf{1}, f([\mathbf{1}, f(\mathbf{X}\mathbf{W}_1)]\mathbf{W}_2)]\mathbf{W}_3)$$

where $\mathbf{X} = \begin{bmatrix} 1 & 2 & 1 \\ 1 & 5 & 1 \end{bmatrix}$, $\mathbf{W}_1 = \begin{bmatrix} -1 & 0 & 1 \\ 0 & -1 & 0 \\ 1 & 0 & -1 \end{bmatrix}$, $\mathbf{W}_2 = \mathbf{W}_3 \begin{bmatrix} -1 & 0 & 1 \\ 0 & -1 & 0 \\ 1 & 0 & 1 \\ 1 & -1 & 1 \end{bmatrix}$.

Suppose the **Sigmoid** has been used as the activation function (f) for all the nodes. Compute the network output matrix $\boldsymbol{F}_{\mathbf{w}}(\mathbf{X})$ (up to 1 decimal place for each entry) based on the given network weights and data.

$$\boldsymbol{F}_{\mathbf{w}}(\mathbf{X}) = \begin{bmatrix} blank1 & blank2 & blank3 \\ blank4 & blank5 & blank6 \end{bmatrix}$$

Q4

$$f(\mathbf{X}\mathbf{W}_1) = f\left(\begin{bmatrix}1 & 2 & 1\\ 1 & 5 & 1\end{bmatrix}\begin{bmatrix}-1 & 0 & 1\\ 0 & -1 & 0\\ 1 & 0 & -1\end{bmatrix}\right)$$

$$= f\left(\begin{bmatrix}0 & -2 & 0\\ 0 & -5 & 0\end{bmatrix}\right)$$

$$= \begin{bmatrix}0.5 & 0.1192 & 0.5\\ 0.5 & 0.0067 & 0.5\end{bmatrix}$$

$$f([\mathbf{1}, f(\mathbf{X}\mathbf{W}_1)]\mathbf{W}_2) = f\left(\begin{bmatrix}1 & 0.5 & 0.1192 & 0.5\\ 1 & 0.5 & 0.0067 & 0.5\end{bmatrix}\begin{bmatrix}-1 & 0 & 1\\ 0 & -1 & 0\\ 1 & 0 & 1\\ 1 & -1 & 1\end{bmatrix}\right)$$

$$= f\left(\begin{bmatrix}-0.3808 & -1.0000 & 1.6192\\ -0.4933 & -1.0000 & 1.5067\end{bmatrix}\right)$$

$$= \begin{bmatrix}0.4059 & 0.2689 & 0.8347\\ 0.3791 & 0.2689 & 0.8186\end{bmatrix}$$

$$f([\mathbf{1}, f([\mathbf{1}, f(\mathbf{X}\mathbf{W}_1)]\mathbf{W}_2)]\mathbf{W}_3) = f\left(\begin{bmatrix}1 & 0.4059 & 0.2689 & 0.8347\\ 1 & 0.3791 & 0.2689 & 0.8186\end{bmatrix}\begin{bmatrix}-1 & 0 & 1\\ 0 & -1 & 0\\ 1 & 0 & 1\\ 1 & -1 & 1\end{bmatrix}\right)$$

$$= \begin{bmatrix}0.5259 & 0.2243 & 0.8913\\ 0.5219 & 0.2319 & 0.8897\end{bmatrix}$$

Q4

```
import numpy as np
beta = 1

X = np.array([[1,2,1],[1,5,1]])
W1 = np.array([[-1,0,1],[0,-1,0],[1,0,-1]])
W23 = np.array([[-1,0,1],[0,-1,0],[1,0,1],[1,-1,1]])

# sigmoid 1/1+exp(-beta*a)
Ftemp = X@W1
F1 = np.hstack((np.ones((2,1)),1/(1 + np.exp(-beta*Ftemp))))
print(F1)
Ftemp = F1@W23
F2 = np.hstack((np.ones((2,1)),1/(1 + np.exp(-beta*Ftemp))))
print(F2)
Ftemp = F2@W23
F3 = 1/(1 + np.exp(-beta*Ftemp))
print(F3)
```

Q5

(MLP classifier, find the best hidden node size, assuming same hidden layer size in each layer, based on cross-validation on the training set and then use it for testing)

Obtain the data set "**from sklearn.datasets import load_iris**".

a) Split the database into two sets: 80% of samples for training, and 20% of samples for testing using random_state=0

b) Perform a 5-fold Cross-validation using only the training set to determine the best 3-layer **MLPClassifier** (**from sklearn.neural_network import MLPClassifier with hidden_layer_sizes = (Nhidd,Nhidd,Nhidd) for Nhidd in range(1,11)**)* for prediction. In other words, partition the **training set** into two sets, 4/5 for training and 1/5 for validation; and repeat this process until each of the 1/5 has been validated.Provide a plot of the average 5-fold training and validation accuracies over the different network sizes.

c) Find the size of **Nhidd** that gives the best validation accuracy for the training set.

d) Use this **Nhidd** in the **MLPClassifier** with **hidden_layer_sizes=(Nhidd,Nhidd,Nhidd)** to compute the prediction accuracy based on the 20% of samples for testing in part (a).

* The assumption of **hidden_layer_sizes=(Nhidd,Nhidd,Nhidd)** is to reduce the search space in this exercise. In field applications, the search should take different sizes for each hidden layer.

Q5

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn import metrics
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import warnings

warnings.filterwarnings("ignore",category=UserWarning)

df = load_iris()
Xdata = df['data']
Ydata = df['target']

acc_train_array = np.zeros(10)
acc_valid_array = np.zeros(10)

X_train, X_test, y_train, y_test = train_test_split(Xdata,
Ydata,train_size=.8,test_size=.2)

validation_size = np.floor(len(y_train)/5)
train_size = len(y_train) - validation_size
```

```
for Nhidd in range (1,11,1):
    clf = MLPClassifier(hidden_layer_sizes=(Nhidd,Nhidd,Nhidd),solver='lbfgs', alpha=1e-5,random_state=1,max_iter=10000)
    acc_train_array_fold = 0
    acc_valid_array_fold = 0
    for fold in range (1,6,1):
        # Construct validation and training matrices
        # int() to eliminate floating point portion, i.e. X.0 -> X
        Y_validation = y_train[(fold-1)*int(validation_size):int(validation_size)*fold]
        X_validation = X_train[(fold-1)*int(validation_size):int(validation_size)*fold,:]
        X_training = []
        Y_training = []
        Y_training = [y_train[Idx] for Idx in range (0,int(train_size),1) if ((Idx < (fold-1)*int(validation_size)) or (Idx >=
int(validation_size)*fold)) ]
        X_training = [X_train[Idx,:] for Idx in range (0,int(train_size),1) if ((Idx < (fold-1)*int(validation_size)) or (Idx >=
int(validation_size)*fold)) ]
        Y_training = np.array(Y_training) # due to Y_training was in list format, need to convert them back to array
        X_training = np.array(X_training) # due to X_training was in list format, need to convert them back to array

        clf.fit(X_training,Y_training)
        ## trained output
        y_train_est = clf.predict(X_training)
        acc_train_array_fold += metrics.accuracy_score(y_train_est,Y_training)
        ## validation output
        y_valid_est = clf.predict(X_validation)
        acc_valid_array_fold += metrics.accuracy_score(y_valid_est,Y_validation)

    acc_train_array[Nhidd-1] = acc_train_array_fold/5
    acc_valid_array[Nhidd-1] = acc_valid_array_fold/5
```

- Nhidd represents the number of neurons per hidden layer, iterating from 1 to 10.
- MLPClassifier creates a neural network with three hidden layers, each containing Nhidd neurons.

- This loop splits the training data into training and validation sets for each fold.

- clf.fit(X_training, Y_training) trains a neural network classifier (MLPClassifier) on the given training data. Here's a breakdown of how this function works:

```
## find the size that gives the best validation accuracy
Nhidden = np.argmax(acc_valid_array,axis=0)+1
print('Number of neuron per hidden layer with best validation accuracy is',Nhidden)
clf = MLPClassifier(hidden_layer_sizes=(Nhidden,Nhidden,Nhidden),solver='lbfgs', alpha=1e-
5,random_state=1,max_iter=10000)
clf.fit(X_train,y_train)
y_train_est = clf.predict(X_train)
y_test_est = clf.predict(X_test)
acc_train = metrics.accuracy_score(y_train_est,y_train)
acc_test = metrics.accuracy_score(y_test_est,y_test)
print('Test accuracy for Nhidd = ',Nhidden,' is ',acc_test*100,' %')

# plotting
hiddensize = [x for x in range(1,11)]
plt.plot(hiddensize, acc_train_array, color='blue', marker='o', linewidth=3, label='Training')
plt.plot(hiddensize, acc_valid_array, color='orange', marker='x', linewidth=3,    label='Validation')
plt.plot(Nhidden,acc_train, color='green', marker='o', markersize = 10, label ='Final Training')
plt.plot(Nhidden,acc_train, color='red', marker='X', markersize = 10,label = 'Test')
plt.xlabel('Number of hidden nodes in each layer')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracies')
plt.legend()
plt.show()
```

- Nhidden selects the number of neurons per layer that yields the highest validation accuracy.
- The final model is then trained on the entire training set and evaluated on both the training and test sets.

Q5


```
Number of neuron per hidden layer with best validation accuracy is 10
Test accuracy for Nhidd =  10  is  100.0  %
```

Q6

(An example of handwritten digit image classification using CNN)

Please go through the baseline example in the following link to get a feel of how the Convolutional Neural Network (CNN) can be used for handwritten digit image classification.

https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-from-scratch-for-mnist-handwritten-digit-classification/

Note: This example assumes that you are using standalone Keras running on top of TensorFlow with Python 3 (you might need **conda install -c condaforge keras tensorflow** to get the Keras library installed). The following codes might be useful for warnings suppression if you find them annoying:

```
import warnings

warnings.filterwarnings("ignore",category=UserWarning)
```

As the data size and the network size are relatively large comparing with previous assignments, the codes can take quite some time to run (e.g., several minutes running on the latest notebook).

Q6

```
# load train and test dataset
def load_dataset():
    # load dataset
    (trainX, trainY), (testX, testY) = mnist.load_data()
    # reshape dataset to have a single channel
    trainX = trainX.reshape((trainX.shape[0], 28, 28, 1))
    testX = testX.reshape((testX.shape[0], 28, 28, 1))
    # one hot encode target values
    trainY = to_categorical(trainY)
    testY = to_categorical(testY)
    return trainX, trainY, testX, testY
```

- load the images and reshape the data arrays to have a single color channel.
- use a one hot encoding for the class element of each sample, transforming the integer into a 10 element binary vector with a 1 for the index of the class value, and 0 values for all other classes.

Q6

```
# scale pixels
def prep_pixels(train, test):
    # convert from integers to floats
    train_norm = train.astype('float32')
    test_norm = test.astype('float32')
    # normalize to range 0-1
    train_norm = train_norm / 255.0
    test_norm = test_norm / 255.0
    # return normalized images
    return train_norm, test_norm
```

- Nsormalize the pixel values of grayscale images, e.g. rescale them to the range [0,1]

Q6

```
# define cnn model
def define_model():
    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu',
kernel_initializer='he_uniform', input_shape=(28, 28, 1)))
    model.add(MaxPooling2D((2, 2)))
    model.add(Flatten())
    model.add(Dense(100, activation='relu', kernel_initializer='he_uniform'))
    model.add(Dense(10, activation='softmax'))
    # compile model
    opt = SGD(learning_rate=0.01, momentum=0.9)
    model.compile(optimizer=opt, loss='categorical_crossentropy',
metrics=['accuracy'])
    return model
```

Define **a baseline convolutional neural network model** for the problem. The model has two main aspects:

- The feature extraction front end comprised of convolutional and pooling layers,
- The classifier backend that will make a prediction.

```
# evaluate a model using k-fold cross-validation
def evaluate_model(dataX, dataY, n_folds=5):
    scores, histories = list(), list()
    # prepare cross validation
    kfold = KFold(n_folds, shuffle=True, random_state=1)
    # enumerate splits
    for train_ix, test_ix in kfold.split(dataX):
        # define model
        model = define_model()
        # select rows for train and test
        trainX, trainY, testX, testY = dataX[train_ix], dataY[train_ix],
dataX[test_ix], dataY[test_ix]
        # fit model
        history = model.fit(trainX, trainY, epochs=10, batch_size=32,
validation_data=(testX, testY), verbose=0)
        # evaluate model
        _, acc = model.evaluate(testX, testY, verbose=0)
        print('> %.3f' % (acc * 100.0))
        # stores scores
        scores.append(acc)
        histories.append(history)
    return scores, histories
```

The model will be evaluated using [five-fold cross-validation](). The value of *k=5* was chosen to provide a baseline for both repeated evaluation and to not be so large as to require a long running time. Each test set will be 20% of the training datase or about 12,000 examples, close to the size of the actual test set for this problem.

Q6

```
# plot diagnostic learning curves
def summarize_diagnostics(histories):
    for i in range(len(histories)):
        # plot loss
        plt.subplot(2, 1, 1)
        plt.title('Cross Entropy Loss')
        plt.plot(histories[i].history['loss'], color='blue', label='train')
        plt.plot(histories[i].history['val_loss'], color='orange', label='test')
        # plot accuracy
        plt.subplot(2, 1, 2)
        plt.title('Classification Accuracy')
        plt.plot(histories[i].history['accuracy'], color='blue', label='train')
        plt.plot(histories[i].history['val_accuracy'], color='orange', label='test')
    plt.show()
```

Once the model has been evaluated, we can present the results. There are two key aspects to present:

- the diagnostics of the learning behavior of the model during training
- the estimation of the model performance.

Q6

```
# summarize model performance
def summarize_performance(scores):
    # print summary
    print('Accuracy: mean=%.3f std=%.3f, n=%d' % (mean(scores)*100,
std(scores)*100, len(scores)))
    # box and whisker plots of results
    plt.boxplot(scores)
    plt.show()
```

- The classification accuracy scores collected during each fold can be summarized by calculating the mean and standard deviation. This provides an estimate of the average expected performance of the model trained on this dataset, with an estimate of the average variance in the mean.
- We will also summarize the distribution of scores by creating and showing a box and whisker plot.

Q6

```
# run the test harness for evaluating a model
def run_test_harness():
    # load dataset
    trainX, trainY, testX, testY = load_dataset()
    # prepare pixel data
    trainX, testX = prep_pixels(trainX, testX)
    # evaluate model
    scores, histories = evaluate_model(trainX, trainY)
    # learning curves
    summarize_diagnostics(histories)
    # summarize estimated performance
    summarize_performance(scores)

# entry point, run the test harness
run_test_harness()
```

This involves calling all of the define functions.

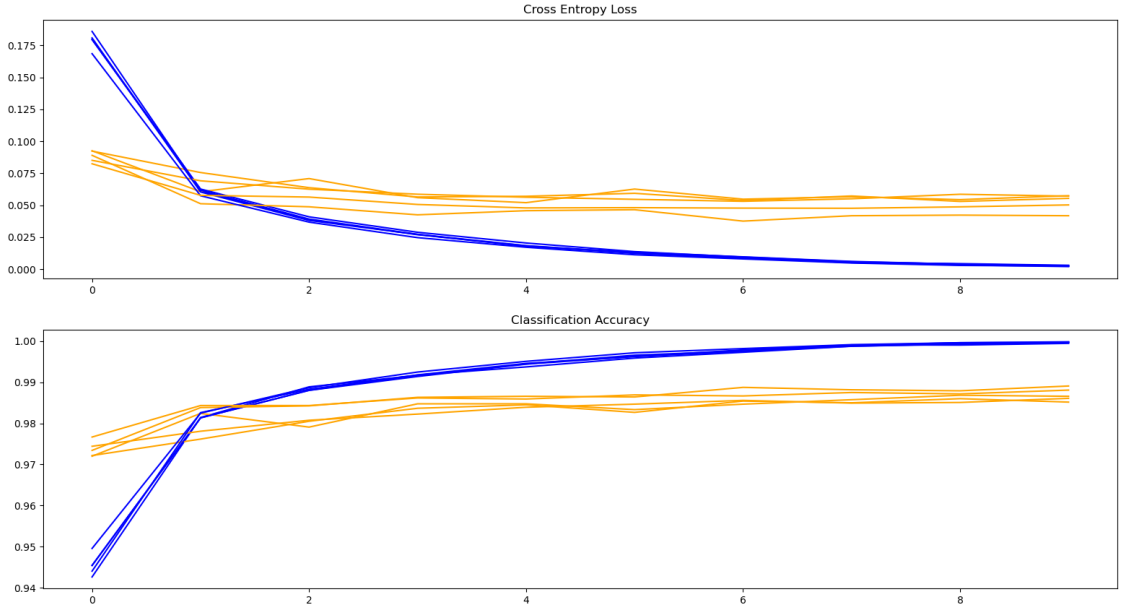
Q6

Running the example prints the classification accuracy for each fold of the cross-validation process. This is helpful to get an idea that the model evaluation is progressing

> 98.558
> 98.600
> 98.600
> 98.883
> 98.700
Accuracy: mean=98.668 std=0.117, n=5

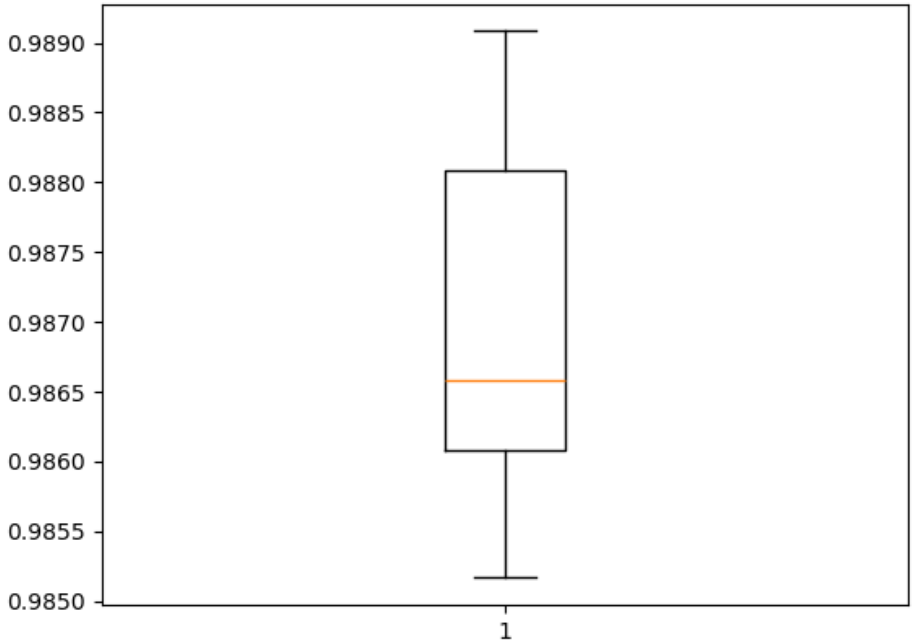
Q6

A diagnostic plot is shown, giving insight into the learning behavior of the model across each fold. In this case, we can see that the model generally achieves a good fit, with train and test learning curves converging. There is no obvious sign of over- or underfitting.



Q6

A box and whisker plot is created to summarize the distribution of accuracy scores.



THANK YOU