EE2211 Tutorial 12

Question 1: The convolutional neural network is particularly useful for applications related to image and text processing due to its dense connections.

a) True

b) False

↳ sparser (fewer) connections not Dense

Ans: b).

Question 2: 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 → nope, applied to each neuron

c) help to introduce non-linearity into the model

d) always output values between 0 and 1 → not always! while sigmoid does output (0,1), Relu outputs from 0 to ∞, etc

Ans: c.

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.

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

Answer:

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

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

→ ReLu = max(0, a) for each entry in matrix, hence -1, -2 become 0

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

matrix multiply ↓

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

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

$$= \begin{bmatrix} 2 & 0 & 6 \\ 2 & 0 & 5 \end{bmatrix}$$ → ReLu = max(0, a) for each entry in matrix, hence no change here

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.

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

Answer:

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

$$= f\left(\begin{bmatrix} 0 & -2 & 0 \\ 0 & -5 & 0 \end{bmatrix}\right)$$ → Sigmoid : $\frac{1}{1+e^{-\beta a}} = \frac{1}{1+e^{-a}}$ assume $\therefore \beta = 1$

$$= \begin{bmatrix} 0.5 & 0.1192 & 0.5 \\ 0.5 & 0.0067 & 0.5 \end{bmatrix}$$ ← for all entries

matrix multiply ↓

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

↓ bias

$$= f\left(\begin{bmatrix} -0.3808 & -1.0000 & 1.6192 \\ -0.4933 & -1.0000 & 1.5067 \end{bmatrix}\right)$$ → Sigmoid : $\frac{1}{1+e^{-\beta a}} = \frac{1}{1+e^{-a}}$ assume $\therefore \beta = 1$

$$= \begin{bmatrix} 0.4059 & 0.2689 & 0.8347 \\ 0.3791 & 0.2689 & 0.8186 \end{bmatrix}$$ ← for all entries

E.g. $\frac{1}{1+e^{-(-0.3808)}}$

$= 0.4059$

matrix multiplication

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

bias

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

(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)

Question 5:

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.

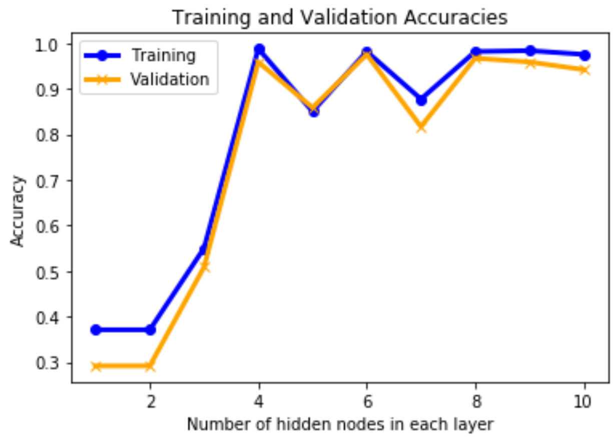
Answer:

```
## load data from scikit
import numpy as np
import pandas as pd
print("pandas version: {}".format(pd.__version__))
import sklearn
print("scikit-learn version: {}".format(sklearn.__version__))
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier # neural network
from sklearn import metrics
```

```
def find_network_size(X_train, y_train):
   acc_train_array = []
   acc_valid_array = []
   for Nhidd in range(1,11):
      acc_train_array_fold = []
      acc_valid_array_fold = []
      ## Random permutation of data
      Idx = np.random.RandomState(seed=8).permutation(len(y_train))
      ## Tuning: perform 5-fold cross-validation on the training set to determine the best network
size
      for k in range(0,5):
         N = np.around((k+1)*len(y_train)/5)
         N = N.astype(int)
         Xvalid = X_train[Idx[N-24:N]] # validation features
         Yvalid = y_train[Idx[N-24:N]] # validation targets
         Idxtrn = np.setdiff1d(Idx, Idx[N-24:N])
         Xtrain = X_train[Idxtrn] # training features in tuning loop
         Ytrain = y_train[Idxtrn] # training targets in tuning loop
         ## MLP Classification with same size for each hidden-layer (specified in question)
         clf = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(Nhidd,Nhidd,Nhidd),
random_state=1)
         clf.fit(Xtrain, Ytrain)
         ## trained output
         y_est_p = clf.predict(Xtrain)
         acc_train_array_fold += [metrics.accuracy_score(y_est_p,Ytrain)]
         ## validation output
         yt_est_p = clf.predict(Xvalid)
         acc_valid_array_fold += [metrics.accuracy_score(yt_est_p,Yvalid)]
      acc_train_array += [np.mean(acc_train_array_fold)]
      acc_valid_array += [np.mean(acc_valid_array_fold)]
   ## find the size that gives the best validation accuracy
   Nhidden = np.argmax(acc_valid_array,axis=0)+1

   ## plotting
   import matplotlib.pyplot as plt
   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.xlabel('Number of hidden nodes in each layer')
   plt.ylabel('Accuracy')
   plt.title('Training and Validation Accuracies')
   plt.legend()
   plt.show()
   return Nhidden

## load data
iris_dataset = load_iris()
## split dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(iris_dataset['data'],
                                                    iris_dataset['target'],
                                                    test_size=0.20,
                                                    random_state=0)
## find the best hidden node size using only the training set
Nhidden = find_network_size(X_train, y_train)
print('best hidden node size =', Nhidden, 'based on 5-fold cross-validation on training set')
## perform evaluation
clf  =  MLPClassifier(solver='lbfgs',  alpha=1e-5,  hidden_layer_sizes=(Nhidden,Nhidden,Nhidden),
random_state=1)
clf.fit(X_train, y_train)
## trained output
y_test_predict = clf.predict(X_test)
test_accuracy = metrics.accuracy_score(y_test_predict,y_test)
print('test accuracy =', test_accuracy)
```




```
>> best hidden node size = 6 based on 5-fold cross-validation on training set
>> test accuracy = 1.0
```

(An example of handwritten digit image classification using CNN)

Question 6:

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 conda-forge keras tensorflow` to get the Keras library installed).

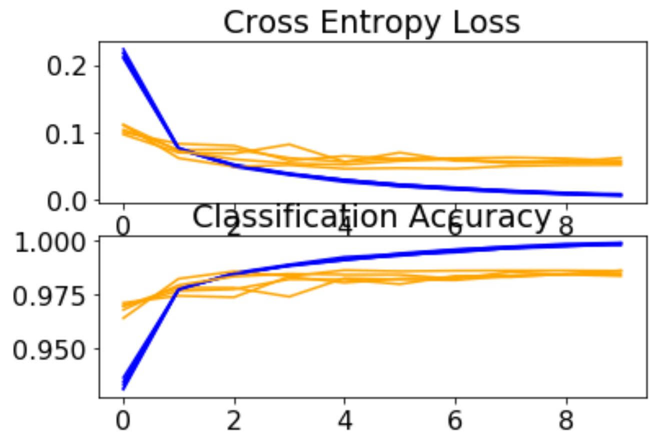
The following codes might be useful for warnings suppression if you find them annoying:

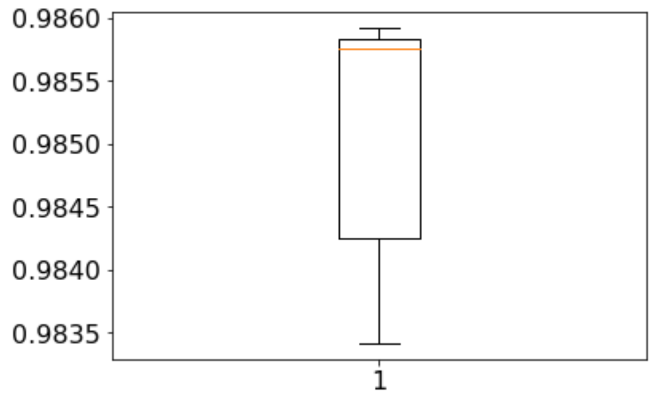
```
import warnings
warnings.filterwarnings("ignore",category=UserWarning)
```

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).

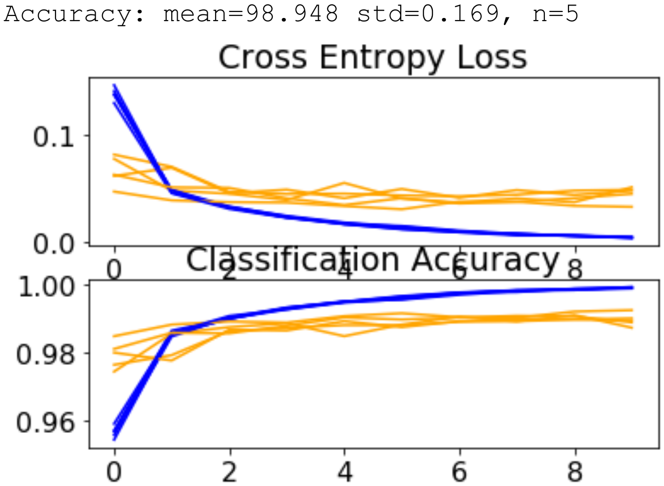
Results:

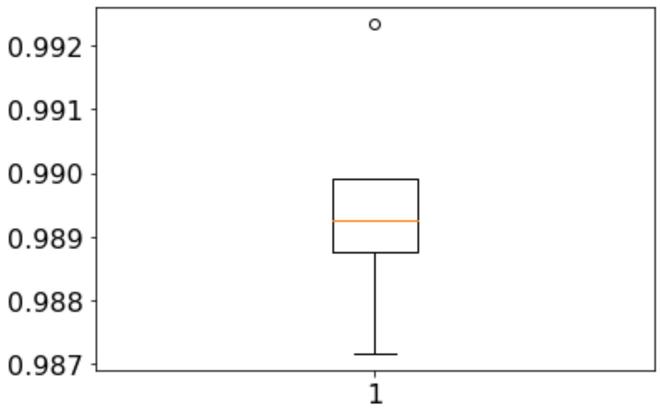
```
Accuracy for each fold:
> 98.583
> 98.425
> 98.342
> 98.575
> 98.592
Accuracy: mean=98.503 std=0.102, n=5
```






```
Improved version (network of larger size):
Accuracy for each fold:
> 98.992
> 98.717
> 98.925
> 99.233
> 98.875
Accuracy: mean=98.948 std=0.169, n=5
```




0.992
0.991
0.990
0.989
0.988
0.987
1