Building, training and testing feed forward neural networks -219393M

Q1)

Creating the NN architecture therefore means coming up with values for the number of layers of each type and the number of nodes in each of these layers.

The Input Layer

- With respect to the number of neurons comprising this layer, this parameter is completely and uniquely determined once we know the shape of your training data.
- Specifically, *the number of neurons comprising that layer is equal to the number of features (columns) in your data*. Some NN configurations add one additional node for a bias term.

The output layer

- Like the Input layer, every NN has *exactly one* output layer. Determining its size (number of neurons) is simple; it is completely determined by the chosen model configuration.
- Is our NN going to run in *Machine* Mode or *Regression* Mode (the ML convention of using a term that is also used in statistics but assigning a different meaning to it is very confusing). Machine mode: returns a class label (e.g., "Premium Account"/"Basic Account"). Regression Mode returns a value (e.g., price).
- If the NN is a regressor, then the output layer has a **single node**.
- If the NN is a classifier, then it also has a single node unless *softmax* is used in which case the output layer has **one node per class label in your model**.

Hidden layers

- If our data is linearly separable (which we often know by the time we begin coding a NN) then we don't need any hidden layers at all. Of course, we don't need an NN to resolve our data either, but it will still do the job.
- One issue within this subject on which there is a consensus is the **performance** difference from adding additional hidden layers: the situations in which performance improves with a second (or third, etc.) hidden layer are very few. ***One hidden layer is sufficient for the large majority of problems***.
- *There are some empirically-derived rules-of-thumb, of these, the most commonly relied on is* ***'the optimal size of the hidden layer is usually between the size of the input and size of the output layers'****. Jeff Heaton, author of Introduction to Neural Networks in Java offers a few more.*

- *In sum, for most problems, one could probably get decent performance (even without a second optimization step) by setting the hidden layer configuration using just two rules:*
  (i) number of hidden layers equals one; and
  (ii) the number of neurons in that layer is the mean of the neurons in the input and output layers.

Reference :
https://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw

- More epochs will be more computationally expensive, but should allow for a better fit.
- When we do the transformation at the same scale for all features then the model will perform better.

Q2) Initially, remove the null values and convert the categorical values to integers by using one hot encoding and label encoding. After doing the preprocessing construct the ANN model using, with 3 layers.
The layers structure as below,

```
# split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=1)
NN_model = Sequential()
# The Input Layer :
NN_model.add(Dense(200, kernel_initializer='normal',input_dim = n_features, activation='relu'))
# The Hidden Layers :
NN_model.add(Dense(256, kernel_initializer='normal',activation='relu'))
# The Output Layer :
NN_model.add(Dense(1, kernel_initializer='normal',activation='sigmoid'))
```

By using 100 epochs, run the model and choosed the best checkpoint.
Using the best checkpoint fit and compile the model. Here NN is a regressor.

```
# Compile the network :
NN_model.compile(loss='mean_squared_error', optimizer='adam', metrics=['mean_squared_error'])
NN_model.summary()
checkpoint_name = 'Weights-new-{epoch:03d}--{val_loss:.5f}.hdf5'
checkpoint = ModelCheckpoint(checkpoint_name, monitor='val_loss', verbose = 1, save_best_only = True, mode ='auto')
callbacks_list = [checkpoint]
NN_model.fit(X_train, y_train, epochs=100, batch_size=32, validation_split = 0.25, callbacks=callbacks_list)
print("\nLoad wights file of the best model :\n")
wights_file = 'Weights-new-038--0.11824.hdf5' # choose the best checkpoint
NN_model.load_weights(wights_file) # load it
NN_model.compile(loss='mean_squared_error', optimizer='adam', metrics=['mean_squared_error'])
predictions = NN_model.predict(X_test)
###here we are saving the output ##########
make_submission(predictions[:,0],'Final_ANN_Prediction')
```

Got the validation loss as “0.11824”.

Basically implemented as above.

Then improved the model as below, by considering the problem, this is Classification problem. So, I implemented NN as a classifier.
After doing the preprocessing as above, encoded the class variable using label encoding.

Then split the data and construct the NN as below with 3 layers.

```
# split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=1)
NN_model = Sequential()
# The Input Layer :
NN_model.add(Dense(200, kernel_initializer='normal',input_dim = n_features, activation='relu'))
# The Hidden Layers :
NN_model.add(Dense(256, kernel_initializer='normal',activation='relu'))
# The Output Layer :
NN_model.add(Dense(n_class, activation='softmax'))
```

Used the **softmax** activation function for output layers as per the reference.

By using 200 epochs, and "**sparse_categorical_crossentropy**" used as lossfunction and run the model and save the best checkpoint by considering validation loss.

```
# Compile the network :
NN_model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
NN_model.summary()
checkpoint_name = 'Weights-classification-{epoch:03d}--{val_loss:.5f}.hdf5'
checkpoint = ModelCheckpoint(checkpoint_name, monitor='val_loss', verbose = 1, save_best_only = True, mode ='auto')
callbacks_list = [checkpoint]
NN_model.fit(X_train, y_train, epochs=200, batch_size=32, validation_split = 0.25, callbacks=callbacks_list)
print("\nLoad wights file of the best model :\n")
wights_file = 'Weights-classification-031--0.40587.hdf5' # choose the best checkpoint
NN_model.load_weights(wights_file) # load it
# NN_model.compile(loss='mean_squared_error', optimizer='adam', metrics=['mean_squared_error'])
NN_model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
# evaluate on test set
y_pred = NN_model.predict(X_test, batch_size=64, verbose=1)
y_pred_bool = np.argmax(y_pred, axis=1)
print(classification_report(y_test, y_pred_bool))
###here we are saving the output ##########
make_submission(y_test,y_pred_bool, 'Final_ANN_Classification_Prediction');
```

The best checkpoint got with 0.40587 val loss

```
Epoch 00030: val_loss did not improve from 0.43094
Epoch 31/200
11/11 [==============================] - 0s 5ms/step - loss: 0.6756 - accuracy: 0.7936 - val_loss: 0.4059 - val_accuracy: 0.8091

Epoch 00031: val_loss improved from 0.43094 to 0.40587, saving model to Weights-classification-031--0.40587.hdf5
Epoch 32/200
11/11 [==============================] - 0s 6ms/step - loss: 0.6728 - accuracy: 0.8260 - val_loss: 0.9194 - val_accuracy: 0.6273
```

FI score is

	precision	recall	f1-score	support
0	0.82	0.82	0.82	123
1	0.76	0.76	0.76	93
accuracy			0.80	216
macro avg	0.79	0.79	0.79	216
weighted avg	0.80	0.80	0.80	216

Then changed the loss = “mean_squared_error” and get the best checkpoint and f1 Score as below,

```
Epoch 185/200
11/11 [==============================] - 0s 22ms/step - loss: 0.2500 - accuracy: 0.4735 - val_loss: 0.2500 - val_accuracy: 0.5000

Epoch 00185: val_loss improved from 0.25000 to 0.25000, saving model to Weights-classification-185--0.25000.hdf5
```

	precision	recall	f1-score	support
0	0.68	0.49	0.57	123
1	0.51	0.70	0.59	93
accuracy			0.58	216
macro avg	0.59	0.59	0.58	216
weighted avg	0.61	0.58	0.58	216

So when we use the loss = “mean_squared_error” the F1 score is reduced.
So I got loss = “sparse_categorical_crossentropy” as a better parameter.

Q3) Yes, as per the finding when we used the activation function for output layer as “Softmax” and loss = “sparse_categorical_crossentropy” then F1 score got improved.

Q4)
As finally the F1 is as below,

	precision	recall	f1-score	support
0	0.82	0.82	0.82	123
1	0.76	0.76	0.76	93
accuracy			0.80	216
macro avg	0.79	0.79	0.79	216
weighted avg	0.80	0.80	0.80	216

```python
# split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=1)
NN_model = Sequential()
# The Input Layer :
NN_model.add(Dense(200, kernel_initializer='normal',input_dim = n_features, activation='relu'))
# The Hidden Layers :
NN_model.add(Dense(256, kernel_initializer='normal',activation='relu'))
# The Output Layer :
NN_model.add(Dense(n_class, activation='softmax'))
# Compile the network :
NN_model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
NN_model.summary()
checkpoint_name = 'Weights-classification-{epoch:03d}--{val_loss:.5f}.hdf5'
checkpoint = ModelCheckpoint(checkpoint_name, monitor='val_loss', verbose = 1, save_best_only = True, mode ='auto')
callbacks_list = [checkpoint]
NN_model.fit(X_train, y_train, epochs=200, batch_size=32, validation_split = 0.25, callbacks=callbacks_list)
print("\nLoad wights file of the best model :\n")
wights_file = 'Weights-classification-031--0.40587.hdf5'# choose the best checkpoint //0.18623.hdf5 when the 100 epoch
NN_model.load_weights(wights_file)# load it
# NN_model.compile(loss='mean_squared_error', optimizer='adam', metrics=['mean_squared_error'])
NN_model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
# evaluate on test set
y_pred = NN_model.predict(X_test, batch_size=64, verbose=1)
y_pred_bool = np.argmax(y_pred, axis=1)
print(classification_report(y_test, y_pred_bool))
```

Also, I used the `StandardScaler for` standardizing the input features. So all input features are on the same scale.
Then the f1 Score improved as below,

	precision	recall	f1-score	support
0	0.87	0.83	0.85	123
1	0.79	0.84	0.81	93
accuracy			0.83	216
macro avg	0.83	0.83	0.83	216
weighted avg	0.84	0.83	0.83	216

Finally, regularization helped to improve the F1 score.

The link for the code as below
https://github.com/Rifaza/NueralNetworkAssignment01_creditcardapproval

The right file is : 219393M- Assignment - 01.py

The codings as below,

```
from keras.models import Sequential
from keras.layers import Dense
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from numpy import unique
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import OrdinalEncoder
from keras.callbacks import ModelCheckpoint
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from numpy import argmax
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.model_selection import KFold, StratifiedKFold
from sklearn.preprocessing import StandardScaler

df= pd.read_csv(r'crx.data', header = None, na_values="?")
df.head()
df.columns = ['A1','A2','A3','A4','A5','A6','A7','A8', 'A9', 'A10','A11','A12','A13','A14','A15','Class']
df = df.to_csv(r'Assignment01.csv', index=None);
df=pd.read_csv("Assignment01.csv");

print('\n\n data types', df.dtypes);
print('\n\nsome of null', df.isnull().sum());
```

```
print('\n\n data types', df.dtypes);
print('\n\nsome of null', df.isnull().sum());
print('\n\nshape of data', df.shape);
df["A14"].value_counts();
print('\n\ndifferent type in A1 column',df["A1"].value_counts());
df.dropna(inplace=True)
print("\n\nShape of the data after removing the null values", df.shape);
#label encoding
df["A1"] = df["A1"].astype('category');
df["A1"] = df["A1"].cat.codes

df["A9"] = df["A9"].astype('category');
df["A9"] = df["A9"].cat.codes
df["A10"] = df["A10"].astype('category');
df["A10"] = df["A10"].cat.codes

#categorical data
categorical_cols = ['A4', 'A5', 'A6', 'A7','A13', 'A12']
#import pandas as pd
df = pd.get_dummies(df, columns = categorical_cols)

df["Class"] = df["Class"].map({
    "-": 0,
    "+": 1
}).astype(int);
print('\n\nnew data types', df.dtypes);
```

```
print('\n\nnew data types', df.dtypes);
print(df.head());
print('Final column namses' , df.columns);
print('after preprocessing the dataset', df.head(10));
print('\n\nafter preprocessing the data types', df.shape);

#using best check point we are going to test the data
def make_submission(actual, prediction, sub_name):
  my_submission = pd.DataFrame({ 'actual': actual, 'Class':prediction})
  my_submission.to_csv('{}.csv'.format(sub_name),index=False)
  print('A submission file has been made')

X = df.loc[:, df.columns != 'Class']

X = np.asarray(X).astype(np.float32);
sc= StandardScaler()
X= sc.fit_transform(X)
y = df['Class']
# encode strings to integer
y = LabelEncoder().fit_transform(y)
n_class = len(unique(y))
n_features = X.shape[1]
kf = KFold(n_splits = 5)
# split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=1)
```

```
NN_model = Sequential()
# The Input Layer :
NN_model.add(Dense(200, kernel_initializer='normal',input_dim = n_features, activation='relu'))
# The Hidden Layers :
NN_model.add(Dense(256, kernel_initializer='normal',activation='relu'))
# The Output Layer :
NN_model.add(Dense(n_class, activation='softmax'))
# Compile the network :
NN_model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
NN_model.summary()
checkpoint_name = 'Weights-classification-{epoch:03d}--{val_loss:.5f}.hdf5'
checkpoint = ModelCheckpoint(checkpoint_name, monitor='val_loss', verbose = 1, save_best_only = True, mode ='auto')
callbacks_list = [checkpoint]
NN_model.fit(X_train, y_train, epochs=200, batch_size=32, validation_split = 0.25, callbacks=callbacks_list)
print("\nLoad wights file of the best model :\n")
wights_file = 'Weights-classification-031--0.40587.hdf5' # choose the best checkpoint
NN_model.load_weights(wights_file) # load it
# NN_model.compile(loss='mean_squared_error', optimizer='adam', metrics=['mean_squared_error'])
NN_model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
# evaluate on test set
y_pred = NN_model.predict(X_test, batch_size=64, verbose=1)
y_pred_bool = np.argmax(y_pred, axis=1)
print(classification_report(y_test, y_pred_bool))
###here we are saving the output ##########
make_submission(y_test,y_pred_bool, 'Final_ANN_Classification_Prediction');
```