DNN Lab

Objectives

- Understand basic DNN model building process using Keras
- Analyze model performance and capacity vs generalization tradeoff
- Modify models to reduce overfitting and improve performance

Exercises

- Build a DNN model for slump Test Problem
- Start with a model consisting of one hidden layer with 7 neurons
- Analyze results and explore improvements to model in terms of capacity, regularization

Double-click (or enter) to edit

Step 1: Import Libraries

```
%tensorflow_version 2.x
from numpy.random import seed
seed(2)
import tensorflow as tf
from tensorflow import keras
from IPython import display
from matplotlib import cm
from matplotlib import gridspec
from matplotlib import pyplot as plt
import numpy as np
import pandas as pd
import os
import datetime
from tensorflow.python.data import Dataset
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler, StandardScaler, Normalizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn import metrics
from sklearn.dummy import DummyRegressor

print(tf.__version__)
```

```
2.6.0
```

Step 2: Import Data

```
pd.options.display.max_rows = 10
pd.options.display.float_format = '{:.4f}'.format

clinic_data = pd.read_excel("OnlineUse.xlsx", sheet_name='OnlineUseData')

clinic_data = clinic_data.reindex(
    np.random.permutation(clinic_data.index))
```

Step 3: Preprocess

```
#missing data values is -98 or -97
clinic_data[clinic_data.eq(-98).any(1)]
clinic_data[clinic_data.eq(-97).any(1)]

#deleted records with missing Online Appointment use
delete_records =  clinic_data[clinic_data.OnlineAppointmentUse <-96].index
clinic_data.drop(delete_records, inplace = True)
```

```
#replacing the missing data value of -98 and -97 with np.nan
#can run a regression to predict the value instead of ultimately using the mean value!!!!!!!!
clinic_data = clinic_data.replace({-98 : np.NaN, -97 : np.NaN})

#checking if NaN values replacement worked
#checking which columns have NaN values
clinic_data[clinic_data.isnull().any(axis=1)]
#checking to see the # of NaN values present
len(clinic_data[clinic_data.isnull().any(axis=1)])

#replace the nan values with the mean
#change to np.NaN
clinic_data.fillna(clinic_data.mean(), inplace=True)
```

```
#remove columns vendor and numpats
clinic_data = clinic_data.drop(['vendor','numpats'],axis=1)

#clinic_data.shape
```

```
#baseline accuracy measure
clinic_data.iloc[:,1:2].mean()
```

```
OnlineAppointmentUse   0.1371
dtype: float64
```

Train/Validation Split

```
#Creating a training and validation dataset with a 80/20 split
X_train,X_test, y_train, y_test = train_test_split(clinic_data.iloc[:,2:],clinic_data.iloc[:,
```

```
X_train.head()
```

	malepct	unemp	age16to24	age25to34	age35to44	age45to54	age55to64	age65to74
5129	0.4665	0.0137	0.0774	0.1087	0.1005	0.1838	0.2341	0.1994
4375	0.5677	0.0272	0.0935	0.2117	0.2163	0.1839	0.1059	0.0947
571	0.4646	0.0321	0.1099	0.1373	0.0707	0.2095	0.2085	0.1269
3926	0.4517	0.0280	0.0366	0.1619	0.1416	0.2058	0.2210	0.1210
3921	0.4278	0.0103	0.0952	0.1379	0.0643	0.1776	0.2155	0.1528

Step 4: Build Model

https://www.tensorflow.org/api_docs/python/tf/keras/Model

https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense

https://keras.io/optimizers/

Build Model

```
#model is continuous, so which is best
model1 = keras.Sequential()
model1.add(keras.layers.Dense(18, activation=tf.nn.leaky_relu,
                       input_shape=(X_train.shape[1],)))
keras.layers.Dropout(0.75),
model1.add(keras.layers.Dense(9, activation=tf.nn.relu
                       ))
keras.layers.Dropout(0.75),
model1.add(keras.layers.Dense(5, activation=tf.nn.leaky_relu
                       ))
keras.layers.Dropout(0.75),
#model1.add(keras.layers.Dense(3, activation=tf.nn.relu
#                       ))
#keras.layers.Dropout(0.75),
model1.add(keras.layers.Dense(1, activation=tf.nn.sigmoid))
```

```
#optimizer = tf.keras.optimizers.RMSprop(0.001)
#optimizer = tf.keras.optimizers.Adam()

model1.compile(loss='mse',
               optimizer='sgd',
               metrics=['mae'])
model1.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 18)                342
_________________________________________________________________
dense_1 (Dense)              (None, 9)                 171
_________________________________________________________________
dense_2 (Dense)              (None, 5)                 50
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 6
=================================================================
Total params: 569
Trainable params: 569
Non-trainable params: 0
_________________________________________________________________
```

Fit Model

```
class PrintDot(keras.callbacks.Callback):
  def on_epoch_end(self, epoch, logs):
    if epoch % 100 == 0: print('')
    print('.', end='')

EPOCHS = 200
tf.random.set_seed(1)

# Store training stats
m1_history = model1.fit(X_train, y_train, epochs=EPOCHS,
                    validation_data= (X_test, y_test), verbose=1)
                    #callbacks=[PrintDot()])
```

```
171/171 [==============================] - 0s 2ms/step - loss: 0.0075 - mae: 0.0683 -
Epoch 51/200
171/171 [==============================] - 0s 2ms/step - loss: 0.0075 - mae: 0.0682 -
Epoch 52/200
171/171 [==============================] - 0s 2ms/step - loss: 0.0075 - mae: 0.0681 -
Epoch 53/200
171/171 [==============================] - 0s 2ms/step - loss: 0.0075 - mae: 0.0680 -
Epoch 54/200
171/171 [==============================] - 0s 2ms/step - loss: 0.0075 - mae: 0.0679 -
```

```
Epoch 55/200
171/171 [==============================] - 0s 2ms/step - loss: 0.0074 - mae: 0.0678 -
Epoch 56/200
171/171 [==============================] - 0s 2ms/step - loss: 0.0074 - mae: 0.0677 -
Epoch 57/200
171/171 [==============================] - 0s 2ms/step - loss: 0.0074 - mae: 0.0676 -
Epoch 58/200
171/171 [==============================] - 0s 2ms/step - loss: 0.0074 - mae: 0.0676 -
Epoch 59/200
171/171 [==============================] - 0s 2ms/step - loss: 0.0074 - mae: 0.0674 -
Epoch 60/200
171/171 [==============================] - 0s 2ms/step - loss: 0.0073 - mae: 0.0673 -
Epoch 61/200
171/171 [==============================] - 0s 2ms/step - loss: 0.0073 - mae: 0.0673 -
Epoch 62/200
171/171 [==============================] - 0s 2ms/step - loss: 0.0073 - mae: 0.0672 -
Epoch 63/200
171/171 [==============================] - 0s 2ms/step - loss: 0.0073 - mae: 0.0670 -
Epoch 64/200
171/171 [==============================] - 0s 2ms/step - loss: 0.0072 - mae: 0.0669 -
Epoch 65/200
171/171 [==============================] - 0s 2ms/step - loss: 0.0072 - mae: 0.0668 -
Epoch 66/200
171/171 [==============================] - 0s 2ms/step - loss: 0.0072 - mae: 0.0667 -
Epoch 67/200
171/171 [==============================] - 0s 2ms/step - loss: 0.0072 - mae: 0.0667 -
Epoch 68/200
171/171 [==============================] - 0s 2ms/step - loss: 0.0072 - mae: 0.0665 -
Epoch 69/200
171/171 [==============================] - 0s 2ms/step - loss: 0.0071 - mae: 0.0664 -
Epoch 70/200
171/171 [==============================] - 0s 2ms/step - loss: 0.0071 - mae: 0.0663 -
Epoch 71/200
171/171 [==============================] - 0s 2ms/step - loss: 0.0071 - mae: 0.0662 -
Epoch 72/200
171/171 [==============================] - 0s 2ms/step - loss: 0.0071 - mae: 0.0661 -
Epoch 73/200
171/171 [==============================] - 0s 2ms/step - loss: 0.0070 - mae: 0.0660 -
Epoch 74/200
171/171 [==============================] - 0s 2ms/step - loss: 0.0070 - mae: 0.0659 -
Epoch 75/200
171/171 [==============================] - 0s 2ms/step - loss: 0.0070 - mae: 0.0658 -
Epoch 76/200
171/171 [==============================] - 0s 2ms/step - loss: 0.0070 - mae: 0.0657 -
Epoch 77/200
171/171 [==============================] - 0s 2ms/step - loss: 0.0070 - mae: 0.0655 -
Epoch 78/200
171/171 [==============================] - 0s 2ms/step - loss: 0.0069 - mae: 0.0654 -
Epoch 79/200
```

Lowest Validation Error

Step 5: Plot Results

```
import matplotlib.pyplot as plt

def plot_history(histories, key='loss'):
  plt.figure(figsize=(16,10))
  for name, history in histories:
    val = plt.plot(m1_history.epoch, m1_history.history['val_'+key],
                   '--', label=name.title()+' Val')
    plt.plot(m1_history.epoch, m1_history.history[key], color=val[0].get_color(),
             label=name.title()+' Train')

  plt.xlabel('Epochs')
  plt.ylabel(key.replace('_',' ').title())
  plt.legend()

  plt.xlim([0,max(m1_history.epoch)])
  plt.ylim([0,0.013])

plot_history([('Basic Model', m1_history)])

#Plot Multiple Model Results

#plot_history([('Plain', m1_history),('L1',model1)])
```


```
y_pred = np.round(model1.predict_on_batch(X_test),5)
print(y_pred)
```

```
[[0.17196]
 [0.21574]
 [0.15165]
 ...
 [0.09469]
 [0.10696]
 [0.07183]]
```


```
#Our accuracy measure for the validation dataset
print(min(m1_history.history['val_mae']))
```

```
0.0440857820212841
```

The goal of our model is to predict the percen
the online appointment use system. One method
of a Regression based neural network is to use
as the mean. In the code much farther above in
the mean Online Appointment use is 13.71%.

Our final model has a valuation Mean Absolute
in the graph above that our training validatio
Absolute Error. Given that the training and va
have any overfitting, underfitting and general

The goal of our model is to predict the percentage of patients that will use the online appointment use system. One method for the baseline accuracy measure of a Regression based neural network is to use a central tendancy measure, such as the mean. In the code much farther above in the python notebook you can see the mean Online Appointment use is 13.71%.

Our final model has a valuation Mean Absolute Error of 0.044. You can also see in the graph above that our training validation dataset have converged on Mean Absolute Error. Given that the training and validation model converged we don't have any overfitting, underfitting and generalization issues.