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
from sklearn.model_selection import train_test_split
from collections import Counter
from imblearn.over_sampling import SMOTE, RandomOverSampler

print(tf.__version__)
```

```
2.6.0
/usr/local/lib/python3.7/dist-packages/sklearn/externals/six.py:31: FutureWarning: The
  "(https://pypi.org/project/six/).", FutureWarning)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:144: FutureWarning:
  warnings.warn(message, FutureWarning)
```

Step 2: Import Data

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

train_data = pd.read_csv("ticdata2000.txt", sep="\t",header=None)

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

```
train_data.shape[0]
#train_data[:][58].max()
train_data[:][55].min()
```

```
0
```

Step 3: Preprocess

```
#checking which columns have NaN values
train_data[train_data.isnull().any(axis=1)]
#checking to see the # of NaN values present
len(train_data[train_data.isnull().any(axis=1)])
```

```
0
```

```
Ov= RandomOverSampler(random_state=42)
features = train_data.iloc[:,:-2]
target = train_data.iloc[:,85:]
# fit and apply the transform
X_sm, y_sm = Ov.fit_resample(features,target)

print(f'''Shape of X before SMOTE: {features.shape}
Shape of X after SMOTE: {X_sm.shape}''')
```

```
Shape of X before SMOTE: (5822, 84)
Shape of X after SMOTE: (10948, 84)
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:760: DataConversionW
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning:
  warnings.warn(msg, category=FutureWarning)
```

```
y_df= pd.DataFrame(y_sm)
X_df = pd.DataFrame(X_sm)


#changing predictor variable into dummy vars
y_df.rename(columns = { 0 : 'Insurance_Purchased'}, inplace=True)
y_df= pd.get_dummies(y_df, columns=['Insurance_Purchased'])


#Baseline accuracy measure
naive_app_min= y_df['Insurance_Purchased_0'].value_counts().max()/len(y_df)
naive_app_min
```

```
0.5
```

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(X_df,y_df, test_size=0.2, random_state=1)
```

```
X_train.head()
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
10414	6	1	3	2	2	0	4	2	4	9	0	0	0	5	4	3	5	2	7	0	0	2	0
4736	38	1	4	2	9	0	4	0	5	8	0	1	1	1	8	1	4	5	1	0	0	2	0
5240	33	2	3	3	8	0	4	0	5	7	0	2	2	1	6	0	2	7	0	0	0	3	6
1576	33	1	2	4	8	0	5	0	5	9	0	0	0	9	0	0	9	0	0	0	0	9	0
6337	4	2	2	5	1	0	7	0	2	7	0	2	2	7	0	2	6	2	6	0	0	3	0

5 rows × 84 columns

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/

```
#Standardizing training dataset
#scaler = StandardScaler()
```

```
#scaledf = scaler.fit_transform(X_train)
#X_train = pd.DataFrame(scaledf, index=X_train.index, columns=X_train.columns)

print(X_train)
#Standardizing validation dataset
#vscaled = scaler.transform(X_test.values)
#X_test = pd.DataFrame(vscaled, index=X_test.index, columns=X_test.columns)
#print(X_test)
```

```
        0   1   2   3   4   5   6   7   8  ...  75  76  77  78  79  80  81  82  83
10414   6   1   3   2   2   0   4   2   4  ...   0   0   0   0   0   0   0   0   0
4736   38   1   4   2   9   0   4   0   5  ...   0   0   0   0   1   0   0   0   0
5240   33   2   3   3   8   0   4   0   5  ...   0   0   0   0   0   0   0   1   0
1576   33   1   2   4   8   0   5   0   5  ...   0   0   0   0   0   0   0   1   0
6337    4   2   2   5   1   0   7   0   2  ...   0   0   0   0   0   0   0   0   0
...    ..  ..  ..  ..  ..  ..  ..  ..  ..  ...  ..  ..  ..  ..  ..  ..  ..  ..  ..
2895   33   2   4   2   8   0   5   1   4  ...   0   0   0   0   0   0   0   0   0
7813   12   2   4   2   3   2   4   2   2  ...   0   0   1   0   1   0   0   0   0
905    39   1   3   3   9   0   7   0   2  ...   0   0   0   0   1   0   0   0   0
5192   41   1   4   2  10   1   5   1   3  ...   0   0   0   0   1   0   0   0   0
235    33   1   2   3   8   0   7   0   2  ...   0   0   0   0   1   0   0   0   0

[8758 rows x 84 columns]
```

Build Model

```
l2_model = keras.Sequential([
    keras.layers.Dense(86, activation=tf.nn.relu,
                       input_shape=(X_train.shape[1],)),
    keras.layers.Dropout(0.75),
    keras.layers.Dense(43, activation=tf.nn.leaky_relu),
    keras.layers.Dropout(0.75),
    keras.layers.Dense(20, activation=tf.nn.sigmoid),
    keras.layers.Dropout(0.75),
    keras.layers.Dense(10, activation=tf.nn.leaky_relu),
    keras.layers.Dropout(0.75),
    keras.layers.Dense(5, activation=tf.nn.sigmoid),
    keras.layers.Dropout(0.75),
    keras.layers.Dense(2, activation=tf.nn.sigmoid)
  ])

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

l2_model.compile(loss=tf.keras.losses.BinaryCrossentropy(),
                optimizer='sgd',
                metrics=['accuracy'])
l2_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
```

```
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 86)                7310
_________________________________________________________________
dropout (Dropout)            (None, 86)                0
_________________________________________________________________
dense_1 (Dense)              (None, 43)                3741
_________________________________________________________________
dropout_1 (Dropout)          (None, 43)                0
_________________________________________________________________
dense_2 (Dense)              (None, 20)                880
_________________________________________________________________
dropout_2 (Dropout)          (None, 20)                0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                210
_________________________________________________________________
dropout_3 (Dropout)          (None, 10)                0
_________________________________________________________________
dense_4 (Dense)              (None, 5)                 55
_________________________________________________________________
dropout_4 (Dropout)          (None, 5)                 0
_________________________________________________________________
dense_5 (Dense)              (None, 2)                 12
=================================================================
Total params: 12,208
Trainable params: 12,208
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

# Store training stats
l2_history = l2_model.fit(X_train, y_train, epochs=EPOCHS,
                    validation_data= (X_test, y_test),verbose=1)
```

```
Epoch 172/200
274/274 [==============================] - 0s 2ms/step - loss: 0.6935 - accuracy: 0.4
Epoch 173/200
274/274 [==============================] - 0s 2ms/step - loss: 0.6933 - accuracy: 0.5
Epoch 174/200
274/274 [==============================] - 0s 2ms/step - loss: 0.6931 - accuracy: 0.5
Epoch 175/200
274/274 [==============================] - 0s 2ms/step - loss: 0.6934 - accuracy: 0.4
Epoch 176/200
274/274 [==============================] - 0s 2ms/step - loss: 0.6932 - accuracy: 0.4
Epoch 177/200
```

```
274/274 [==============================] - 0s 2ms/step - loss: 0.6933 - accuracy: 0.4
Epoch 178/200
274/274 [==============================] - 0s 2ms/step - loss: 0.6933 - accuracy: 0.4
Epoch 179/200
274/274 [==============================] - 0s 2ms/step - loss: 0.6933 - accuracy: 0.4
Epoch 180/200
274/274 [==============================] - 0s 2ms/step - loss: 0.6932 - accuracy: 0.4
Epoch 181/200
274/274 [==============================] - 0s 1ms/step - loss: 0.6932 - accuracy: 0.4
Epoch 182/200
274/274 [==============================] - 0s 2ms/step - loss: 0.6933 - accuracy: 0.4
Epoch 183/200
274/274 [==============================] - 0s 2ms/step - loss: 0.6932 - accuracy: 0.5
Epoch 184/200
274/274 [==============================] - 0s 2ms/step - loss: 0.6933 - accuracy: 0.5
Epoch 185/200
274/274 [==============================] - 0s 1ms/step - loss: 0.6931 - accuracy: 0.5
Epoch 186/200
274/274 [==============================] - 0s 2ms/step - loss: 0.6933 - accuracy: 0.4
Epoch 187/200
274/274 [==============================] - 0s 2ms/step - loss: 0.6933 - accuracy: 0.4
Epoch 188/200
274/274 [==============================] - 0s 2ms/step - loss: 0.6933 - accuracy: 0.4
Epoch 189/200
274/274 [==============================] - 0s 2ms/step - loss: 0.6933 - accuracy: 0.4
Epoch 190/200
274/274 [==============================] - 0s 2ms/step - loss: 0.6932 - accuracy: 0.4
Epoch 191/200
274/274 [==============================] - 0s 2ms/step - loss: 0.6932 - accuracy: 0.4
Epoch 192/200
274/274 [==============================] - 0s 1ms/step - loss: 0.6933 - accuracy: 0.4
Epoch 193/200
274/274 [==============================] - 0s 2ms/step - loss: 0.6933 - accuracy: 0.4
Epoch 194/200
274/274 [==============================] - 0s 2ms/step - loss: 0.6933 - accuracy: 0.4
Epoch 195/200
274/274 [==============================] - 0s 2ms/step - loss: 0.6933 - accuracy: 0.4
Epoch 196/200
274/274 [==============================] - 0s 2ms/step - loss: 0.6932 - accuracy: 0.5
Epoch 197/200
274/274 [==============================] - 0s 2ms/step - loss: 0.6932 - accuracy: 0.4
Epoch 198/200
274/274 [==============================] - 0s 1ms/step - loss: 0.6932 - accuracy: 0.4
Epoch 199/200
274/274 [==============================] - 0s 2ms/step - loss: 0.6932 - accuracy: 0.4
Epoch 200/200
274/274 [==============================] - 0s 1ms/step - loss: 0.6933 - accuracy: 0.4
```

Lowest Validation Error

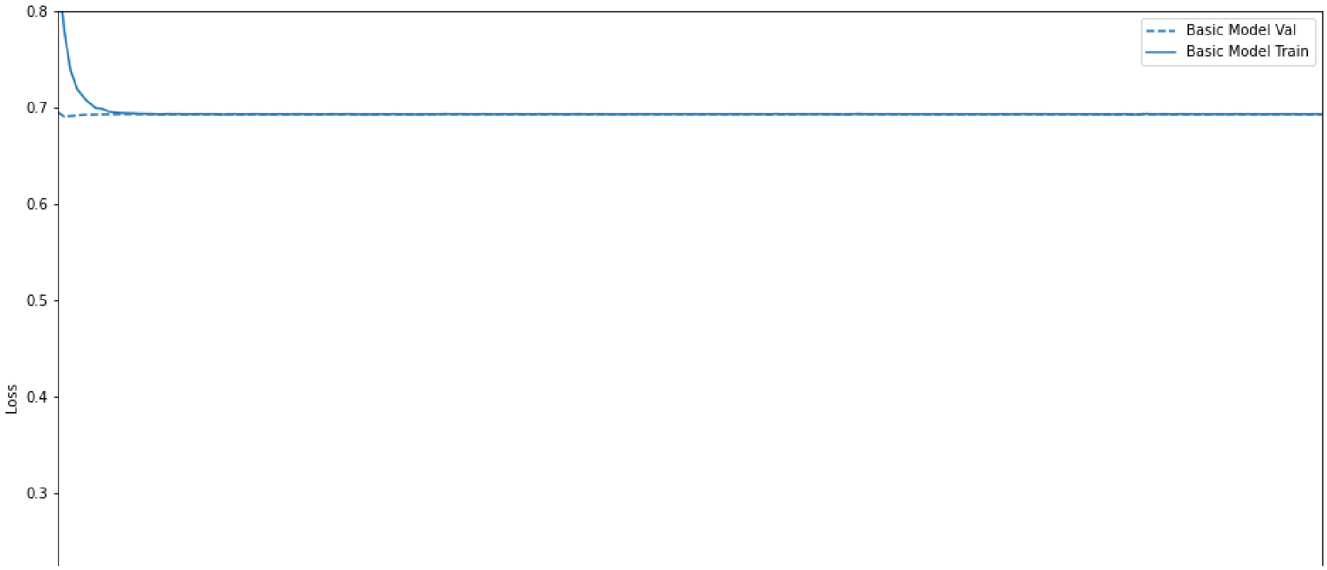
```
y_pred = np.round(l2_model.predict_on_batch(X_test),5)
print(y_pred)
```

```
[[0.50046 0.49954]
```

```
 [0.50069 0.49931]
 [0.5004  0.4996 ]
 ...
 [0.5003  0.4997 ]
 [0.5005  0.4995 ]
 [0.50057 0.49943]]
```

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(l2_history.epoch, l2_history.history['val_'+key],
                   '--', label=name.title()+' Val')
    plt.plot(l2_history.epoch, l2_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(l2_history.epoch)])
  plt.ylim([0,0.8])

plot_history([('Basic Model', l2_history)])

#Plot Multiple Model Results

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




```
print(max(l2_history.history['val_accuracy']))
```

    0.6114155054092407

The goal of the model is to predict whether the person will purchase the Caravan Insurance policy. The naive approach is that 50% of the people will choose to opt for Caravan Insurance. We used random oversampling to even out the dataset, so 50% of the observations are people who purchased caravan insurance and 50% of the people didn't purchase caravan insurance.

As you can see in the code above our best model had an evaluation data accuracy of 61%, which is higher than the naive approach of 50%.

The goal of the model is to predict whether th
purchase the Caravan Insurance policy. The nai
50% of the people will choose to opt for Carav
used random oversampling to even out the datas
observations are people who purchased caravan
the people didn't purchase caravan insurance.

As you can see in the code above our best mode
data accuracy of 61%, which is higher than the
50%.

✓ 0s completed at 11:38 PM