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

Step 1: Import Libraries

In [112]:
```
%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

print(tf.__version__)
```

```
2.6.0
```

Step 2: Import Data

In [113]:
```
pd.options.display.max_rows = 10
pd.options.display.float_format = '{:.1f}'.format

hcv_data = pd.read_csv("hcvdat0.csv")

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

In [114]:
```
hcv_data.shape[0]
```

Out[114]:
```
615
```

In [115]:
```
#removing the redundent index column
hcv_data.drop('Unnamed: 0', axis=1, inplace=True)
print(hcv_data)
```

```
                    Category  Age Sex  ALB   ALP  ...  CHE  CHOL  CREA   GGT
PROT
469            0=Blood Donor   52   f 51.5  81.8  ...  6.7   5.9  88.0  16.3
82.2
592              3=Cirrhosis   47   m 42.0   nan  ...  6.3   5.5  58.0 201.0
79.0
265            0=Blood Donor   58   m 41.3  58.9  ...  8.2   5.7  60.0  10.8
70.1
84             0=Blood Donor   39   m 43.9  90.1  ...  9.9   4.6  98.0  99.3
66.2
109            0=Blood Donor   42   m 44.1  46.8  ... 10.8   6.3  95.0  19.7
73.0
..                       ...  ...  ..  ...   ...  ...  ...   ...   ...   ...
...
534  0s=suspect Blood Donor   48   m 24.9 116.9  ...  3.4   5.2  29.0  83.0
47.8
584               2=Fibrosis   75   f 36.0   nan  ...  6.7   nan  57.0 177.0
72.0
493            0=Blood Donor   56   f 34.7  90.3  ...  8.1   5.5  67.0   9.0
69.4
527            0=Blood Donor   63   f 27.8  85.7  ...  6.1   4.0  63.0  46.0
56.9
168            0=Blood Donor   47   m 48.3  59.3  ... 11.1   5.6  88.0  91.5
73.0

[615 rows x 13 columns]
```

In [116]:
```
#2.3 Minimum accuracy the model needs to be
naive_app_min= hcv_data.Category.value_counts().max()/len(hcv_data)
naive_app_min
```

Out[116]:
```
0.8666666666666667
```

In [117]:
```
#changing Sex column to 0 and 1s
hcv_data= pd.get_dummies(hcv_data, columns=['Sex'], drop_first=True)
#changing predictor variable into dummy vars
hcv_data= pd.get_dummies(hcv_data, columns=['Category'])
print(hcv_data)
```

```
     Age   ALB  ...  Category_2=Fibrosis  Category_3=Cirrhosis
469   52  51.5  ...                    0                     0
592   47  42.0  ...                    0                     1
265   58  41.3  ...                    0                     0
84    39  43.9  ...                    0                     0
109   42  44.1  ...                    0                     0
..   ...   ...  ...                  ...                   ...
534   48  24.9  ...                    0                     0
584   75  36.0  ...                    1                     0
493   56  34.7  ...                    0                     0
527   63  27.8  ...                    0                     0
168   47  48.3  ...                    0                     0

[615 rows x 17 columns]
```

In [118]:
```
hcv_data = hcv_data[hcv_data.columns[::-1]]
```

Step 3: Preprocess

In [119]:
```
#removed any NaN values and replaced it with the column mean
for cols in hcv_data.columns[hcv_data.isnull().any()]:
    hcv_data[cols].replace(np.NaN, hcv_data[cols].mean(),inplace=True)

hcv_data[hcv_data.isnull().any(axis=1)]
```

Out[119]:

Category_3=Cirrhosis	Category_2=Fibrosis	Category_1=Hepatitis	Category_0s=suspect Blood Donor	Categoı

In [120]:
```
#checking to see if any NaN values are present
len(hcv_data[hcv_data.isnull().any(axis=1)])
```

Out[120]: 0

Train/Validation Split

In [121]:
```
#Creating a training and validation dataset with a 80/20 split
X_train,X_test, y_train, y_test = train_test_split(hcv_data.iloc[:,5:],hcv_dat
a.iloc[:,:5], test_size=0.2, random_state=1)
```

In [122]:
```
#2.2 Print first few rows of training data
X_train.head()
```

Out[122]:

	Sex_m	PROT	GGT	CREA	CHOL	CHE	BIL	AST	ALT	ALP	ALB	Age
246	1	72.2	30.2	80.0	6.3	7.5	4.5	21.0	36.9	87.1	46.2	55
344	0	72.8	12.4	64.0	5.1	10.0	3.0	22.0	19.2	62.6	43.4	35
75	1	70.1	17.3	67.0	4.1	6.9	16.7	35.1	47.4	69.4	44.7	38
216	1	67.4	87.8	77.0	6.1	8.9	7.8	23.7	37.0	82.2	82.2	52
444	0	78.2	14.5	69.0	6.8	9.1	5.8	15.9	14.3	45.9	45.4	49

In [123]:
```
#normalizing training dataset
scaler = StandardScaler()

scaledf = scaler.fit_transform(X_train.iloc[:,1:])
X_train.iloc[:,1:] = pd.DataFrame(scaledf, index=X_train.iloc[:,1:].index, col
umns=X_train.iloc[:,1:].columns)

#print(X_train)
#normalizing validation dataset
vscaled = scaler.transform(X_test.iloc[:,1:].values)
X_test.iloc[:,1:] = pd.DataFrame(vscaled, index=X_test.iloc[:,1:].index, colum
ns=X_test.iloc[:,1:].columns)
#print(X_test)
```

Step 4: Build Model

[https://www.tensorflow.org/api_docs/python/tf/keras/Model (https://www.tensorflow.org/api_docs/python/tf/keras/Model)](https://www.tensorflow.org/api_docs/python/tf/keras/Model)

[https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense (https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense)](https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense)

[https://keras.io/optimizers/ (https://keras.io/optimizers/)](https://keras.io/optimizers/)

Build Model

In [124]:
```
l2_model = keras.Sequential([
    keras.layers.Dense(24, kernel_regularizer=keras.regularizers.l2(0.001), ac
tivation=tf.nn.relu,
                       input_shape=(X_train.shape[1],)),
    keras.layers.Dropout(0.25),
    keras.layers.Dense(5, kernel_regularizer=keras.regularizers.l2(0.001), act
ivation=tf.nn.relu),
    keras.layers.Dropout(0.25),
    #keras.layers.Dense(3, kernel_regularizer=keras.regularizers.l2(0.01), act
ivation=tf.nn.relu),
    #keras.layers.Dropout(0.25),
    #keras.layers.Dense(6, kernel_regularizer=keras.regularizers.l2(0.01), act
ivation=tf.nn.softmax),
    #keras.layers.Dropout(0.50),
    #keras.layers.Dense(4, kernel_regularizer=keras.regularizers.l2(0.01), act
ivation=tf.nn.softmax),
    #keras.layers.Dropout(0.50),
    keras.layers.Dense(5, activation=tf.nn.softmax)
  ])



l2_model.compile(loss=tf.keras.losses.CategoricalCrossentropy(label_smoothing=
0.1),
                optimizer='sgd',
                metrics=[tf.keras.metrics.CategoricalAccuracy()])
```

Fit Model

In [125]:
```
logdir = os.path.join("logs", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"
))
tensorboard_callback =  tf.keras.callbacks.TensorBoard(logdir, histogram_freq=
1)
```

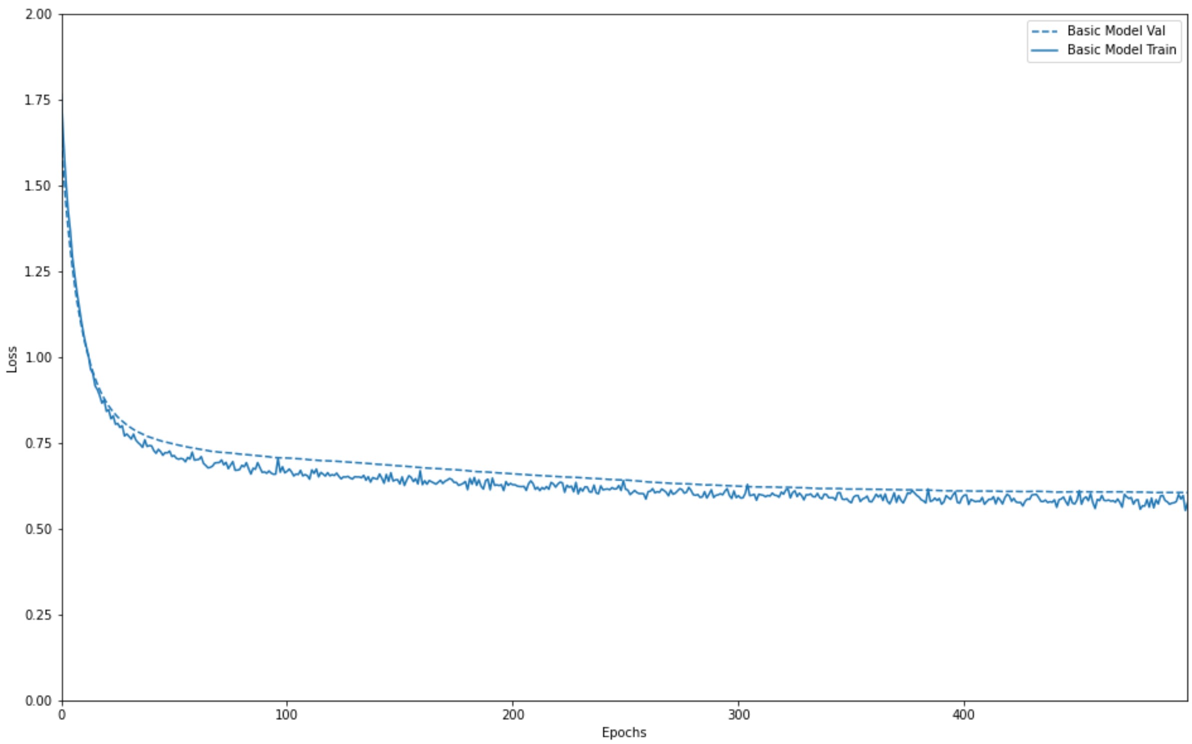
In [126]:
```
l2_model.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_22 (Dense)             (None, 24)                312
_________________________________________________________________
dropout_15 (Dropout)         (None, 24)                0
_________________________________________________________________
dense_23 (Dense)             (None, 5)                 125
_________________________________________________________________
dropout_16 (Dropout)         (None, 5)                 0
_________________________________________________________________
dense_24 (Dense)             (None, 5)                 30
=================================================================
Total params: 467
Trainable params: 467
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:
```
class PrintDot(keras.callbacks.Callback):
  def on_epoch_end(self, epoch, logs):
    if epoch % 10 == 0:
      print('')
      print(logs)

EPOCHS = 500
tf.random.set_seed(1)



# Store training stats
l2_history = l2_model.fit(X_train, y_train, epochs=EPOCHS,
                    validation_data= (X_test, y_test),verbose=2) #PRINTDOTS NO
T WORKING RIGHT NOW
#try 0 and 1, verboe 1 gives data as its running
```

Step 5: Plot Results

In [128]:
```
#2.4 Chart of training and validation error and accuracy (or appropriate metri
c based on problem)

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([0,2])

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

#Plot Multiple Model Results

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



Predictions

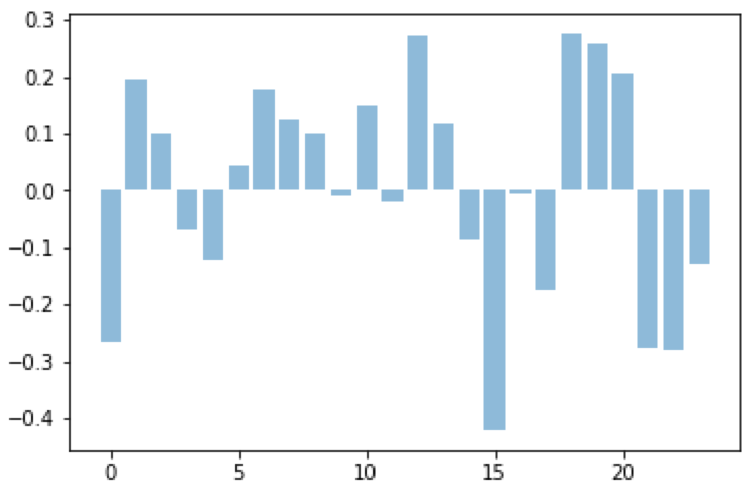
In [138]:
```
valpreds = np.round(l2_model.predict_on_batch(X_test),3)
print(valpreds[:5])
```

```
[[0.015 0.024 0.029 0.02  0.913]
 [0.016 0.025 0.029 0.021 0.909]
 [0.021 0.031 0.033 0.021 0.894]
 [0.018 0.028 0.031 0.02  0.903]
 [0.016 0.025 0.03  0.019 0.911]]
```

In [ ]:
```
with pd.option_context('display.max_rows', None, 'display.max_columns', None):
    print(y_test)
```

In [130]:
```
# Plot Weights
nfw = l2_model.get_weights()[0][0]
y_pos = np.arange(len(nfw))

plt.bar(y_pos, nfw, align='center', alpha=0.5)
```

Out[130]: <BarContainer object of 24 artists>



In [133]:
```
print(min(l2_history.history['val_loss']))
```

```
0.6051108837127686
```

In [135]:
```
print(max(l2_history.history['val_categorical_accuracy']))
```

```
0.8943089246749878
```

In [ ]:
```
l
```

Goal: Predict whether the patient belongs in 1 of 5 categories: Blood Donor, Suspected Blood Donor, Hephatitis, Fibrosis, and Cirrhosis. Given the 5 categories we're trying to predict the a model >naive approach accuracy of 86.67%.

As you can see in the code above the categorical accuracy was 89.43%, which is slightly higher than the baseline accuracy of 86.67%.

The graph shows very little generalization issues since they are converging. Label Smoothing made it smoother then prior because the spikes were frequent. I also had to use less layers as the accuracy was pretty low and the generalization error was significant.

Changed the amount of nodes as a larger amount of nodes had a an accuracy below the baseline.