T81-558: Applications of Deep Neural Networks

Module 8: Kaggle Data Sets

- Instructor: Jeff Heaton, McKelvey School of Engineering, Washington University in St. Louis
- For more information visit the class website.

Module 8 Material

- Part 8.1: Introduction to Kaggle [Video] [Notebook]
- Part 8.2: Building Ensembles with Scikit-Learn and Keras [Video] [Notebook]
- Part 8.3: How Should you Architect Your Keras Neural Network: Hyperparameters [Video] [Notebook]
- **Part 8.4: Bayesian Hyperparameter Optimization for Keras** [Video] [Notebook]
- Part 8.5: Current Semester's Kaggle [Video] [Notebook]

Google CoLab Instructions

The following code ensures that Google CoLab is running the correct version of TensorFlow.

In [1]:
```
# Startup Google CoLab
try:
    %tensorflow_version 2.x
    COLAB = True
    print("Note: using Google CoLab")
except:
    print("Note: not using Google CoLab")
    COLAB = False

# Nicely formatted time string
def hms_string(sec_elapsed):
    h = int(sec_elapsed / (60 * 60))
    m = int((sec_elapsed % (60 * 60)) / 60)
    s = sec_elapsed % 60
    return "{}:{:>02}:{:>05.2f}".format(h, m, s)
```

```
Note: using Google CoLab
```

Part 8.4: Bayesian Hyperparameter Optimization for Keras

Bayesian Hyperparameter Optimization is a method of finding hyperparameters more efficiently than a grid search. Because each candidate set of hyperparameters requires a retraining of the neural network, it is best to keep the number of candidate sets to a minimum. Bayesian Hyperparameter Optimization achieves this by training a model to predict good candidate sets of hyperparameters. [Cite:snoek2012practical]

- bayesian-optimization
- hyperopt
- spearmint

In [2]:

```
# Ignore useless W0819 warnings generated by TensorFlow 2.0.
# Hopefully can remove this ignore in the future.
# See https://github.com/tensorflow/tensorflow/issues/31308
import logging, os
logging.disable(logging.WARNING)
os.environ["TF_CPP_MIN_LOG_LEVEL"] = "3"

import pandas as pd
from scipy.stats import zscore

# Read the data set
df = pd.read_csv(
    "https://data.heatonresearch.com/data/t81-558/jh-simple-dataset.csv",
    na_values=['NA','?'])
```

```
# Generate dummies for job
df = pd.concat([df,pd.get_dummies(df['job'],prefix="job")],axis=1)
df.drop('job', axis=1, inplace=True)

# Generate dummies for area
df = pd.concat([df,pd.get_dummies(df['area'],prefix="area")],axis=1)
df.drop('area', axis=1, inplace=True)

# Missing values for income
med = df['income'].median()
df['income'] = df['income'].fillna(med)

# Standardize ranges
df['income'] = zscore(df['income'])
df['aspect'] = zscore(df['aspect'])
df['save_rate'] = zscore(df['save_rate'])
df['age'] = zscore(df['age'])
df['subscriptions'] = zscore(df['subscriptions'])

# Convert to numpy - Classification
x_columns = df.columns.drop('product').drop('id')
x = df[x_columns].values
dummies = pd.get_dummies(df['product']) # Classification
products = dummies.columns
y = dummies.values
```

Now that we've preprocessed the data, we can begin the hyperparameter optimization. We start by creating a function that generates the model based on just three parameters. Bayesian optimization works on a vector of numbers, not on a problematic notion like how many layers and neurons are on each layer. To represent this complex neuron structure as a vector, we use several numbers to describe this structure.

- **dropout** - The dropout percent for each layer.
- **neuronPct** - What percent of our fixed 5,000 maximum number of neurons do we wish to use? This parameter specifies the total count of neurons in the entire network.
- **neuronShrink** - Neural networks usually start with more neurons on the first hidden layer and then decrease this count for additional layers. This percent specifies how much to shrink subsequent layers based on the previous layer. We stop adding more layers once we run out of neurons (the count specified by neuronPct).

These three numbers define the structure of the neural network. The commands in the below code show exactly how the

These three numbers define the structure of the neural network. The commends in the below code show exactly how the program constructs the network.

In [3]:

```
import pandas as pd
import os
import numpy as np
import time
import tensorflow.keras.initializers
import statistics
import tensorflow.keras
from sklearn import metrics
from sklearn.model_selection import StratifiedKFold
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation, Dropout, InputLayer
from tensorflow.keras import regularizers
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.model_selection import ShuffleSplit
from tensorflow.keras.layers import LeakyReLU,PReLU
from tensorflow.keras.optimizers import Adam

def generate_model(dropout, neuronPct, neuronShrink):
    # We start with some percent of 5000 starting neurons on
    # the first hidden layer.
    neuronCount = int(neuronPct * 5000)

    # Construct neural network
    model = Sequential()

    # So long as there would have been at least 25 neurons and
    # fewer than 10
    # layers, create a new layer.
    layer = 0
    while neuronCount>25 and layer<10:
        # The first (0th) layer needs an input input_dim(neuronCount)
        if layer==0:
            model.add(Dense(neuronCount,
                input_dim=x.shape[1],
                activation=PReLU()))
        else:
            model.add(Dense(neuronCount, activation=PReLU()))
        layer += 1
        # Add dropout after each hidden layer
        model.add(Dropout(dropout))

        # Shrink neuron count for each layer
        neuronCount = neuronCount * neuronShrink

    model.add(Dense(y.shape[1],activation='softmax')) # Output
    return model
```

We can test this code to see how it creates a neural network based on three such parameters.

In [4]:

```
# Generate a model and see what the resulting structure looks like.
model = generate_model(dropout=0.2, neuronPct=0.1, neuronShrink=0.25)
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 500)               24500

 dropout (Dropout)           (None, 500)               0

 dense_1 (Dense)             (None, 125)               62750

 dropout_1 (Dropout)         (None, 125)               0

 dense_2 (Dense)             (None, 31)                3937

 dropout_2 (Dropout)         (None, 31)                0

 dense_3 (Dense)             (None, 7)                 224

=================================================================
Total params: 91,411
Trainable params: 91,411
Non-trainable params: 0
_________________________________________________________________
```

We will now create a function to evaluate the neural network using three such parameters. We use bootstrapping because one training run might have "bad luck" with the assigned random weights. We use this function to train and then evaluate the neural network.

In [5]:

```
SPLITS = 2
EPOCHS = 500
PATIENCE = 10

def evaluate_network(dropout,learning_rate,neuronPct,neuronShrink):
    # Bootstrap

    # for Classification
    boot = StratifiedShuffleSplit(n_splits=SPLITS, test_size=0.1)
    # for Regression
    # boot = ShuffleSplit(n_splits=SPLITS, test_size=0.1)

    # Track progress
    mean_benchmark = []
    epochs_needed = []
    num = 0

    # Loop through samples
    for train, test in boot.split(x,df['product']):
        start_time = time.time()
        num+=1

        # Split train and test
        x_train = x[train]
        y_train = y[train]
        x_test = x[test]
        y_test = y[test]

        model = generate_model(dropout, neuronPct, neuronShrink)
        model.compile(loss='categorical_crossentropy',
                      optimizer=Adam(learning_rate=learning_rate))
        monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3,
        patience=PATIENCE, verbose=0, mode='auto',
                                restore_best_weights=True)

        # Train on the bootstrap sample
        model.fit(x_train,y_train,validation_data=(x_test,y_test),
                  callbacks=[monitor],verbose=0,epochs=EPOCHS)
        epochs = monitor.stopped_epoch
```

```
            epochs = monitor.stopped_epoch
            epochs_needed.append(epochs)

            # Predict on the out of boot (validation)
            pred = model.predict(x_test)

            # Measure this bootstrap's log loss
            y_compare = np.argmax(y_test,axis=1) # For log loss calculation
            score = metrics.log_loss(y_compare, pred)
            mean_benchmark.append(score)
            m1 = statistics.mean(mean_benchmark)
            m2 = statistics.mean(epochs_needed)
            mdev = statistics.pstdev(mean_benchmark)

            # Record this iteration
            time_took = time.time() - start_time

    tensorflow.keras.backend.clear_session()
    return (-m1)
```

You can try any combination of our three hyperparameters, plus the learning rate, to see how effective these four numbers are. Of course, our goal is not to manually choose different combinations of these four hyperparameters; we seek to automate.

In [6]:
```
print(evaluate_network(
    dropout=0.2,
    learning_rate=1e-3,
    neuronPct=0.2,
    neuronShrink=0.2))
```

```
-0.6668764846259546
```

First, we must install the Bayesian optimization package if we are in Colab.

In [7]:
```
# HIDE OUTPUT
!pip install bayesian-optimization
```

```
Requirement already satisfied: bayesian-optimization in /usr/local/lib/python3.7/dist-packages (1.2.0)
Requirement already satisfied: scipy>=0.14.0 in /usr/local/lib/python3.7/dist-packages (from bayesian-optimizat
ion) (1.4.1)
Requirement already satisfied: scikit-learn>=0.18.0 in /usr/local/lib/python3.7/dist-packages (from bayesian-op
timization) (1.0.2)
Requirement already satisfied: numpy>=1.9.0 in /usr/local/lib/python3.7/dist-packages (from bayesian-optimizati
on) (1.21.5)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-packages (from scikit-lear
n>=0.18.0->bayesian-optimization) (3.1.0)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages (from scikit-learn>=0.1
8.0->bayesian-optimization) (1.1.0)
```

We will now automate this process. We define the bounds for each of these four hyperparameters and begin the Bayesian optimization. Once the program finishes, the best combination of hyperparameters found is displayed. The **optimize** function accepts two parameters that will significantly impact how long the process takes to complete:

- **n_iter** - How many steps of Bayesian optimization that you want to perform. The more steps, the more likely you will find a reasonable maximum.
- **init_points**: How many steps of random exploration that you want to perform. Random exploration can help by diversifying the exploration space.

In [8]:

```python
from bayes_opt import BayesianOptimization
import time

# Supress NaN warnings
import warnings
warnings.filterwarnings("ignore",category =RuntimeWarning)

# Bounded region of parameter space
pbounds = {'dropout': (0.0, 0.499),
           'learning_rate': (0.0, 0.1),
           'neuronPct': (0.01, 1),
           'neuronShrink': (0.01, 1)
          }

optimizer = BayesianOptimization(
    f=evaluate_network,
    pbounds=pbounds,
    verbose=2,  # verbose = 1 prints only when a maximum
    # is observed, verbose = 0 is silent
    random_state=1,
)

start_time = time.time()
```

```
optimizer.maximize(init_points=10, n_iter=20,)
time_took = time.time() - start_time

print(f"Total runtime: {hms_string(time_took)}")
print(optimizer.max)
```

```
|   iter    |  target   |  dropout  | learni... | neuronPct | neuron... |
-------------------------------------------------------------------------
|  1        | -0.8092   |  0.2081   |  0.07203  |  0.01011  |  0.3093   |
|  2        | -0.7167   |  0.07323  |  0.009234 |  0.1944   |  0.3521   |
|  3        | -17.87    |  0.198    |  0.05388  |  0.425    |  0.6884   |
|  4        | -0.8022   |  0.102    |  0.08781  |  0.03711  |  0.6738   |
|  5        | -0.9209   |  0.2082   |  0.05587  |  0.149    |  0.2061   |
|  6        | -17.96    |  0.3996   |  0.09683  |  0.3203   |  0.6954   |
|  7        | -4.223    |  0.4373   |  0.08946  |  0.09419  |  0.04866  |
|  8        | -0.7025   |  0.08475  |  0.08781  |  0.1074   |  0.4269   |
|  9        | -8.666    |  0.478    |  0.05332  |  0.695    |  0.3224   |
|  10       | -9.785    |  0.3426   |  0.08346  |  0.02811  |  0.7526   |
|  11       | -4.881    |  0.0      |  0.0      |  0.01     |  0.01     |
|  12       | -21.59    |  0.2208   |  0.04135  |  0.5523   |  0.7468   |
|  13       | -1.819    |  0.0      |  0.0      |  1.0      |  0.01     |
|  14       | -33.33    |  0.01058  |  0.08079  |  0.9652   |  0.7051   |
|  15       | -1.418    |  0.4963   |  0.02476  |  0.9744   |  0.01896  |
|  16       | -1.876    |  0.1247   |  0.0      |  0.5781   |  0.01     |
|  17       | -1.898    |  0.0      |  0.0      |  0.01     |  1.0      |
|  18       | -17.7     |  0.1621   |  0.06358  |  0.4065   |  0.754    |
|  19       | -2.674    |  0.0      |  0.0      |  0.01     |  0.5464   |
|  20       | -1.931    |  0.499    |  0.0      |  0.5538   |  0.01     |
|  21       | -3.402    |  0.004722 |  0.05502  |  0.1704   |  0.483    |
|  22       | -2.8      |  0.08639  |  0.0838   |  0.04668  |  0.6864   |
|  23       | -20.98    |  0.1168   |  0.0447   |  0.5546   |  0.9497   |
|  24       | -4.565    |  0.2554   |  0.1      |  0.8418   |  0.01     |
|  25       | -4.724    |  0.0      |  0.1      |  0.3368   |  0.01     |
|  26       | -0.6956   |  0.2505   |  0.007623 |  0.01265  |  0.523    |
|  27       | -0.7139   |  0.2967   |  0.01162  |  0.3735   |  0.01708  |
|  28       | -2.145    |  0.499    |  0.0      |  0.3053   |  0.2207   |
|  29       | -2.069    |  0.0      |  0.0      |  0.4808   |  0.2473   |
|  30       | -0.7155   |  0.4082   |  0.01635  |  0.02488  |  0.1694   |
=========================================================================
Total runtime: 1:36:11.56
{'target': -0.6955536706512794, 'params': {'dropout': 0.2504561773412203, 'learning_rate': 0.007623234670914292
4, 'neuronPct': 0.012648791521811826, 'neuronShrink': 0.5229748831552032}}
```

As you can see, the algorithm performed 30 total iterations. This total iteration count includes ten random and 20 optimization iterations.