Open in Colab

T81-558: Applications of Deep Neural Networks

Module 5: Regularization and Dropout

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

Module 5 Material

- Part 5.1: Part 5.1: Introduction to Regularization: Ridge and Lasso [Video] [Notebook]
- Part 5.2: Using K-Fold Cross Validation with Keras [Video] [Notebook]
- Part 5.3: Using L1 and L2 Regularization with Keras to Decrease Overfitting [Video] [Notebook]
- Part 5.4: Drop Out for Keras to Decrease Overfitting [Video] [Notebook]
- **Part 5.5: Benchmarking Keras Deep Learning Regularization Techniques** [Video] [Notebook]

Google CoLab Instructions

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

In [5]:
```
try:
    %tensorflow_version 2.x
    COLAB = True
    print("Note: using Google CoLab")
except:
    print("Note: not using Google CoLab")
    COLAB = False
```

```
Note: using Google CoLab
```

Part 5.5: Benchmarking Regularization Techniques

Quite a few hyperparameters have been introduced so far. Tweaking each of these values can have an effect on the score obtained by your neural networks. Some of the hyperparameters seen so far include:

- Number of layers in the neural network
- How many neurons in each layer
- What activation functions to use on each layer
- Dropout percent on each layer
- L1 and L2 values on each layer

To try out each of these hyperparameters you will need to run train neural networks with multiple settings for each hyperparameter. However, you may have noticed that neural networks often produce somewhat different results when trained multiple times. This is because the neural networks start with random weights. Because of this it is necessary to fit and evaluate a neural network times to ensure that one set of hyperparameters are actually better than another. Bootstrapping can be an effective means of benchmarking (comparing) two sets of hyperparameters.

Bootstrapping is similar to cross-validation. Both go through a number of cycles/folds providing validation and training sets. However, bootstrapping can have an unlimited number of cycles. Bootstrapping chooses a new train and validation split each cycle, with replacement. The fact that each cycle is chosen with replacement means that, unlike cross validation, there will often be repeated rows selected between cycles. If you run the bootstrap for enough cycles, there will be duplicate cycles.

In this part we will use bootstrapping for hyperparameter benchmarking. We will train a neural network for a specified number of splits (denoted by the SPLITS constant). For these examples we use 100. We will compare the average score at the end of the 100. By the end of the cycles the mean score will have converged somewhat. This ending score will be a much better basis of comparison than a single cross-validation. Additionally, the average number of epochs will be tracked to give an idea of a possible optimal value. Because the early stopping validation set is also used to evaluate the the neural network as well, it might be slightly inflated. This is because we are both stopping and evaluating on the same sample. However, we are using the scores only as relative measures to determine the superiority of one set of hyperparameters to another, so this slight inflation should not present too much of a problem.

Because we are benchmarking, we will display the amount of time taken for each cycle. The following function can be used to nicely format a time span.

In [6]:
```
# 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)
```

Bootstrapping for Regression

Regression bootstrapping uses the **ShuffleSplit** object to perform the splits. This technique is similar to **KFold** for cross-validation; no balancing occurs. We will attempt to predict the age column for the **jh-simple-dataset**; the following code loads this data.

In [7]:

```
import pandas as pd
from scipy.stats import zscore
from sklearn.model_selection import train_test_split

# 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)

# Generate dummies for product
df = pd.concat([df,pd.get_dummies(df['product'],prefix="product")],axis=1)
df.drop('product', 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['subscriptions'] = zscore(df['subscriptions'])

# Convert to numpy - Classification
x_columns = df.columns.drop('age').drop('id')
x = df[x_columns].values
y = df['age'].values
```

The following code performs the bootstrap. The architecture of the neural network can be adjusted to compare many different configurations.

In [8]:

```
import pandas as pd
import os
import numpy as np
import time
import statistics
from sklearn import metrics
from sklearn.model_selection import StratifiedKFold
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation
from tensorflow.keras import regularizers
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.model_selection import ShuffleSplit

SPLITS = 50

# Bootstrap
boot = ShuffleSplit(n_splits=SPLITS, test_size=0.1, random_state=42)

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

# Loop through samples
for train, test in boot.split(x):
    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]

    # Construct neural network
    model = Sequential()
    model.add(Dense(20, input_dim=x_train.shape[1], activation='relu'))
    model.add(Dense(10, activation='relu'))
    model.add(Dense(1))
    model.compile(loss='mean_squared_error', optimizer='adam')

    monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3,
        patience=5, 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=1000)
    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
    score = np.sqrt(metrics.mean_squared_error(pred,y_test))
```

```
    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
    print(f"#{num}: score={score:.6f}, mean score={m1:.6f},"
          f" stdev={mdev:.6f}",
          f" epochs={epochs}, mean epochs={int(m2)}",
          f" time={hms_string(time_took)}")
```

```
#1: score=0.630750, mean score=0.630750, stdev=0.000000  epochs=147, mean ep
ochs=147  time=0:00:12.56
#2: score=1.020895, mean score=0.825823, stdev=0.195072  epochs=101, mean ep
ochs=124  time=0:00:08.70
#3: score=0.803801, mean score=0.818482, stdev=0.159614  epochs=155, mean ep
ochs=134  time=0:00:20.85
#4: score=0.540871, mean score=0.749079, stdev=0.183188  epochs=122, mean ep
ochs=131  time=0:00:10.64
#5: score=0.802589, mean score=0.759781, stdev=0.165240  epochs=116, mean ep
ochs=128  time=0:00:10.84
#6: score=0.862807, mean score=0.776952, stdev=0.155653  epochs=108, mean ep
ochs=124  time=0:00:10.65
#7: score=0.550373, mean score=0.744584, stdev=0.164478  epochs=131, mean ep
ochs=125  time=0:00:10.85
#8: score=0.659148, mean score=0.733904, stdev=0.156428  epochs=118, mean ep
ochs=124  time=0:00:10.10
#9: score=0.606425, mean score=0.719740, stdev=0.152826  epochs=99, mean epo
chs=121  time=0:00:10.64
#10: score=1.169816, mean score=0.764748, stdev=0.198120  epochs=101, mean e
pochs=119  time=0:00:10.65
#11: score=0.985013, mean score=0.784772, stdev=0.199231  epochs=106, mean e
pochs=118  time=0:00:09.02
#12: score=0.857432, mean score=0.790827, stdev=0.191803  epochs=113, mean e
pochs=118  time=0:00:09.46
#13: score=0.495272, mean score=0.768092, stdev=0.200402  epochs=151, mean e
pochs=120  time=0:00:20.88
#14: score=1.079376, mean score=0.790326, stdev=0.209092  epochs=104, mean e
pochs=119  time=0:00:10.65
#15: score=0.616606, mean score=0.778745, stdev=0.206597  epochs=130, mean e
pochs=120  time=0:00:10.70
#16: score=0.781853, mean score=0.778939, stdev=0.200038  epochs=123, mean e
pochs=120  time=0:00:10.69
#17: score=0.781730, mean score=0.779103, stdev=0.194067  epochs=116, mean e
pochs=120  time=0:00:10.64
#18: score=0.845470, mean score=0.782790, stdev=0.189211  epochs=143, mean e
pochs=121  time=0:00:20.89
#19: score=0.643181, mean score=0.775442, stdev=0.186784  epochs=124, mean e
pochs=121  time=0:00:10.63
#20: score=1.026157, mean score=0.787978, stdev=0.190078  epochs=91, mean ep
ochs=119  time=0:00:10.63
#21: score=0.587819, mean score=0.778447, stdev=0.190332  epochs=106, mean e
pochs=119  time=0:00:10.62
#22: score=0.600830, mean score=0.770373, stdev=0.189600  epochs=117, mean e
pochs=119  time=0:00:10.32
#23: score=0.662913, mean score=0.765701, stdev=0.186723  epochs=126, mean e
pochs=119  time=0:00:20.89
#24: score=0.671352, mean score=0.761770, stdev=0.183762  epochs=130, mean e
pochs=119  time=0:00:20.87
#25: score=0.647940, mean score=0.757217, stdev=0.181425  epochs=143, mean e
pochs=120  time=0:00:12.29
#26: score=0.684534, mean score=0.754421, stdev=0.178450  epochs=94, mean ep
ochs=119  time=0:00:08.29
#27: score=0.534195, mean score=0.746265, stdev=0.179986  epochs=149, mean e
pochs=120  time=0:00:20.91
#28: score=0.901485, mean score=0.751808, stdev=0.179074  epochs=110, mean e
pochs=120  time=0:00:10.66
```

```
#29: score=0.696614, mean score=0.749905, stdev=0.176248  epochs=117, mean e
pochs=120  time=0:00:10.46
#30: score=0.656065, mean score=0.746777, stdev=0.174102  epochs=109, mean e
pochs=120  time=0:00:10.63
#31: score=0.749652, mean score=0.746870, stdev=0.171272  epochs=118, mean e
pochs=119  time=0:00:10.66
#32: score=0.508090, mean score=0.739408, stdev=0.173619  epochs=106, mean e
pochs=119  time=0:00:10.66
#33: score=0.732891, mean score=0.739210, stdev=0.170971  epochs=124, mean e
pochs=119  time=0:00:10.76
#34: score=1.089590, mean score=0.749516, stdev=0.178539  epochs=95, mean ep
ochs=118  time=0:00:08.24
#35: score=0.568665, mean score=0.744349, stdev=0.178530  epochs=115, mean e
pochs=118  time=0:00:10.64
#36: score=0.523255, mean score=0.738207, stdev=0.179744  epochs=108, mean e
pochs=118  time=0:00:09.23
#37: score=1.082163, mean score=0.747503, stdev=0.185865  epochs=87, mean ep
ochs=117  time=0:00:10.62
#38: score=0.752920, mean score=0.747646, stdev=0.183405  epochs=125, mean e
pochs=117  time=0:00:10.66
#39: score=0.587106, mean score=0.743529, stdev=0.182808  epochs=118, mean e
pochs=117  time=0:00:10.18
#40: score=0.781335, mean score=0.744474, stdev=0.180605  epochs=103, mean e
pochs=117  time=0:00:10.64
#41: score=1.209243, mean score=0.755810, stdev=0.192257  epochs=82, mean ep
ochs=116  time=0:00:07.39
#42: score=0.650733, mean score=0.753308, stdev=0.190628  epochs=141, mean e
pochs=117  time=0:00:21.19
#43: score=0.622103, mean score=0.750257, stdev=0.189434  epochs=116, mean e
pochs=117  time=0:00:09.85
#44: score=0.519172, mean score=0.745005, stdev=0.190409  epochs=135, mean e
pochs=117  time=0:00:11.94
#45: score=0.926205, mean score=0.749032, stdev=0.190167  epochs=87, mean ep
ochs=116  time=0:00:07.78
#46: score=0.604350, mean score=0.745887, stdev=0.189268  epochs=78, mean ep
ochs=116  time=0:00:10.64
#47: score=0.690874, mean score=0.744716, stdev=0.187412  epochs=136, mean e
pochs=116  time=0:00:20.86
#48: score=0.719645, mean score=0.744194, stdev=0.185484  epochs=112, mean e
pochs=116  time=0:00:09.33
#49: score=0.911419, mean score=0.747607, stdev=0.185098  epochs=124, mean e
pochs=116  time=0:00:10.66
#50: score=0.599252, mean score=0.744639, stdev=0.184411  epochs=132, mean e
pochs=116  time=0:00:20.91
```

The bootstrapping process for classification is similar, and I present it in the next section.

Bootstrapping for Classification

Regression bootstrapping uses the **StratifiedShuffleSplit** class to perform the splits. This class is similar to **StratifiedKFold** for cross-validation, as the classes are balanced so that the sampling does not affect proportions. To demonstrate this technique, we will attempt to predict the product column for the **jh-simple-dataset**; the following code loads this data.

In [9]:
```
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
```

We now run this data through a number of splits specified by the SPLITS variable. We track the average error through each of these splits.

In [10]:
```
import pandas as pd
import os
import numpy as np
import time
import statistics
from sklearn import metrics
from sklearn.model_selection import StratifiedKFold
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation
from tensorflow.keras import regularizers
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.model_selection import StratifiedShuffleSplit

SPLITS = 50
```

```python
# Bootstrap
boot = StratifiedShuffleSplit(n_splits=SPLITS, test_size=0.1,
                                random_state=42)

# 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]

    # Construct neural network
    model = Sequential()
    model.add(Dense(50, input_dim=x.shape[1], activation='relu')) # Hidden 1
    model.add(Dense(25, activation='relu')) # Hidden 2
    model.add(Dense(y.shape[1],activation='softmax')) # Output
    model.compile(loss='categorical_crossentropy', optimizer='adam')
    monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3,
        patience=25, 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=1000)
    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
    print(f"#{num}: score={score:.6f}, mean score={m1:.6f}," +\
          f"stdev={mdev:.6f}, epochs={epochs}, mean epochs={int(m2)}," +\
          f" time={hms_string(time_took)}")
```

```
#1: score=0.666342, mean score=0.666342,stdev=0.000000, epochs=66, mean epoc
hs=66, time=0:00:06.31
#2: score=0.645598, mean score=0.655970,stdev=0.010372, epochs=59, mean epoc
hs=62, time=0:00:10.63
#3: score=0.676924, mean score=0.662955,stdev=0.013011, epochs=66, mean epoc
hs=63, time=0:00:10.64
#4: score=0.672602, mean score=0.665366,stdev=0.012017, epochs=84, mean epoc
hs=68, time=0:00:08.20
#5: score=0.667274, mean score=0.665748,stdev=0.010776, epochs=73, mean epoc
hs=69, time=0:00:10.65
#6: score=0.706372, mean score=0.672518,stdev=0.018055, epochs=50, mean epoc
hs=66, time=0:00:04.81
#7: score=0.687937, mean score=0.674721,stdev=0.017565, epochs=71, mean epoc
hs=67, time=0:00:06.89
#8: score=0.734794, mean score=0.682230,stdev=0.025781, epochs=43, mean epoc
hs=64, time=0:00:05.51
#9: score=0.623972, mean score=0.675757,stdev=0.030431, epochs=65, mean epoc
hs=64, time=0:00:10.66
#10: score=0.650303, mean score=0.673212,stdev=0.029862, epochs=109, mean ep
ochs=68, time=0:00:10.63
#11: score=0.679500, mean score=0.673783,stdev=0.028529, epochs=83, mean epo
chs=69, time=0:00:10.63
#12: score=0.736851, mean score=0.679039,stdev=0.032403, epochs=51, mean epo
chs=68, time=0:00:05.51
#13: score=0.703048, mean score=0.680886,stdev=0.031782, epochs=92, mean epo
chs=70, time=0:00:08.48
#14: score=0.733015, mean score=0.684609,stdev=0.033439, epochs=52, mean epo
chs=68, time=0:00:05.13
#15: score=0.664863, mean score=0.683293,stdev=0.032679, epochs=77, mean epo
chs=69, time=0:00:10.62
#16: score=0.740248, mean score=0.686853,stdev=0.034514, epochs=79, mean epo
chs=70, time=0:00:10.94
#17: score=0.639677, mean score=0.684078,stdev=0.035276, epochs=82, mean epo
chs=70, time=0:00:10.64
#18: score=0.648893, mean score=0.682123,stdev=0.035216, epochs=64, mean epo
chs=70, time=0:00:06.14
#19: score=0.603215, mean score=0.677970,stdev=0.038541, epochs=60, mean epo
chs=69, time=0:00:10.72
#20: score=0.691074, mean score=0.678625,stdev=0.037673, epochs=49, mean epo
chs=68, time=0:00:05.07
#21: score=0.649008, mean score=0.677215,stdev=0.037302, epochs=54, mean epo
chs=68, time=0:00:05.51
#22: score=0.745487, mean score=0.680318,stdev=0.039121, epochs=39, mean epo
chs=66, time=0:00:05.54
#23: score=0.588884, mean score=0.676343,stdev=0.042563, epochs=74, mean epo
chs=67, time=0:00:07.11
#24: score=0.697504, mean score=0.677224,stdev=0.041881, epochs=61, mean epo
chs=66, time=0:00:05.86
#25: score=0.569334, mean score=0.672909,stdev=0.046161, epochs=64, mean epo
chs=66, time=0:00:10.67
#26: score=0.632199, mean score=0.671343,stdev=0.045936, epochs=65, mean epo
chs=66, time=0:00:06.16
#27: score=0.707666, mean score=0.672688,stdev=0.045597, epochs=74, mean epo
chs=66, time=0:00:07.34
#28: score=0.747781, mean score=0.675370,stdev=0.046894, epochs=48, mean epo
chs=66, time=0:00:05.56
```

```
#29: score=0.648160, mean score=0.674432,stdev=0.046345, epochs=61, mean epo
chs=66, time=0:00:05.80
#30: score=0.695912, mean score=0.675148,stdev=0.045729, epochs=70, mean epo
chs=66, time=0:00:06.72
#31: score=0.692880, mean score=0.675720,stdev=0.045094, epochs=61, mean epo
chs=66, time=0:00:10.65
#32: score=0.675613, mean score=0.675717,stdev=0.044384, epochs=73, mean epo
chs=66, time=0:00:10.66
#33: score=0.625625, mean score=0.674199,stdev=0.044542, epochs=57, mean epo
chs=65, time=0:00:05.34
#34: score=0.571148, mean score=0.671168,stdev=0.047210, epochs=130, mean ep
ochs=67, time=0:00:20.88
#35: score=0.542365, mean score=0.667488,stdev=0.051240, epochs=75, mean epo
chs=68, time=0:00:10.67
#36: score=0.645099, mean score=0.666866,stdev=0.050657, epochs=59, mean epo
chs=67, time=0:00:05.59
#37: score=0.639249, mean score=0.666119,stdev=0.050168, epochs=78, mean epo
chs=68, time=0:00:10.68
#38: score=0.684326, mean score=0.666598,stdev=0.049589, epochs=75, mean epo
chs=68, time=0:00:10.63
#39: score=0.728835, mean score=0.668194,stdev=0.049928, epochs=79, mean epo
chs=68, time=0:00:07.78
#40: score=0.706089, mean score=0.669142,stdev=0.049654, epochs=46, mean epo
chs=67, time=0:00:04.59
#41: score=0.727177, mean score=0.670557,stdev=0.049855, epochs=68, mean epo
chs=67, time=0:00:10.69
#42: score=0.653240, mean score=0.670145,stdev=0.049329, epochs=53, mean epo
chs=67, time=0:00:05.17
#43: score=0.692113, mean score=0.670656,stdev=0.048864, epochs=51, mean epo
chs=67, time=0:00:05.56
#44: score=0.745355, mean score=0.672353,stdev=0.049572, epochs=66, mean epo
chs=67, time=0:00:10.66
#45: score=0.631125, mean score=0.671437,stdev=0.049393, epochs=75, mean epo
chs=67, time=0:00:07.16
#46: score=0.664004, mean score=0.671276,stdev=0.048865, epochs=57, mean epo
chs=67, time=0:00:05.69
#47: score=0.686937, mean score=0.671609,stdev=0.048395, epochs=52, mean epo
chs=66, time=0:00:05.07
#48: score=0.760827, mean score=0.673468,stdev=0.049555, epochs=40, mean epo
chs=66, time=0:00:04.14
#49: score=0.665493, mean score=0.673305,stdev=0.049060, epochs=60, mean epo
chs=66, time=0:00:10.65
#50: score=0.692625, mean score=0.673691,stdev=0.048642, epochs=55, mean epo
chs=65, time=0:00:05.22
```

Benchmarking

Now that we've seen how to bootstrap with both classification and regression, we can start to try to optimize the hyperparameters for the **jh-simple-dataset** data. For this example, we will encode for classification of the product column. Evaluation will be in log loss.

In [11]:

```
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
```

I performed some optimization, and the code has the best settings that I could determine. Later in this book, we will see how we can use an automatic process to optimize the hyperparameters.

In [13]:

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

```python
SPLITS = 100

# Bootstrap
boot = StratifiedShuffleSplit(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]

    # Construct neural network
    model = Sequential()
    model.add(Dense(100, input_dim=x.shape[1], activation=PReLU(), \
        kernel_regularizer=regularizers.l2(1e-4))) # Hidden 1
    model.add(Dropout(0.5))
    model.add(Dense(100, activation=PReLU(), \
        activity_regularizer=regularizers.l2(1e-4))) # Hidden 2
    model.add(Dropout(0.5))
    model.add(Dense(100, activation=PReLU(), \
        activity_regularizer=regularizers.l2(1e-4)
    )) # Hidden 3
#    model.add(Dropout(0.5)) - Usually better performance
# without dropout on final layer
    model.add(Dense(y.shape[1],activation='softmax')) # Output
    model.compile(loss='categorical_crossentropy', optimizer='adam')
    monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3,
        patience=100, 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=1000)
    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
    print(f"#{num}: score={score:.6f}, mean score={m1:.6f},"
          f"stdev={mdev:.6f}, epochs={epochs},"
          f"mean epochs={int(m2)}, time={hms_string(time_took)}")
```

```
#1: score=0.642887, mean score=0.642887,stdev=0.000000, epochs=325,mean epoc
hs=325, time=0:00:42.10
#2: score=0.555518, mean score=0.599202,stdev=0.043684, epochs=208,mean epoc
hs=266, time=0:00:41.74
#3: score=0.605537, mean score=0.601314,stdev=0.035793, epochs=187,mean epoc
hs=240, time=0:00:24.22
#4: score=0.609415, mean score=0.603339,stdev=0.031195, epochs=250,mean epoc
hs=242, time=0:00:41.72
#5: score=0.619657, mean score=0.606603,stdev=0.028655, epochs=201,mean epoc
hs=234, time=0:00:26.10
#6: score=0.638641, mean score=0.611943,stdev=0.028755, epochs=172,mean epoc
hs=223, time=0:00:41.73
#7: score=0.671137, mean score=0.620399,stdev=0.033731, epochs=203,mean epoc
hs=220, time=0:00:26.58
#8: score=0.635294, mean score=0.622261,stdev=0.031935, epochs=209,mean epoc
hs=219, time=0:00:41.74
#9: score=0.633694, mean score=0.623531,stdev=0.030322, epochs=162,mean epoc
hs=213, time=0:00:41.78
#10: score=0.596081, mean score=0.620786,stdev=0.029921, epochs=197,mean epo
chs=211, time=0:00:41.74
#11: score=0.583717, mean score=0.617416,stdev=0.030454, epochs=232,mean epo
chs=213, time=0:00:41.77
#12: score=0.686736, mean score=0.623193,stdev=0.034889, epochs=216,mean epo
chs=213, time=0:00:28.25
#13: score=0.684454, mean score=0.627905,stdev=0.037284, epochs=134,mean epo
chs=207, time=0:00:21.58
#14: score=0.573696, mean score=0.624033,stdev=0.038545, epochs=184,mean epo
chs=205, time=0:00:23.81
#15: score=0.723944, mean score=0.630694,stdev=0.044808, epochs=170,mean epo
chs=203, time=0:00:41.80
#16: score=0.659891, mean score=0.632519,stdev=0.043957, epochs=203,mean epo
chs=203, time=0:00:41.80
#17: score=0.569637, mean score=0.628820,stdev=0.045139, epochs=204,mean epo
chs=203, time=0:00:41.77
#18: score=0.608905, mean score=0.627713,stdev=0.044103, epochs=233,mean epo
chs=205, time=0:00:41.76
#19: score=0.734381, mean score=0.633328,stdev=0.049092, epochs=193,mean epo
chs=204, time=0:00:25.25
#20: score=0.587099, mean score=0.631016,stdev=0.048899, epochs=252,mean epo
chs=206, time=0:00:42.07
#21: score=0.661902, mean score=0.632487,stdev=0.048171, epochs=211,mean epo
chs=206, time=0:00:41.79
#22: score=0.656783, mean score=0.633591,stdev=0.047335, epochs=145,mean epo
chs=204, time=0:00:19.19
#23: score=0.611230, mean score=0.632619,stdev=0.046519, epochs=201,mean epo
chs=204, time=0:00:41.77
#24: score=0.638759, mean score=0.632875,stdev=0.045556, epochs=223,mean epo
chs=204, time=0:00:28.67
#25: score=0.635676, mean score=0.632987,stdev=0.044639, epochs=240,mean epo
chs=206, time=0:00:41.77
#26: score=0.599321, mean score=0.631692,stdev=0.044248, epochs=199,mean epo
chs=205, time=0:00:42.10
#27: score=0.696892, mean score=0.634107,stdev=0.045133, epochs=146,mean epo
chs=203, time=0:00:21.28
#28: score=0.637397, mean score=0.634224,stdev=0.044324, epochs=179,mean epo
chs=202, time=0:00:23.53
```

```
#29: score=0.645323, mean score=0.634607,stdev=0.043600, epochs=256,mean epo
chs=204, time=0:00:32.44
#30: score=0.588104, mean score=0.633057,stdev=0.043672, epochs=199,mean epo
chs=204, time=0:00:25.96
#31: score=0.676097, mean score=0.634445,stdev=0.043630, epochs=229,mean epo
chs=205, time=0:00:41.74
#32: score=0.667709, mean score=0.635485,stdev=0.043331, epochs=155,mean epo
chs=203, time=0:00:20.61
#33: score=0.616544, mean score=0.634911,stdev=0.042793, epochs=283,mean epo
chs=206, time=0:00:36.55
#34: score=0.622340, mean score=0.634541,stdev=0.042212, epochs=174,mean epo
chs=205, time=0:00:22.82
#35: score=0.665123, mean score=0.635415,stdev=0.041916, epochs=205,mean epo
chs=205, time=0:00:27.05
#36: score=0.573597, mean score=0.633698,stdev=0.042560, epochs=205,mean epo
chs=205, time=0:00:41.81
#37: score=0.617111, mean score=0.633249,stdev=0.042067, epochs=253,mean epo
chs=206, time=0:00:31.92
#38: score=0.627494, mean score=0.633098,stdev=0.041520, epochs=205,mean epo
chs=206, time=0:00:41.76
#39: score=0.669212, mean score=0.634024,stdev=0.041380, epochs=193,mean epo
chs=206, time=0:00:42.14
#40: score=0.684894, mean score=0.635296,stdev=0.041624, epochs=171,mean epo
chs=205, time=0:00:21.86
#41: score=0.648313, mean score=0.635613,stdev=0.041162, epochs=205,mean epo
chs=205, time=0:00:41.74
#42: score=0.679919, mean score=0.636668,stdev=0.041226, epochs=251,mean epo
chs=206, time=0:00:41.74
#43: score=0.701787, mean score=0.638183,stdev=0.041909, epochs=146,mean epo
chs=204, time=0:00:21.29
#44: score=0.660646, mean score=0.638693,stdev=0.041566, epochs=168,mean epo
chs=204, time=0:00:41.80
#45: score=0.660335, mean score=0.639174,stdev=0.041225, epochs=136,mean epo
chs=202, time=0:00:21.67
#46: score=0.656875, mean score=0.639559,stdev=0.040856, epochs=154,mean epo
chs=201, time=0:00:21.33
#47: score=0.679169, mean score=0.640402,stdev=0.040821, epochs=286,mean epo
chs=203, time=0:00:41.78
#48: score=0.608082, mean score=0.639728,stdev=0.040656, epochs=173,mean epo
chs=202, time=0:00:22.20
#49: score=0.590421, mean score=0.638722,stdev=0.040839, epochs=185,mean epo
chs=202, time=0:00:24.23
#50: score=0.616646, mean score=0.638281,stdev=0.040546, epochs=273,mean epo
chs=203, time=0:00:41.76
#51: score=0.683312, mean score=0.639163,stdev=0.040629, epochs=163,mean epo
chs=202, time=0:00:41.76
#52: score=0.686289, mean score=0.640070,stdev=0.040754, epochs=166,mean epo
chs=202, time=0:00:22.03
#53: score=0.701892, mean score=0.641236,stdev=0.041235, epochs=185,mean epo
chs=201, time=0:00:24.02
#54: score=0.647809, mean score=0.641358,stdev=0.040861, epochs=171,mean epo
chs=201, time=0:00:22.38
#55: score=0.678673, mean score=0.642036,stdev=0.040793, epochs=160,mean epo
chs=200, time=0:00:41.77
#56: score=0.594752, mean score=0.641192,stdev=0.040910, epochs=185,mean epo
chs=200, time=0:00:25.42
```

```
#57: score=0.719842, mean score=0.642572,stdev=0.041843, epochs=124,mean epo
chs=198, time=0:00:17.03
#58: score=0.689348, mean score=0.643378,stdev=0.041926, epochs=223,mean epo
chs=199, time=0:00:29.47
#59: score=0.657452, mean score=0.643617,stdev=0.041608, epochs=220,mean epo
chs=199, time=0:00:28.31
#60: score=0.611100, mean score=0.643075,stdev=0.041470, epochs=226,mean epo
chs=200, time=0:00:29.49
#61: score=0.660965, mean score=0.643368,stdev=0.041191, epochs=162,mean epo
chs=199, time=0:00:21.28
#62: score=0.669189, mean score=0.643785,stdev=0.040987, epochs=147,mean epo
chs=198, time=0:00:21.31
#63: score=0.652563, mean score=0.643924,stdev=0.040675, epochs=187,mean epo
chs=198, time=0:00:41.74
#64: score=0.590525, mean score=0.643090,stdev=0.040896, epochs=275,mean epo
chs=199, time=0:00:35.78
#65: score=0.699827, mean score=0.643963,stdev=0.041176, epochs=182,mean epo
chs=199, time=0:00:23.86
#66: score=0.665028, mean score=0.644282,stdev=0.040944, epochs=214,mean epo
chs=199, time=0:00:28.54
#67: score=0.729557, mean score=0.645554,stdev=0.041932, epochs=225,mean epo
chs=199, time=0:00:41.79
#68: score=0.586906, mean score=0.644692,stdev=0.042217, epochs=219,mean epo
chs=200, time=0:00:28.43
#69: score=0.717007, mean score=0.645740,stdev=0.042792, epochs=124,mean epo
chs=199, time=0:00:21.30
#70: score=0.670428, mean score=0.646093,stdev=0.042586, epochs=198,mean epo
chs=199, time=0:00:41.92
#71: score=0.717004, mean score=0.647091,stdev=0.043103, epochs=203,mean epo
chs=199, time=0:00:42.09
#72: score=0.582071, mean score=0.646188,stdev=0.043474, epochs=174,mean epo
chs=198, time=0:00:41.77
#73: score=0.723909, mean score=0.647253,stdev=0.044110, epochs=199,mean epo
chs=198, time=0:00:41.78
#74: score=0.685384, mean score=0.647768,stdev=0.044032, epochs=145,mean epo
chs=198, time=0:00:19.13
#75: score=0.584444, mean score=0.646924,stdev=0.044336, epochs=205,mean epo
chs=198, time=0:00:41.77
#76: score=0.681646, mean score=0.647381,stdev=0.044221, epochs=160,mean epo
chs=197, time=0:00:21.30
#77: score=0.585961, mean score=0.646583,stdev=0.044480, epochs=195,mean epo
chs=197, time=0:00:42.19
#78: score=0.626380, mean score=0.646324,stdev=0.044252, epochs=231,mean epo
chs=198, time=0:00:41.76
#79: score=0.700790, mean score=0.647014,stdev=0.044391, epochs=188,mean epo
chs=197, time=0:00:24.41
#80: score=0.664455, mean score=0.647232,stdev=0.044155, epochs=164,mean epo
chs=197, time=0:00:21.95
#81: score=0.601657, mean score=0.646669,stdev=0.044169, epochs=205,mean epo
chs=197, time=0:00:41.80
#82: score=0.661004, mean score=0.646844,stdev=0.043927, epochs=151,mean epo
chs=197, time=0:00:19.90
#83: score=0.693299, mean score=0.647404,stdev=0.043955, epochs=161,mean epo
chs=196, time=0:00:21.32
#84: score=0.732184, mean score=0.648413,stdev=0.044649, epochs=147,mean epo
chs=196, time=0:00:20.04
```

```
#85: score=0.628028, mean score=0.648173,stdev=0.044440, epochs=197,mean epo
chs=196, time=0:00:25.61
#86: score=0.626073, mean score=0.647916,stdev=0.044245, epochs=176,mean epo
chs=195, time=0:00:23.27
#87: score=0.632806, mean score=0.647742,stdev=0.044019, epochs=261,mean epo
chs=196, time=0:00:41.76
#88: score=0.694768, mean score=0.648277,stdev=0.044051, epochs=204,mean epo
chs=196, time=0:00:41.80
#89: score=0.699703, mean score=0.648855,stdev=0.044137, epochs=183,mean epo
chs=196, time=0:00:23.10
#90: score=0.611230, mean score=0.648437,stdev=0.044068, epochs=270,mean epo
chs=197, time=0:00:34.62
#91: score=0.637264, mean score=0.648314,stdev=0.043841, epochs=257,mean epo
chs=197, time=0:00:41.78
#92: score=0.678976, mean score=0.648647,stdev=0.043718, epochs=158,mean epo
chs=197, time=0:00:21.24
#93: score=0.627937, mean score=0.648424,stdev=0.043534, epochs=218,mean epo
chs=197, time=0:00:41.74
#94: score=0.644387, mean score=0.648381,stdev=0.043304, epochs=197,mean epo
chs=197, time=0:00:41.76
#95: score=0.660005, mean score=0.648504,stdev=0.043092, epochs=167,mean epo
chs=197, time=0:00:21.69
#96: score=0.674187, mean score=0.648771,stdev=0.042946, epochs=174,mean epo
chs=197, time=0:00:22.79
#97: score=0.654942, mean score=0.648835,stdev=0.042729, epochs=162,mean epo
chs=196, time=0:00:21.30
#98: score=0.644139, mean score=0.648787,stdev=0.042513, epochs=173,mean epo
chs=196, time=0:00:22.70
#99: score=0.697473, mean score=0.649279,stdev=0.042577, epochs=172,mean epo
chs=196, time=0:00:41.79
#100: score=0.678298, mean score=0.649569,stdev=0.042462, epochs=169,mean ep
ochs=196, time=0:00:21.90
```