Open in Colab

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]:
```
# Start 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 8.5: Current Semester's Kaggle

Kaggke competition site for current semester:

- Spring 2023 Kaggle Assignment

Previous Kaggle competition sites for this class (NOT this semester's assignment, feel free to use code):

- Fall 2022 Kaggle Assignment
- Spring 2022 Kaggle Assignment
- Fall 2021 Kaggle Assignment
- Spring 2021 Kaggle Assignment
- Fall 2020 Kaggle Assignment
- Spring 2020 Kaggle Assignment
- Fall 2019 Kaggle Assignment
- Spring 2019 Kaggle Assignment
- Fall 2018 Kaggle Assignment
- Spring 2018 Kaggle Assignment
- Fall 2017 Kaggle Assignment
- Spring 2017 Kaggle Assignment
- Fall 2016 Kaggle Assignment

Iris as a Kaggle Competition

If I used the Iris data as a Kaggle, I would give you the following three files:

- kaggle_iris_test.csv - The data that Kaggle will evaluate you on. It contains only input; you must provide answers. (contains x)
- kaggle_iris_train.csv - The data that you will use to train. (contains x and y)
- kaggle_iris_sample.csv - A sample submission for Kaggle. (contains x and y)

Important features of the Kaggle iris files (that differ from how we've previously seen files):

- The iris species is already index encoded.
- Your training data is in a separate file.
- You will load the test data to generate a submission file.

The following program generates a submission file for "Iris Kaggle". You can use it as a starting point for assignment 3.

In [2]:
```
import os
import pandas as pd
from sklearn.model_selection import train_test_split
import tensorflow as tf
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation
from tensorflow.keras.callbacks import EarlyStopping

df_train = pd.read_csv(
    "https://data.heatonresearch.com/data/t81-558/datasets/"+\
    "kaggle_iris_train.csv", na_values=['NA','?'])

# Encode feature vector
df_train.drop('id', axis=1, inplace=True)

num_classes = len(df_train.groupby('species').species.nunique())

print("Number of classes: {}".format(num_classes))

# Convert to numpy - Classification
x = df_train[['sepal_l', 'sepal_w', 'petal_l', 'petal_w']].values
dummies = pd.get_dummies(df_train['species']) # Classification
species = dummies.columns
y = dummies.values

# Split into train/test
x_train, x_test, y_train, y_test = train_test_split(
    x, y, test_size=0.25, random_state=45)

# Train, with early stopping
model = Sequential()
model.add(Dense(50, input_dim=x.shape[1], activation='relu'))
model.add(Dense(25))
model.add(Dense(y.shape[1],activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam')
monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3,
                        patience=5, verbose=1, mode='auto',
                       restore_best_weights=True)

model.fit(x_train,y_train,validation_data=(x_test,y_test),
          callbacks=[monitor],verbose=0,epochs=1000)
```

```
Number of classes: 3
Restoring model weights from the end of the best epoch: 103.
Epoch 108: early stopping
```

```
Out[2]: <keras.callbacks.History at 0x7f05e7452710>
```

Now that we've trained the neural network, we can check its log loss.

```
In [3]:
from sklearn import metrics

# Calculate multi log loss error
pred = model.predict(x_test)
score = metrics.log_loss(y_test, pred)
print("Log loss score: {}".format(score))
```

```
Log loss score: 0.10988010508939623
```

Now we are ready to generate the Kaggle submission file. We will use the iris test data that does not contain a y target value. It is our job to predict this value and submit it to Kaggle.

In [4]:
```
# Generate Kaggle submit file

# Encode feature vector
df_test = pd.read_csv(
    "https://data.heatonresearch.com/data/t81-558/datasets/"+\
    "kaggle_iris_test.csv", na_values=['NA','?'])

# Convert to numpy - Classification
ids = df_test['id']
df_test.drop('id', axis=1, inplace=True)
x = df_test[['sepal_l', 'sepal_w', 'petal_l', 'petal_w']].values
y = dummies.values

# Generate predictions
pred = model.predict(x)
#pred

# Create submission data set

df_submit = pd.DataFrame(pred)
df_submit.insert(0,'id',ids)
df_submit.columns = ['id','species-0','species-1','species-2']

# Write submit file locally
df_submit.to_csv("iris_submit.csv", index=False)

print(df_submit[:5])
```

```
    id  species-0  species-1  species-2
0  100   0.022300   0.777859   0.199841
1  101   0.001309   0.273849   0.724842
2  102   0.001153   0.319349   0.679498
3  103   0.958006   0.041989   0.000005
4  104   0.976932   0.023066   0.000002
```

MPG as a Kaggle Competition (Regression)

If the Auto MPG data were used as a Kaggle, you would be given the following three files:

- kaggle_mpg_test.csv - The data that Kaggle will evaluate you on. Contains only input, you must provide answers. (contains x)
- kaggle_mpg_train.csv - The data that you will use to train. (contains x and y)
- kaggle_mpg_sample.csv - A sample submission for Kaggle. (contains x and y)

Important features of the Kaggle iris files (that differ from how we've previously seen files):

The following program generates a submission file for "MPG Kaggle".

In [5]:
```
# HIDE OUTPUT
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation
from sklearn.model_selection import train_test_split
from tensorflow.keras.callbacks import EarlyStopping
```

```
import pandas as pd
import io
import os
import requests
import numpy as np
from sklearn import metrics

save_path = "."

df = pd.read_csv(
    "https://data.heatonresearch.com/data/t81-558/datasets/"+\
    "kaggle_auto_train.csv",
    na_values=['NA', '?'])

cars = df['name']

# Handle missing value
df['horsepower'] = df['horsepower'].fillna(df['horsepower'].median())

# Pandas to Numpy
x = df[['cylinders', 'displacement', 'horsepower', 'weight',
       'acceleration', 'year', 'origin']].values
y = df['mpg'].values # regression

# Split into train/test
x_train, x_test, y_train, y_test = train_test_split(
    x, y, test_size=0.25, random_state=42)

# Build the neural network
model = Sequential()
model.add(Dense(25, input_dim=x.shape[1], activation='relu')) # Hidden
model.add(Dense(10, activation='relu')) # Hidden 2
model.add(Dense(1)) # Output
model.compile(loss='mean_squared_error', optimizer='adam')
monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3, patience=5,
                        verbose=1, mode='auto', restore_best_weights=Tr
model.fit(x_train,y_train,validation_data=(x_test,y_test),
          verbose=2,callbacks=[monitor],epochs=1000)

# Predict
pred = model.predict(x_test)
```

```
Epoch 1/1000
9/9 - 1s - loss: 1797.5945 - val_loss: 1272.4421 - 1s/epoch - 144ms/step
Epoch 2/1000
9/9 - 0s - loss: 574.7726 - val_loss: 734.3082 - 92ms/epoch - 10ms/step
Epoch 3/1000
9/9 - 0s - loss: 487.3118 - val_loss: 446.3558 - 76ms/epoch - 8ms/step
Epoch 4/1000
9/9 - 0s - loss: 326.7128 - val_loss: 321.7191 - 96ms/epoch - 11ms/step
Epoch 5/1000
9/9 - 0s - loss: 294.8217 - val_loss: 271.3473 - 70ms/epoch - 8ms/step
Epoch 6/1000
9/9 - 0s - loss: 259.8376 - val_loss: 239.6796 - 116ms/epoch - 13ms/step
Epoch 7/1000
9/9 - 0s - loss: 250.4708 - val_loss: 227.4295 - 73ms/epoch - 8ms/step
Epoch 8/1000
9/9 - 0s - loss: 227.1252 - val_loss: 198.4167 - 125ms/epoch - 14ms/step
Epoch 9/1000
```

```
9/9 - 0s - loss: 225.6681 - val_loss: 195.5055 - 95ms/epoch - 11ms/step
Epoch 10/1000
9/9 - 0s - loss: 209.1198 - val_loss: 184.1092 - 121ms/epoch - 13ms/step
Epoch 11/1000
9/9 - 0s - loss: 195.4801 - val_loss: 176.0311 - 108ms/epoch - 12ms/step
Epoch 12/1000
9/9 - 0s - loss: 198.6493 - val_loss: 168.1613 - 163ms/epoch - 18ms/step
Epoch 13/1000
9/9 - 0s - loss: 198.5606 - val_loss: 196.0306 - 114ms/epoch - 13ms/step
Epoch 14/1000
9/9 - 0s - loss: 184.3067 - val_loss: 179.8450 - 99ms/epoch - 11ms/step
Epoch 15/1000
9/9 - 0s - loss: 178.6627 - val_loss: 148.1014 - 80ms/epoch - 9ms/step
Epoch 16/1000
9/9 - 0s - loss: 154.0201 - val_loss: 129.9253 - 74ms/epoch - 8ms/step
Epoch 17/1000
9/9 - 0s - loss: 145.2373 - val_loss: 124.0609 - 79ms/epoch - 9ms/step
Epoch 18/1000
9/9 - 0s - loss: 140.0318 - val_loss: 116.7844 - 86ms/epoch - 10ms/step
Epoch 19/1000
9/9 - 0s - loss: 135.1688 - val_loss: 115.0745 - 136ms/epoch - 15ms/step
Epoch 20/1000
9/9 - 0s - loss: 132.8391 - val_loss: 106.9831 - 169ms/epoch - 19ms/step
Epoch 21/1000
9/9 - 0s - loss: 123.6673 - val_loss: 105.7211 - 95ms/epoch - 11ms/step
Epoch 22/1000
9/9 - 0s - loss: 123.7169 - val_loss: 99.6713 - 112ms/epoch - 12ms/step
Epoch 23/1000
9/9 - 0s - loss: 118.0815 - val_loss: 96.0683 - 150ms/epoch - 17ms/step
Epoch 24/1000
9/9 - 0s - loss: 114.6363 - val_loss: 99.1486 - 153ms/epoch - 17ms/step
Epoch 25/1000
9/9 - 0s - loss: 112.3965 - val_loss: 93.8642 - 180ms/epoch - 20ms/step
Epoch 26/1000
9/9 - 0s - loss: 111.2470 - val_loss: 88.3417 - 139ms/epoch - 15ms/step
Epoch 27/1000
9/9 - 0s - loss: 107.8639 - val_loss: 86.7927 - 135ms/epoch - 15ms/step
Epoch 28/1000
9/9 - 0s - loss: 103.0426 - val_loss: 89.0441 - 101ms/epoch - 11ms/step
Epoch 29/1000
9/9 - 0s - loss: 110.6277 - val_loss: 82.4294 - 159ms/epoch - 18ms/step
Epoch 30/1000
9/9 - 0s - loss: 100.3681 - val_loss: 90.8037 - 82ms/epoch - 9ms/step
Epoch 31/1000
9/9 - 0s - loss: 105.4711 - val_loss: 79.2106 - 76ms/epoch - 8ms/step
Epoch 32/1000
9/9 - 0s - loss: 98.7603 - val_loss: 79.9620 - 73ms/epoch - 8ms/step
Epoch 33/1000
9/9 - 0s - loss: 94.7678 - val_loss: 76.8616 - 78ms/epoch - 9ms/step
Epoch 34/1000
9/9 - 0s - loss: 93.8199 - val_loss: 77.0823 - 76ms/epoch - 8ms/step
Epoch 35/1000
9/9 - 0s - loss: 94.8746 - val_loss: 73.9967 - 62ms/epoch - 7ms/step
Epoch 36/1000
9/9 - 0s - loss: 95.3178 - val_loss: 73.0059 - 60ms/epoch - 7ms/step
Epoch 37/1000
9/9 - 0s - loss: 91.1315 - val_loss: 80.8389 - 57ms/epoch - 6ms/step
Epoch 38/1000
9/9 - 0s - loss: 96.4810 - val_loss: 77.8854 - 59ms/epoch - 7ms/step
Epoch 39/1000
```

```
9/9 - 0s - loss: 91.1039 - val_loss: 69.9539 - 40ms/epoch - 4ms/step
Epoch 40/1000
9/9 - 0s - loss: 86.9596 - val_loss: 69.3511 - 43ms/epoch - 5ms/step
Epoch 41/1000
9/9 - 0s - loss: 87.6142 - val_loss: 70.1390 - 57ms/epoch - 6ms/step
Epoch 42/1000
9/9 - 0s - loss: 88.0185 - val_loss: 73.6168 - 38ms/epoch - 4ms/step
Epoch 43/1000
9/9 - 0s - loss: 92.8655 - val_loss: 67.5213 - 38ms/epoch - 4ms/step
Epoch 44/1000
9/9 - 0s - loss: 88.5278 - val_loss: 69.9708 - 59ms/epoch - 7ms/step
Epoch 45/1000
9/9 - 0s - loss: 82.9339 - val_loss: 70.3786 - 39ms/epoch - 4ms/step
Epoch 46/1000
9/9 - 0s - loss: 81.7092 - val_loss: 63.3550 - 59ms/epoch - 7ms/step
Epoch 47/1000
9/9 - 0s - loss: 81.1514 - val_loss: 78.7681 - 59ms/epoch - 7ms/step
Epoch 48/1000
9/9 - 0s - loss: 99.3562 - val_loss: 62.8894 - 64ms/epoch - 7ms/step
Epoch 49/1000
9/9 - 0s - loss: 96.8292 - val_loss: 67.8047 - 55ms/epoch - 6ms/step
Epoch 50/1000
9/9 - 0s - loss: 88.7995 - val_loss: 67.5249 - 56ms/epoch - 6ms/step
Epoch 51/1000
9/9 - 0s - loss: 80.6064 - val_loss: 96.2975 - 58ms/epoch - 6ms/step
Epoch 52/1000
9/9 - 0s - loss: 95.2732 - val_loss: 62.4323 - 39ms/epoch - 4ms/step
Epoch 53/1000
9/9 - 0s - loss: 75.1992 - val_loss: 64.0174 - 39ms/epoch - 4ms/step
Epoch 54/1000
9/9 - 0s - loss: 75.5173 - val_loss: 57.8594 - 40ms/epoch - 4ms/step
Epoch 55/1000
9/9 - 0s - loss: 72.6369 - val_loss: 56.2216 - 47ms/epoch - 5ms/step
Epoch 56/1000
9/9 - 0s - loss: 72.8636 - val_loss: 55.3956 - 54ms/epoch - 6ms/step
Epoch 57/1000
9/9 - 0s - loss: 69.0251 - val_loss: 70.7940 - 56ms/epoch - 6ms/step
Epoch 58/1000
9/9 - 0s - loss: 75.8152 - val_loss: 63.7728 - 37ms/epoch - 4ms/step
Epoch 59/1000
9/9 - 0s - loss: 71.6866 - val_loss: 59.5908 - 41ms/epoch - 5ms/step
Epoch 60/1000
9/9 - 0s - loss: 69.3349 - val_loss: 52.7848 - 38ms/epoch - 4ms/step
Epoch 61/1000
9/9 - 0s - loss: 67.8410 - val_loss: 53.5977 - 54ms/epoch - 6ms/step
Epoch 62/1000
9/9 - 0s - loss: 68.4640 - val_loss: 53.6664 - 39ms/epoch - 4ms/step
Epoch 63/1000
9/9 - 0s - loss: 63.7229 - val_loss: 52.4224 - 44ms/epoch - 5ms/step
Epoch 64/1000
9/9 - 0s - loss: 69.8485 - val_loss: 59.1973 - 53ms/epoch - 6ms/step
Epoch 65/1000
9/9 - 0s - loss: 75.7193 - val_loss: 70.1342 - 37ms/epoch - 4ms/step
Epoch 66/1000
9/9 - 0s - loss: 87.7418 - val_loss: 55.3687 - 38ms/epoch - 4ms/step
Epoch 67/1000
9/9 - 0s - loss: 72.8599 - val_loss: 52.9028 - 44ms/epoch - 5ms/step
Epoch 68/1000
9/9 - 0s - loss: 69.9528 - val_loss: 49.9109 - 38ms/epoch - 4ms/step
Epoch 69/1000
9/9 - 0s - loss: 62.7782 - val_loss: 46.6361 - 39ms/epoch - 4ms/step
```

```
Epoch 70/1000
9/9 - 0s - loss: 58.4024 - val_loss: 50.8190 - 38ms/epoch - 4ms/step
Epoch 71/1000
9/9 - 0s - loss: 63.5687 - val_loss: 46.6161 - 44ms/epoch - 5ms/step
Epoch 72/1000
9/9 - 0s - loss: 65.9290 - val_loss: 47.1278 - 40ms/epoch - 4ms/step
Epoch 73/1000
9/9 - 0s - loss: 74.9235 - val_loss: 61.1282 - 42ms/epoch - 5ms/step
Epoch 74/1000
9/9 - 0s - loss: 63.6773 - val_loss: 45.0233 - 39ms/epoch - 4ms/step
Epoch 75/1000
9/9 - 0s - loss: 55.8287 - val_loss: 59.8986 - 41ms/epoch - 5ms/step
Epoch 76/1000
9/9 - 0s - loss: 58.9969 - val_loss: 52.0535 - 39ms/epoch - 4ms/step
Epoch 77/1000
9/9 - 0s - loss: 60.7104 - val_loss: 43.0530 - 46ms/epoch - 5ms/step
Epoch 78/1000
9/9 - 0s - loss: 59.7358 - val_loss: 45.3669 - 41ms/epoch - 5ms/step
Epoch 79/1000
9/9 - 0s - loss: 60.9792 - val_loss: 40.7967 - 41ms/epoch - 5ms/step
Epoch 80/1000
9/9 - 0s - loss: 58.0294 - val_loss: 49.0612 - 42ms/epoch - 5ms/step
Epoch 81/1000
9/9 - 0s - loss: 57.6733 - val_loss: 41.7604 - 44ms/epoch - 5ms/step
Epoch 82/1000
9/9 - 0s - loss: 50.3309 - val_loss: 39.1461 - 38ms/epoch - 4ms/step
Epoch 83/1000
9/9 - 0s - loss: 54.2316 - val_loss: 40.8561 - 36ms/epoch - 4ms/step
Epoch 84/1000
9/9 - 0s - loss: 66.4084 - val_loss: 38.1869 - 60ms/epoch - 7ms/step
Epoch 85/1000
9/9 - 0s - loss: 50.0778 - val_loss: 37.8852 - 56ms/epoch - 6ms/step
Epoch 86/1000
9/9 - 0s - loss: 47.0763 - val_loss: 37.3743 - 39ms/epoch - 4ms/step
Epoch 87/1000
9/9 - 0s - loss: 46.1752 - val_loss: 45.8444 - 45ms/epoch - 5ms/step
Epoch 88/1000
9/9 - 0s - loss: 49.4047 - val_loss: 37.3778 - 40ms/epoch - 4ms/step
Epoch 89/1000
9/9 - 0s - loss: 46.5478 - val_loss: 36.2859 - 38ms/epoch - 4ms/step
Epoch 90/1000
9/9 - 0s - loss: 44.7429 - val_loss: 47.2213 - 38ms/epoch - 4ms/step
Epoch 91/1000
9/9 - 0s - loss: 49.7726 - val_loss: 52.5501 - 42ms/epoch - 5ms/step
Epoch 92/1000
9/9 - 0s - loss: 53.5449 - val_loss: 62.3078 - 57ms/epoch - 6ms/step
Epoch 93/1000
9/9 - 0s - loss: 54.7558 - val_loss: 51.2010 - 43ms/epoch - 5ms/step
Epoch 94/1000
Restoring model weights from the end of the best epoch: 89.
9/9 - 0s - loss: 52.3631 - val_loss: 42.2640 - 69ms/epoch - 8ms/step
Epoch 94: early stopping
```

Now that we've trained the neural network, we can check its RMSE error.

In [6]:

```python
import numpy as np

# Measure RMSE error.  RMSE is common for regression.
score = np.sqrt(metrics.mean_squared_error(pred,y_test))
```

```
print("Final score (RMSE): {}".format(score))
```

```
Final score (RMSE): 6.023776405947501
```

Now we are ready to generate the Kaggle submission file. We will use the MPG test data that does not contain a y target value. It is our job to predict this value and submit it to Kaggle.

In [7]:

```
import pandas as pd

# Generate Kaggle submit file

# Encode feature vector
df_test = pd.read_csv(
    "https://data.heatonresearch.com/data/t81-558/datasets/"+\
    "kaggle_auto_test.csv", na_values=['NA','?'])

# Convert to numpy - regression
ids = df_test['id']
df_test.drop('id', axis=1, inplace=True)

# Handle missing value
df_test['horsepower'] = df_test['horsepower'].\
    fillna(df['horsepower'].median())

x = df_test[['cylinders', 'displacement', 'horsepower', 'weight',
       'acceleration', 'year', 'origin']].values

# Generate predictions
pred = model.predict(x)
#pred
```