Open in Colab

T81-558: Applications of Deep Neural Networks

Module 2: Python for Machine Learning

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

Module 2 Material

Main video lecture:

- **Part 2.1: Introduction to Pandas** [Video] [Notebook]
- Part 2.2: Categorical Values [Video] [Notebook]
- Part 2.3: Grouping, Sorting, and Shuffling in Python Pandas [Video] [Notebook]
- Part 2.4: Using Apply and Map in Pandas for Keras [Video] [Notebook]
- Part 2.5: Feature Engineering in Pandas for Deep Learning in Keras [Video] [Notebook]

Google CoLab Instructions

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

In [1]:

```
try:
    from google.colab import drive
    %tensorflow_version 2.x
    COLAB = True
    print("Note: using Google CoLab")
except:
    print("Note: not using Google CoLab")
    COLAB = False
```

```
Note: not using Google CoLab
```

Part 2.1: Introduction to Pandas

Pandas is an open-source library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language. It is based on the dataframe concept found in the R programming language. For this class, Pandas will be the primary means by which we manipulate data to be processed by neural networks.

The data frame is a crucial component of Pandas. We will use it to access the auto-mpg dataset. You can find this dataset on the UCI machine learning repository. For this class, we will use a version of the Auto MPG dataset, where I added column headers. You can find my version at https://data.heatonresearch.com/.

UCI took this dataset from the StatLib library, which Carnegie Mellon University maintains. The dataset was used in the 1983 American Statistical Association Exposition. It contains data for 398 cars, including mpg, cylinders, displacement, horsepower , weight, acceleration, model year, origin and the car's name.

The following code loads the MPG dataset into a data frame:

In [2]:

```
# Simple dataframe
import os
import pandas as pd

pd.set_option('display.max_columns', 7)
df = pd.read_csv(
    "https://data.heatonresearch.com/data/t81-558/auto-mpg.csv")
display(df[0:5])
```

	mpg	cylinders	displacement	...	year	origin	name
0	18.0	8	307.0	...	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	...	70	1	buick skylark 320
2	18.0	8	318.0	...	70	1	plymouth satellite
3	16.0	8	304.0	...	70	1	amc rebel sst
4	17.0	8	302.0	...	70	1	ford torino

5 rows × 9 columns

The **display** function provides a cleaner display than merely printing the data frame. Specifying the maximum rows and columns allows you to achieve greater control over the display.

In [3]:

```
pd.set_option('display.max_columns', 7)
pd.set_option('display.max_rows', 5)
display(df)
```

	mpg	cylinders	displacement	...	year	origin	name
0	18.0	8	307.0	...	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	...	70	1	buick skylark 320
...	...	...	...	...	...	...	...
396	28.0	4	120.0	...	82	1	ford ranger
397	31.0	4	119.0	...	82	1	chevy s-10

398 rows × 9 columns

It is possible to generate a second data frame to display statistical information about the first data frame.

In [4]:

```
# Strip non-numerics
df = df.select_dtypes(include=['int', 'float'])

headers = list(df.columns.values)
fields = []

for field in headers:
    fields.append({
        'name' : field,
        'mean': df[field].mean(),
        'var': df[field].var(),
        'sdev': df[field].std()
    })

for field in fields:
    print(field)
```

```
{'name': 'mpg', 'mean': 23.514572864321615, 'var': 61.089610774274405, 'sde
v': 7.815984312565782}
{'name': 'cylinders', 'mean': 5.454773869346734, 'var': 2.893415439919943,
'sdev': 1.7010042445332094}
{'name': 'displacement', 'mean': 193.42587939698493, 'var': 10872.1991522473
64, 'sdev': 104.26983817119581}
{'name': 'weight', 'mean': 2970.424623115578, 'var': 717140.9905256768, 'sde
v': 846.8417741973271}
{'name': 'acceleration', 'mean': 15.568090452261291, 'var': 7.60484823361138
1, 'sdev': 2.7576889298126757}
{'name': 'year', 'mean': 76.01005025125629, 'var': 13.672442818627143, 'sde
v': 3.697626646732623}
{'name': 'origin', 'mean': 1.5728643216080402, 'var': 0.6432920268850575, 's
dev': 0.8020548777266163}
```

This code outputs a list of dictionaries that hold this statistical information. This information looks similar to the JSON code seen in Module 1. If proper JSON is needed, the program should add these records to a list and call the Python JSON library's **dumps** command.

The Python program can convert this JSON-like information to a data frame for better display.

In [5]:

```
pd.set_option('display.max_columns', 0)
pd.set_option('display.max_rows', 0)
df2 = pd.DataFrame(fields)
display(df2)
```

	name	mean	var	sdev
0	mpg	23.514573	61.089611	7.815984
1	cylinders	5.454774	2.893415	1.701004
2	displacement	193.425879	10872.199152	104.269838
3	weight	2970.424623	717140.990526	846.841774
4	acceleration	15.568090	7.604848	2.757689
5	year	76.010050	13.672443	3.697627
6	origin	1.572864	0.643292	0.802055

Missing Values

Missing values are a reality of machine learning. Ideally, every row of data will have values for all columns. However, this is rarely the case. Most of the values are present in the MPG database. However, there are missing values in the horsepower column. A common practice is to replace missing values with the median value for that column. The program calculates the median. The following code replaces any NA values in horsepower with the median:

In [6]:

```
import os
import pandas as pd

df = pd.read_csv(
    "https://data.heatonresearch.com/data/t81-558/auto-mpg.csv",
    na_values=['NA', '?'])
print(f"horsepower has na? {pd.isnull(df['horsepower']).values.any()}")

print("Filling missing values...")
med = df['horsepower'].median()
df['horsepower'] = df['horsepower'].fillna(med)
# df = df.dropna() # you can also simply drop NA values

print(f"horsepower has na? {pd.isnull(df['horsepower']).values.any()}")
```

```
horsepower has na? True
Filling missing values...
horsepower has na? False
```

Dealing with Outliers

Outliers are values that are unusually high or low. We typically consider outliers to be a value that is several standard deviations from the mean. Sometimes outliers are simply errors; this is a result of observation error. Outliers can also be truly large or small values that may be difficult to address. The following function can remove such values.

In [7]:

```
# Remove all rows where the specified column is +/- sd standard deviations
def remove_outliers(df, name, sd):
    drop_rows = df.index[(np.abs(df[name] - df[name].mean())
                          >= (sd * df[name].std()))]
    df.drop(drop_rows, axis=0, inplace=True)
```

The code below will drop every row from the Auto MPG dataset where the horsepower is two standard deviations or more above or below the mean.

In [8]:

```
import pandas as pd
import os
import numpy as np
from sklearn import metrics
from scipy.stats import zscore

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

# create feature vector
med = df['horsepower'].median()
df['horsepower'] = df['horsepower'].fillna(med)

# Drop the name column
df.drop('name',1,inplace=True)

# Drop outliers in horsepower
print("Length before MPG outliers dropped: {}".format(len(df)))
remove_outliers(df,'mpg',2)
print("Length after MPG outliers dropped: {}".format(len(df)))

pd.set_option('display.max_columns', 0)
pd.set_option('display.max_rows', 5)
display(df)
```

```
Length before MPG outliers dropped: 398
Length after MPG outliers dropped: 388
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	or
0	18.0	8	307.0	130.0	3504	12.0	70	
1	15.0	8	350.0	165.0	3693	11.5	70	
...	...	...	...	...	...	...	...	
396	28.0	4	120.0	79.0	2625	18.6	82	
397	31.0	4	119.0	82.0	2720	19.4	82	

388 rows × 8 columns

Dropping Fields

You must drop fields that are of no value to the neural network. The following code removes the name column from the MPG dataset.

In [9]:

```
import os
import pandas as pd

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

print(f"Before drop: {list(df.columns)}")
df.drop('name', 1, inplace=True)
print(f"After drop: {list(df.columns)}")
```

```
Before drop: ['mpg', 'cylinders', 'displacement', 'horsepower', 'weight', 'a
cceleration', 'year', 'origin', 'name']
After drop: ['mpg', 'cylinders', 'displacement', 'horsepower', 'weight', 'ac
celeration', 'year', 'origin']
```

Concatenating Rows and Columns

Python can concatenate rows and columns together to form new data frames. The code below creates a new data frame from the **name** and **horsepower** columns from the Auto MPG dataset. The program does this by concatenating two columns together.

In [10]:

```
# Create a new dataframe from name and horsepower

import os
import pandas as pd

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

col_horsepower = df['horsepower']
```

```
col_name = df['name']
result = pd.concat([col_name, col_horsepower], axis=1)

pd.set_option('display.max_columns', 0)
pd.set_option('display.max_rows', 5)
display(result)
```

	name	horsepower
0	chevrolet chevelle malibu	130.0
1	buick skylark 320	165.0
...	...	...
396	ford ranger	79.0
397	chevy s-10	82.0

398 rows × 2 columns

The **concat** function can also concatenate rows together. This code concatenates the first two rows and the last two rows of the Auto MPG dataset.

In [11]:

```
# Create a new dataframe from first 2 rows and last 2 rows

import os
import pandas as pd

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

result = pd.concat([df[0:2],df[-2:]], axis=0)

pd.set_option('display.max_columns', 7)
pd.set_option('display.max_rows', 0)
display(result)
```

	mpg	cylinders	displacement	...	year	origin	name
0	18.0	8	307.0	...	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	...	70	1	buick skylark 320
396	28.0	4	120.0	...	82	1	ford ranger
397	31.0	4	119.0	...	82	1	chevy s-10

4 rows × 9 columns

Training and Validation

We must evaluate a machine learning model based on its ability to predict values that it has never seen before. Because of this, we often divide the training data into a validation and training set. The machine learning model will learn from the training data but ultimately be evaluated based on the validation data.

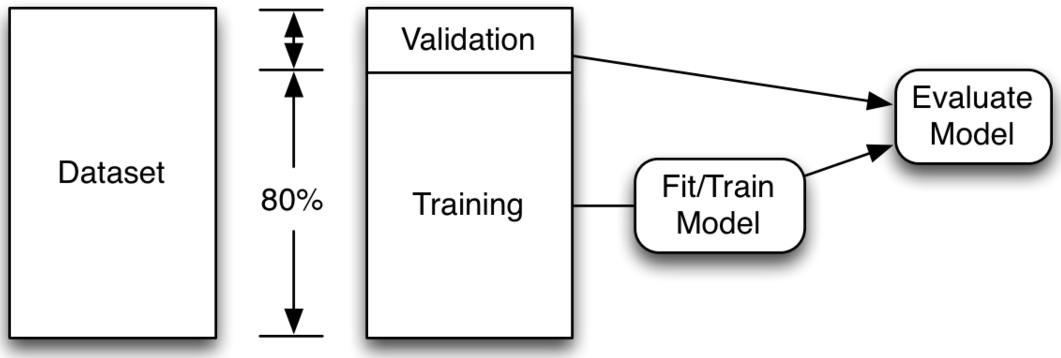
- **Training Data** - **In Sample Data** - The data that the neural network used to train.
- **Validation Data** - **Out of Sample Data** - The data that the machine learning model is evaluated upon after it is fit to the training data.

There are two effective means of dealing with training and validation data:

- **Training/Validation Split** - The program splits the data according to some ratio between a training and validation (hold-out) set. Typical rates are 80% training and 20% validation.
- **K-Fold Cross Validation** - The program splits the data into several folds and models. Because the program creates the same number of models as folds, the program can generate out-of-sample predictions for the entire dataset.

The code below splits the MPG data into a training and validation set. The training set uses 80% of the data, and the validation set uses 20%. Figure 2.TRN-VAL shows how we train a model on 80% of the data and then validated against the remaining 20%.

Figure 2.TRN-VAL: Training and Validation



In [12]:

```
import os
import pandas as pd
import numpy as np

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

# Usually a good idea to shuffle
df = df.reindex(np.random.permutation(df.index))
```

```
mask = np.random.rand(len(df)) < 0.8
trainDF = pd.DataFrame(df[mask])
validationDF = pd.DataFrame(df[~mask])

print(f"Training DF: {len(trainDF)}")
print(f"Validation DF: {len(validationDF)}")
```

```
Training DF: 333
Validation DF: 65
```

Converting a Dataframe to a Matrix

Neural networks do not directly operate on Python data frames. A neural network requires a numeric matrix. The program uses a data frame's **values** property to convert the data to a matrix.

```
In [13]: df.values
```

```
Out[13]: array([[20.2, 6, 232.0, ..., 79, 1, 'amc concord dl 6'],
               [14.0, 8, 304.0, ..., 74, 1, 'amc matador (sw)'],
               [14.0, 8, 351.0, ..., 71, 1, 'ford galaxie 500'],
               ...,
               [20.2, 6, 200.0, ..., 78, 1, 'ford fairmont (auto)'],
               [26.0, 4, 97.0, ..., 70, 2, 'volkswagen 1131 deluxe sedan'],
               [19.4, 6, 232.0, ..., 78, 1, 'amc concord']], dtype=object)
```

You might wish only to convert some of the columns, to leave out the name column, use the following code.

```
In [14]: df[['mpg', 'cylinders', 'displacement', 'horsepower', 'weight',
                'acceleration', 'year', 'origin']].values
```

```
Out[14]: array([[ 20.2,   6. , 232. , ...,  18.2,  79. ,   1. ],
               [ 14. ,   8. , 304. , ...,  15.5,  74. ,   1. ],
               [ 14. ,   8. , 351. , ...,  13.5,  71. ,   1. ],
               ...,
               [ 20.2,   6. , 200. , ...,  15.8,  78. ,   1. ],
               [ 26. ,   4. ,  97. , ...,  20.5,  70. ,   2. ],
               [ 19.4,   6. , 232. , ...,  17.2,  78. ,   1. ]])
```

Saving a Dataframe to CSV

Many of the assignments in this course will require that you save a data frame to submit to the instructor. The following code performs a shuffle and then saves a new copy.

```
In [15]: import os
         import pandas as pd
         import numpy as np
```

```
path = "."

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

filename_write = os.path.join(path, "auto-mpg-shuffle.csv")
df = df.reindex(np.random.permutation(df.index))
# Specify index = false to not write row numbers
df.to_csv(filename_write, index=False)
```

```
Done
```

Saving a Dataframe to Pickle

A variety of software programs can use text files stored as CSV. However, they take longer to generate and can sometimes lose small amounts of precision in the conversion. Generally, you will output to CSV because it is very compatible, even outside of Python. Another format is Pickle. The code below stores the Dataframe to Pickle. Pickle stores data in the exact binary representation used by Python. The benefit is that there is no loss of data going to CSV format. The disadvantage is that generally, only Python programs can read Pickle files.

In [16]:
```
import os
import pandas as pd
import numpy as np
import pickle

path = "."

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

filename_write = os.path.join(path, "auto-mpg-shuffle.pkl")
df = df.reindex(np.random.permutation(df.index))

with open(filename_write,"wb") as fp:
    pickle.dump(df, fp)
```

Loading the pickle file back into memory is accomplished by the following lines of code. Notice that the index numbers are still jumbled from the previous shuffle? Loading the CSV rebuilt (in the last step) did not preserve these values.

In [17]:
```
import os
import pandas as pd
import numpy as np
import pickle

path = "."

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

filename_read = os.path.join(path, "auto-mpg-shuffle.pkl")

with open(filename_write,"rb") as fp:
    df = pickle.load(fp)

pd.set_option('display.max_columns', 7)
pd.set_option('display.max_rows', 5)
display(df)
```

	mpg	cylinders	displacement	...	year	origin	name
387	38.0	6	262.0	...	82	1	oldsmobile cutlass ciera (diesel)
361	25.4	6	168.0	...	81	3	toyota cressida
...	...	...	...	...	...	...	...
358	31.6	4	120.0	...	81	3	mazda 626
237	30.5	4	98.0	...	77	1	chevrolet chevette

398 rows × 9 columns

Module 2 Assignment

You can find the first assignment here: assignment 2

```
In [ ]:
```