

CS3315 Project: MLP with Keras

```
In [1]: import numpy as np
import pandas as pd
from sklearn.linear_model import Perceptron
```

```
In [2]: # import data
filename = 'data/2020.06.19.csv'
df = pd.read_csv(filename)

# sample small subset
# df = df.sample(500000, random_state=78)
df.info()
df.head(2)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 765360 entries, 0 to 765359
Data columns (total 16 columns):
#   Column                Non-Null Count  Dtype
---  -
0   avg_ip_t              765360 non-null float64
1   bytes_in              765360 non-null int64
2   bytes_out            765360 non-null int64
3   dest_ip              765360 non-null int64
4   dest_port            740863 non-null float64
5   entropy              765360 non-null float64
6   num_pkts_out         765360 non-null int64
7   num_pkts_in          765360 non-null int64
8   proto                765360 non-null int64
9   src_ip              765360 non-null int64
10  src_port             740863 non-null float64
11  time_end             765360 non-null int64
12  time_start           765360 non-null int64
13  total_entropy        765360 non-null float64
14  label                765360 non-null object
15  duration             765360 non-null float64
dtypes: float64(6), int64(9), object(1)
memory usage: 93.4+ MB
```

```
Out[2]:
```

	avg_ip_t	bytes_in	bytes_out	dest_ip	dest_port	entropy	num_pkts_out	num_pkts_in	protc
0	7.5	342	3679	786	9200.0	5.436687	2	2	€
1	0.0	0	0	786	55972.0	0.000000	1	1	€

```
In [3]: # clean data
df.dropna(inplace=True)
df.isna().sum()

# need to clean for features that are 0 and don't make sense (bytes = 0?)
```

```
Out[3]: avg_ipt      0
        bytes_in    0
        bytes_out    0
        dest_ip      0
        dest_port    0
        entropy      0
        num_pkts_out  0
        num_pkts_in  0
        proto        0
        src_ip       0
        src_port     0
        time_end     0
        time_start   0
        total_entropy 0
        label        0
        duration     0
        dtype: int64
```

```
In [4]: print('label values:', df['label'].unique())

def ordinal_encoder(category):
    dict = {'benign':0, 'outlier':1, 'malicious':1}
    return dict[category]

print('benign', ordinal_encoder('benign'))
print('outlier', ordinal_encoder('outlier'))
print('malicious', ordinal_encoder('malicious'))
df['label'] = df['label'].apply(ordinal_encoder)

label values: ['benign' 'outlier' 'malicious']
benign 0
outlier 1
malicious 1
```

```
In [5]: features = ['avg_ipt',
                    'bytes_in',
                    'bytes_out',
                    'dest_ip',
                    'dest_port',
                    'entropy',
                    'num_pkts_in',
                    'num_pkts_out',
                    'proto',
                    'src_ip',
                    'src_port',
                    'time_end',
                    'time_start',
                    'total_entropy',
                    'duration']

X = df.loc[:, features]
y = df.loc[:, 'label']
```

```
In [6]: # Scale features
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
# try PolyScaler?

scaler = StandardScaler()
scaler.fit(X)
X = scaler.transform(X)
```

```
In [7]: # test/train split

from sklearn.model_selection import train_test_split

# 80/20 training/validation split
X_train, X_val, y_train, y_val = train_test_split(X, y, train_size=.8, test_s

y_train = y_train.to_numpy()
y_val = y_val.to_numpy()

# should print number of shape: (num features, num entries)
print('Training set: ', 'X: ', X_train.shape, 'y: ', y_train.shape, 'Validat

print(X_train[1])
print(y_train[1])

Training set: X: (592690, 15) y: (592690,) Validation set: X: (148173,
15) printy: (148173,)
[-0.05565213 -0.24851657 -0.4012423 -0.13191521 -0.1413975 -1.3449873
 -0.27015576 -0.2850236 -0.40847424 -0.50115682  0.54239382  0.33245924
  0.33237538 -0.20292837 -0.33436206]
0
```

```
In [8]: # import tensorflow and keras

import tensorflow as tf
from tensorflow import keras
print(tf.__version__)
print(keras.__version__)

import os
```

```
2022-12-01 21:55:24.369885: I tensorflow/core/platform/cpu_feature_guard.c
c:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network
Library (oneDNN) to use the following CPU instructions in performance-criti
cal operations: AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
2.12.0-dev20221112
2.12.0
```

```
In [9]: # keras sequential API model (pg 299)
```

```
model = keras.models.Sequential([
    keras.layers.InputLayer(input_shape=(X_train.shape[1])),
    keras.layers.Dense(300, activation="relu"),
    keras.layers.Dense(200, activation="relu"),
    keras.layers.Dense(2, activation="softmax")
])
print(X_train.shape[1])

model.summary()
```

15

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 300)	4800
dense_1 (Dense)	(None, 200)	60200
dense_2 (Dense)	(None, 2)	402

=====
Total params: 65,402
Trainable params: 65,402
Non-trainable params: 0
=====

```
In [10]: # TODO: adjust kernel initializer or bias initializer?
# https://keras.io/initializers/
```

```
In [11]: # compile the model
model.compile(loss="sparse_categorical_crossentropy",
              optimizer = "sgd",
              metrics="accuracy")
```

```
In [12]: # train the model
num_epochs=30
print(X_train.shape)
history = model.fit(X_train, y_train,
                    epochs=num_epochs,
                    validation_data=(X_val, y_val))

# loss is nan -> probably indicative of exploding gradients -- try again whe
```

(592690, 15)
Epoch 1/30
18522/18522 [=====] - 35s 2ms/step - loss: 0.0394
- accuracy: 0.9914 - val_loss: 0.0135 - val_accuracy: 0.9969
Epoch 2/30
18522/18522 [=====] - 33s 2ms/step - loss: 0.0101
- accuracy: 0.9975 - val_loss: 0.0103 - val_accuracy: 0.9972
Epoch 3/30
18522/18522 [=====] - 33s 2ms/step - loss: 0.0084
- accuracy: 0.9979 - val_loss: 0.0088 - val_accuracy: 0.9977
Epoch 4/30
18522/18522 [=====] - 33s 2ms/step - loss: 0.0076
- accuracy: 0.9980 - val_loss: 0.0088 - val_accuracy: 0.9977
Epoch 5/30
18522/18522 [=====] - 33s 2ms/step - loss: 0.0072
- accuracy: 0.9981 - val_loss: 0.0090 - val_accuracy: 0.9978
Epoch 6/30
18522/18522 [=====] - 33s 2ms/step - loss: 0.0069
- accuracy: 0.9982 - val_loss: 0.0075 - val_accuracy: 0.9979
Epoch 7/30
18522/18522 [=====] - 33s 2ms/step - loss: 0.0066
- accuracy: 0.9982 - val_loss: 0.0073 - val_accuracy: 0.9980
Epoch 8/30
18522/18522 [=====] - 33s 2ms/step - loss: 0.0064
- accuracy: 0.9982 - val_loss: 0.0084 - val_accuracy: 0.9977
Epoch 9/30
18522/18522 [=====] - 33s 2ms/step - loss: 0.0062
- accuracy: 0.9983 - val_loss: 0.0245 - val_accuracy: 0.9961
Epoch 10/30
18522/18522 [=====] - 33s 2ms/step - loss: 0.0062
- accuracy: 0.9983 - val_loss: 0.0080 - val_accuracy: 0.9975
Epoch 11/30
18522/18522 [=====] - 33s 2ms/step - loss: 0.0060
- accuracy: 0.9983 - val_loss: 0.0069 - val_accuracy: 0.9980
Epoch 12/30
18522/18522 [=====] - 33s 2ms/step - loss: 0.0058
- accuracy: 0.9983 - val_loss: 0.0076 - val_accuracy: 0.9981
Epoch 13/30
18522/18522 [=====] - 33s 2ms/step - loss: 0.0057
- accuracy: 0.9984 - val_loss: 0.0075 - val_accuracy: 0.9979
Epoch 14/30
18522/18522 [=====] - 33s 2ms/step - loss: 0.0056
- accuracy: 0.9984 - val_loss: 0.0064 - val_accuracy: 0.9982
Epoch 15/30
18522/18522 [=====] - 33s 2ms/step - loss: 0.0055
- accuracy: 0.9984 - val_loss: 0.0065 - val_accuracy: 0.9982
Epoch 16/30
18522/18522 [=====] - 33s 2ms/step - loss: 0.0054
- accuracy: 0.9985 - val_loss: 0.0062 - val_accuracy: 0.9982
Epoch 17/30
18522/18522 [=====] - 32s 2ms/step - loss: 0.0054
- accuracy: 0.9985 - val_loss: 0.0069 - val_accuracy: 0.9980
Epoch 18/30
18522/18522 [=====] - 32s 2ms/step - loss: 0.0052
- accuracy: 0.9985 - val_loss: 0.0061 - val_accuracy: 0.9984
Epoch 19/30

```

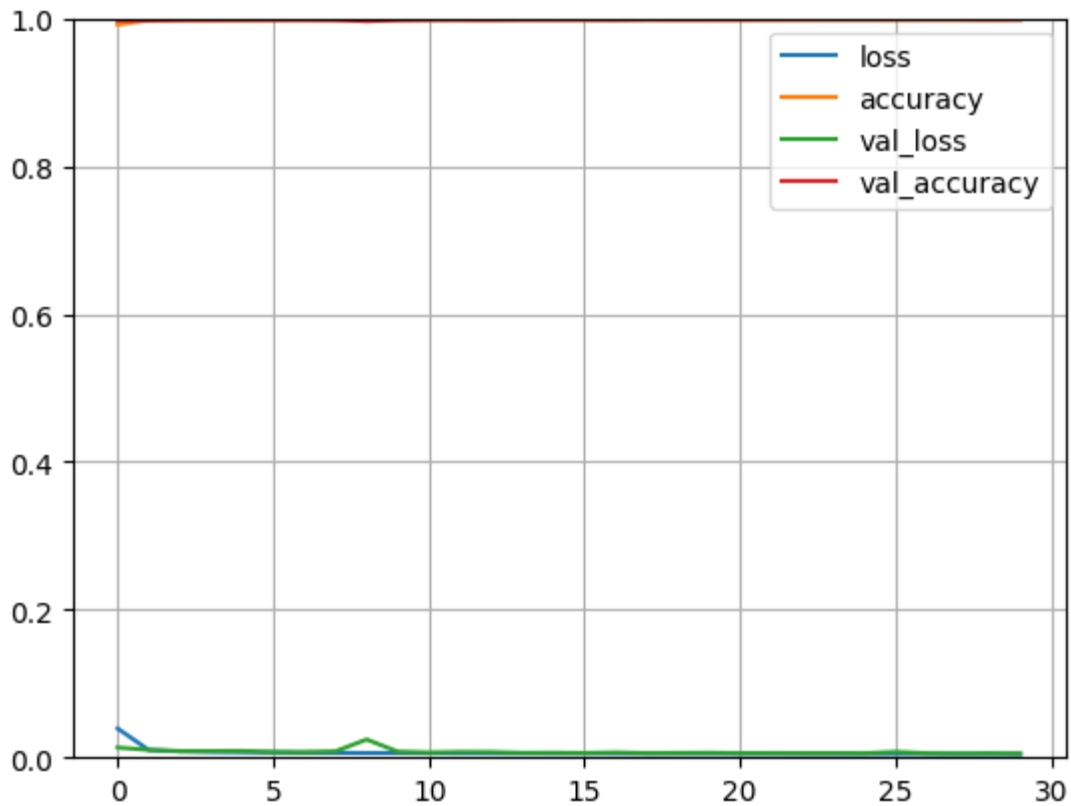
18522/18522 [=====] - 32s 2ms/step - loss: 0.0052
- accuracy: 0.9985 - val_loss: 0.0063 - val_accuracy: 0.9982
Epoch 20/30
18522/18522 [=====] - 32s 2ms/step - loss: 0.0052
- accuracy: 0.9985 - val_loss: 0.0064 - val_accuracy: 0.9983
Epoch 21/30
18522/18522 [=====] - 32s 2ms/step - loss: 0.0051
- accuracy: 0.9986 - val_loss: 0.0060 - val_accuracy: 0.9983
Epoch 22/30
18522/18522 [=====] - 32s 2ms/step - loss: 0.0050
- accuracy: 0.9986 - val_loss: 0.0061 - val_accuracy: 0.9983
Epoch 23/30
18522/18522 [=====] - 32s 2ms/step - loss: 0.0050
- accuracy: 0.9986 - val_loss: 0.0057 - val_accuracy: 0.9984
Epoch 24/30
18522/18522 [=====] - 32s 2ms/step - loss: 0.0048
- accuracy: 0.9986 - val_loss: 0.0059 - val_accuracy: 0.9984
Epoch 25/30
18522/18522 [=====] - 32s 2ms/step - loss: 0.0048
- accuracy: 0.9987 - val_loss: 0.0057 - val_accuracy: 0.9984
Epoch 26/30
18522/18522 [=====] - 32s 2ms/step - loss: 0.0048
- accuracy: 0.9986 - val_loss: 0.0077 - val_accuracy: 0.9984
Epoch 27/30
18522/18522 [=====] - 32s 2ms/step - loss: 0.0048
- accuracy: 0.9987 - val_loss: 0.0058 - val_accuracy: 0.9985
Epoch 28/30
18522/18522 [=====] - 32s 2ms/step - loss: 0.0047
- accuracy: 0.9987 - val_loss: 0.0054 - val_accuracy: 0.9985
Epoch 29/30
18522/18522 [=====] - 32s 2ms/step - loss: 0.0047
- accuracy: 0.9987 - val_loss: 0.0057 - val_accuracy: 0.9982
Epoch 30/30
18522/18522 [=====] - 32s 2ms/step - loss: 0.0046
- accuracy: 0.9987 - val_loss: 0.0055 - val_accuracy: 0.9983

```

```

In [13]: # plot loss vs. accuracy (HOML p. 305)
import matplotlib.pyplot as plt
pd.DataFrame(history.history).plot()
plt.grid(True)
plt.gca().set_ylim(0,1)
plt.show()

```



```
In [14]: # save model
#model.save("project_mlp_2_layers")
```

```
In [15]: # test predictions
X_new = X_val
test_pred = np.argmax(model.predict(X_new), axis=-1)

4631/4631 [=====] - 4s 775us/step
```

```
In [16]: from sklearn.metrics import *
print("Predicted labels:\t", test_pred)
print("Actual labels:\t\t", y_val)
print(classification_report(y_val, test_pred))
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	75511
1	1.00	1.00	1.00	72662
accuracy			1.00	148173
macro avg	1.00	1.00	1.00	148173
weighted avg	1.00	1.00	1.00	148173

Validate Model with Data from June 2022

```
In [17]: # import data
filename = 'data/2020.06.20.csv'
df2 = pd.read_csv(filename)

# sample small subset
#df2 = df2.sample(n=100000, random_state=78)
df2.info()
df2.head(2)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 770853 entries, 0 to 770852
Data columns (total 16 columns):
#   Column                Non-Null Count  Dtype
---  -
0   avg_ipt                770853 non-null float64
1   bytes_in               770853 non-null int64
2   bytes_out              770853 non-null int64
3   dest_ip                770853 non-null int64
4   dest_port              770853 non-null int64
5   entropy                770853 non-null float64
6   num_pkts_out           770853 non-null int64
7   num_pkts_in            770853 non-null int64
8   proto                  770853 non-null int64
9   src_ip                 770853 non-null int64
10  src_port                770853 non-null int64
11  time_end                770853 non-null int64
12  time_start              770853 non-null int64
13  total_entropy           770853 non-null float64
14  label                   770853 non-null object
15  duration                770853 non-null float64
dtypes: float64(4), int64(11), object(1)
memory usage: 94.1+ MB
```

```
Out[17]:
```

	avg_ipt	bytes_in	bytes_out	dest_ip	dest_port	entropy	num_pkts_out	num_pkts_in	pro
0	34.57143	34	29	786	5900	5.040459	7	10	
1	37.00000	34	29	786	5900	5.127916	7	10	

```
In [18]: # clean data
df2.dropna(inplace=True)
df2.isna().sum()
```



```
Out[18]: avg_ipt      0
         bytes_in    0
         bytes_out    0
         dest_ip      0
         dest_port    0
         entropy      0
         num_pkts_out  0
         num_pkts_in  0
         proto        0
         src_ip       0
         src_port     0
         time_end     0
         time_start   0
         total_entropy 0
         label        0
         duration     0
         dtype: int64
```

```
In [19]: print('label values:', df2['label'].unique())

def ordinal_encoder(category):
    dict = {'benign':0, 'outlier':1, 'malicious':1}
    return dict[category]

print('benign', ordinal_encoder('benign'))
print('outlier', ordinal_encoder('outlier'))
print('malicious', ordinal_encoder('malicious'))
df2['label'] = df2['label'].apply(ordinal_encoder)

label values: ['malicious' 'benign' 'outlier']
benign 0
outlier 1
malicious 1
```

```
In [20]: features = ['avg_ipt',
                     'bytes_in',
                     'bytes_out',
                     'dest_ip',
                     'dest_port',
                     'entropy',
                     'num_pkts_in',
                     'num_pkts_out',
                     'proto',
                     'src_ip',
                     'src_port',
                     'time_end',
                     'time_start',
                     'total_entropy',
                     'duration']

X_22 = df2.loc[:, features]
y_22 = df2.loc[:, 'label']
```

```
In [21]: # Scale features
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
# try PolyScaler?

scaler = StandardScaler()
scaler.fit(X_22)
X_22 = scaler.transform(X_22)

# change labels to numpy
y_22 = y_22.to_numpy()
```

```
In [22]: # test predictions
X_test_new = X_22
test_pred_22 = np.argmax(model.predict(X_test_new), axis=-1)

24090/24090 [=====] - 19s 775us/step
```

```
In [23]: print("Predicted labels:\t", test_pred_22)
print("Actual labels:\t\t", y_22)
print(classification_report(y_22, test_pred_22))
```

Predicted labels:	[1 1 1 ... 1 1 1]			
Actual labels:	[1 1 1 ... 1 1 1]			
	precision	recall	f1-score	support
0	1.00	0.99	1.00	366310
1	0.99	1.00	1.00	404543
accuracy			1.00	770853
macro avg	1.00	1.00	1.00	770853
weighted avg	1.00	1.00	1.00	770853

```
In [ ]:
```