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_ipt        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_ipt	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	6
1	0.0	0	0	786	55972.0	0.000000	1	1	6

In [3]:
```python
# 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]:
```python
print('label values:', df['label'].unique())

def ordinal_encoder(category):
    dict = {'benign':0, 'outlier':1, 'malicious':2}
    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 2
```

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, t

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, 'Va

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

```
Training set:  X:  (592690, 15) y:  (592690,) Validation set:  X:  (14
8173, 15) printy:  (148173,)
[-0.05565213 -0.24851657 -0.4012423  -0.13191521 -0.1413975  -1.344987
3
 -0.27015576 -0.2850236  -0.40847424 -0.50115682  0.54239382  0.332459
24
  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 19:22:25.620736: I tensorflow/core/platform/cpu_feature_gua
rd.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural
Network Library (oneDNN) to use the following CPU instructions in perf
ormance-critical operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the approp
riate 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(3, 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, 3)                 603

=================================================================
Total params: 65,603
Trainable params: 65,603
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")
```

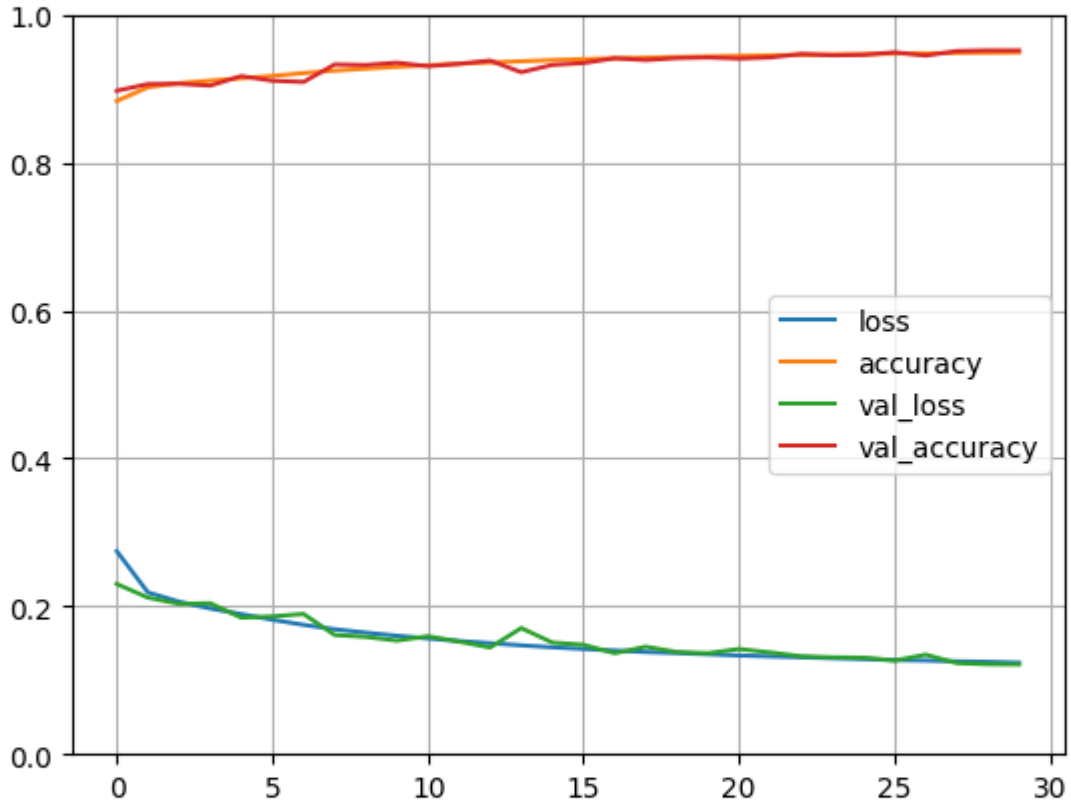
In [12]:
```
# train the model
num_epochs=30
print(X_train.shape)
history = model.fit(X_train, y_train,
                    epochs=num_epochs,
                    validation_data=(X_val, y_val))

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

```
(592690, 15)
Epoch 1/30
18522/18522 [==============================] - 35s 2ms/step - loss: 0.
2745 - accuracy: 0.8832 - val_loss: 0.2302 - val_accuracy: 0.8971
Epoch 2/30
18522/18522 [==============================] - 36s 2ms/step - loss: 0.
2188 - accuracy: 0.9018 - val_loss: 0.2118 - val_accuracy: 0.9061
Epoch 3/30
18522/18522 [==============================] - 33s 2ms/step - loss: 0.
2063 - accuracy: 0.9073 - val_loss: 0.2033 - val_accuracy: 0.9065
Epoch 4/30
18522/18522 [==============================] - 32s 2ms/step - loss: 0.
1971 - accuracy: 0.9108 - val_loss: 0.2041 - val_accuracy: 0.9040
Epoch 5/30
18522/18522 [==============================] - 33s 2ms/step - loss: 0.
1895 - accuracy: 0.9140 - val_loss: 0.1847 - val_accuracy: 0.9169
Epoch 6/30
18522/18522 [==============================] - 33s 2ms/step - loss: 0.
1818 - accuracy: 0.9171 - val_loss: 0.1864 - val_accuracy: 0.9107
Epoch 7/30
18522/18522 [==============================] - 33s 2ms/step - loss: 0.
1750 - accuracy: 0.9209 - val_loss: 0.1897 - val_accuracy: 0.9089
Epoch 8/30
18522/18522 [==============================] - 32s 2ms/step - loss: 0.
1690 - accuracy: 0.9243 - val_loss: 0.1613 - val_accuracy: 0.9326
Epoch 9/30
18522/18522 [==============================] - 33s 2ms/step - loss: 0.
1644 - accuracy: 0.9270 - val_loss: 0.1587 - val_accuracy: 0.9319
Epoch 10/30
18522/18522 [==============================] - 33s 2ms/step - loss: 0.
1602 - accuracy: 0.9294 - val_loss: 0.1537 - val_accuracy: 0.9350
Epoch 11/30
18522/18522 [==============================] - 32s 2ms/step - loss: 0.
1566 - accuracy: 0.9316 - val_loss: 0.1596 - val_accuracy: 0.9303
Epoch 12/30
18522/18522 [==============================] - 33s 2ms/step - loss: 0.
1531 - accuracy: 0.9336 - val_loss: 0.1523 - val_accuracy: 0.9334
Epoch 13/30
18522/18522 [==============================] - 32s 2ms/step - loss: 0.
1502 - accuracy: 0.9354 - val_loss: 0.1443 - val_accuracy: 0.9378
Epoch 14/30
18522/18522 [==============================] - 34s 2ms/step - loss: 0.
1473 - accuracy: 0.9369 - val_loss: 0.1706 - val_accuracy: 0.9221
Epoch 15/30
18522/18522 [==============================] - 34s 2ms/step - loss: 0.
1446 - accuracy: 0.9386 - val_loss: 0.1510 - val_accuracy: 0.9321
Epoch 16/30
```

```
18522/18522 [==============================] - 32s 2ms/step - loss: 0.
1424 - accuracy: 0.9398 - val_loss: 0.1479 - val_accuracy: 0.9346
Epoch 17/30
18522/18522 [==============================] - 32s 2ms/step - loss: 0.
1403 - accuracy: 0.9410 - val_loss: 0.1366 - val_accuracy: 0.9411
Epoch 18/30
18522/18522 [==============================] - 32s 2ms/step - loss: 0.
1382 - accuracy: 0.9419 - val_loss: 0.1454 - val_accuracy: 0.9386
Epoch 19/30
18522/18522 [==============================] - 32s 2ms/step - loss: 0.
1368 - accuracy: 0.9428 - val_loss: 0.1382 - val_accuracy: 0.9412
Epoch 20/30
18522/18522 [==============================] - 32s 2ms/step - loss: 0.
1352 - accuracy: 0.9435 - val_loss: 0.1366 - val_accuracy: 0.9421
Epoch 21/30
18522/18522 [==============================] - 32s 2ms/step - loss: 0.
1335 - accuracy: 0.9442 - val_loss: 0.1423 - val_accuracy: 0.9406
Epoch 22/30
18522/18522 [==============================] - 32s 2ms/step - loss: 0.
1323 - accuracy: 0.9450 - val_loss: 0.1376 - val_accuracy: 0.9421
Epoch 23/30
18522/18522 [==============================] - 32s 2ms/step - loss: 0.
1312 - accuracy: 0.9455 - val_loss: 0.1324 - val_accuracy: 0.9468
Epoch 24/30
18522/18522 [==============================] - 32s 2ms/step - loss: 0.
1298 - accuracy: 0.9461 - val_loss: 0.1311 - val_accuracy: 0.9450
Epoch 25/30
18522/18522 [==============================] - 32s 2ms/step - loss: 0.
1287 - accuracy: 0.9466 - val_loss: 0.1306 - val_accuracy: 0.9455
Epoch 26/30
18522/18522 [==============================] - 32s 2ms/step - loss: 0.
1274 - accuracy: 0.9474 - val_loss: 0.1266 - val_accuracy: 0.9490
Epoch 27/30
18522/18522 [==============================] - 32s 2ms/step - loss: 0.
1268 - accuracy: 0.9476 - val_loss: 0.1346 - val_accuracy: 0.9446
Epoch 28/30
18522/18522 [==============================] - 32s 2ms/step - loss: 0.
1256 - accuracy: 0.9482 - val_loss: 0.1232 - val_accuracy: 0.9508
Epoch 29/30
18522/18522 [==============================] - 32s 2ms/step - loss: 0.
1249 - accuracy: 0.9485 - val_loss: 0.1217 - val_accuracy: 0.9516
Epoch 30/30
18522/18522 [==============================] - 32s 2ms/step - loss: 0.
1238 - accuracy: 0.9489 - val_loss: 0.1215 - val_accuracy: 0.9516
```

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")
```

```
INFO:tensorflow:Assets written to: project_mlp_2_layers/assets
```

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

```
4631/4631 [==============================] - 4s 754us/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))
```

```
Predicted labels:        [0 1 0 ... 2 2 2]
Actual labels:           [0 1 0 ... 2 2 1]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     75511
           1       0.91      0.80      0.85     24572
           2       0.90      0.96      0.93     48090

    accuracy                           0.95    148173
   macro avg       0.94      0.92      0.93    148173
weighted avg       0.95      0.95      0.95    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':2}
    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 2
```

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 [==============================] - 18s 767us/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:        [2 2 2 ... 2 2 2]
Actual labels:           [2 2 2 ... 1 1 2]
              precision    recall  f1-score   support

           0       1.00      0.99      1.00    366310
           1       0.28      0.26      0.27     69389
           2       0.84      0.86      0.85    335154

    accuracy                           0.87    770853
   macro avg       0.71      0.70      0.71    770853
weighted avg       0.87      0.87      0.87    770853
```

In [ ]: