Training Processed Data in a Neural Network

[Link to Notebook]

```
import pandas as pd
from tensorflow.python.keras import models
from tensorflow.python.keras import layers
from keras.optimizers import Adam


%load_ext tensorboard
import tensorboard
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

Load Output Data

```
acre = pd.read_csv("https://nathanpersonalbucket.s3-us-west-2.amazonaws.com/acres.csv
acre
```

	region	YEAR_	Month	GIS_ACRES
0	1	1909	5	59.738968
1	1	1909	7	4.978550
2	1	1910	7	147285.675293
3	1	1910	8	1819.406002
4	1	1910	9	1633.051666
...	...	...	...	...
2128	3	2019	7	2604.718104
2129	3	2019	8	625.997220
2130	3	2019	9	3958.133733
2131	3	2019	10	28948.924878
2132	3	2019	11	3301.654997

2133 rows × 4 columns

```
count = pd.read_csv("https://nathanpersonalbucket.s3-us-west-2.amazonaws.com/count.cs
count
```

```
count
```

	region	YEAR_	Month	OBJECTID
0	1	1909	5	1
1	1	1909	7	1
2	1	1910	7	3
3	1	1910	8	6
4	1	1910	9	5
...	...	...	...	...
2128	3	2019	7	10
2129	3	2019	8	10
2130	3	2019	9	12
2131	3	2019	10	22
2132	3	2019	11	9

2133 rows × 4 columns

```
output = pd.merge(acre, count).rename({"OBJECTID":"Count"})
output = output.rename(columns={"YEAR_":"YEAR", "Month":"MONTH"})
```

Load Input Data

```
input = pd.read_csv("https://nathanpersonalbucket.s3-us-west-2.amazonaws.com/weather.
input
```

	region	YEAR	MONTH	PRCP	TAVG	TMAX	TMIN
0	1	1989	9	0.000000	32.000000	32.000000	32.000000
1	1	1989	10	0.200000	32.000000	32.000000	32.000000

Merge Data

```
def normalizeColumn(x):
  return (x - x.mean()) / x.std()
```

```
def getFireClass(FireSize):
  if FireSize < .25:
    return 1
  elif (FireSize >= .25 and FireSize <100):
    return 1
  elif FireSize < 300:
    return 2
  elif FireSize < 1000:
    return 3
  elif FireSize < 5000:
    return 3
  else:
    return 4
```

```
WDF_Acres = pd.merge(input, output, how ='left', left_on=('YEAR','MONTH','region'), r
Binary = WDF_Acres

WDF_Acres["AvgSize"] = WDF_Acres["GIS_ACRES"] / WDF_Acres["OBJECTID"]
WDF_Acres = WDF_Acres.fillna(0)
WDF_Acres['Severity'] = WDF_Acres['AvgSize'].apply(getFireClass)
WDF_Acres
```

	region	YEAR	MONTH	PRCP	TAVG	TMAX	TMIN	GIS_ACRES	OBJE
0	1	1989	9	0.000000	32.000000	32.000000	32.000000	5306.863037	
1	1	1989	10	0.200000	32.000000	32.000000	32.000000	41.270145	
2	1	1989	11	0.150000	32.000000	32.000000	32.000000	121.872509	
3	1	1989	12	0.003226	32.000000	32.000000	32.000000	20.844431	
4	1	1990	1	0.258065	32.000000	32.000000	32.000000	0.000000	

```
X", "TMIN", "GIS_ACRES", "OBJECTID",]] = WDF_Acres[["PRCP", "TAVG", "TMAX", "TMIN", "
```

```
WDF_Acres = WDF_Acres.drop(["AvgSize", "OBJECTID", "GIS_ACRES", "YEAR"], axis=1)
WDF_Acres
```

	region	MONTH	PRCP	TAVG	TMAX	TMIN	Severity
0	1	9	-0.677493	-1.458965	-1.951860	-0.895084	3
1	1	10	0.989406	-1.458965	-1.951860	-0.895084	1
2	1	11	0.572682	-1.458965	-1.951860	-0.895084	1
3	1	12	-0.650607	-1.458965	-1.951860	-0.895084	1
4	1	1	1.473345	-1.458965	-1.951860	-0.895084	1
...	...	...	...	...	...	...	...
792	3	9	-0.668463	1.526032	1.284450	1.834153	3
793	3	10	-0.677493	1.102869	1.102721	1.081488	3
794	3	11	-0.116303	0.617837	0.573681	0.677820	3
795	3	12	0.303827	0.276545	0.116749	0.395072	1
796	3	1	-0.677493	0.223971	0.082662	0.138066	1

797 rows × 7 columns

Vectorize Data into Columns

```
def vectorize(x):
  if x["Severity"] == 1:
    x["1"] = 1
  else:
    x["1"] = 0
```

```
  if x["Severity"] == 2:
    x["2"] = 1
  else:
    x["2"] = 0

  if x["Severity"] == 3:
    x["3"] = 1
  else:
    x["3"] = 0

  if x["Severity"] == 4:
    x["4"] = 1
  else:
    x["4"] = 0

  return x
```

```
WDF_Acres = WDF_Acres.apply(vectorize, axis=1)
WDF_Acres = WDF_Acres.drop("Severity", axis=1)
```

Split Data

```
print(WDF_Acres)

trainX = WDF_Acres.iloc[:636].drop(["1", "2", "3", "4"], axis=1)
trainY = WDF_Acres.iloc[:636][["1", "2", "3", "4"]]

testX = WDF_Acres.iloc[636:].drop(["1", "2", "3", "4"], axis=1)
testY = WDF_Acres.iloc[636:][["1", "2", "3", "4"]]


trainX
```

```
     region  MONTH      PRCP      TAVG      TMAX      TMIN    1    2    3    4
0       1.0    9.0 -0.677493 -1.458965 -1.951860 -0.895084  0.0  0.0  1.0  0.0
1       1.0   10.0  0.989406 -1.458965 -1.951860 -0.895084  1.0  0.0  0.0  0.0
2       1.0   11.0  0.572682 -1.458965 -1.951860 -0.895084  1.0  0.0  0.0  0.0
3       1.0   12.0 -0.650607 -1.458965 -1.951860 -0.895084  1.0  0.0  0.0  0.0
4       1.0    1.0  1.473345 -1.458965 -1.951860 -0.895084  1.0  0.0  0.0  0.0
..      ...    ...       ...       ...       ...       ...  ...  ...  ...  ...
792     3.0    9.0 -0.668463  1.526032  1.284450  1.834153  0.0  0.0  1.0  0.0
793     3.0   10.0 -0.677493  1.102869  1.102721  1.081488  0.0  0.0  1.0  0.0
794     3.0   11.0 -0.116303  0.617837  0.573681  0.677820  0.0  0.0  1.0  0.0
795     3.0   12.0  0.303827  0.276545  0.116749  0.395072  1.0  0.0  0.0  0.0
796     3.0    1.0 -0.677493  0.223971  0.082662  0.138066  1.0  0.0  0.0  0.0

[797 rows x 10 columns]
```

	region	MONTH	PRCP	TAVG	TMAX	TMIN
0	1.0	9.0	-0.677493	-1.458965	-1.951860	-0.895084
1	1.0	10.0	0.989406	-1.458965	-1.951860	-0.895084
2	1.0	11.0	0.572682	-1.458965	-1.951860	-0.895084
3	1.0	12.0	-0.650607	-1.458965	-1.951860	-0.895084

Train Data

```
from tensorflow.python.keras import models
from tensorflow.python.keras import layers

from tensorflow import keras
from keras.optimizers import Adam

model = models.Sequential()
model.add(layers.Dense(7, activation="relu", input_shape=(6, )))
model.add(layers.Dense(25, activation='relu'))
model.add(layers.Dense(64, activation='sigmoid'))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(16, activation='sigmoid'))
model.add(layers.Dense(4, activation="softmax"))

model.compile(optimizer='Adam', loss='categorical_crossentropy',metrics=['accuracy'])
```

```
from datetime import datetime
logdir="logs/fit/" + datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = keras.callbacks.TensorBoard(log_dir=logdir)



model.fit(trainX, trainY, epochs=25, batch_size=35, callbacks=[tensorboard_callback])
```

```
Epoch 1/25
```

```
 1/19 [>.............................] - ETA: 0s - loss: 1.3812 - accuracy: 0.20
19/19 [==============================] - 0s 4ms/step - loss: 1.2455 - accuracy:
Epoch 2/25
19/19 [==============================] - 0s 1ms/step - loss: 1.1405 - accuracy:
Epoch 3/25
19/19 [==============================] - 0s 1ms/step - loss: 1.1270 - accuracy:
Epoch 4/25
19/19 [==============================] - 0s 2ms/step - loss: 1.1177 - accuracy:
Epoch 5/25
19/19 [==============================] - 0s 2ms/step - loss: 1.1046 - accuracy:
Epoch 6/25
19/19 [==============================] - 0s 1ms/step - loss: 1.0807 - accuracy:
Epoch 7/25
19/19 [==============================] - 0s 2ms/step - loss: 1.0408 - accuracy:
Epoch 8/25
19/19 [==============================] - 0s 2ms/step - loss: 0.9856 - accuracy:
Epoch 9/25
19/19 [==============================] - 0s 2ms/step - loss: 0.9442 - accuracy:
Epoch 10/25
19/19 [==============================] - 0s 2ms/step - loss: 0.9189 - accuracy:
Epoch 11/25
19/19 [==============================] - 0s 2ms/step - loss: 0.9063 - accuracy:
Epoch 12/25
19/19 [==============================] - 0s 1ms/step - loss: 0.8996 - accuracy:
Epoch 13/25
19/19 [==============================] - 0s 2ms/step - loss: 0.8934 - accuracy:
Epoch 14/25
19/19 [==============================] - 0s 1ms/step - loss: 0.8872 - accuracy:
Epoch 15/25
19/19 [==============================] - 0s 2ms/step - loss: 0.8823 - accuracy:
Epoch 16/25
19/19 [==============================] - 0s 1ms/step - loss: 0.8845 - accuracy:
Epoch 17/25
19/19 [==============================] - 0s 2ms/step - loss: 0.8782 - accuracy:
Epoch 18/25
19/19 [==============================] - 0s 2ms/step - loss: 0.8772 - accuracy:
Epoch 19/25
19/19 [==============================] - 0s 2ms/step - loss: 0.8767 - accuracy:
Epoch 20/25
19/19 [==============================] - 0s 2ms/step - loss: 0.8710 - accuracy:
Epoch 21/25
19/19 [==============================] - 0s 2ms/step - loss: 0.8785 - accuracy:
Epoch 22/25
19/19 [==============================] - 0s 2ms/step - loss: 0.8701 - accuracy:
Epoch 23/25
19/19 [==============================] - 0s 2ms/step - loss: 0.8645 - accuracy:
Epoch 24/25
19/19 [==============================] - 0s 2ms/step - loss: 0.8641 - accuracy:
Epoch 25/25
19/19 [==============================] - 0s 2ms/step - loss: 0.8636 - accuracy:
<tensorflow.python.keras.callbacks.History at 0x7f8a53458438>
```

```
results = model.evaluate(testX, testY)
results
```

```
6/6 [==============================] - 0s 2ms/step - loss: 1.1881 - accuracy: 0.
[1.1880820989608765, 0.5465838313102722]
```

```
%tensorboard --logdir logs
```

```
Reusing TensorBoard on port 6006 (pid 97), started 0:09:44 ago. (Use '!kill 97'
```

TensorBoard SCALARS GRAPHS INACTIVE

Show data download links

Ignore outliers in chart scalin

Tooltip sorting method: default

Smoothing 0.6

Horizontal Axis

STEP RELATIVE

WALL

Runs

Write a regex to filter runs

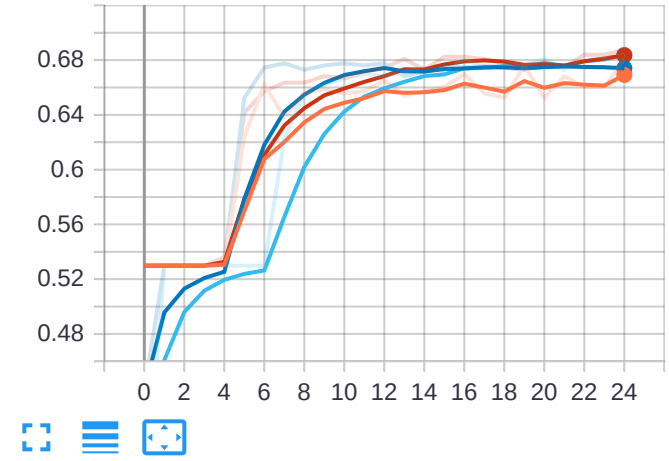
- fit/20201129-224244/train
- fit/20201129-225002/train
- fit/20201129-225138/train
- fit/20201129-225233/train

TOGGLE ALL RUNS

logs

Filter tags (regular expressions supported)

epoch_accuracy



epoch_loss