JHU IDS Module 11 Lab

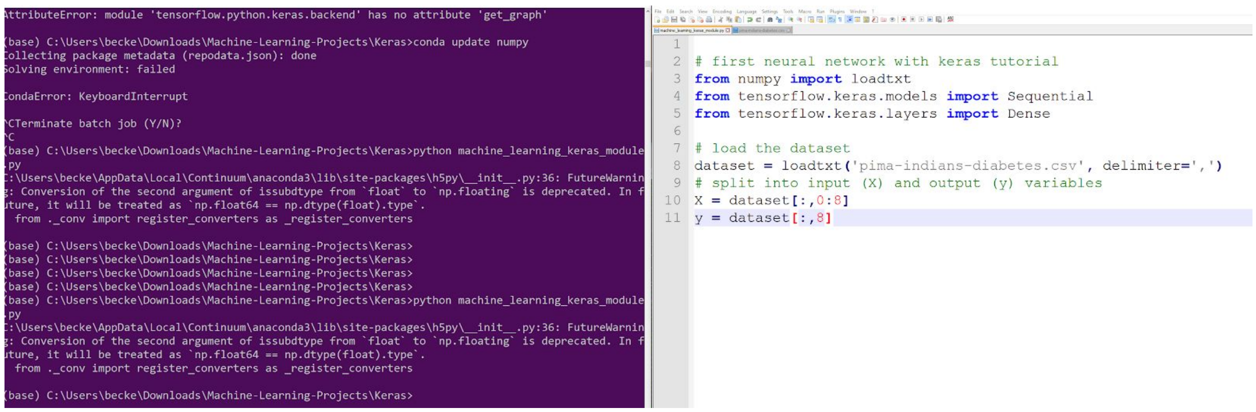
Audrey Long

08/08/2020

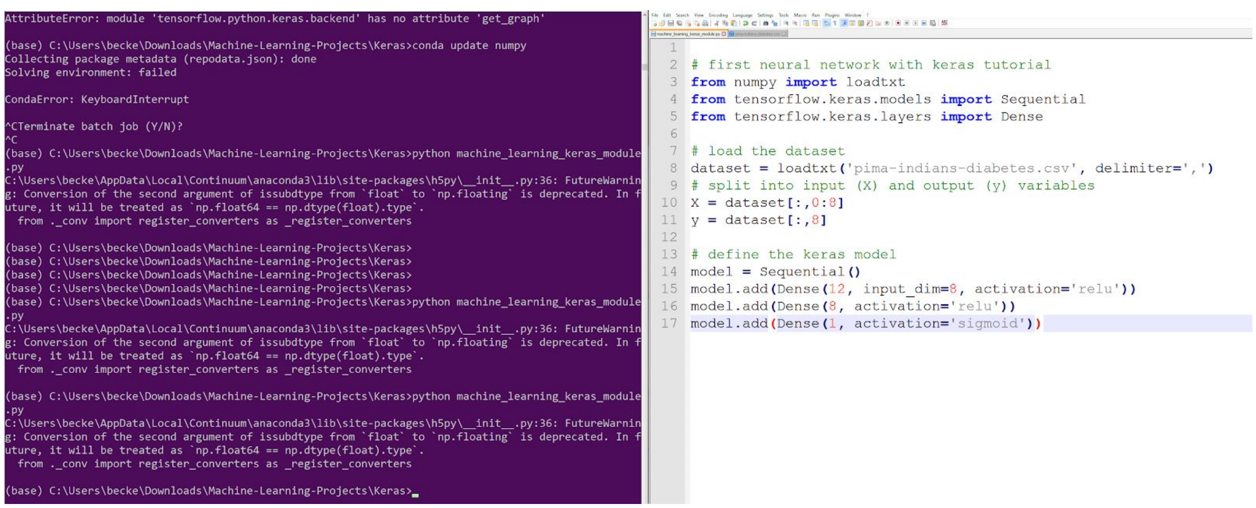
Introduction
Keras is a very popular framework for implementing neural nets in Python. You will use it to build a multilayer neural net, train it on real data, then use your model to make predictions about that data.

Building a Neural Net in Keras
There is a terrific tutorial for using Keras to build a net here:
https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/
Please complete each of the 7 steps: load data, define model, compile model, fit model, evaluate model, then make predictions!
Please provide your Python source code along with a screenshot showing the completion of each step.

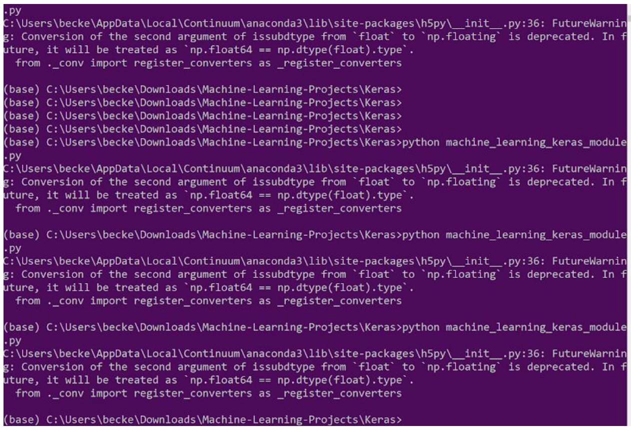
1. Load Data

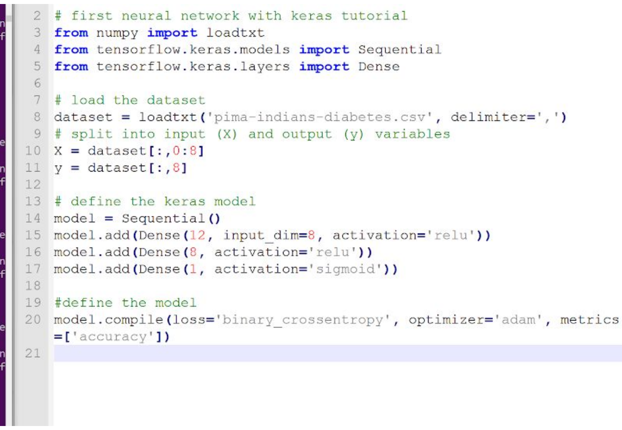


2. Define Model

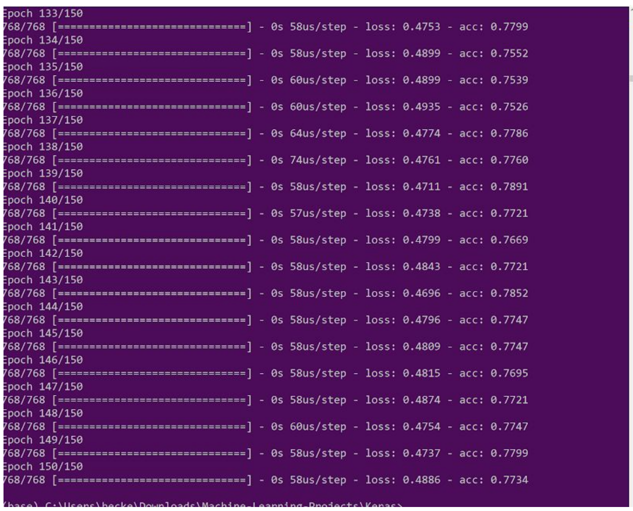


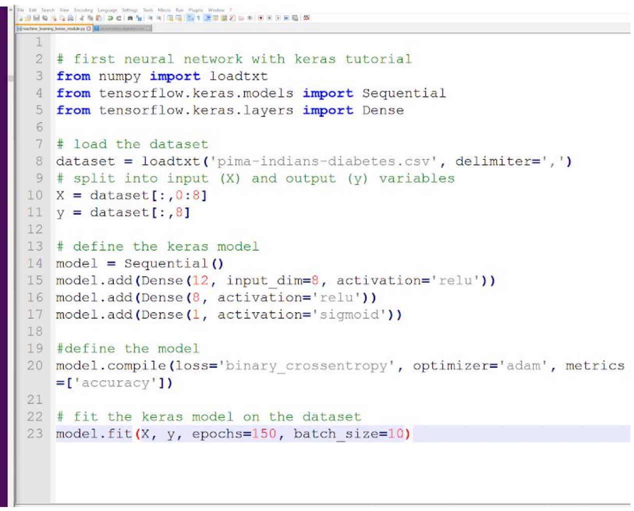
3. Compile Model


```
.py
C:\Users\becke\AppData\Local\Continuum\anaconda3\lib\site-packages\h5py\__init__.py:36: FutureWarnin
g: Conversion of the second argument of issubdtype from `float` to `np.floating` is deprecated. In f
uture, it will be treated as `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters

(base) C:\Users\becke\Downloads\Machine-Learning-Projects\Keras>
(base) C:\Users\becke\Downloads\Machine-Learning-Projects\Keras>
(base) C:\Users\becke\Downloads\Machine-Learning-Projects\Keras>
(base) C:\Users\becke\Downloads\Machine-Learning-Projects\Keras>
(base) C:\Users\becke\Downloads\Machine-Learning-Projects\Keras>python machine_learning_keras_module
.py
C:\Users\becke\AppData\Local\Continuum\anaconda3\lib\site-packages\h5py\__init__.py:36: FutureWarnin
g: Conversion of the second argument of issubdtype from `float` to `np.floating` is deprecated. In f
uture, it will be treated as `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters

(base) C:\Users\becke\Downloads\Machine-Learning-Projects\Keras>python machine_learning_keras_module
.py
C:\Users\becke\AppData\Local\Continuum\anaconda3\lib\site-packages\h5py\__init__.py:36: FutureWarnin
g: Conversion of the second argument of issubdtype from `float` to `np.floating` is deprecated. In f
uture, it will be treated as `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters

(base) C:\Users\becke\Downloads\Machine-Learning-Projects\Keras>python machine_learning_keras_module
.py
C:\Users\becke\AppData\Local\Continuum\anaconda3\lib\site-packages\h5py\__init__.py:36: FutureWarnin
g: Conversion of the second argument of issubdtype from `float` to `np.floating` is deprecated. In f
uture, it will be treated as `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters

(base) C:\Users\becke\Downloads\Machine-Learning-Projects\Keras>
```

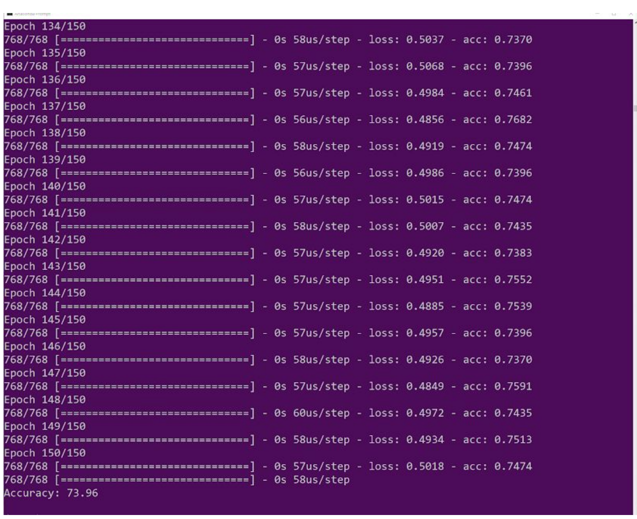

```python
# first neural network with keras tutorial
from numpy import loadtxt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# load the dataset
dataset = loadtxt('pima-indians-diabetes.csv', delimiter=',')
# split into input (X) and output (y) variables
X = dataset[:,0:8]
y = dataset[:,8]

# define the keras model
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

#define the model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics
=['accuracy'])
```

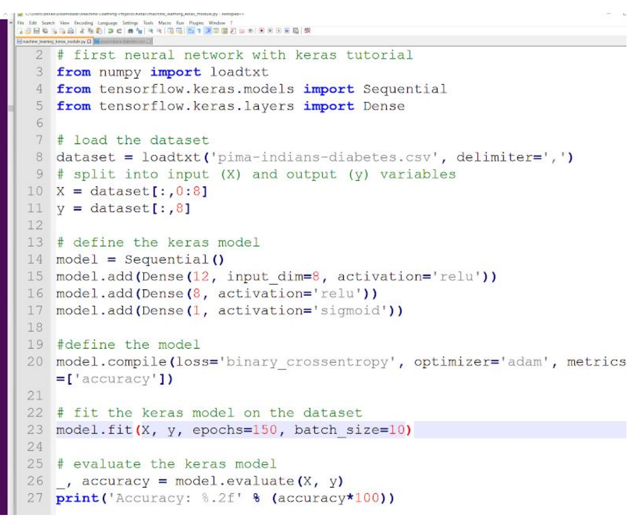
4. Fit Model


```
Epoch 133/150
768/768 [==============================] - 0s 58us/step - loss: 0.4753 - acc: 0.7799
Epoch 134/150
768/768 [==============================] - 0s 58us/step - loss: 0.4899 - acc: 0.7552
Epoch 135/150
768/768 [==============================] - 0s 60us/step - loss: 0.4899 - acc: 0.7539
Epoch 136/150
768/768 [==============================] - 0s 60us/step - loss: 0.4935 - acc: 0.7526
Epoch 137/150
768/768 [==============================] - 0s 64us/step - loss: 0.4774 - acc: 0.7786
Epoch 138/150
768/768 [==============================] - 0s 74us/step - loss: 0.4761 - acc: 0.7760
Epoch 139/150
768/768 [==============================] - 0s 58us/step - loss: 0.4711 - acc: 0.7891
Epoch 140/150
768/768 [==============================] - 0s 57us/step - loss: 0.4738 - acc: 0.7721
Epoch 141/150
768/768 [==============================] - 0s 58us/step - loss: 0.4799 - acc: 0.7669
Epoch 142/150
768/768 [==============================] - 0s 58us/step - loss: 0.4843 - acc: 0.7721
Epoch 143/150
768/768 [==============================] - 0s 58us/step - loss: 0.4696 - acc: 0.7852
Epoch 144/150
768/768 [==============================] - 0s 58us/step - loss: 0.4796 - acc: 0.7747
Epoch 145/150
768/768 [==============================] - 0s 58us/step - loss: 0.4809 - acc: 0.7747
Epoch 146/150
768/768 [==============================] - 0s 58us/step - loss: 0.4815 - acc: 0.7695
Epoch 147/150
768/768 [==============================] - 0s 58us/step - loss: 0.4874 - acc: 0.7721
Epoch 148/150
768/768 [==============================] - 0s 60us/step - loss: 0.4754 - acc: 0.7747
Epoch 149/150
768/768 [==============================] - 0s 58us/step - loss: 0.4737 - acc: 0.7799
Epoch 150/150
768/768 [==============================] - 0s 58us/step - loss: 0.4886 - acc: 0.7734
```


```python
# first neural network with keras tutorial
from numpy import loadtxt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# load the dataset
dataset = loadtxt('pima-indians-diabetes.csv', delimiter=',')
# split into input (X) and output (y) variables
X = dataset[:,0:8]
y = dataset[:,8]

# define the keras model
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

#define the model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics
=['accuracy'])

# fit the keras model on the dataset
model.fit(X, y, epochs=150, batch_size=10)
```

5. Evaluate Model


```
Epoch 134/150
768/768 [==============================] - 0s 58us/step - loss: 0.5037 - acc: 0.7370
Epoch 135/150
768/768 [==============================] - 0s 57us/step - loss: 0.5068 - acc: 0.7396
Epoch 136/150
768/768 [==============================] - 0s 57us/step - loss: 0.4984 - acc: 0.7461
Epoch 137/150
768/768 [==============================] - 0s 56us/step - loss: 0.4856 - acc: 0.7682
Epoch 138/150
768/768 [==============================] - 0s 58us/step - loss: 0.4919 - acc: 0.7474
Epoch 139/150
768/768 [==============================] - 0s 56us/step - loss: 0.4986 - acc: 0.7396
Epoch 140/150
768/768 [==============================] - 0s 57us/step - loss: 0.5015 - acc: 0.7474
Epoch 141/150
768/768 [==============================] - 0s 58us/step - loss: 0.5007 - acc: 0.7435
Epoch 142/150
768/768 [==============================] - 0s 57us/step - loss: 0.4920 - acc: 0.7383
Epoch 143/150
768/768 [==============================] - 0s 57us/step - loss: 0.4951 - acc: 0.7552
Epoch 144/150
768/768 [==============================] - 0s 57us/step - loss: 0.4885 - acc: 0.7539
Epoch 145/150
768/768 [==============================] - 0s 57us/step - loss: 0.4957 - acc: 0.7396
Epoch 146/150
768/768 [==============================] - 0s 58us/step - loss: 0.4926 - acc: 0.7370
Epoch 147/150
768/768 [==============================] - 0s 57us/step - loss: 0.4849 - acc: 0.7591
Epoch 148/150
768/768 [==============================] - 0s 60us/step - loss: 0.4972 - acc: 0.7435
Epoch 149/150
768/768 [==============================] - 0s 58us/step - loss: 0.4934 - acc: 0.7513
Epoch 150/150
768/768 [==============================] - 0s 57us/step - loss: 0.5018 - acc: 0.7474
768/768 [==============================] - 0s 58us/step
Accuracy: 73.96
```


```python
# first neural network with keras tutorial
from numpy import loadtxt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# load the dataset
dataset = loadtxt('pima-indians-diabetes.csv', delimiter=',')
# split into input (X) and output (y) variables
X = dataset[:,0:8]
y = dataset[:,8]

# define the keras model
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

#define the model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics
=['accuracy'])

# fit the keras model on the dataset
model.fit(X, y, epochs=150, batch_size=10)

# evaluate the keras model
_, accuracy = model.evaluate(X, y)
print('Accuracy: %.2f' % (accuracy*100))
```

6. Make Predictions

Please also answer the following question:
In ~1 page, how would you use a neural network in an IDS context to learn about and identify potentially malicious traffic on your network? Compare this approach with at least one other learning technique and tell why an alternative method might be preferred over a Neural Network.

In today's age the intrusion detection systems are used to constantly monitor and detect malicious activity with the use of a solid ruleset to detect true positives. A widely known issue with intrusion detection systems lie in the sheer amount of false positives that appear in the IDS report, which is very problematic when trying to find the true network attacks and threats. Though these rulesets seem to be fruitful, some rulesets such as shellcode injection seem to be almost always missed by IDS including powerful tools such as Snort and Sguil. After doing some thorough research on the topic I really like the suggested solution of introducing a neural network into an existing IDS such as figure 1 below demonstrates.

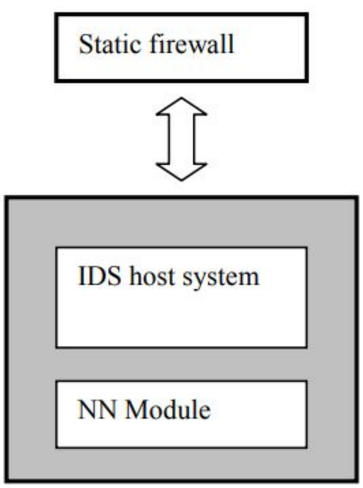


Fig. 1. System design.

The suggested system is able to adapt dynamically to the possible failure of a communication, system attack or more complex forms of infiltration (exploits, out of band communications, the analysis of covert communication channels). The whole proposed system is suitable to build on existing IDS. Of course the existing system must meet certain requirements. For example there must be possibility to program own extension modules, possibility to capture packets and save captured packets. This is an important point because of the possibility of repeating cycles of learning a neural network. As a host system is for us suitable IDS Snort. In general I like this solution because it's built on top of an existing IDS and users can easily integrate the new addition to the system.

The flexibility of neural networks is an advantage - in addition, the neural network is able to analyze incomplete data. Non-linearity of data flows in communication networks is another aspect influencing the selection. Since the neural network output is expressed as probability, neural network outputs can subsequently work as a certain prediction. Because neural networks can improve their abilities by learning, the output information could then be used to generate various actions in cases where a prediction is an alert of an attack attempt.

Though I like the idea of a neural network included in an IDS a lot of optimization must be made in order to implement the solution, along with more core/processors such as GPU parallelization would need to be implemented to ensure the processing speed isn't interrupted to the overall system status and speed. A learning technique instead that I think would work well would be just a big data solution to traffic incoming network traffic. The learning algorithm could be trained to better analyze and parse the data to generate better to follow graphs to show the number of packets, and data that was passed into clusters to better see what the traffic i doing on the network, then the analysts can update the rules or network practices to better accommodate the newly acquired information about the network activity.

References

https://medium.com/cuelogic-technologies/evaluation-of-machine-learning-algorithms-for-intrusion-detection-system-6854645f9211
https://www.sciencedirect.com/science/article/pii/S2405959518300493
https://papers.nips.cc/ cfgpaper/1459-intrusion-detection-with-neural-networks.pdf
https://www.sciencedirect.com/science/article/pii/S1877705814003579