Neural Nets for Exotic Particles search

Ananth Subramanian, Vitaly Briker, Jackson Au & Richard Farrow

November 18, 2019

1. Introduction

Neural networks are a subset of machine learning. The neural network framework aims to build models to interpret and draw conclusions, similar to that of a human mind. The human being has a neural network that is built up by neural circuits. Similarly, neural networks in statistical modeling embodies the same structure – intertwined, connective circuits for interpreting complex datasets.

This case study explores the usage of neural networks in the field of high-energy physics. Performance is of vital importance when it comes to measuring success for methods. With data science as an ever-growing field, this study will compare and contrast performance results from optimizers available at the time of this paper's publication (2014) to optimizers offered present-day (2019). Receiver operating characteristic curve will be the key metric for the performance comparison.

2. Methods

Higgs dataset is used for this analysis. There are in total 29 columns with 10M records. The first column is the label (1 for signal, 0 for background). The 28 columns have the mixture of both high-level and low-level features. The architecture as mentioned in the paper "Searching for Exotic Particles in High-Energy Physics with Deep Learning" is reproduced and the results are compared.

i) **Multi-Layer Perceptron**: Perceptron is a single neuron and multi-layer perceptron is a methodology of building a Neural Net with several hidden layers and neurons. A multi-layer perceptron neural net was used to analyze the Higgs Data with the architecture of 5 dense hidden layers and 300 neurons in each layer (as per the paper architecture). A sample multi-layer perceptron architecture is shown below (Figure 1).

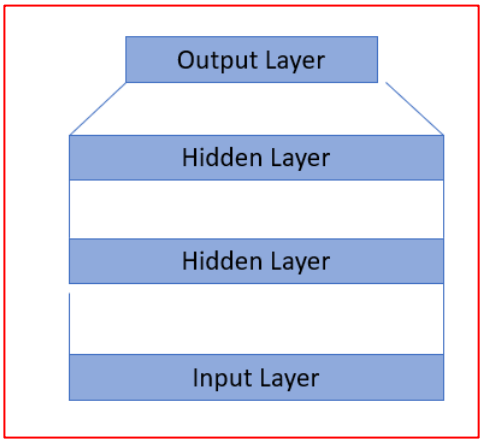


Figure 1 - sample multi-layer perceptron architecture

Within the hidden layers the following parameters were used:

a. ReLU (Rectified Linear Unit) activation

b. L2 Regularization with value 1x10 ** (-5) (as per the paper architecture)

The fully connected layer has “Sigmoid” activation. Stochastic Gradient Descent (SGD) optimizer is used with learning rate of 0.05 (as per the paper architecture).

The loss parameter used is “Binary Cross Entropy” and the model was fitted and run for 10 epochs with a batch size of 1000.

ii) **Multi-Layer Perceptron with Drop Out:** The architecture as explained above was used but with a Drop-Out value of 0.2 within the hidden layers.

iii) **New Optimizer (RAdam):** Rectified Adam (RAdam) is an optimizer that has been recently introduced and as a part of continuous improvement this has been used instead of SGD optimizer and the analysis for repeated

For all the methodologies above, ROCs were plotted and compared with the paper results.

The outputs are within the “Results” section and the “Conclusion” section contains suggestions for further improvements.

3. Results

- Multilayer perceptron with no drop-out:

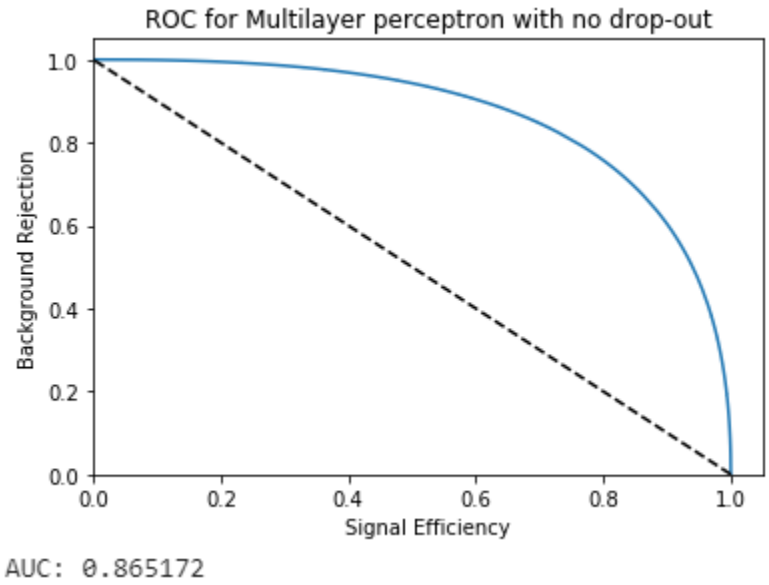
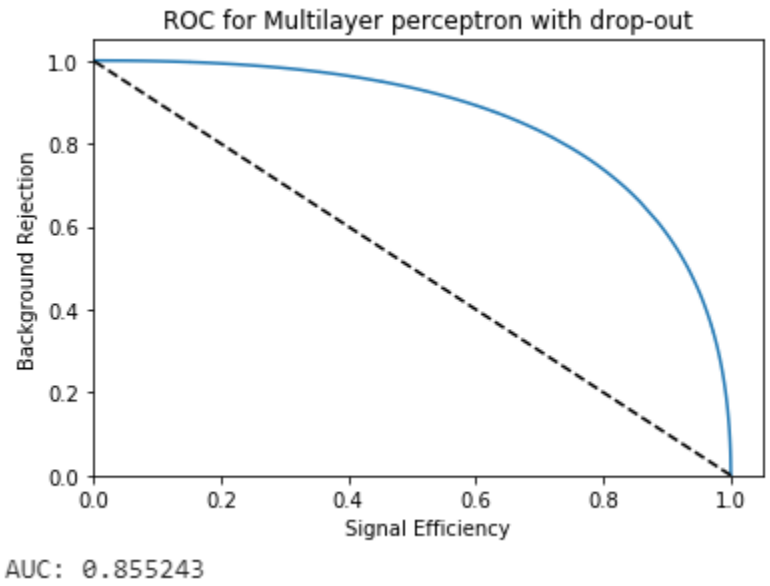


Fig. 2 - ROC for Multilayer perceptron with no drop-out

- Multi-layer perceptron with drop-out



- Fig. 3 - ROC for Multi-layer perceptron with drop-out

- RAdam optimizer and with drop-out:

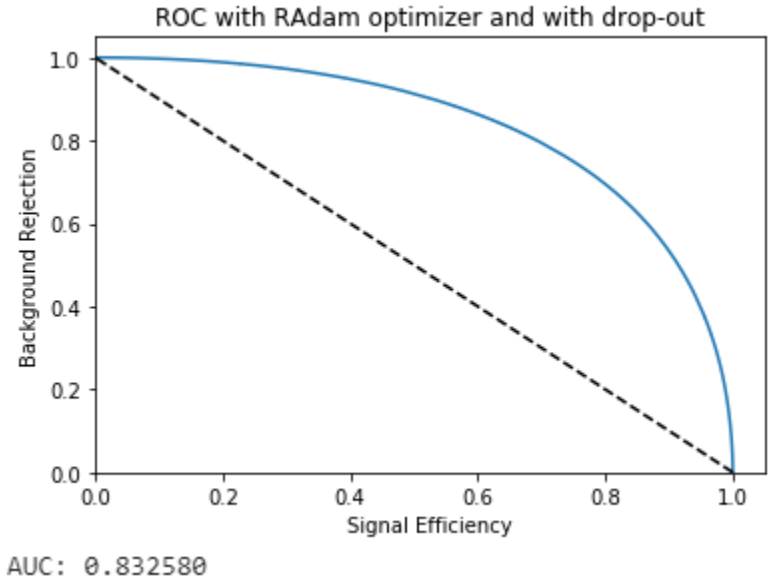


Fig. 4 - ROC with RAdam optimizer and with drop-out

Here is a comparison of the results obtained for the complete dataset with that of the paper.

ROC (Area Under Curve)	Case Study Result	Paper Result
Multilayer Perceptron (No drop-out)	0.863	0.876
Multilayer Perceptron (with drop-out)	0.855	0.873
Multilayer Perceptron (with RAdam opt)	0.833	NA

Table 1 - Model Comparison

4. Conclusion

Looking at the results, there is an opportunity for improvement.

Here are few suggestions:

1. When the paper was written, there was no RAdam optimizer. From the results, RAdam did not do well when compared to SGD. The understanding is that RAdam shows better results with multiple datasets. (References: www.pyimagesearch.com/2019/09/30/rectified-adam-optimizer-with-keras/)
2. Transfer Learning: This helps in using the pre-trained model and reuse the weight. The advantage is that all the 10M records need not be used. This also helps in implementing cross-validation thereby preventing over-fitting
3. Hyperparameters tuning: Other option to consider is tuning of hyperparameters within the neural nets. With GPU/TPU in place, more layers can be added, and the performance observed with tensor board. Tensor board helps to monitor training and validation accuracy and make better decisions on the number of epochs.

5. Appendix – Code

#Python :
#Multi-layer perceptron with no drop-out:

```
data.columns = ['label','lepton pT', 'lepton eta', 'lepton
phi', 'missing energy magnitude', 'missing energy phi', 'jet
1 pt', 'jet 1 eta', 'jet 1 phi','jet 1 b-tag', 'jet 2 pt',
'jet 2 eta','jet 2 phi', 'jet 2 b-tag', 'jet 3 pt', 'jet 3
eta', 'jet 3 phi', 'jet 3 b-tag', 'jet 4 pt', 'jet 4 eta',
'jet 4 phi', 'jet 4 b-tag','m_jj', 'm_jjj', 'm_lv', 'm_jlv',
'm_bb','m_wbb','m_wwbb']

X = data.iloc[:,1:28]

y = data.iloc[:,0]

from keras.models import Sequential
from keras.regularizers import l2
from keras.optimizers import SGD
from keras.layers import Dense
import numpy
# fix random seed for reproducibility
numpy.random.seed(7)


X = X
Y = y
# create model
model = Sequential()
model.add(Dense(300, input_dim=27,
activation='relu',kernel_regularizer=l2(0.00001)))
model.add(Dense(300,
activation='relu',kernel_regularizer=l2(0.00001)))
model.add(Dense(300,
activation='relu',kernel_regularizer=l2(0.00001)))
model.add(Dense(300,
activation='relu',kernel_regularizer=l2(0.00001)))
model.add(Dense(300,
activation='relu',kernel_regularizer=l2(0.00001)))
model.add(Dense(1, activation='sigmoid'))
sgd = SGD(lr=0.05, momentum=0.8, decay=0.0, nesterov=False)

# Compile model
model.compile(loss='binary_crossentropy', optimizer=sgd,
metrics=['accuracy'])

# Fit the model
model.fit(X, Y, epochs=10, batch_size=1000)

# evaluate the model
scores = model.evaluate(X, Y)
print("\n%s: %.2f%%" % (model.metrics_names[1],
scores[1]*100))
```

#Code for plotting ROC:

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
def generate_results(y_test, y_score):
    fpr, tpr, _ = roc_curve(y_test, y_score)
    roc_auc = auc(fpr, tpr)
    plt.figure()
    plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' %
roc_auc)
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.05])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic curve')
    plt.show()
    print('AUC: %f' % roc_auc)
```

#Multi-layer perceptron with drop-out:

```
from keras.models import Sequential
from keras.regularizers import l2
from keras.optimizers import SGD
from keras.layers import Dropout
from keras.layers import Dense
import numpy

# fix random seed for reproducibility
numpy.random.seed(7)


X = X
Y = y
# create model
model = Sequential()
model.add(Dense(300, input_dim=27,
activation='relu',kernel_regularizer=l2(0.00001)))
model.add(Dense(300,
activation='relu',kernel_regularizer=l2(0.00001)))
model.add(Dropout(0.2))
model.add(Dense(300,
activation='relu',kernel_regularizer=l2(0.00001)))
model.add(Dropout(0.2))
model.add(Dense(300,
activation='relu',kernel_regularizer=l2(0.00001)))
model.add(Dropout(0.2))
model.add(Dense(300,
activation='relu',kernel_regularizer=l2(0.00001)))
model.add(Dropout(0.2))
model.add(Dense(1, activation='sigmoid'))
sgd = SGD(lr=0.05, momentum=0.8, decay=0.0, nesterov=False)
# Compile model
model.compile(loss='binary_crossentropy', optimizer=sgd,
metrics=['accuracy'])
# Fit the model
model.fit(X, Y, epochs=10, batch_size=1000)
# evaluate the model
scores = model.evaluate(X, Y)
```

```
print("\n%s: %.2f%%" % (model.metrics_names[1],
scores[1]*100))
```

#With RAdam Optimizer:

```
opt = RAdam(total_steps=5000, warmup_proportion=0.1,
min_lr=1e-5)

X = X
Y = y
# create model
model = Sequential()
model.add(Dense(300, input_dim=27,
activation='relu',kernel_regularizer=l2(0.00001)))
model.add(Dense(300,
activation='relu',kernel_regularizer=l2(0.00001)))
model.add(Dropout(0.2))
model.add(Dense(300,
activation='relu',kernel_regularizer=l2(0.00001)))
model.add(Dropout(0.2))
model.add(Dense(300,
activation='relu',kernel_regularizer=l2(0.00001)))
model.add(Dropout(0.2))
model.add(Dense(300,
activation='relu',kernel_regularizer=l2(0.00001)))
model.add(Dropout(0.2))
model.add(Dense(1, activation='sigmoid'))
sgd = SGD(lr=0.05, momentum=0.8, decay=0.0, nesterov=False)
# Compile model
model.compile(loss='binary_crossentropy', optimizer=opt,
metrics=['accuracy'])
# Fit the model
model.fit(X, Y, epochs=10, batch_size=1000)
# evaluate the model
scores = model.evaluate(X, Y)
print("\n%s: %.2f%%" % (model.metrics_names[1],
scores[1]*100))
```