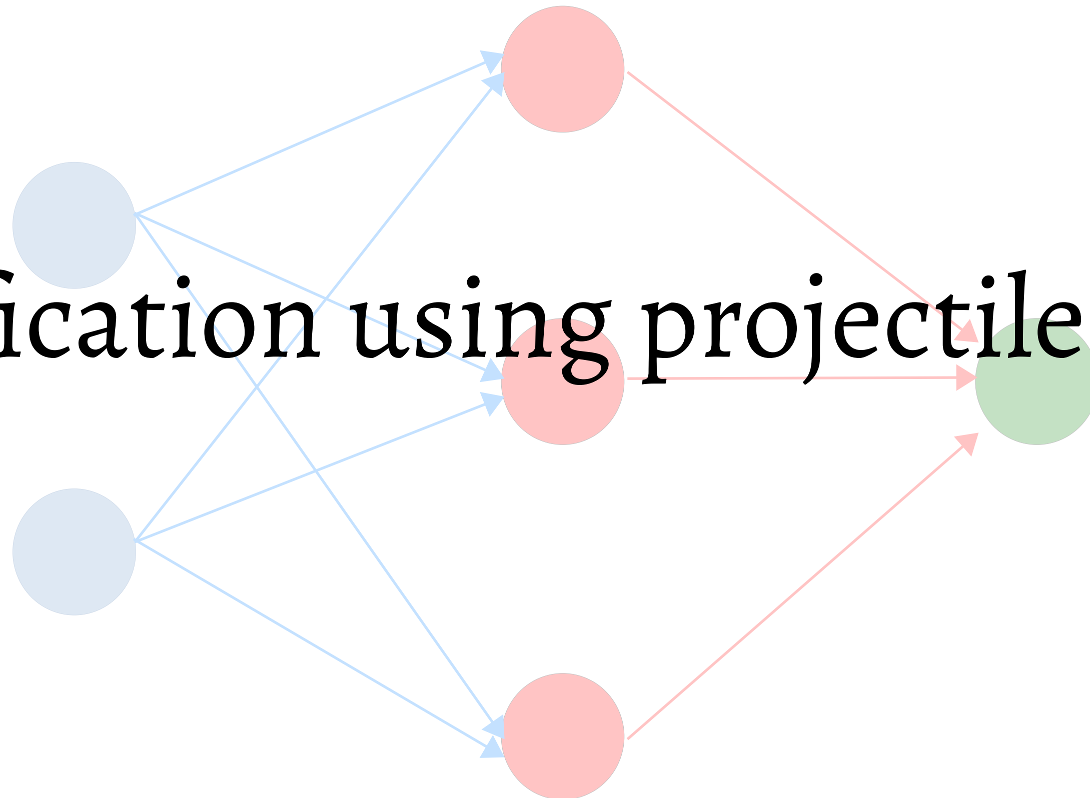
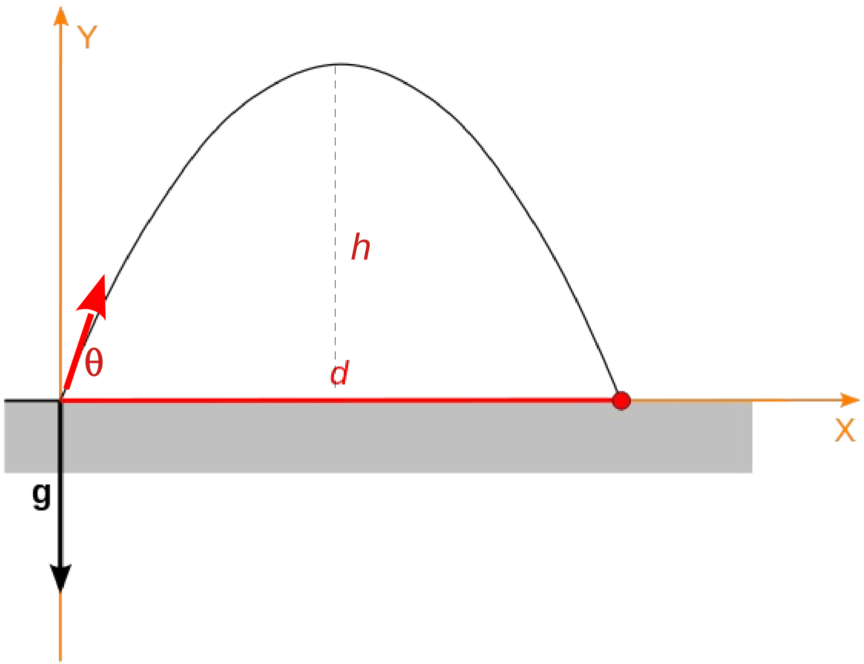
PH6232: Machine Learning for Physics applications



Classification using projectile motion

Feb 17th, 2023

Setup of task



Distance = $v^2 \sin(2\theta) / g$

Height = $v^2 \sin^2(\theta) / (2g)$

Time of Flight = $2 v \sin(\theta) / g$

Two inputs, v and angle (θ) determine the other three parameters

Our task

```
e_classification>head -5 projectile_input4_100k.txt
2.35565 21.0442 107.615 28.2299 41.922
3.88282 27.2298 24.9687 19.1289 45.7724
1.92644 14.0741 31.7454 17.1848 63.839
0.564022 1.20643 12.0678 10.8245 24.6576
4.01677 61.1877 192.268 38.9703 55.6157
```

projectile_input4_100k.txt : (get it from the git folder)
It has 5 columns: tof, height, dist, v, angleDeg
It has 100k entries (*i.e. 100,000 examples*)
"data" is crappier now, more smearing

Our task: Based on the tof, height, dist
can we predict if the initial velocity was between 25.0 and 35.0 m/s?

We want to classify examples into whether $25 < v < 35$ Or not.

Rewording, given the inputs, what is the probability that $25 < v < 35$?

Changes to network definition

Regression

```
model = Sequential()
model.add(Dense(4, activation='relu', kernel_initializer='he_normal', input_dim=n_features))
model.add(Dense(2, activation='relu', kernel_initializer='he_normal'))
model.add(Dense(1, activation='relu'))

model.compile(optimizer='adam', loss='MeanSquaredError')
```

Classification

```
model = Sequential()
model.add(Dense(4, activation='relu', kernel_initializer='he_normal', input_dim=n_features))
model.add(Dense(2, activation='relu', kernel_initializer='he_normal'))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam', loss='binary_crossentropy')
```

We use the `sigmoid` activation for the output neuron. This makes us get a continuous output between 0 and 1

We use the `binary_crossentropy` loss function.

Binary crossentropy

$$\mathrm{BCE} = -(1/N)\,\Sigma_i\big(\, y_i \log(p(y_i)) + (1-y_i)\log(1-p(y_i)) \,\big)$$

Let's understand this

For y=1 (i.e. signal) add $\log(p(y_i))$ to the loss

For y=0 (i.e. background) add $\log(1-p(y_i))$ to the loss

$p(y_i)$	0.5	0.9	0.999
$\log(p(y_i))$	-0.30	-0.05	-0.0004

BCE becomes smaller as the probability is estimated correctly

Binary crossentropy

$$BCE = -(1/N) \Sigma_i(y_i \log(p(y_i)) + (1-y_i) \log(1-p(y_i)))$$

Let's understand this

For y=1 (i.e. signal) add $\log(p(y_i))$ to the loss
For y=0 (i.e. background) add $\log(1-p(y_i))$ to the loss

$p(y_i)$	0.5	0.9	0.999
$\log(p(y_i))$	-0.30	-0.05	-0.0004

BCE becomes smaller as the probability is estimated correctly

Entropy is a measure of uncertainty for a given distribution, say q(y).
Cross-entropy is when we approximate true q(y) with some approximate p(y).

http://colah.github.io/posts/2015-09-Visual-Information/

Train/Test split

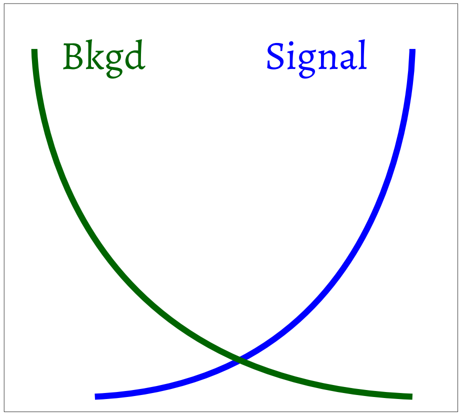
```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.5)
```

This time, we split our given dataset into two parts. We use one for the training, and the other for testing.

The training dataset is the one where weights are updated based on the calculated loss. The testing dataset is used to calculate loss etc. just for monitoring – we will get to see how the network is doing (in on-going way) on independent data.

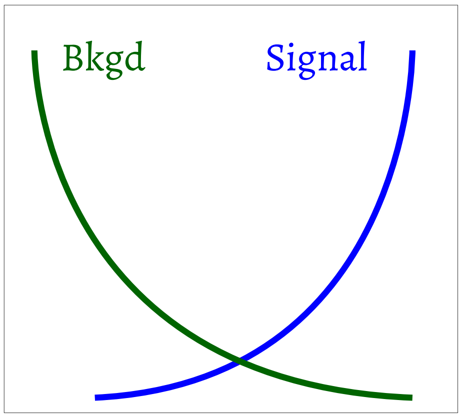
```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
history = model.fit(X_train,y_train,epochs=10,batch_size=100,
                    validation_data=(X_test,y_test),verbose=0)
print(history.history.keys())
```

NN output score

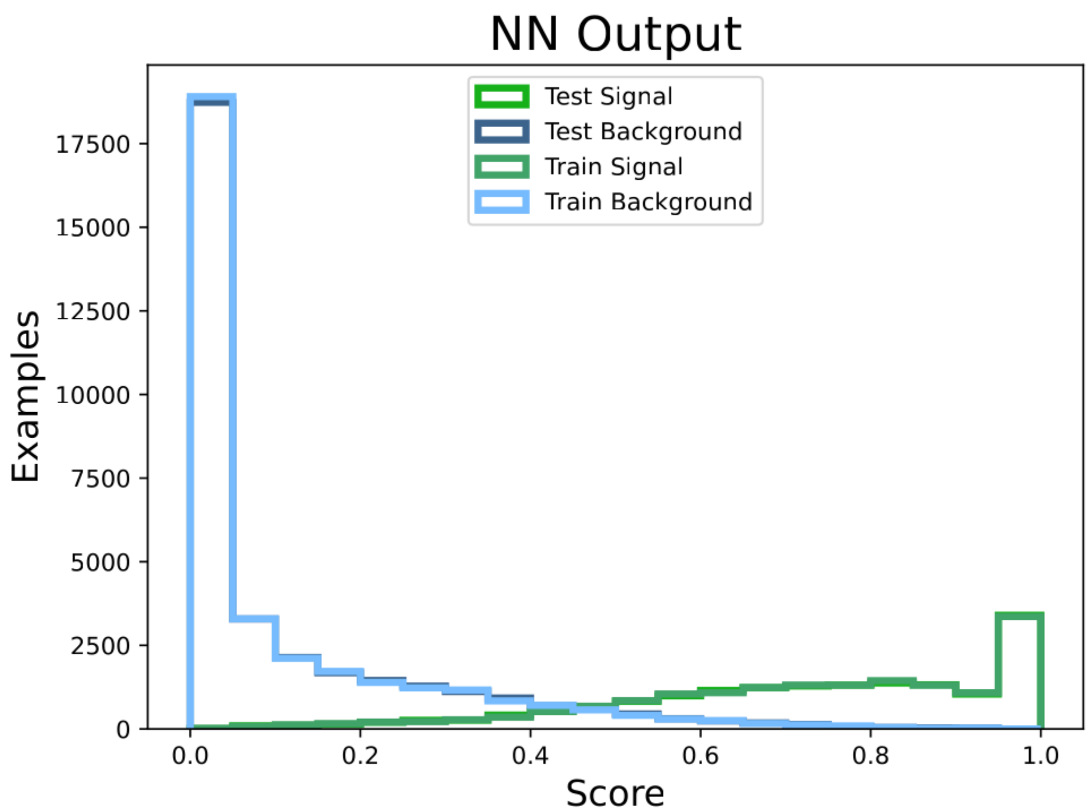


This is what we want as NN output.

NN output score

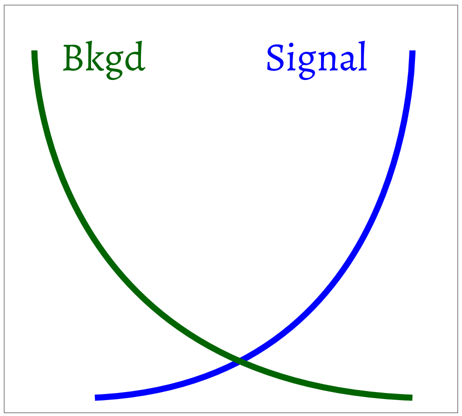


This is what we want as NN output.

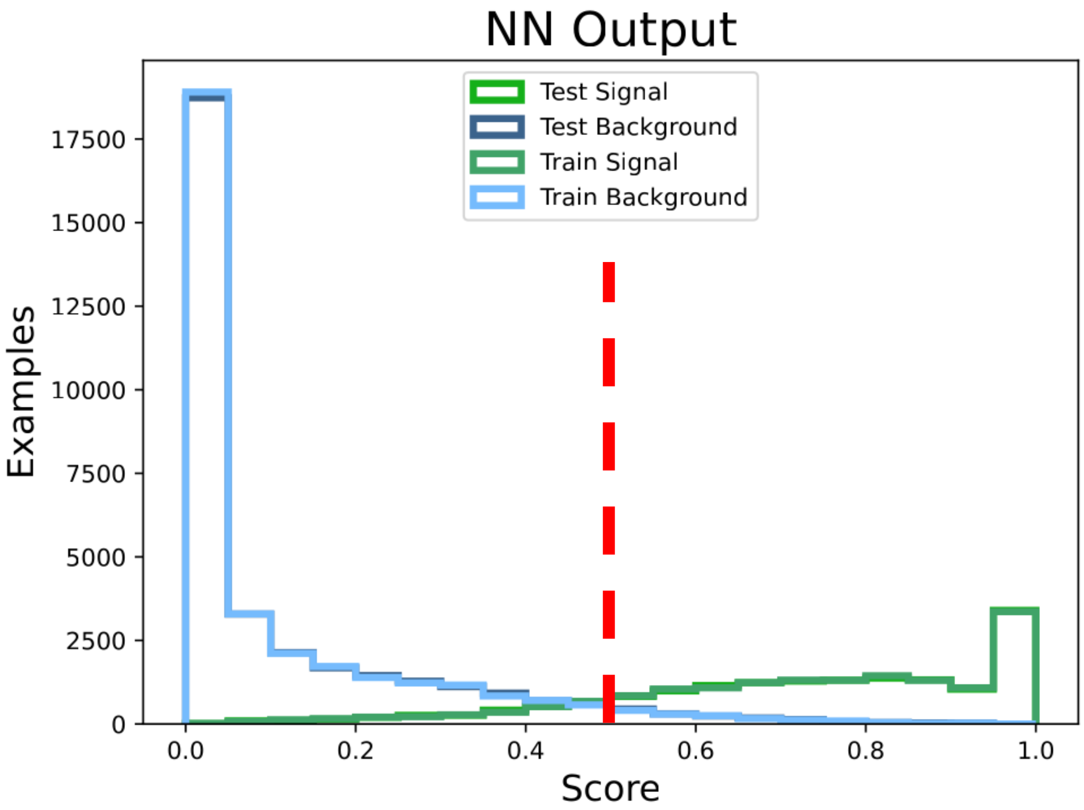


For our problem, we interprete this score as the probability that 25 < v < 35 . We want signal (i.e. true 25<v<35) to have high score, and we want background (i.e. v<25 or v>35) to have low score.

NN output score



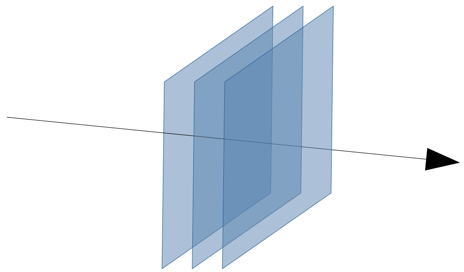
This is what we want as NN output.



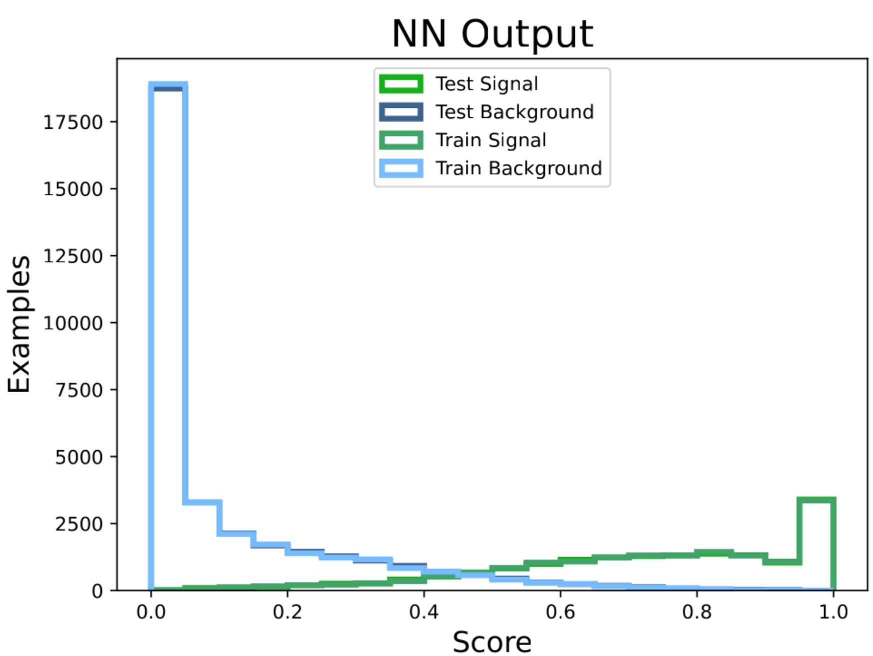
A score of 0.5 is used to define accuracy, i.e if score is above 0.5, its rounded to 1

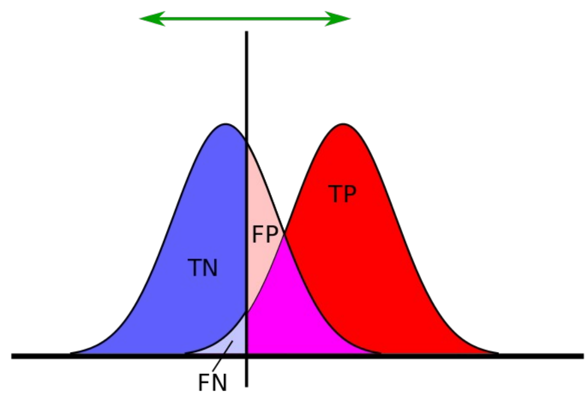
Accuracy = NumCorrect/NumTotal

ROC curve



Two hidden layers
with 8,4 neurons





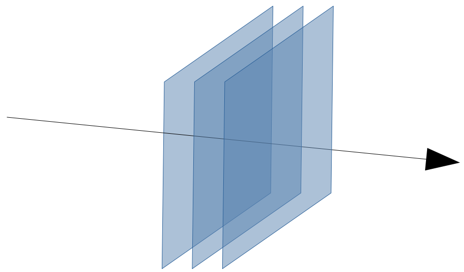
TP	FP
FN	TN

TPR = TP/(TP+FN)
FPR = FP/(FP+TN)

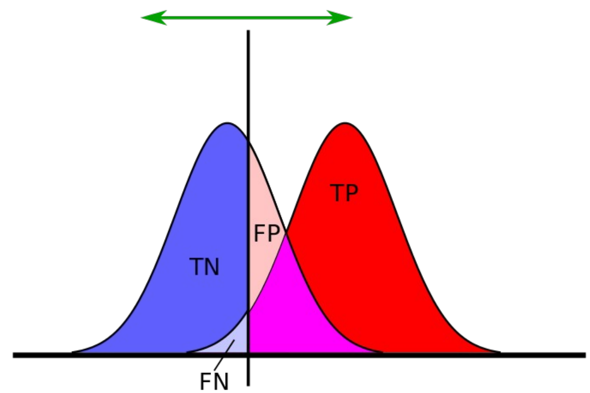
Plot true positive rate vs False positive rate

True positive rate = Efficiency of Signal to pass selection
False positive rate = Efficiency of Bkgd to pass selection

ROC curve



Two hidden layers
with 8,4 neurons

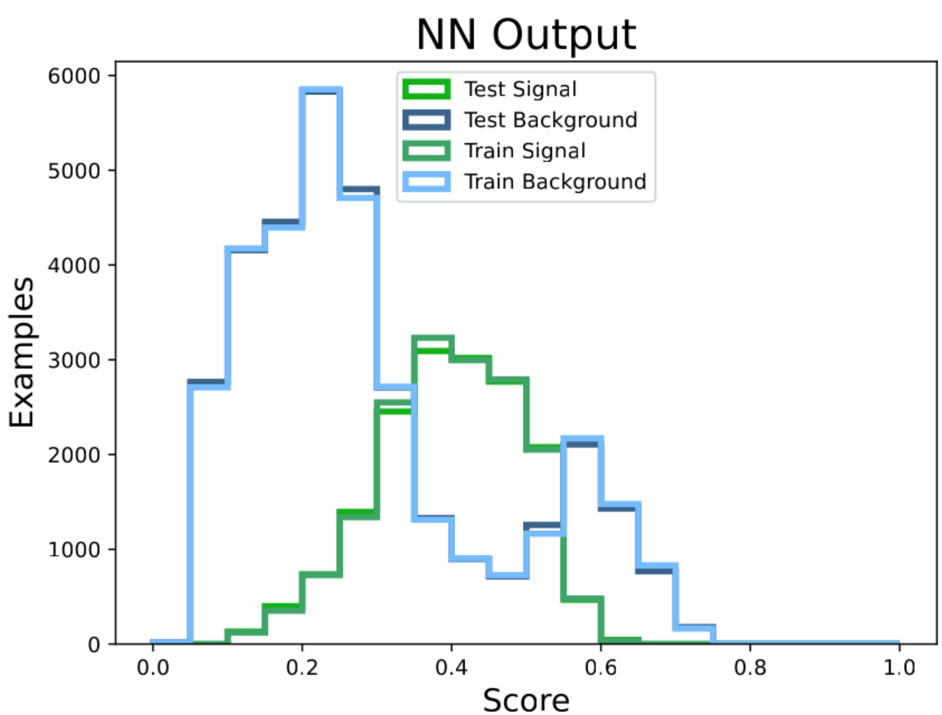


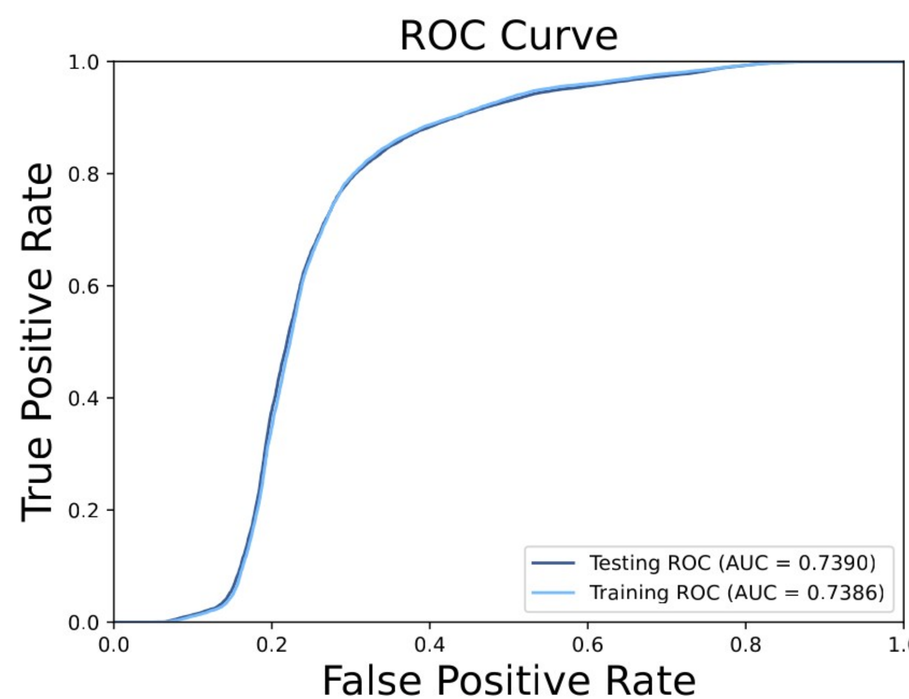
TP	FP
FN	TN

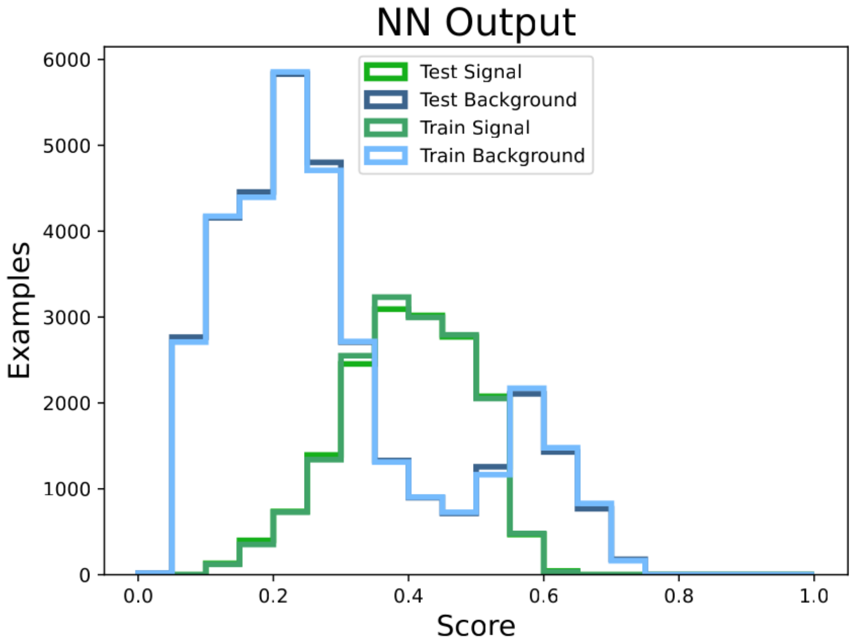
Plot true positive rate vs False positive rate

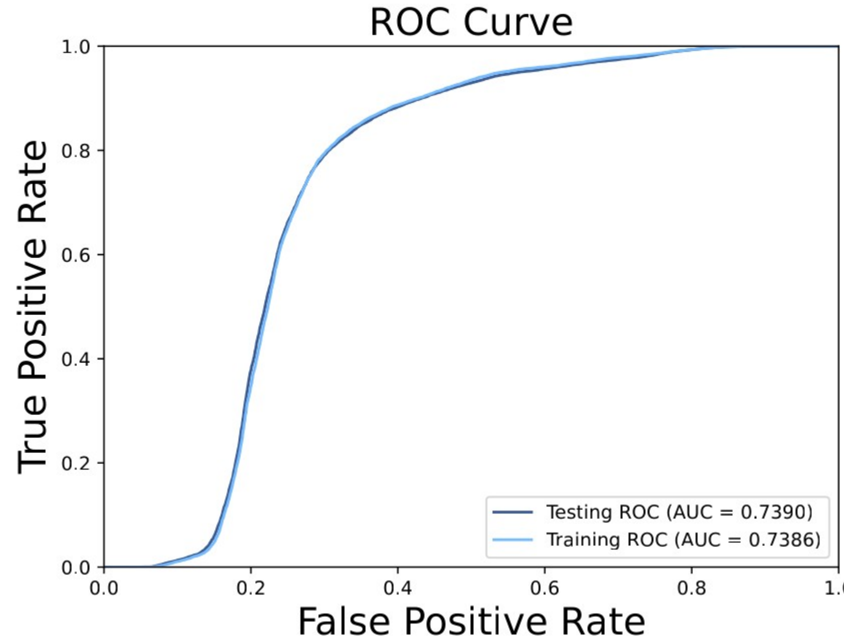
True positive rate = Efficiency of Signal to pass selection
False positive rate = Efficiency of Bkgd to pass selection

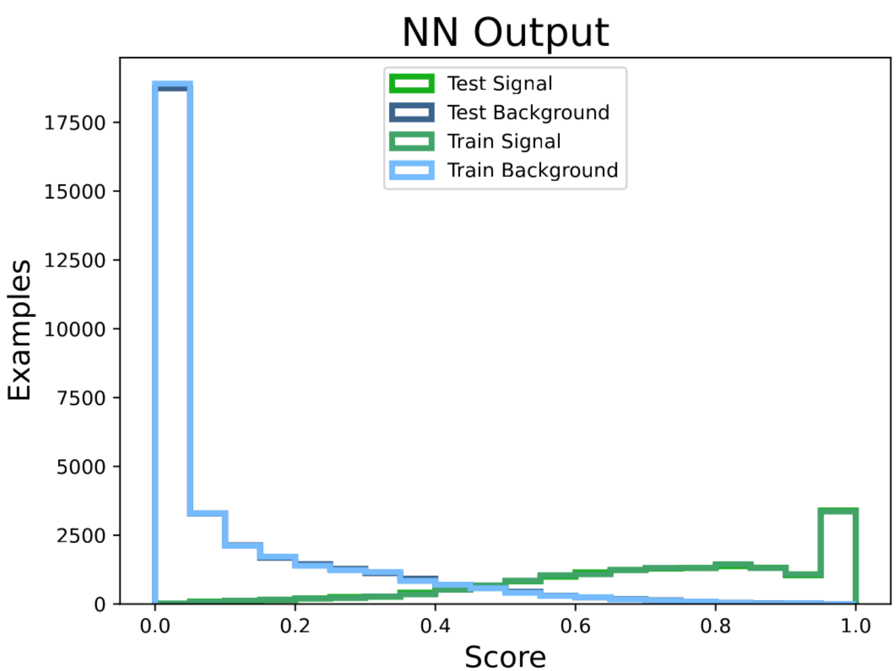
TPR = TP/(TP+FN)
FPR = FP/(FP+TN)

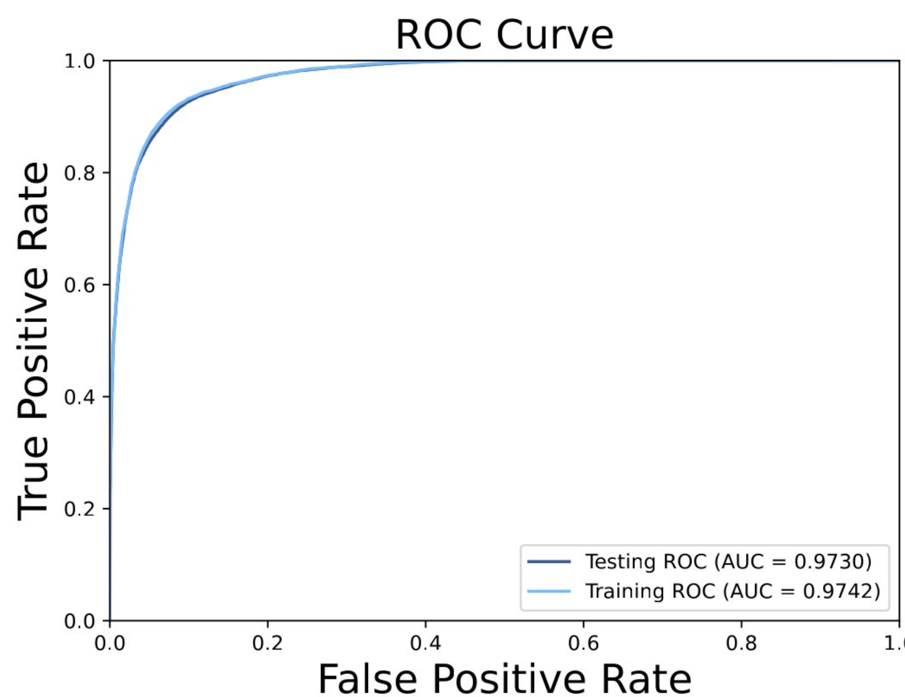




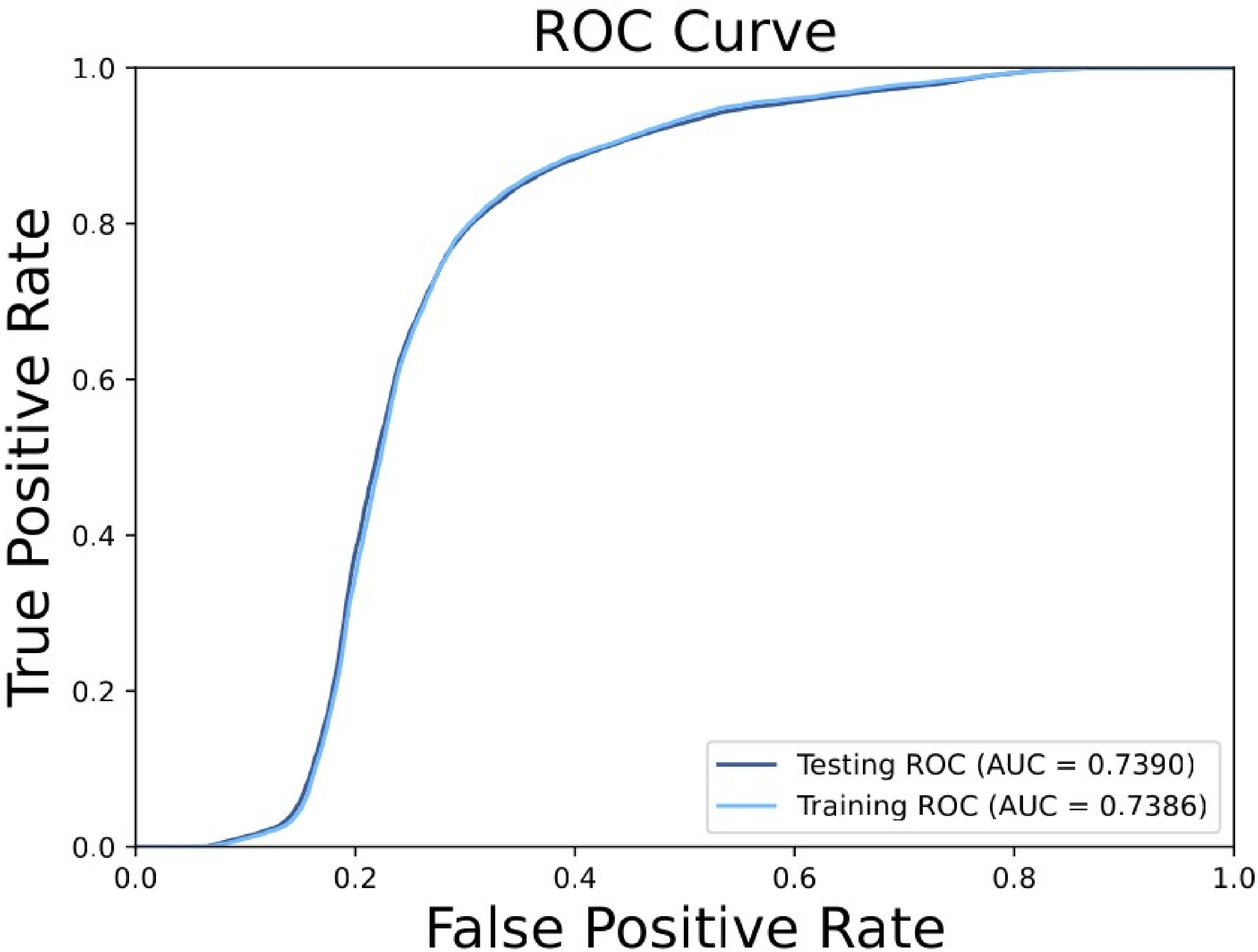

NN Output
Test Signal
Test Background
Train Signal
Train Background
Examples
Score
0
1000
2000
3000
4000
5000
6000
0.0
0.2
0.4
0.6
0.8
1.0


ROC Curve
True Positive Rate
False Positive Rate
Testing ROC (AUC = 0.7390)
Training ROC (AUC = 0.7386)
0.0
0.2
0.4
0.6
0.8
1.0

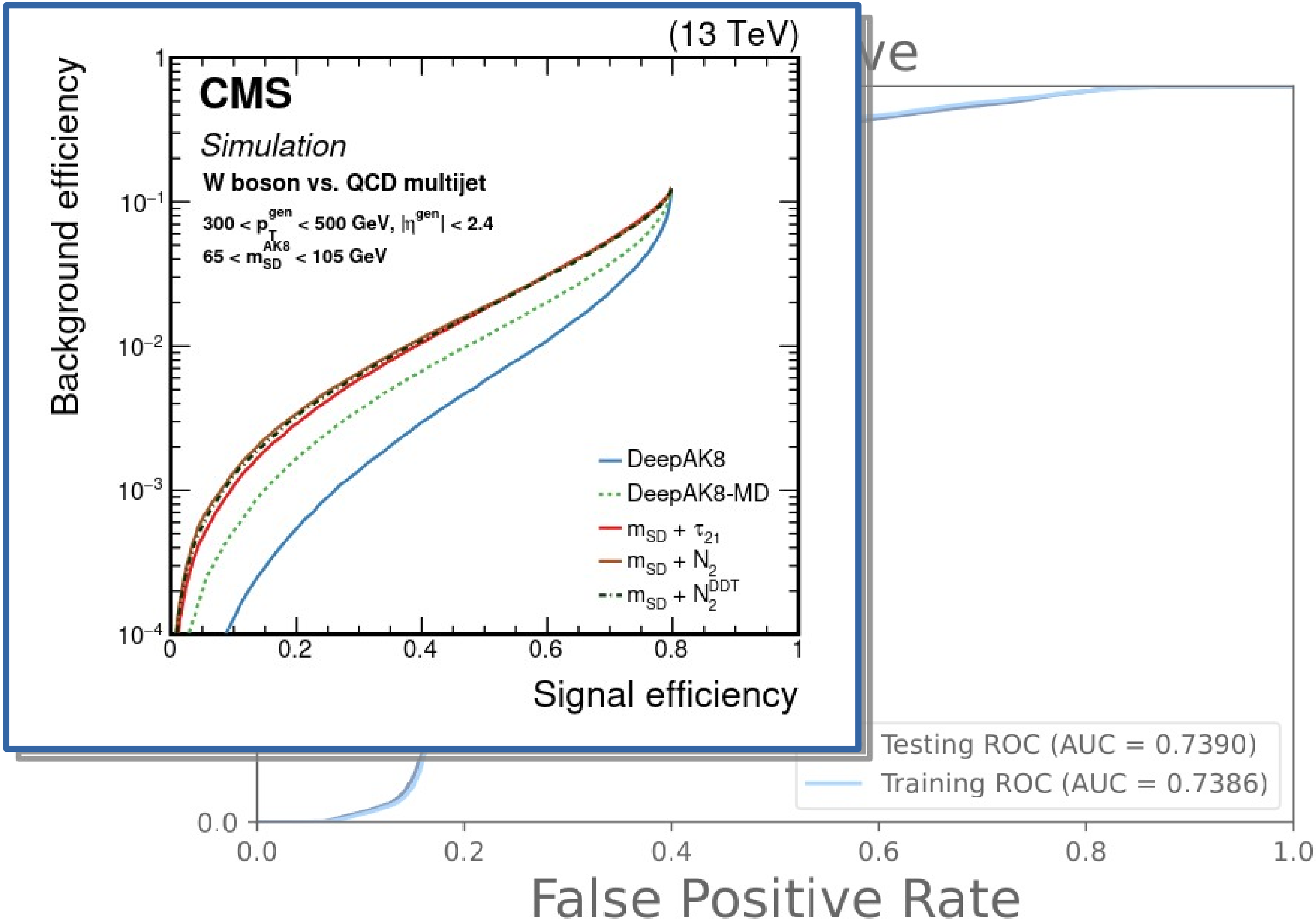

NN Output
Test Signal
Test Background
Train Signal
Train Background
Examples
Score
0
2500
5000
7500
10000
12500
15000
17500
0.0
0.2
0.4
0.6
0.8
1.0


ROC Curve
True Positive Rate
False Positive Rate
Testing ROC (AUC = 0.9730)
Training ROC (AUC = 0.9742)
0.0
0.2
0.4
0.6
0.8
1.0

Check....


ROC Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Testing ROC (AUC = 0.7390)
Training ROC (AUC = 0.7386)

Check....


(13 TeV)
CMS
Simulation
W boson vs. QCD multijet
300 < p_T^{gen} < 500 GeV, $|\eta^{gen}|$ < 2.4
65 < m_{SD}^{AK8} < 105 GeV
Background efficiency
Signal efficiency
DeepAK8
DeepAK8-MD
$m_{SD} + \tau_{21}$
$m_{SD} + N_2$
$m_{SD} + N_2^{DDT}$
Testing ROC (AUC = 0.7390)
Training ROC (AUC = 0.7386)
False Positive Rate

Overtraining

Overtraining is when the network gets very biased by the training examples it has seen… and starts to learn things that are not relevant.

Training to classify cats and dogs –
suppose in half of our training pictures the cat has a black circle on neck.
If we force the network to keep learning, it will eventually "learn" that cats are supposed to have a black circle on the neck.
The network is overtrained.
It will do very well in our training, but as soon as we test it with other random pictures of a cat, its performance will be poor.

Overtraining is when the network has become over-sensitive to qualities in the training data.

Today....

We have two files with code today

1. Usual file, `proj_class.py`
2. Testing file `proj_test_nn.py`

The second file is one where you use an already trained model to make some predictions on a new dataset.

Two input files are given

`projectile_input4_100k.txt` Use this with `proj_class.py` to train/test your network

`projectile_input4_test_20k.txt` Use this with `proj_test_nn.py` to evaluate your network