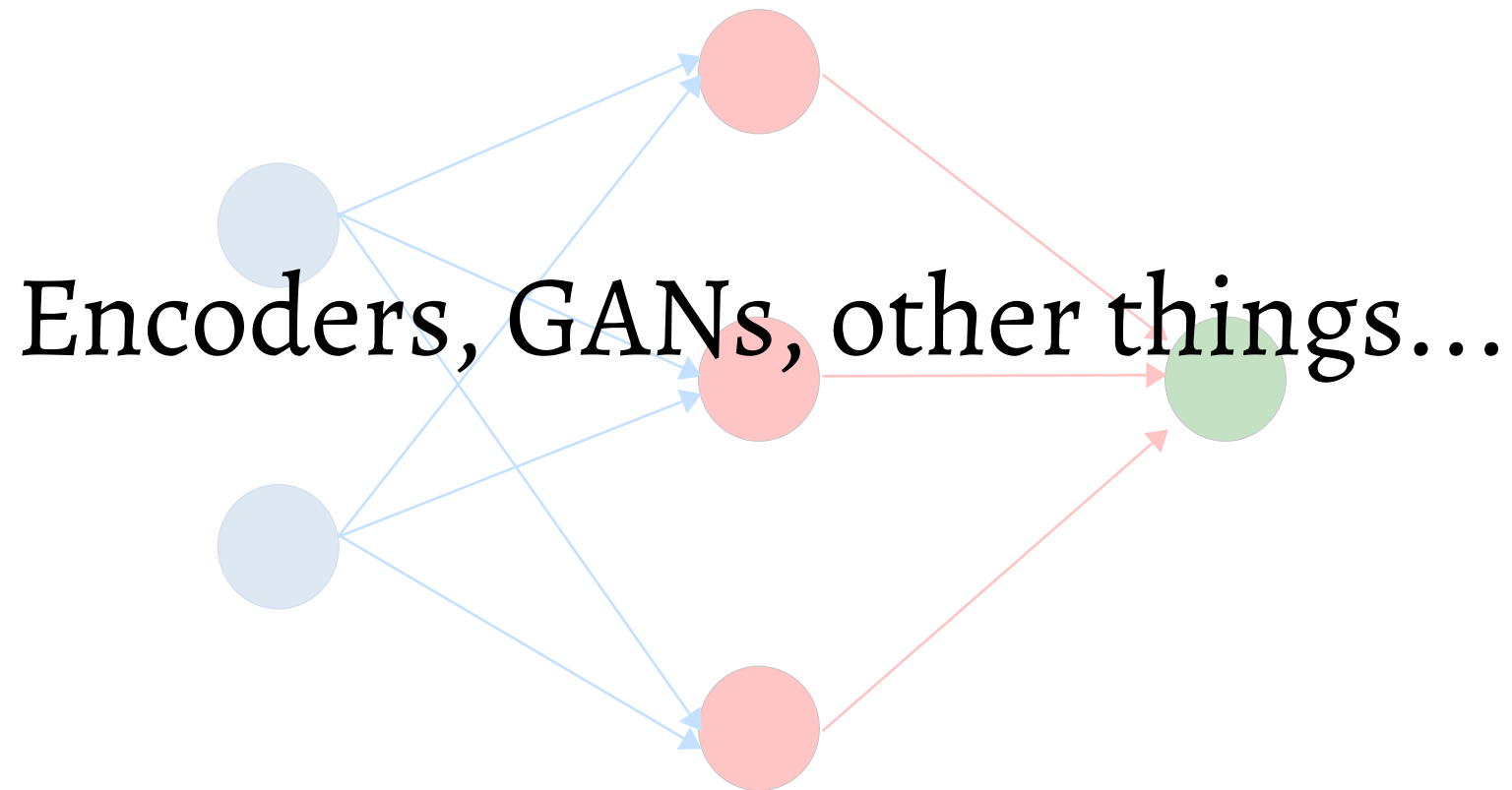
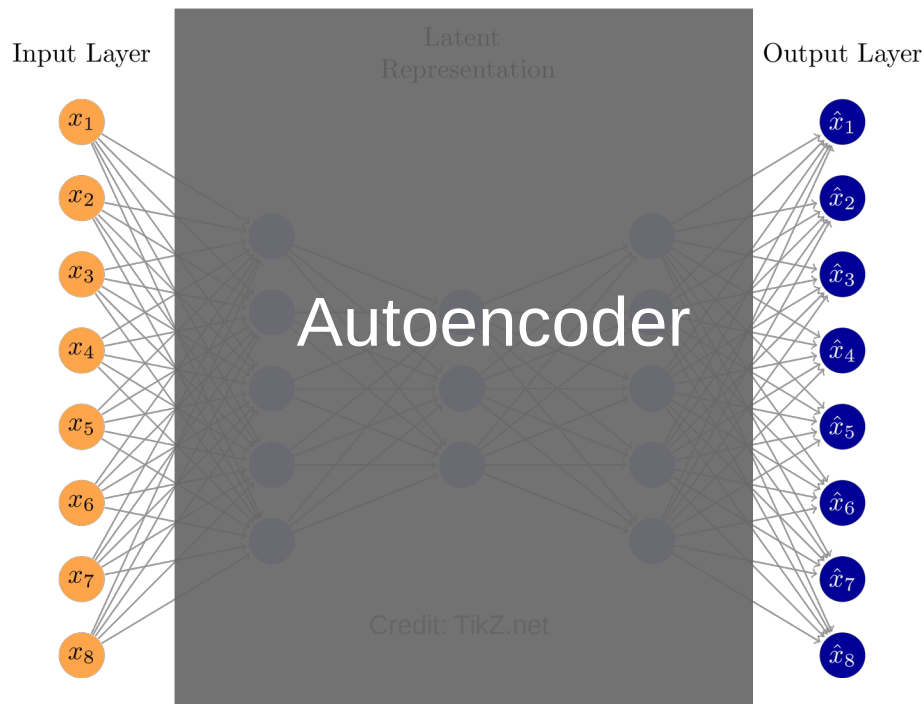


PH6232: Machine Learning for Physics applications



AutoEncoder

Autoencoders are a dimensionality reduction (non linear) algorithm



Goal

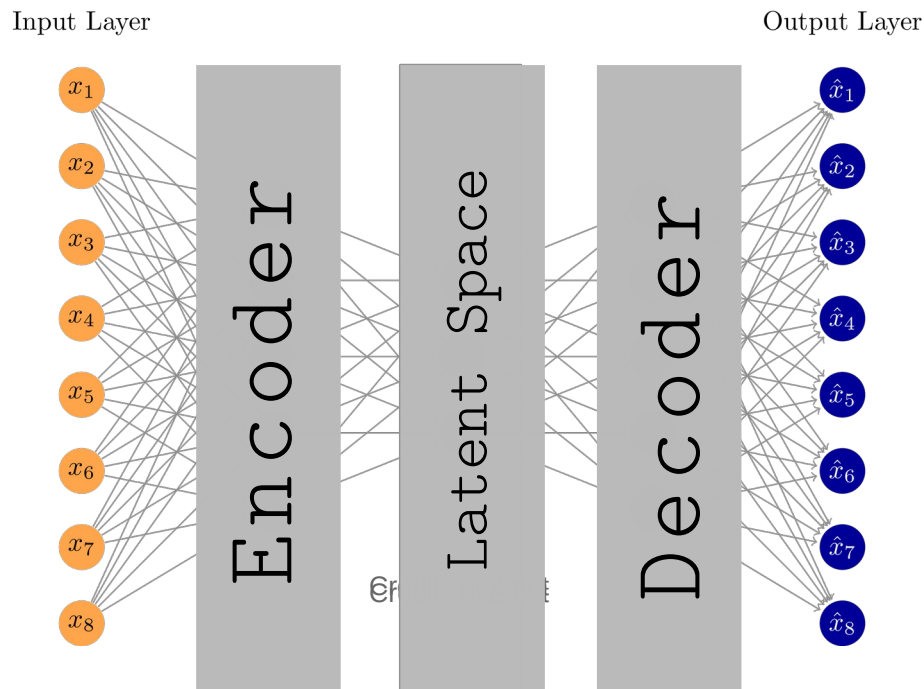
- Reproduce Input Features
- Learn meaningful patterns from data

Usage

- As an anomaly detection technique
- Recovering blurred images
- Preprocessing step before any other ML algorithm, visualizing data in low dimensional latent space

AutoEncoder

Autoencoders are a dimensionality reduction (non linear) algorithm



Latent Representation is
the low dimensional space
you are targetting to compress
the data

Usually lower dimension than Input
features

AutoEncoder architecture

Input = X

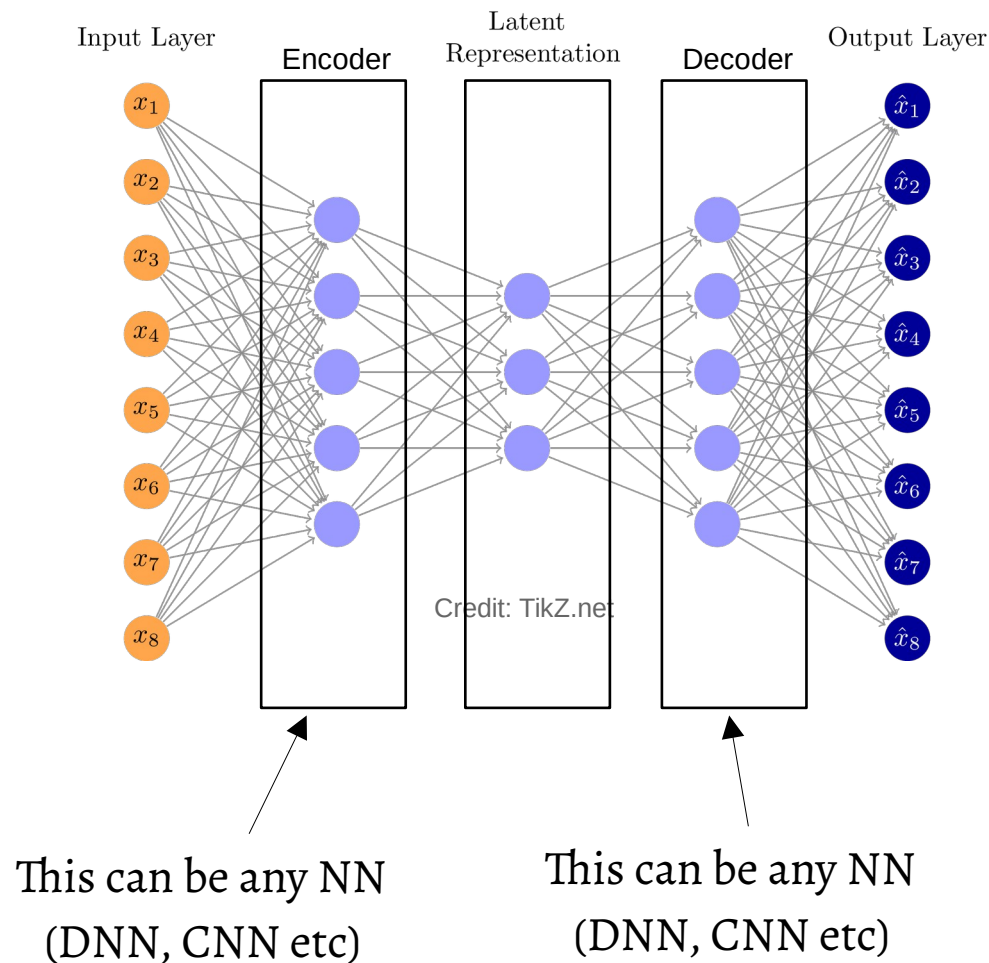
Target = X

What you can change

- Number of layers and neurons in layers for both encoder and decoder
- Encoder and decoder need not to be same class of NN or same num. of layers (or neurons)
- Latent Dimension
- Loss Function

$$MSELoss = \frac{1}{N} \sum_{i=1}^N (X_i - \hat{X}_i)^2$$

Typically captures how well the reconstructed output are compared to input

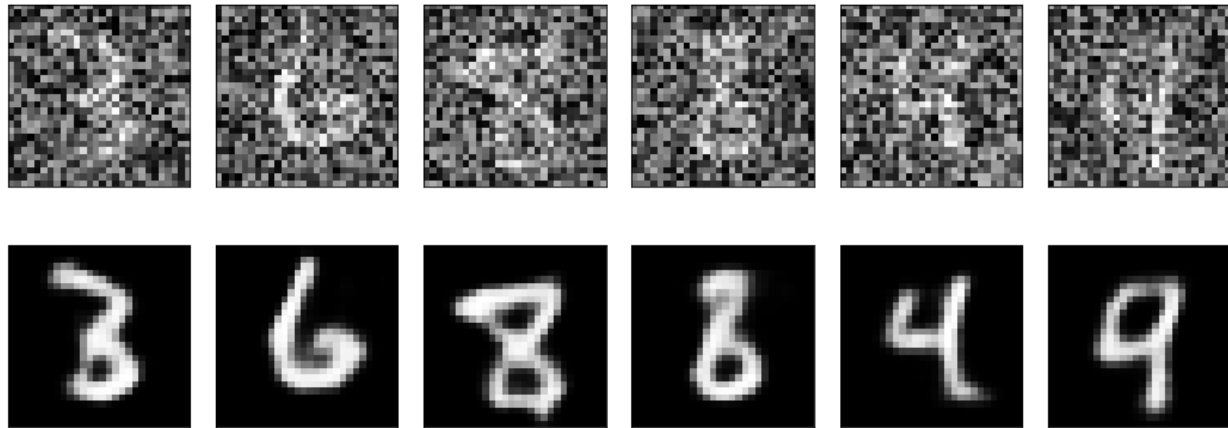


This can be any NN
(DNN, CNN etc)

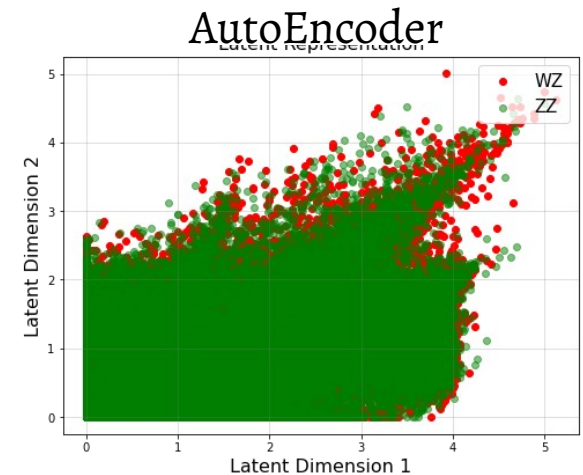
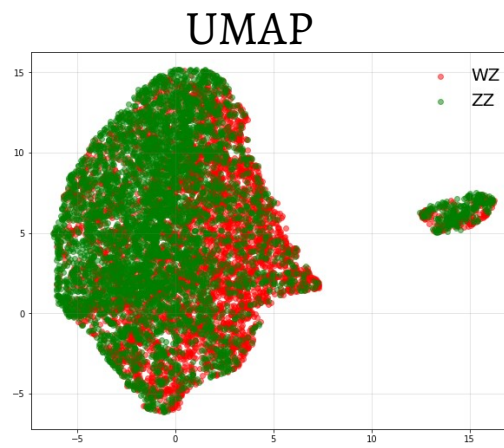
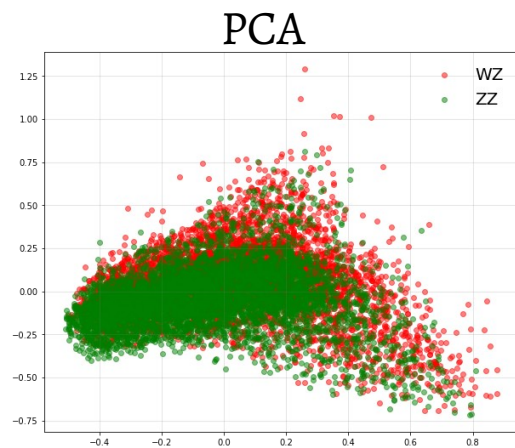
(Typically we use same class
of NN used in Encoder)

Examples...

Recovering blurred Images



WZ-ZZ Problem



WZ and ZZ events in 2DLatent Space

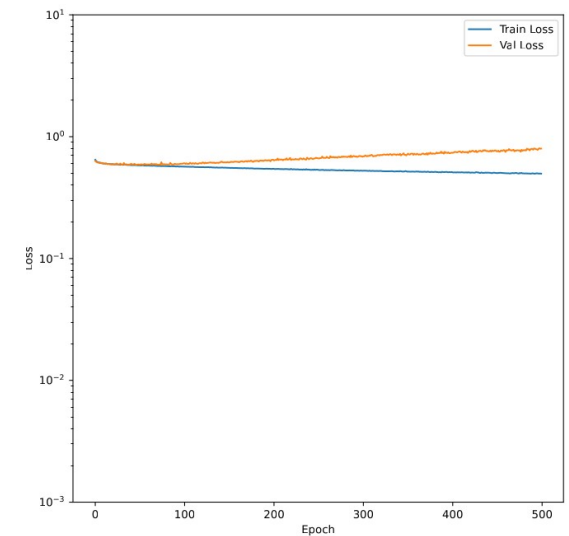
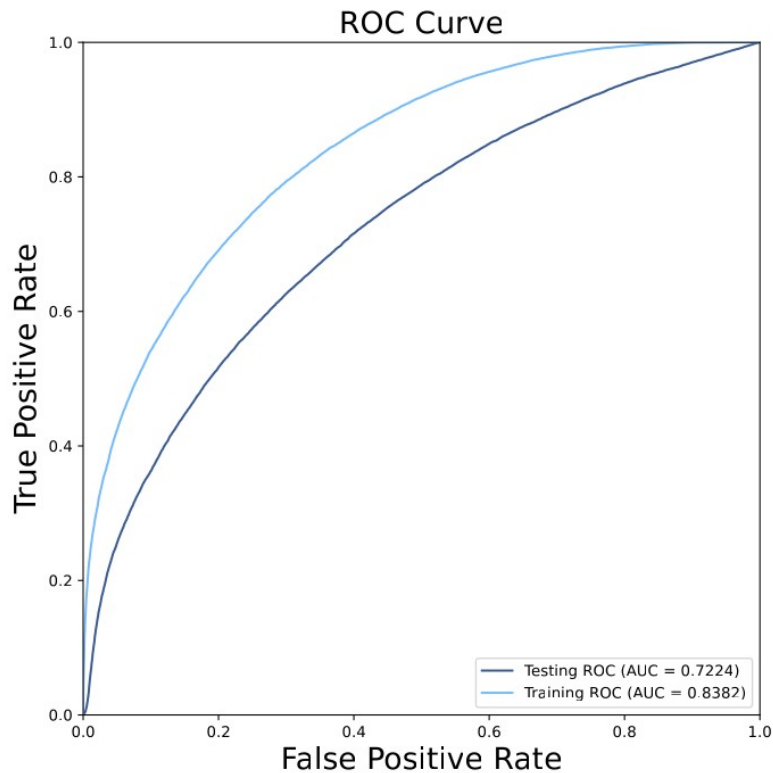
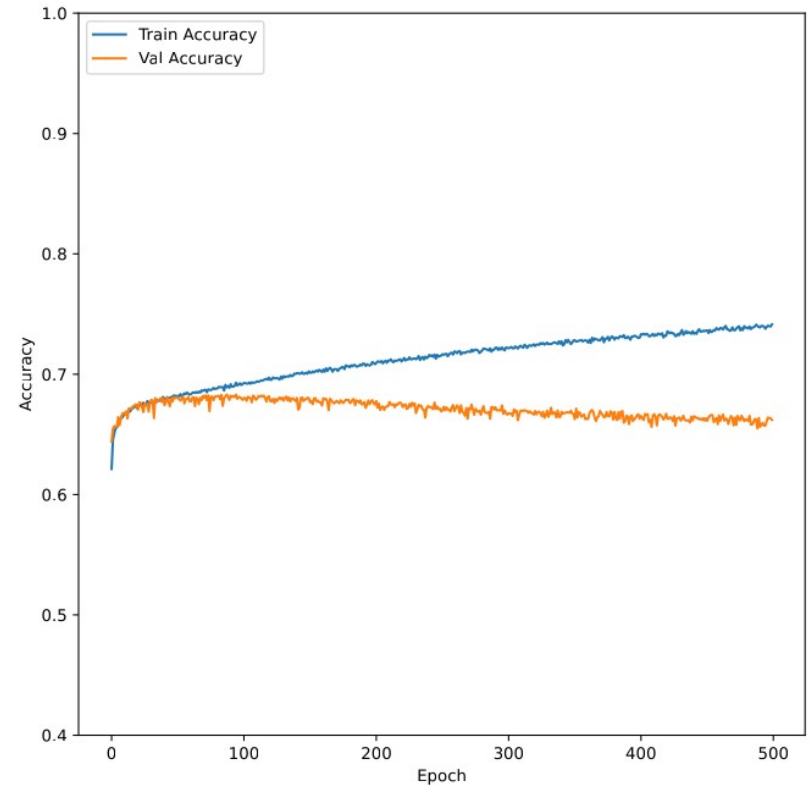
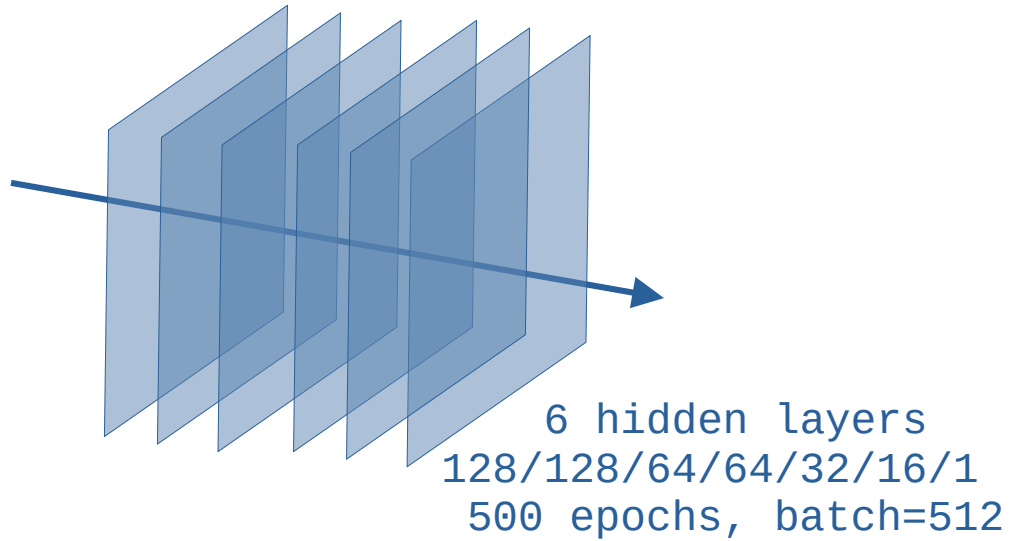
Overtraining

Overtraining is when your network learns specific details of your training dataset that are not general features of the problem.

For example overtraining can occur when you keep training for many epochs. In such a case, the performance on the training dataset can keep improving, but the performance on a test dataset does not improve.

An overtrained network shows good performance on the training dataset, but is unable to reproduce the performance on the testing dataset.

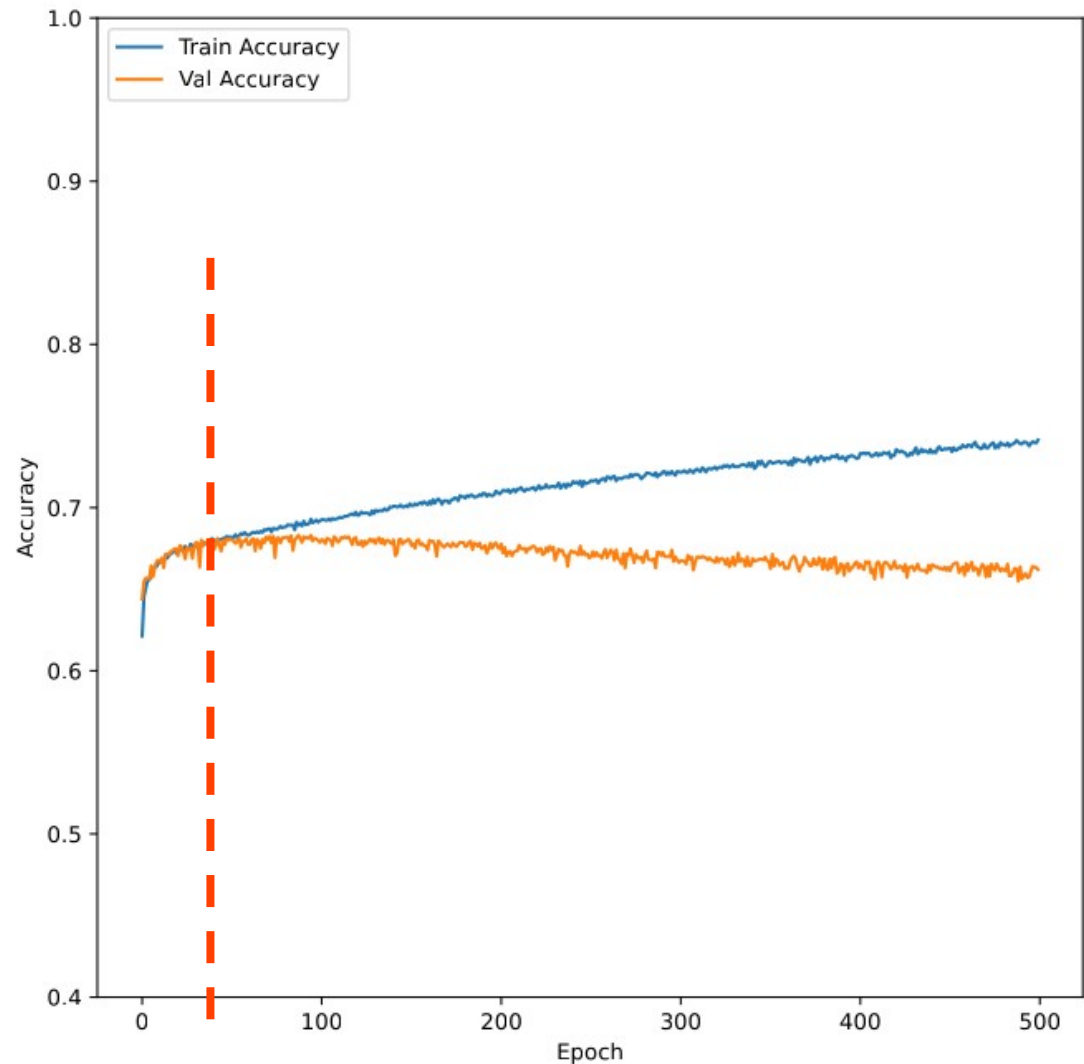
Overtraining – the HEP WZ/ZZ problem



Mitigating overtraining

Early stopping.

Here we could stop at epoch 40 for example...




Mitigating overtraining

Implementing “**dropout**” regularization scheme

At each training step, we drop a fraction of neurons
(set their output to zero).

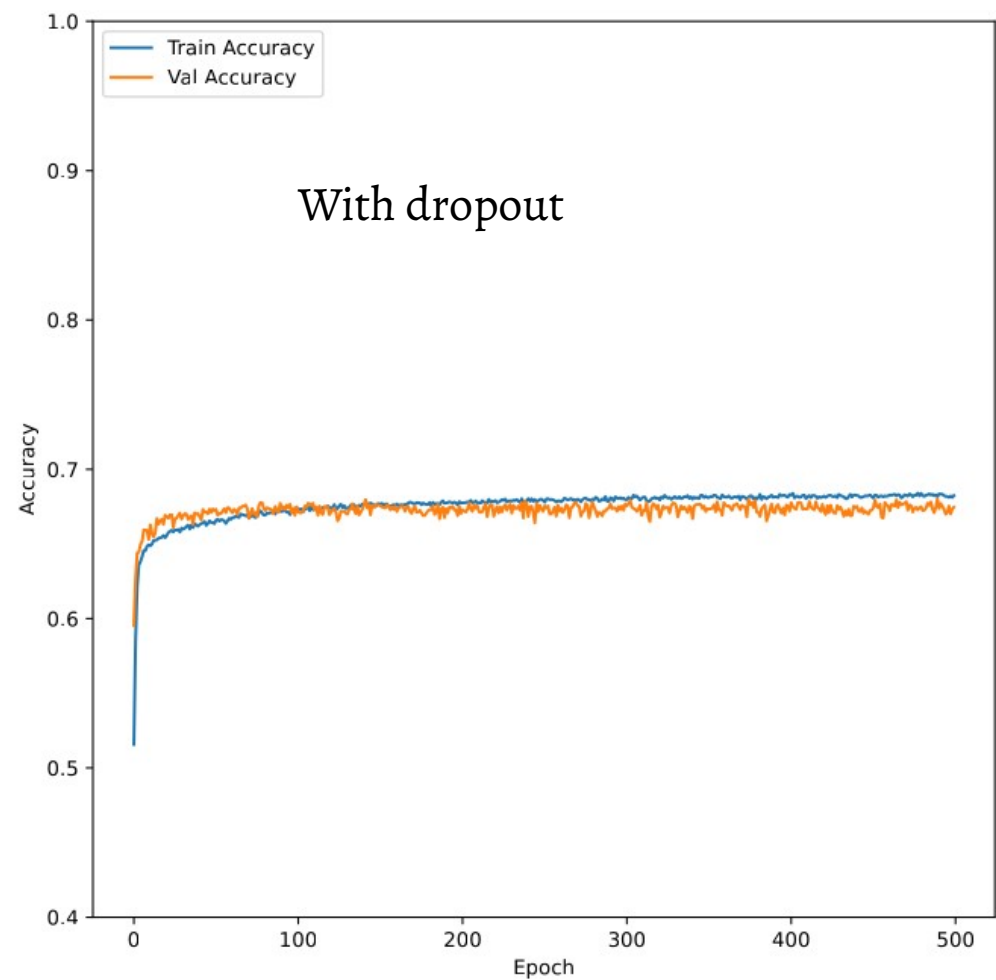
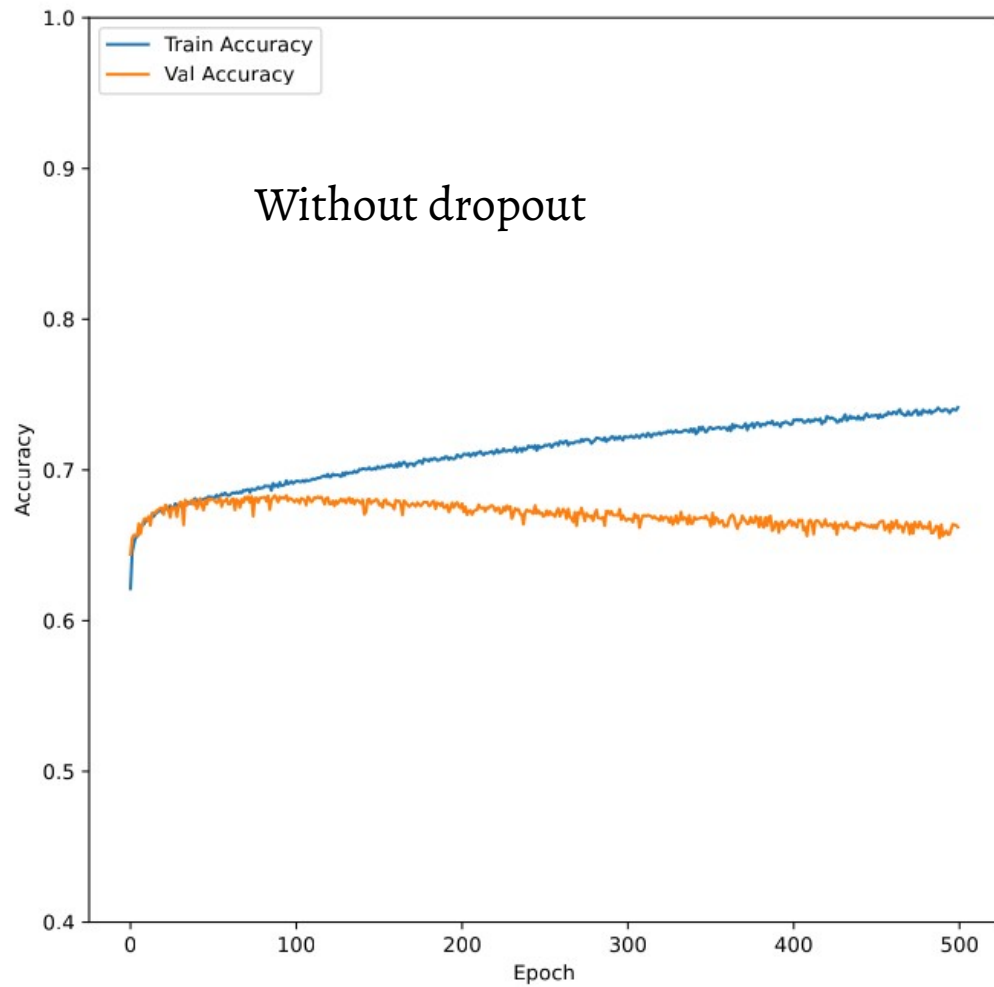
Implemented by adding dropout layers with a probability.

```
model = Sequential()  
model.add(Dense(128, activation='relu', ...  
model.add(Dense(128, activation='relu', ...  
model.add(Dense(64, activation='relu', ...  
model.add(Dense(64, activation='relu', ...  
model.add(Dense(32, activation='relu', ...  
model.add(Dense(16, activation='relu', ...  
model.add(Dense(1, activation='sigmoid'))
```

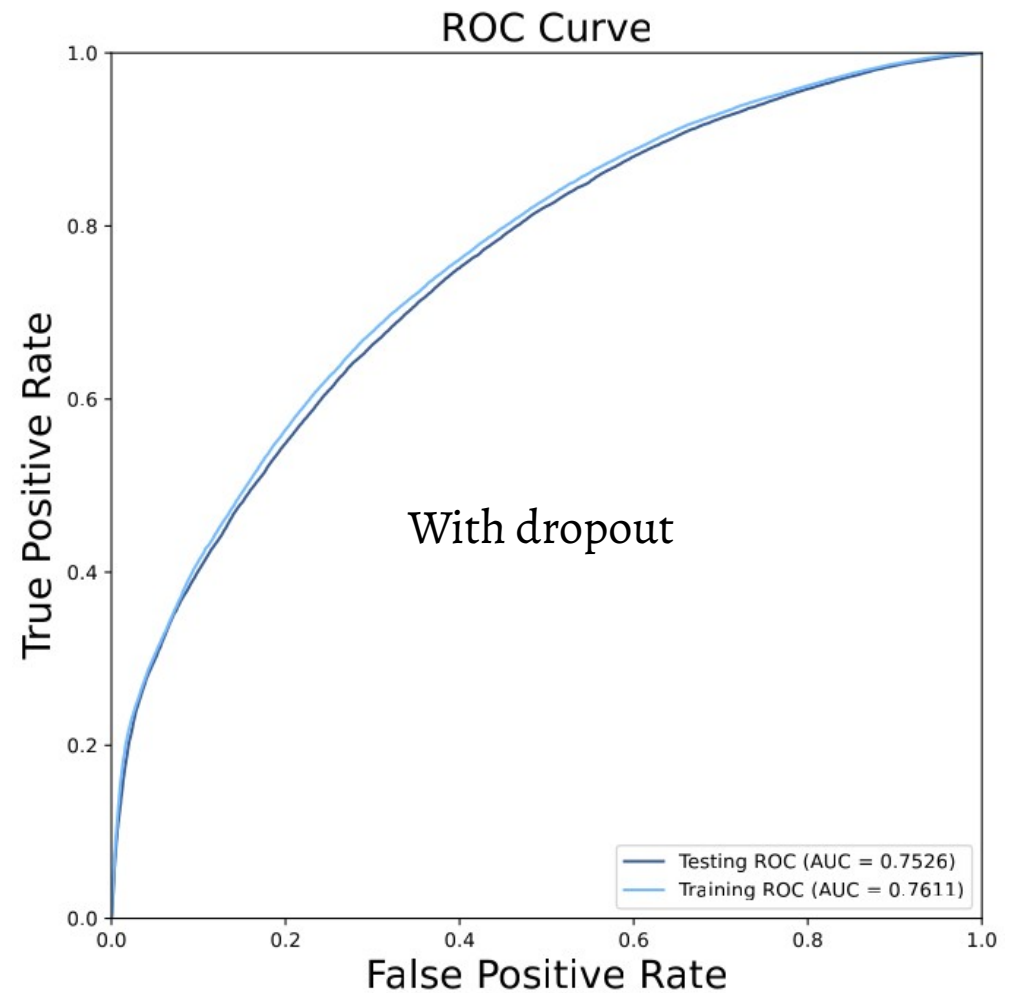
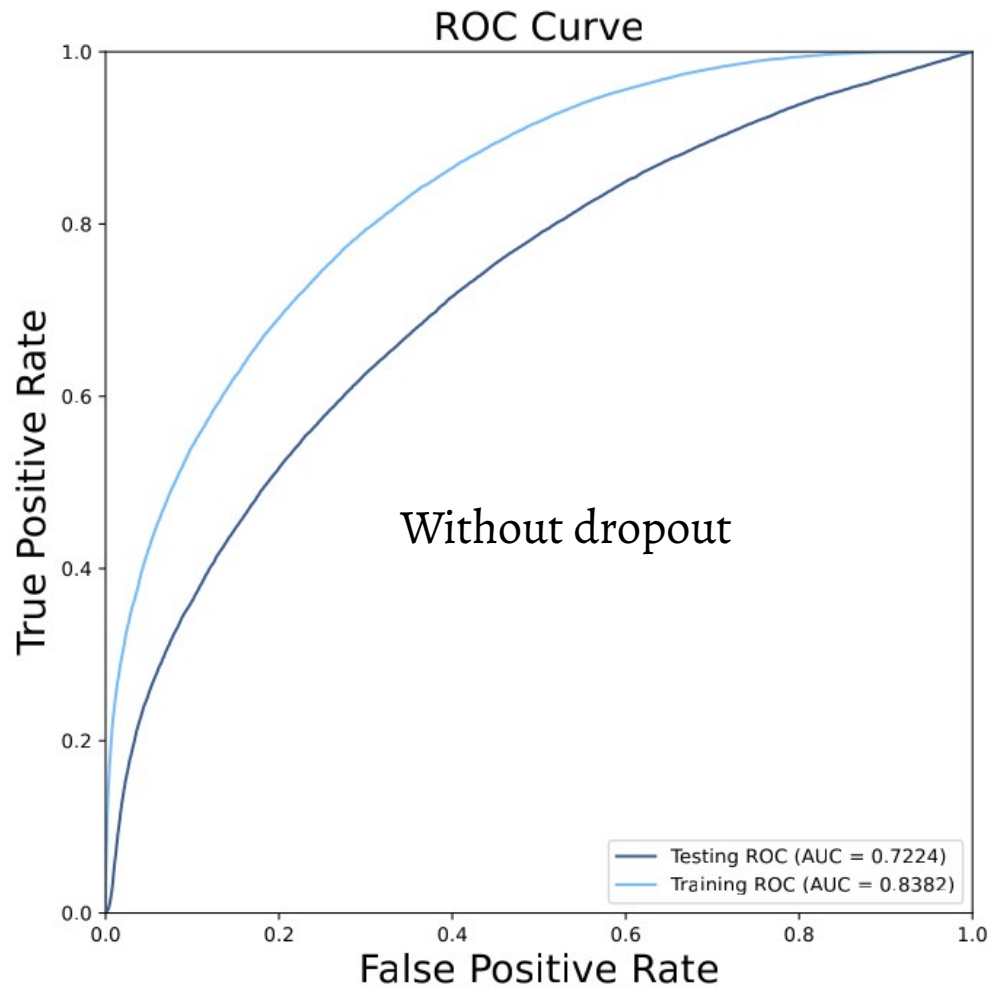


```
model = Sequential()  
model.add(Dense(128, activation='relu', ...  
model.add(Dense(128, activation='relu', ...  
model.add(Dense(64, activation='relu', ...  
model.add(Dropout(0.5))  
model.add(Dense(64, activation='relu', ...  
model.add(Dropout(0.5))  
model.add(Dense(64, activation='relu', ...  
model.add(Dropout(0.5))  
model.add(Dense(32, activation='relu', ...  
model.add(Dense(16, activation='relu', ...  
model.add(Dense(1, activation='sigmoid'))
```

Dropout



Dropout



Ranking variables

The list of input features given to a neural network determine its performance.

For example, for a classifier, an input feature that looks different for the two classes will help the classification.

For a given set of variables, and a given network, can we rank the input features to see which ones are important, and which ones aren't helping?

Permutation_importance

```
from sklearn.inspection import permutation_importance
```

[link](#)

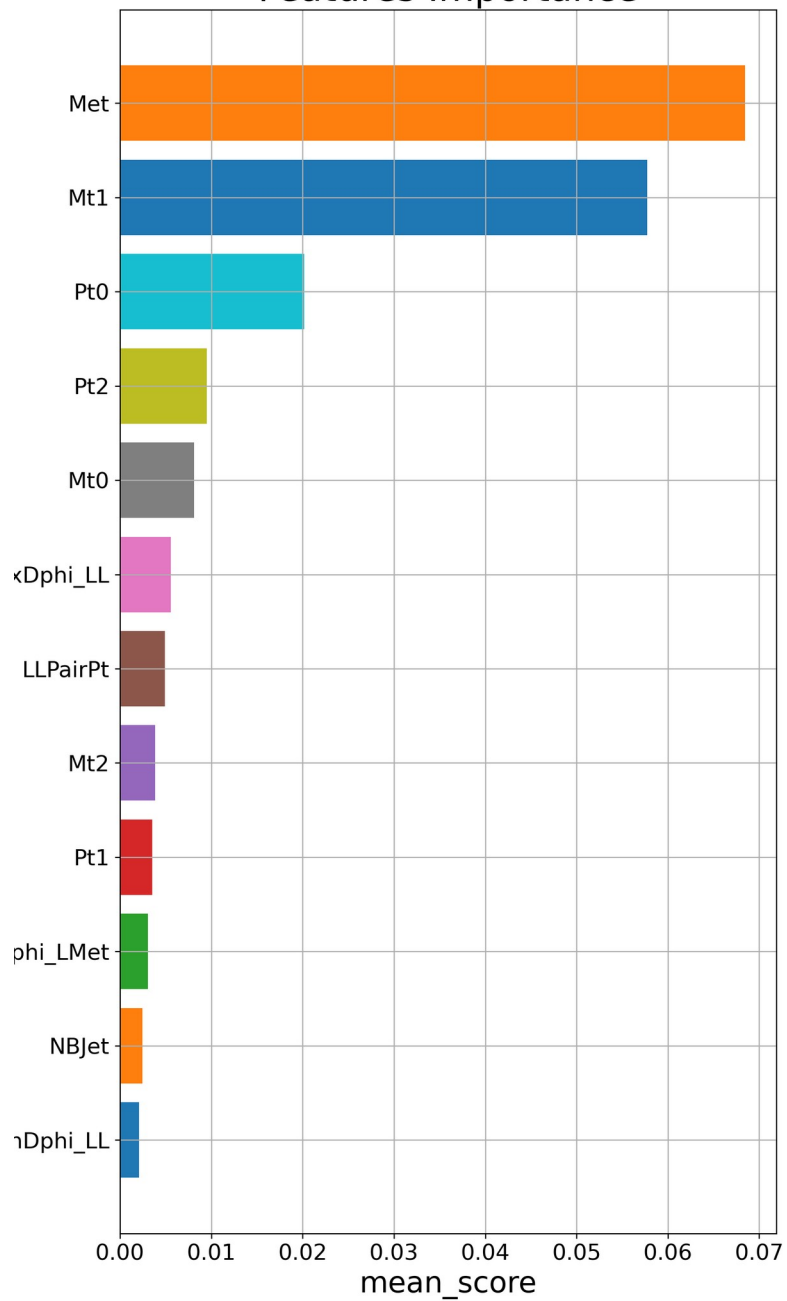
Permutation importance is defined as decrease in model performance when one feature is randomly shuffled.

	X ₁	X ₂	X ₃	X ₄	Output $y = f(x_i)$
0						y^0
1						y^1
2						y^2
3						y^3

	X ₁	X ₂	X ₃	X ₄	Output $y = f(x_i)$
1	.	0	y^0
3	.	1	y^1
0	.	2	y^2
2	.	3	y^3

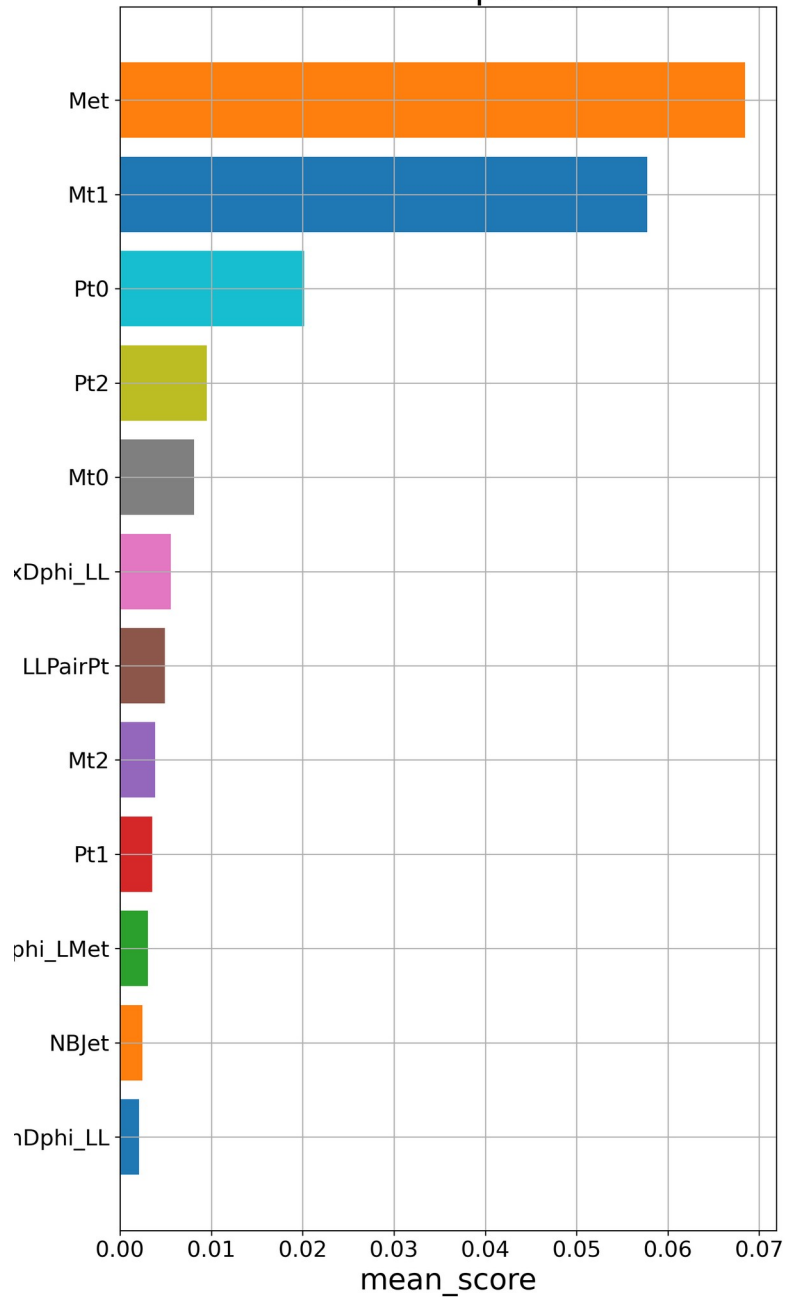
If we shuffle one feature randomly, then how much does the output change?

Features Importance



Permutation_importance WZ/ZZ

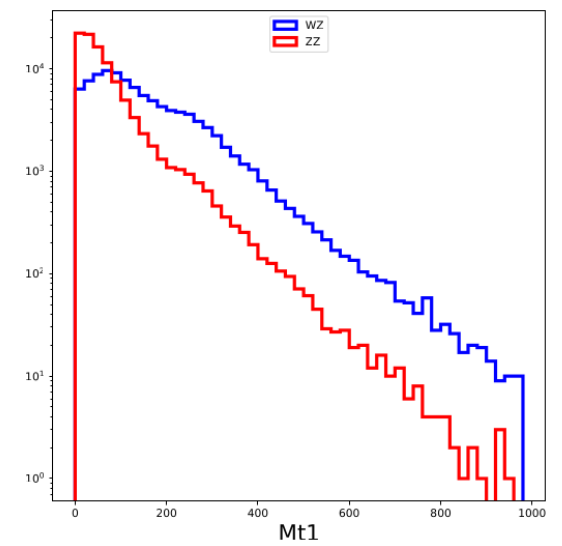
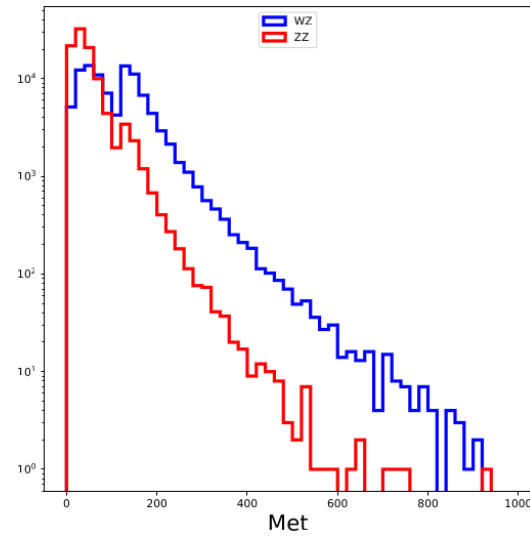
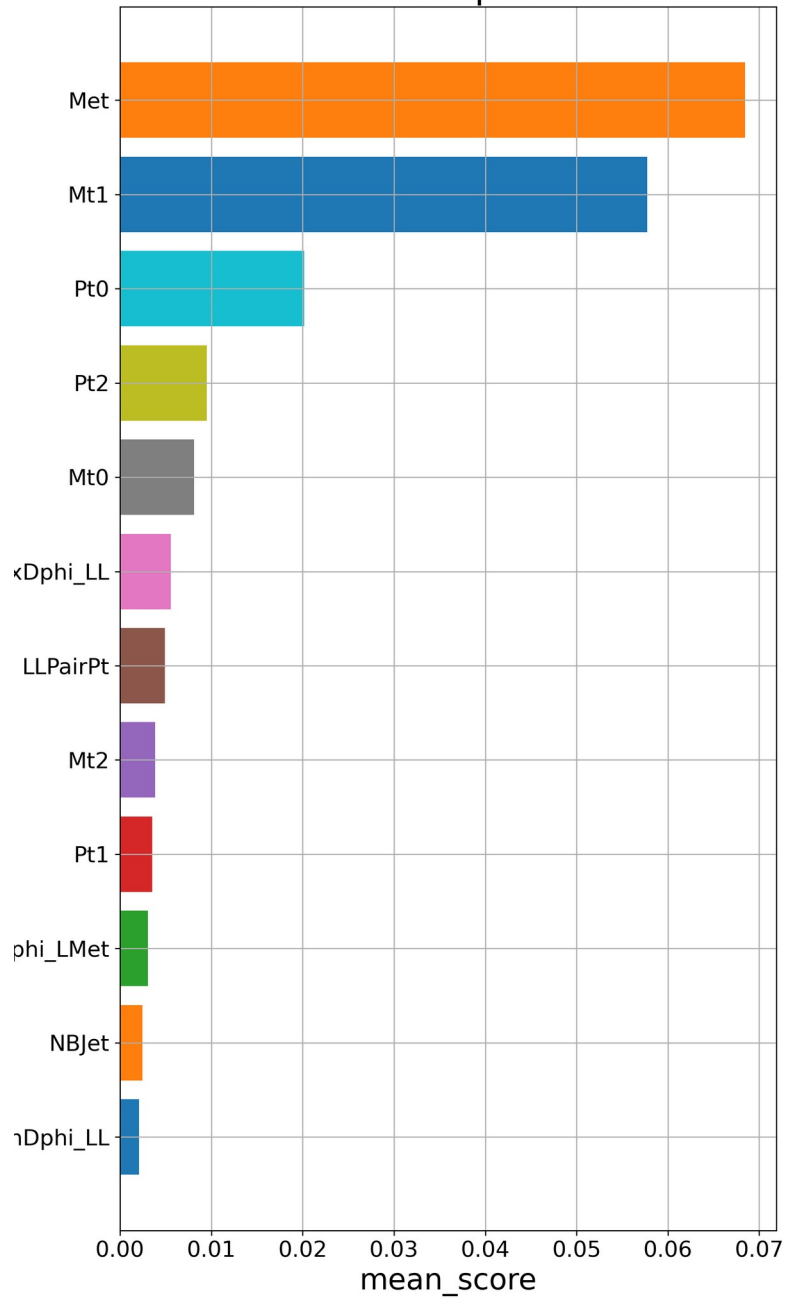
Features Importance



One thing to remember: this tells us how important this feature is for *this* model.

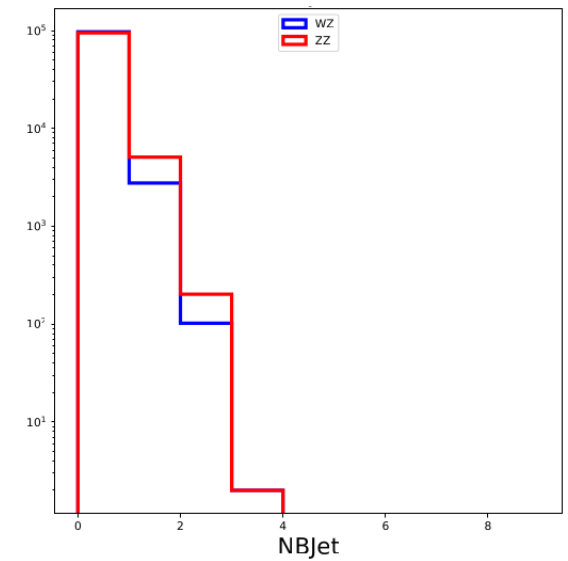
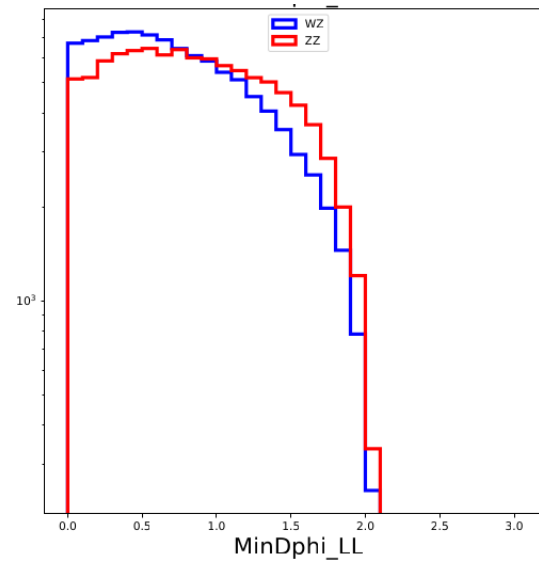
Permutation_importance WZ/ZZ

Features Importance



Top two ↑

Bottom two ↓



Multiclassifiers

So far we have seen binary classifiers: is given example class A or class B? A binary classifier has one output neuron which gives a score (is example of class A or class B?)

In a multiclassifier, we have multiple outputs – as many outputs as there are classes.

Say we have three classes A, B, C.

In a multiclassifier, each of these gets an output

OutputA = score that the example is of class A

OutputB = score that the example is of class B

OutputC = score that the example is of class C

Typically we design $\text{OutputA} + \text{OutputB} + \text{OutputC} = 1$

(then we can think of these as a probability)

Multiclassifier example

Binary classifier

Read in WZ, ZZ dataframes

WZ got label 0, ZZ got label 1

Network was

```
model = Sequential()  
model.add(Dense(32, activation='relu',  
kernel_initializer='he_normal',  
input_dim=n_features))  
model.add(Dense(16, ...  
model.add(Dense(1, activation='sigmoid'))
```

There is one output

```
nnscore = mymodel.predict(df)
```

Say we had three processes WZ, ZZ, and ttZ


Multiclassifier

Read in WZ, ZZ, ttZ dataframes

WZ = 0 , ZZ = 1 , ttZ = 2

Network was

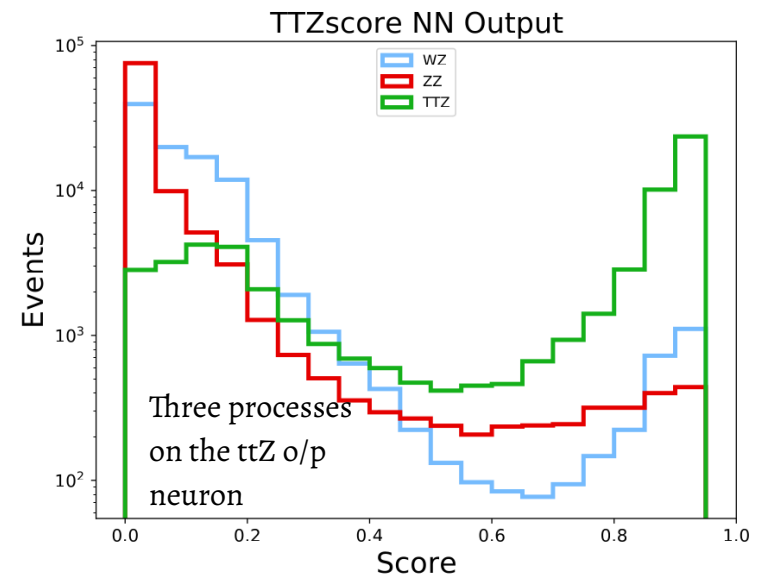
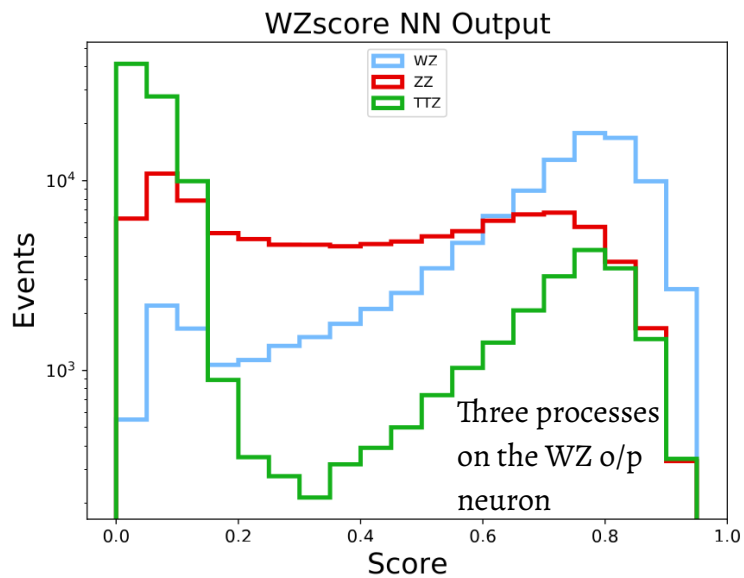
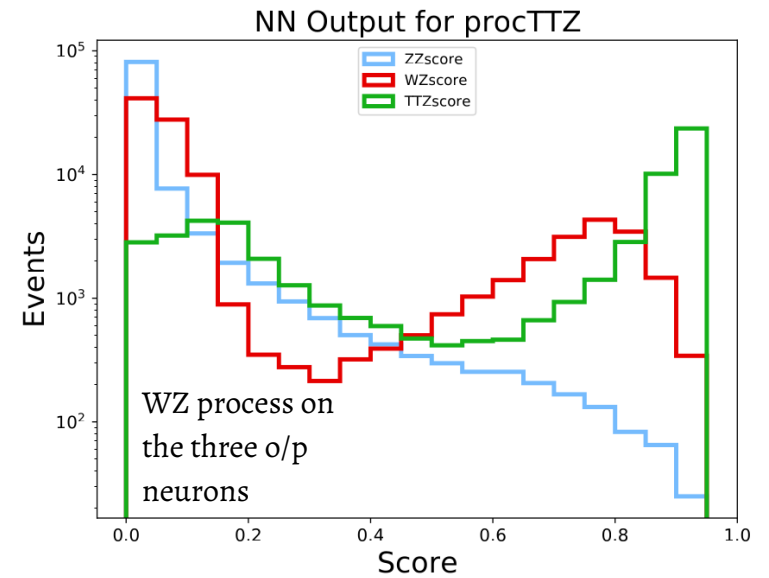
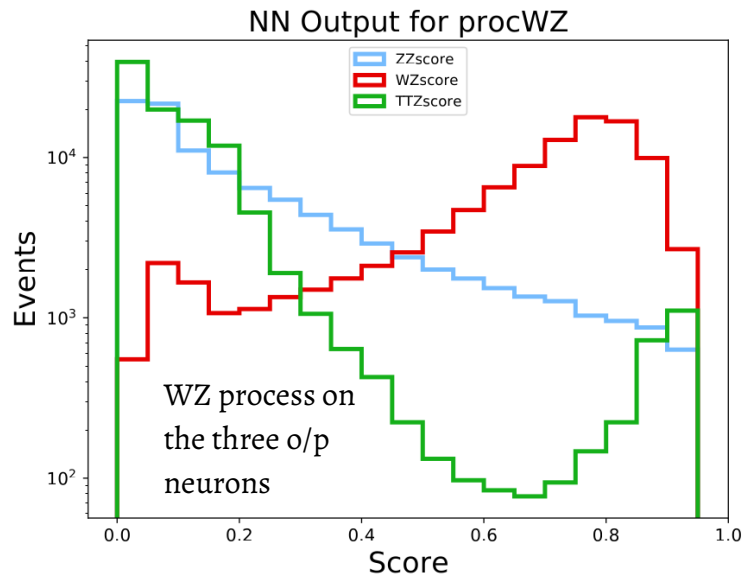
```
model = Sequential()  
model.add(Dense(32, activation='relu',  
kernel_initializer='he_normal',  
input_dim=n_features))  
model.add(Dense(16, ...  
model.add(Dense(3, activation='softmax'))
```



There are three outputs

```
nnscore = mymodel.predict(df)  
nnscore[:,0] label=0  
nnscore[:,1] label=1  
nnscore[:,2] label=2
```

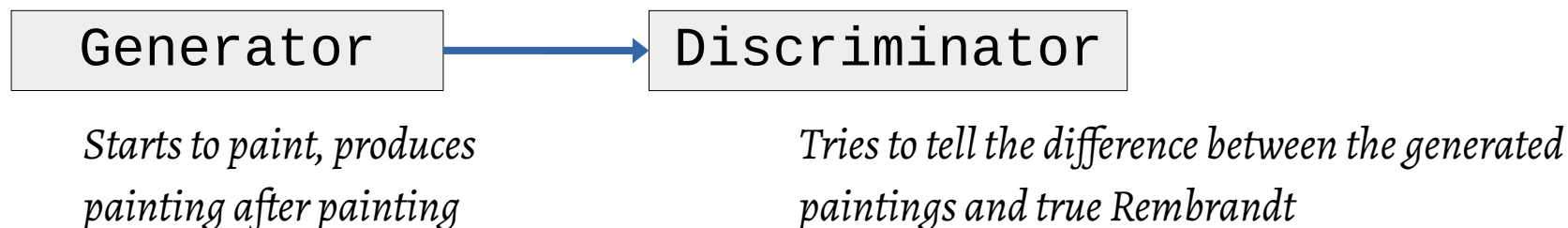
Multiclassifier example



Generative Adversarial Networks

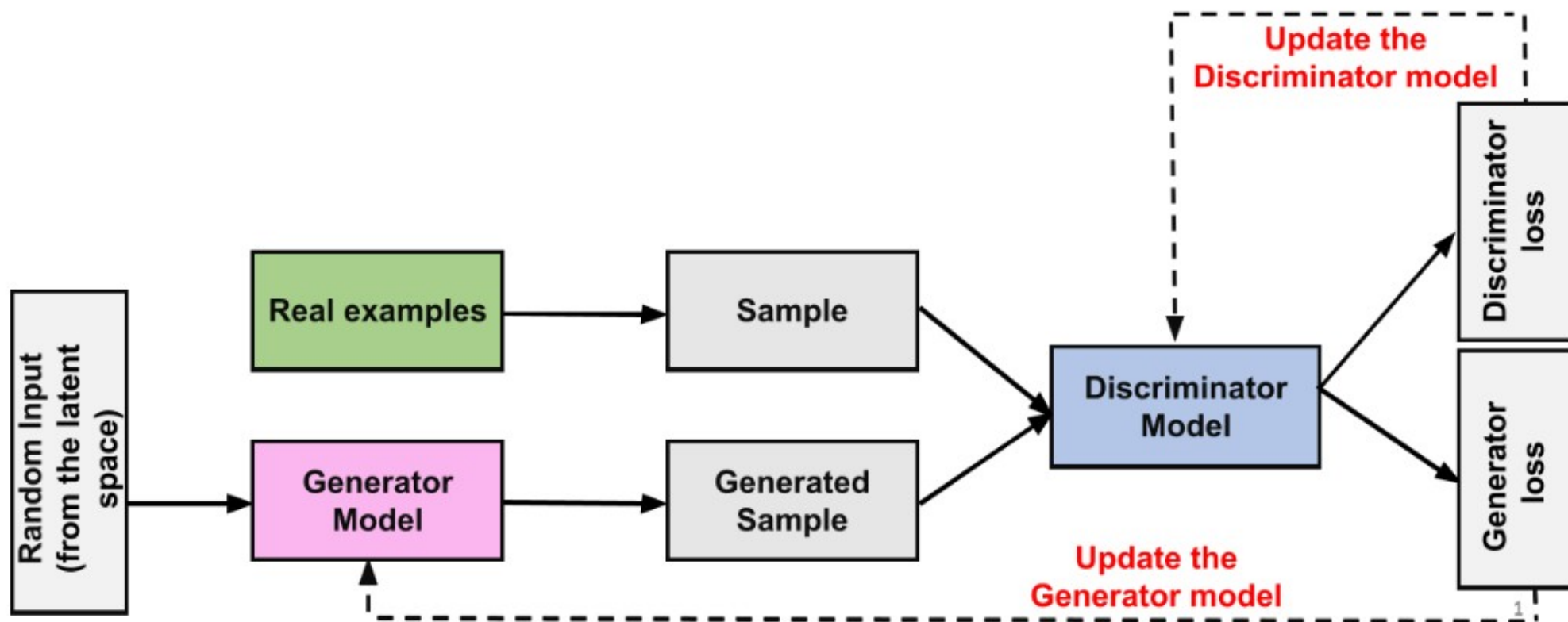
Specific neural network architecture to “generate”

Suppose we want to produce paintings in the style of Rembrandt



Generator's job: produce a painting so good that the discriminator cannot tell the difference between it and a true painting. (thus the adversarial relation between the two)

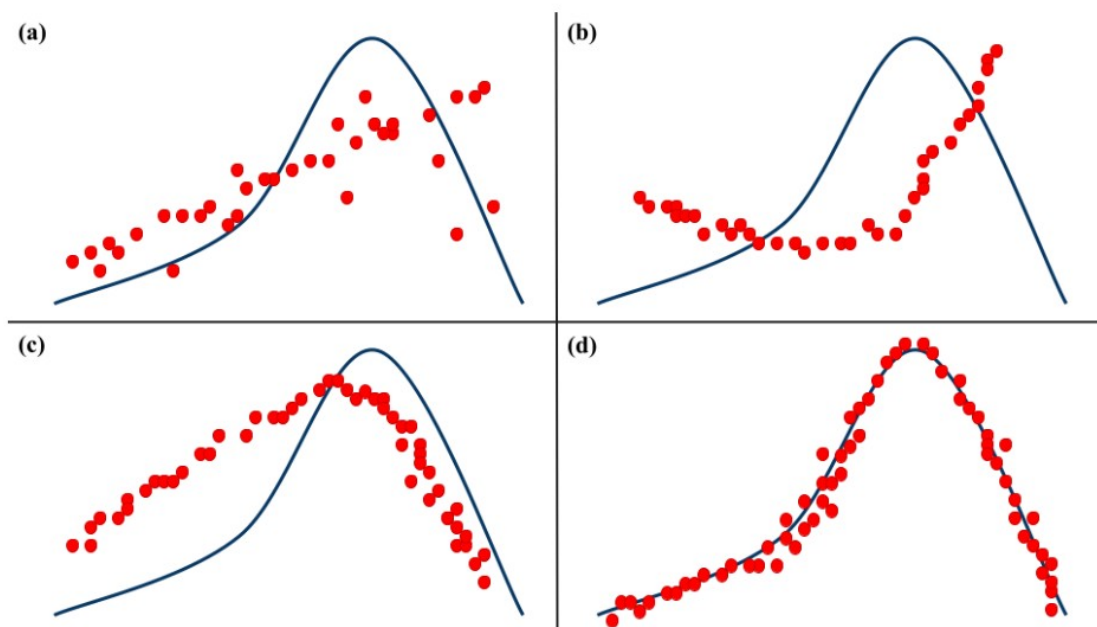
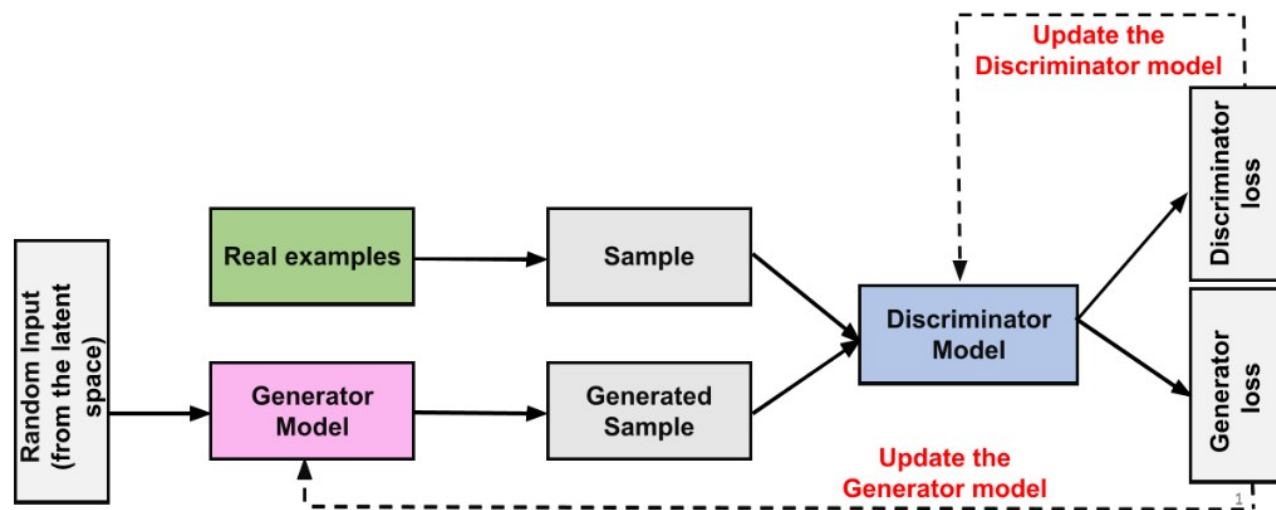
Generative Adversarial Networks



Aparna Jayaraj,
MS thesis 2023

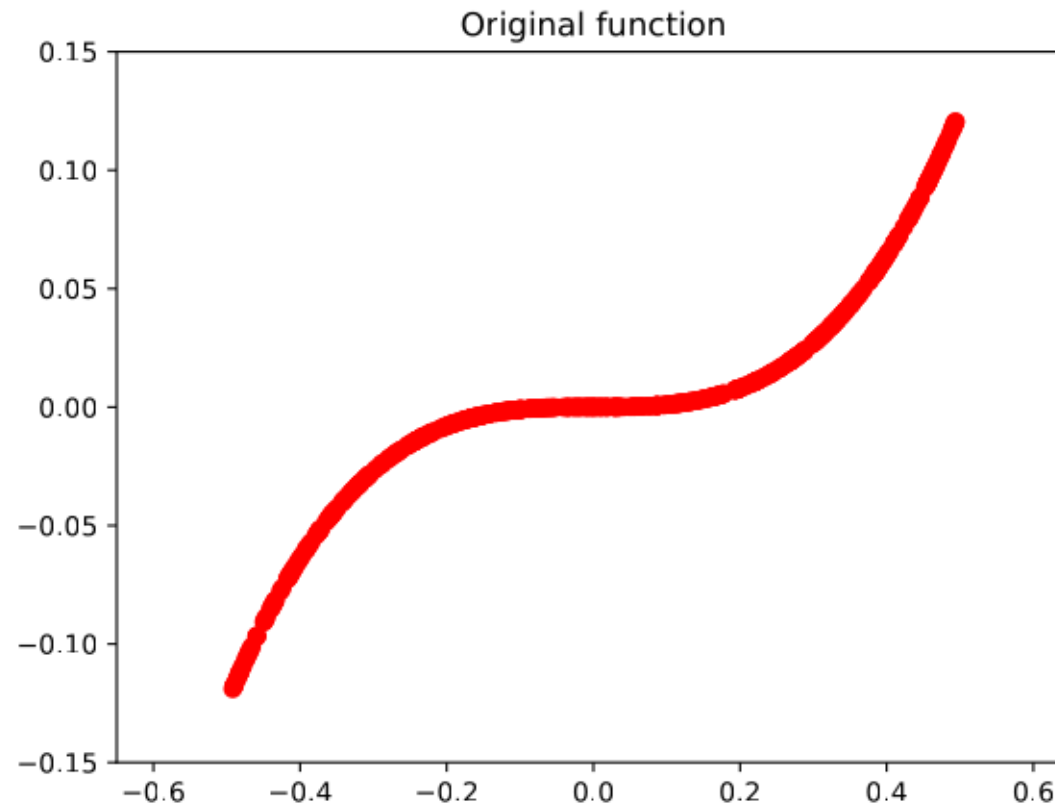
Generative Adversarial Networks

Aparna Jayaraj,
MS thesis 2023

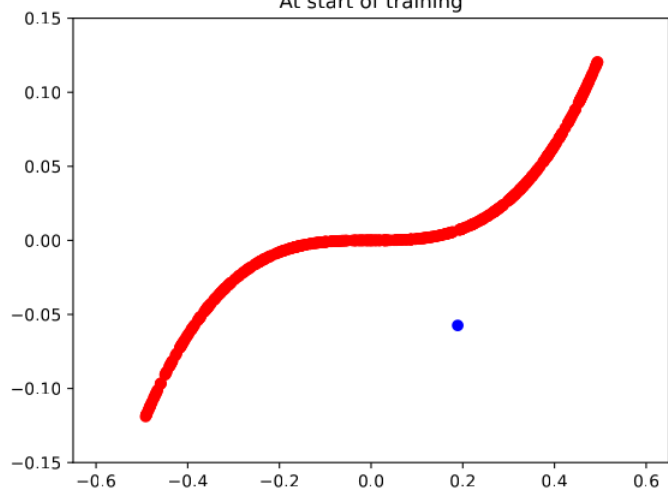


$a \rightarrow b \rightarrow c \rightarrow d$
Increasing epochs

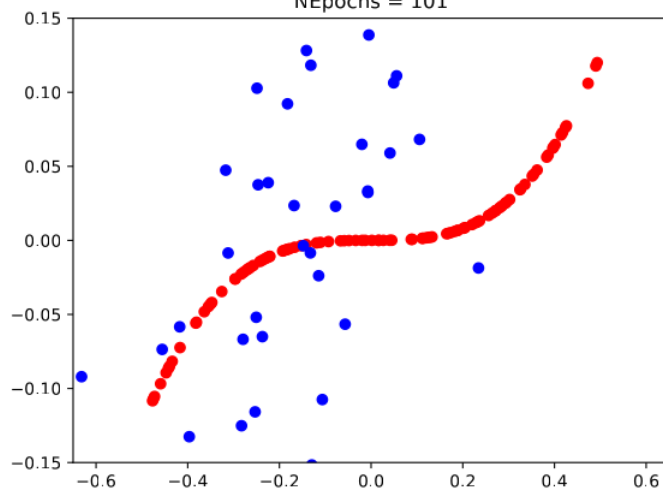
Example: generate x^3 between -0.5 and 0.5



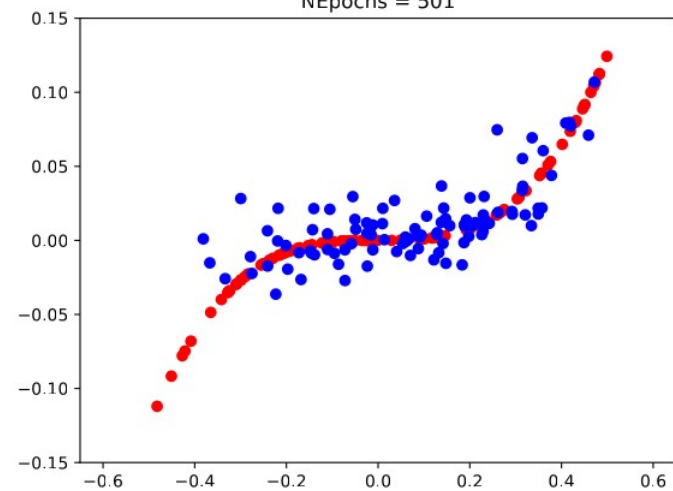
At start of training



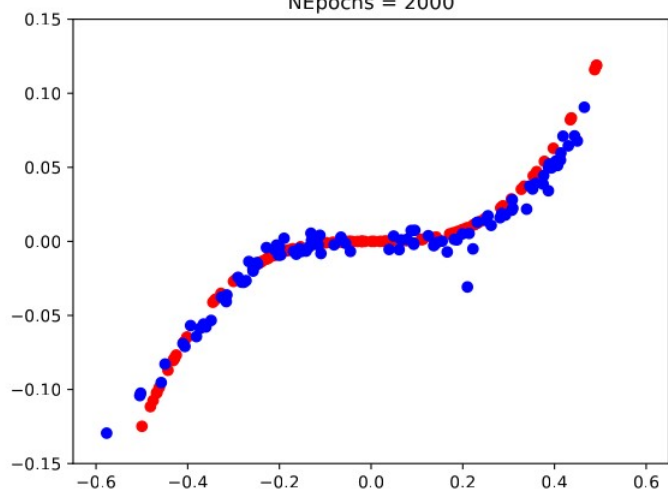
NEpochs = 101



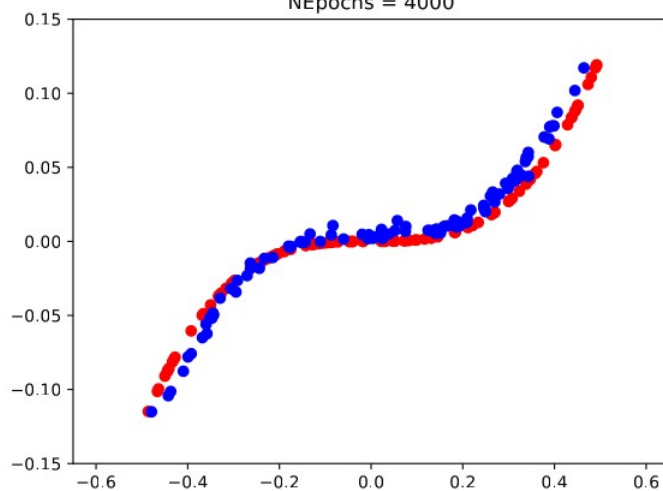
NEpochs = 501



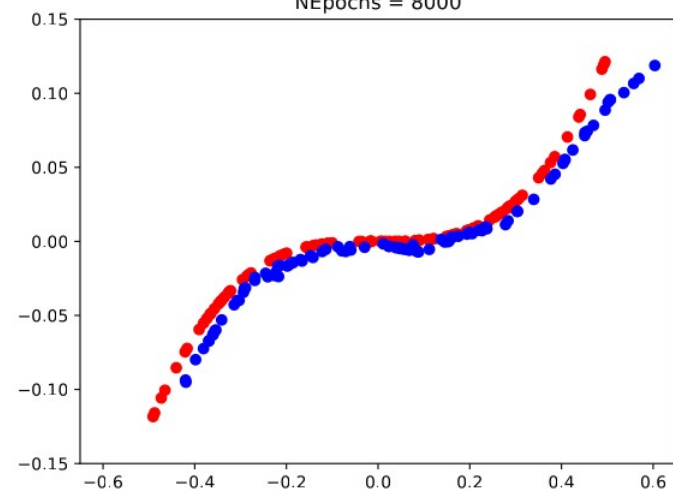
NEpochs = 2000

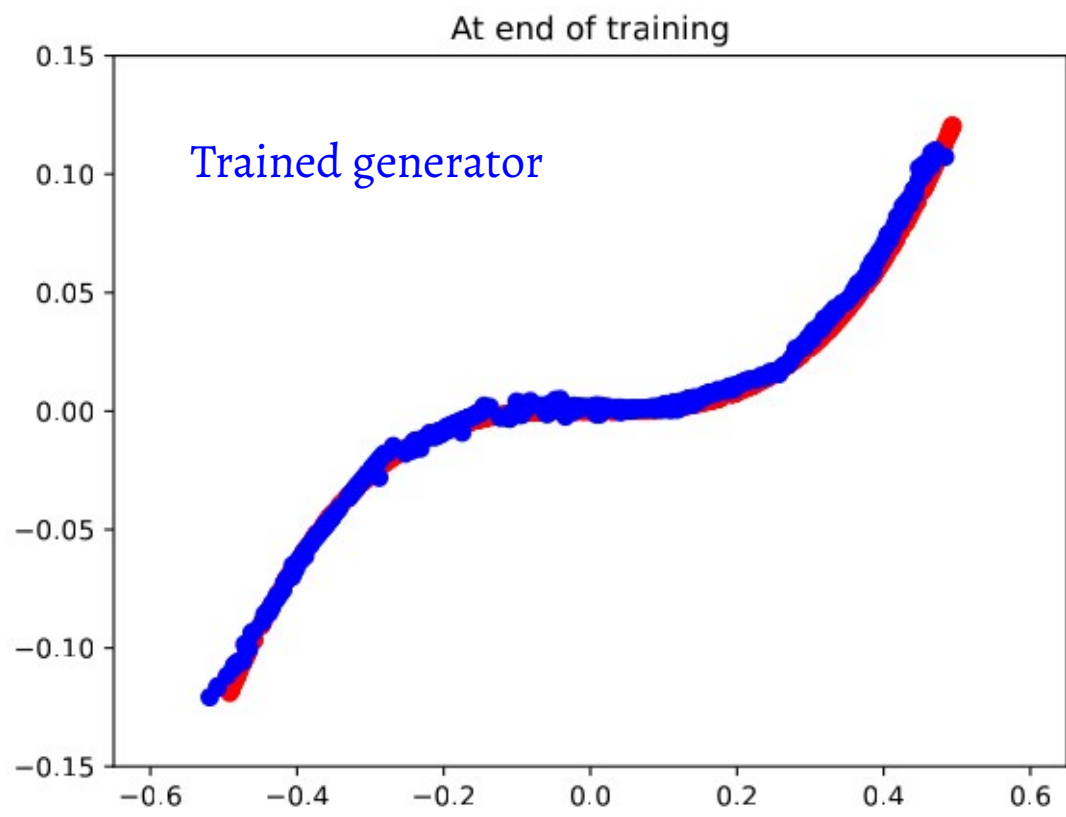
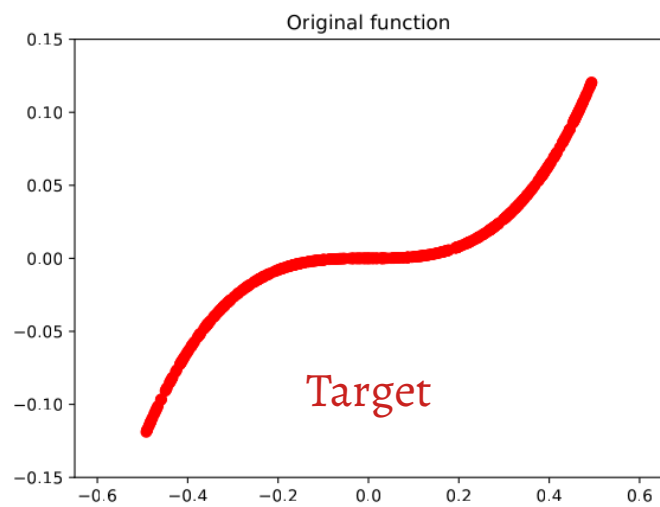


NEpochs = 4000



NEpochs = 8000





Tasks...

1. A simple GAN example is uploaded in the usual place
Either
 - (a) modify the network to see if you get good results in fewer epochs OR
 - (b) Instead of x^3 , try out a different function (such as an ellipse!)
2. Implement the feature importance in the assignments. Drop the worse performing variables, and retrain. Does it make a difference?
3. Does dropout make a difference even if there is no overtraining?

THIS IS YOUR MACHINE LEARNING SYSTEM?

YUP! YOU POUR THE DATA INTO THIS BIG PILE OF LINEAR ALGEBRA, THEN COLLECT THE ANSWERS ON THE OTHER SIDE.

WHAT IF THE ANSWERS ARE WRONG?

JUST STIR THE PILE UNTIL THEY START LOOKING RIGHT.



HOW TO CONFUSE MACHINE LEARNING

