# Detecting the Equation of a Graph Using a Convolutional Neural Network

Brandon Wood and Josias Moukpe

CSE 5290 Artificial Intelligence

Oct 19, 2021 - Update

# Outline

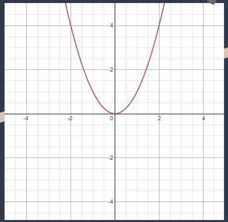
1. Project Scope
2. Achievements
    a. Data source and Input
    b. Network Architecture
    c. Training and Testing Results
3. Working ideas to improve
    a. Data augmentation
    b. More performance metrics
    c. Next design steps
4. References

# Project Scope

$$f(x) = a_0x^0 + a_1x^1 + a_2x^2 + a_3x^3 + \ldots + a_nx^n$$

$$[a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8]$$

CNN Model



- CNN Multiple Regression Model to Detect Equation of Graph from Plot
- First with polynomials (n: integer ≥ 0)
- Starting with 8 coefficients corresponding to 8 degrees.
- Output is vector of coefficients (degree 0 represented by all coefficients = 0)
- Coefficients are integers in [-10, 10] inclusive

# Previously ...

Moving forward we are going to:

- Complete building the dataset
- Build an initial network model
- Build the training loop for the network
- Train the network

# ACHIEVED

# Data Source

```
#create plot
ax.plot(x, (coeff[8]*x**8)+(coeff[7]*x**7)+(coeff[6]*x**6)
        +(coeff[5]*x**5)+(coeff[4]*x**4)+(coeff[3]*x**3)
        +(coeff[2]*x**2)+(coeff[1]*x**1)+(coeff[0]),
        color='#000000')
```

| Filename | coeff1 | coeff2 | coeff3 | coeff4 | coeff5 | coeff6 | coeff7 | coeff8 |
|---|---|---|---|---|---|---|---|---|
| n5n10p3n | -5 | -8 | 5 | -3 | -3 | 3 | -10 | -5 |
| n7p5p4n4 | 9 | 7 | -6 | -1 | -4 | 4 | 5 | -7 |
| p9n3n9p3 | -10 | -3 | 9 | -1 | 3 | -9 | -3 | 9 |
| n7n9p4n6 | -9 | 7 | -5 | 2 | -6 | 4 | -9 | -7 |
| n6n2p3p1 | 9 | -9 | -4 | -3 | 10 | 3 | -2 | -6 |

Python’s Matplotlib library is used to automatically generate polynomial plots.

A script creates randomly generated plots and saves their corresponding filename and coefficients as a label in a csv file.

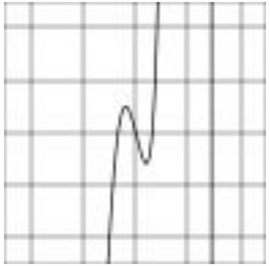
The current generated dataset is split as 85% to train the CNN and 15% for validation and testing.

Data loaders then feed the training set and test set to the CNN.

# Input Data

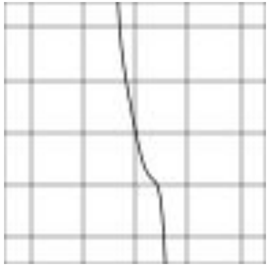
- 2D Single Channel Grayscale Images of polynomial graph plots
- 128×1×96×96 Images Tensor and 128×8 Labels Tensor
- Batchsize = 128
- Fixed standard resolution, dimensions, and centering
- 10 units width for both X-axis and Y-axis of the plot (numbers not displayed)

Current Training Plots:



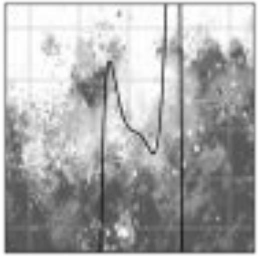
[-4, -1, 7, 1, 6, 6, 0, -1]

[-9, -1, 4, 6, -4, -9, 7, 10]

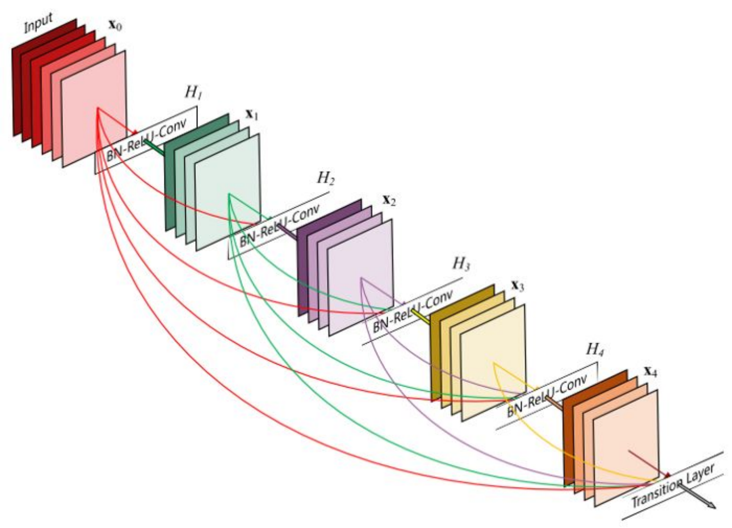


[-5, 2, 4, 0, -7, 7, 4, 0]

Planned Training Plots:



# Network Architecture




Following is our initial network design choices

- Architecture is split into
    - Convolutional Layer Blocks
    - Fully Connected Layer Blocks
- Activation: LeakyReLU (not bounded)
- Loss: Mean Square Error (MSE)

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

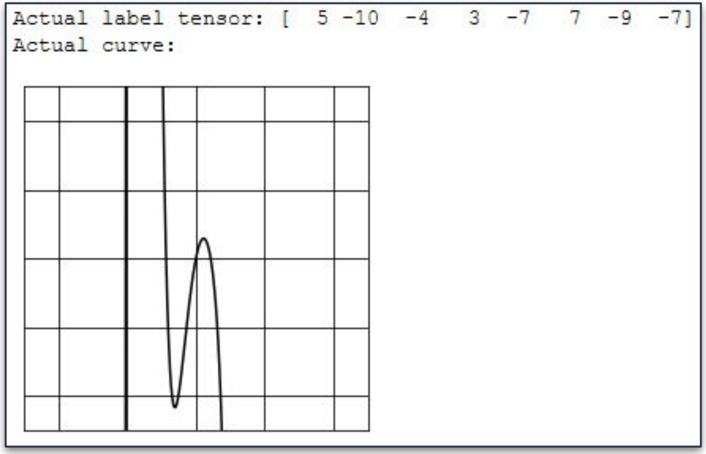
- Optimizer: Adam (.005 Learning Rate)
- Batchsize: 128
- Hyperparameters will be manipulated at training time (includes learning rate, number of epochs, dropout rate, etc)

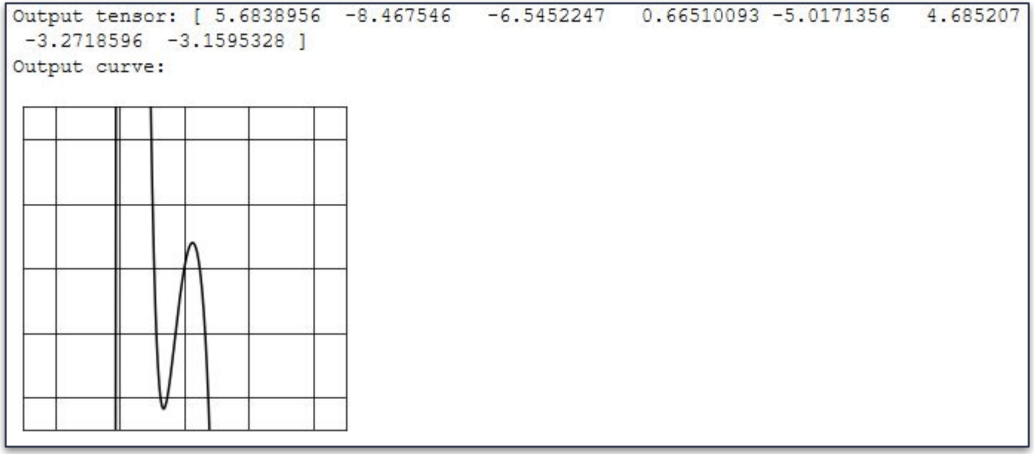
# Network Architecture

```
Network(
 (cnn_layers): Sequential(
  (0): Conv2d(1, 8, kernel_size=(3, 3), stride=(1, 1))
  (1): BatchNorm2d(8, eps=1e-05, momentum=0.1)
  (2): LeakyReLU(negative_slope=0.01)
  (3): MaxPool2d(kernel_size=2, stride=2,)
  (4): Conv2d(8, 16, kernel_size=(3, 3), stride=(1, 1))
  (5): BatchNorm2d(16, eps=1e-05, momentum=0.1,)
  (6): LeakyReLU(negative_slope=0.01)
  (7): MaxPool2d(kernel_size=2, stride=2,)
  (8): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
  (9): BatchNorm2d(32, eps=1e-05, momentum=0.1,)
  (10): LeakyReLU(negative_slope=0.01)
  (11): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
  (12): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
  (13): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
  (14): BatchNorm2d(32, eps=1e-05, momentum=0.1,)
  (15): LeakyReLU(negative_slope=0.01)
  (16): MaxPool2d(kernel_size=2, stride=2, padding=0)
 )
 (fc_layers): Sequential(
  (0): Linear(in_features=1568, out_features=4096, bias=True)
  (1): LeakyReLU(negative_slope=0.01)
  (2): Linear(in_features=4096, out_features=4096, bias=True)
  (3): LeakyReLU(negative_slope=0.01)
  (4): Linear(in_features=4096, out_features=128, bias=True)
  (5): LeakyReLU(negative_slope=0.01)
  (6): Linear(in_features=128, out_features=64, bias=True)
  (7): LeakyReLU(negative_slope=0.01)
  (8): Linear(in_features=64, out_features=8, bias=True)
  (9): LeakyReLU(negative_slope=0.01)
 )
 (criterion): MSELoss()
 (dropout): Dropout(p=0.2, inplace=False)
)
```

# Current Results

- After 1500 epochs of training the CNN, the output is decently close to the actual coefficient values.
- Output curve is created based on the output labels tensor.

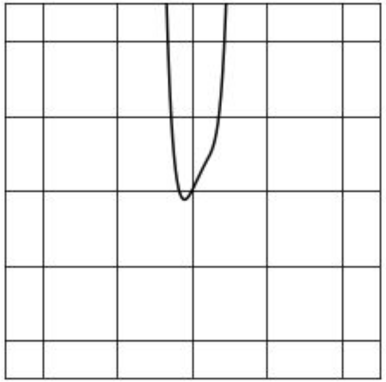


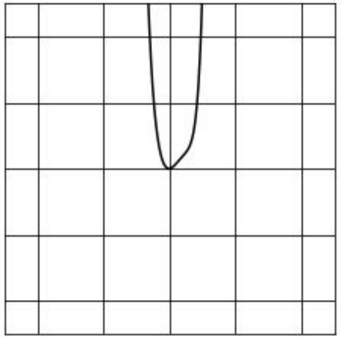
# Current Results

- Additional example at 1500 epochs.
- Output curve is created based on the output labels tensor.

```
Actual label tensor: [ 2  2 -8  7 -1  8 -5  3]
Actual curve:
```



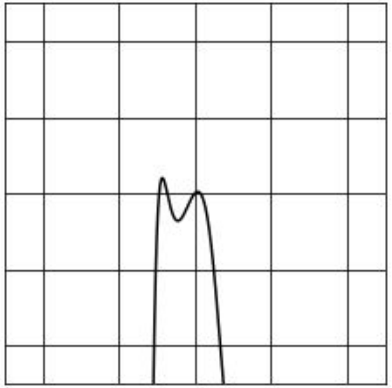
```
Output tensor: [ 0.1622504   4.2711864  -7.430684    3.355888   -0.03072323  5.581914
 -4.868897    4.496127  ]
Output curve:
```



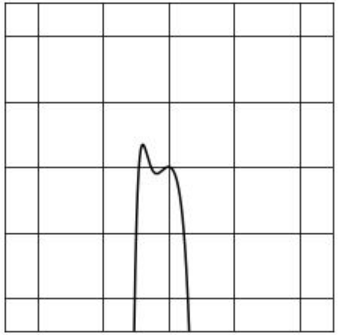
# Current Results

- Additional example at 1500 epochs.
- Output curve is created based on the output labels tensor.

```
Actual label tensor: [  1  -6 -10   0   1   9   5  -7]
Actual curve:
```

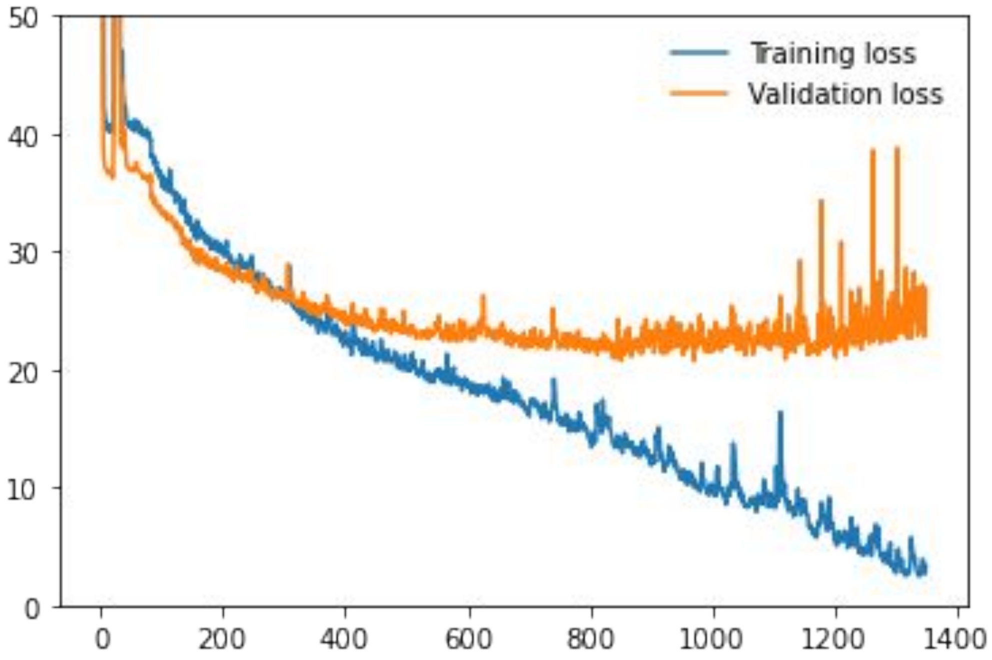


```
Output tensor: [-0.01907877 -4.4737     -6.916153   -0.02837024 -9.947171   -7.3853784
  1.9521291  -5.4963245 ]
Output curve:
```



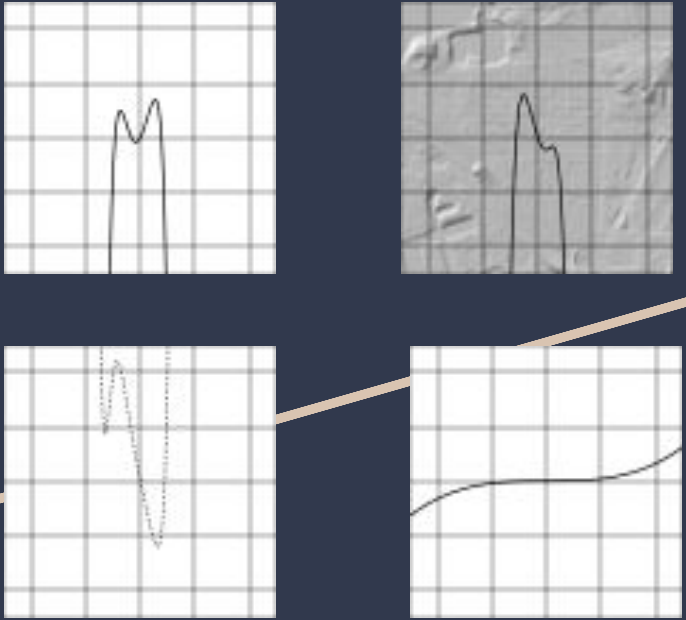
# Current Results

- Training loss and testing loss start very high.
- Training loss gradually approaches 0 as number of epochs increases.
- Testing loss levels at ~25 and even begins to increase towards the end.
- Test loss is too high, so we need to improve the performance of the network



# IN PROGRESS

# Data Augmentation And variability

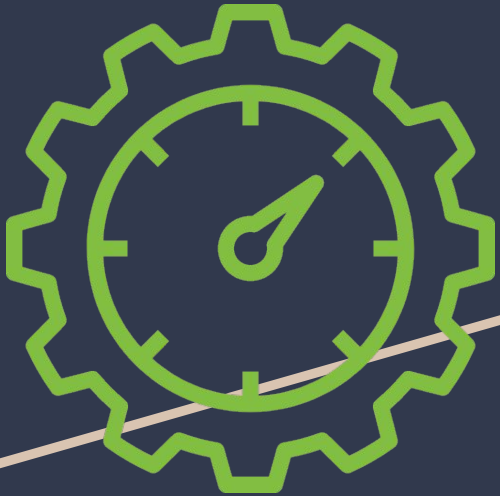


Making our dataset much richer with almost 1 million samples and statistically sound (with no severe bias)

As part of that process, we will:

- Randomize the background with added noise, blur, and occlusion
- Randomize the style, thickness, and hue of the curve
- Randomize the position of the curve on a the x-y plane (variable x and y coordinates)
- Expand to decimal value coefficients
- Possibly randomize the grid line visibility (grid lines might be leveraged by the model)

# Better performance metrics



Additional performance metrics to better understand and evaluate our model

- Find a way to measure accuracy of multiple regression network with some tolerance factor
- Explore statistical measure of error such as standard deviation, p-value, R-squared etc
- Display the layers output to explain what patterns are being picked up by the network.

# Next Design Steps

Moving forward we are going to:

- Incorporate a grid with finer blocks as part of the input pre-processing
- Revise the architecture (continuously)
- Add dropout to the learning process
- Apply cross validation to the training loop
- Building into our new training environment the following

Explore using hyperparameters tuning with Genetic Algorithms.

1. Encode all hyperparameters listed into a string representation "chromosome/gene"
2. Use the new measure of accuracy to gauge the "fitness" of the net
3. Design a function to allow for "crossover" between nets' genes
4. Design a "mutation" function that allows for variations in hyperparameters.

# Thank you!

Any questions?

# References


Matplotlib -https://matplotlib.org/

Train Convolutional Neural Network for Regression - https://www.mathworks.com/help/deeplearning/ug/train-a-convolutional-neural-network-for-regression.html

Neural Network visualization -https://github.com/HarisIqbal88/PlotNeuralNet