Symbolic Regression via Neural Network weights

Barke Saiprasad, Singh Dilpreet

Problem Statement

Can we use neural network weights to represent equations?

Input and Output

Input : Weights of a **neural network** which “represent” an equation.

Output: The equation in the form of character tokens.

For Example: Tokenized form of ax^2+bx+c

[“**(+/-)**”, “**a**”, “*****”, “**x**”, “*****”, “**x**”, “**(+/-)**”, “**b**”, “*****”, “**x**”, “**(+/-)**”, “**c**”]

where a, b, c are further tokenized as “**whole number**”, “**Decimal point**”, “**Fraction**”

Constraining the Problem - Equations

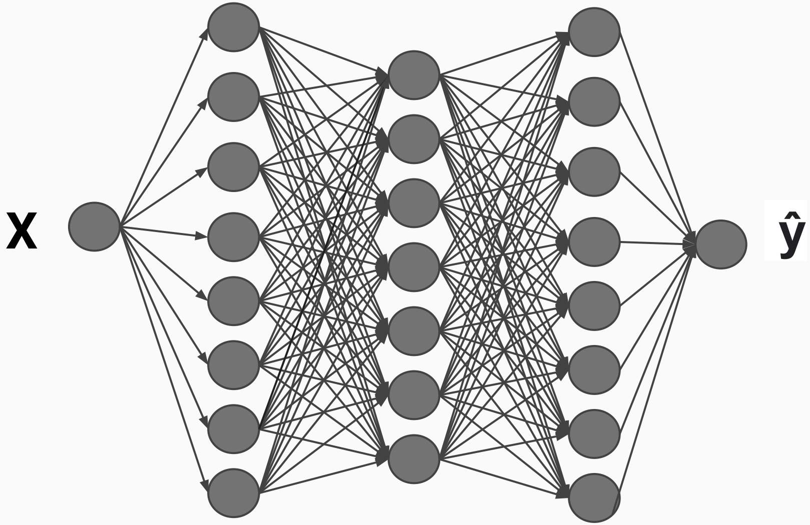
Equations are polynomials up to the **2nd** degree.

$a, b, c \in$ **[-5, 5]**

$x \in$ **[-1, 1]**

How to obtain neural network weights??

From a neural network!



Input: 1
Hidden Layer 1: 8 nodes
Hidden Layer 2: 7 nodes
Hidden Layer 3: 8 nodes
Output : 1

Total Number of Parameters excluding biases: 128

Criterion : MSE loss

Illustration - Phase 1

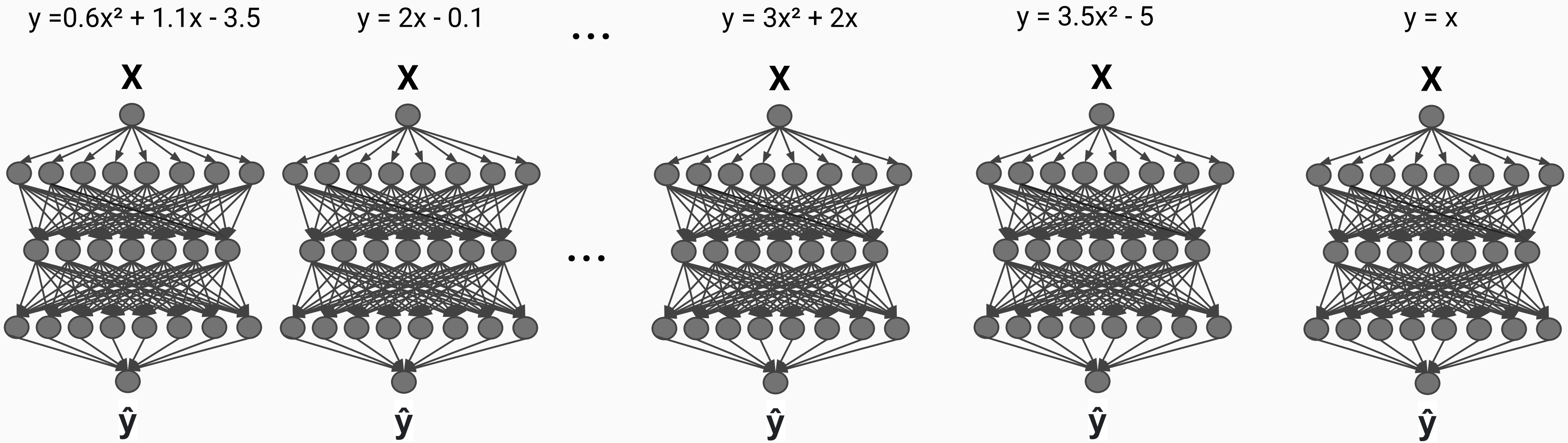


Illustration - Phase 1

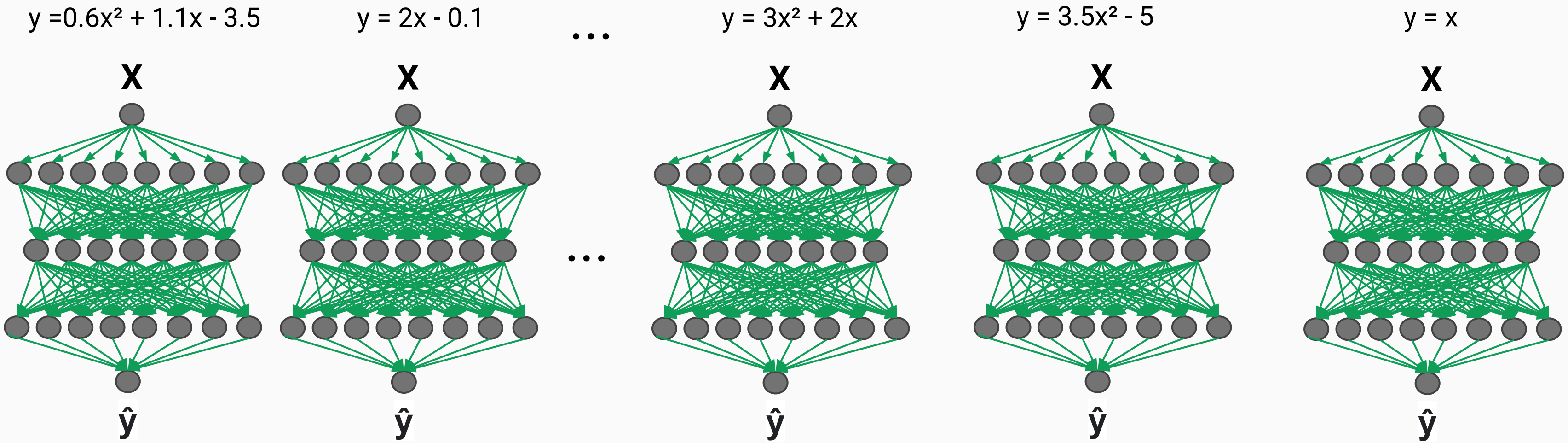
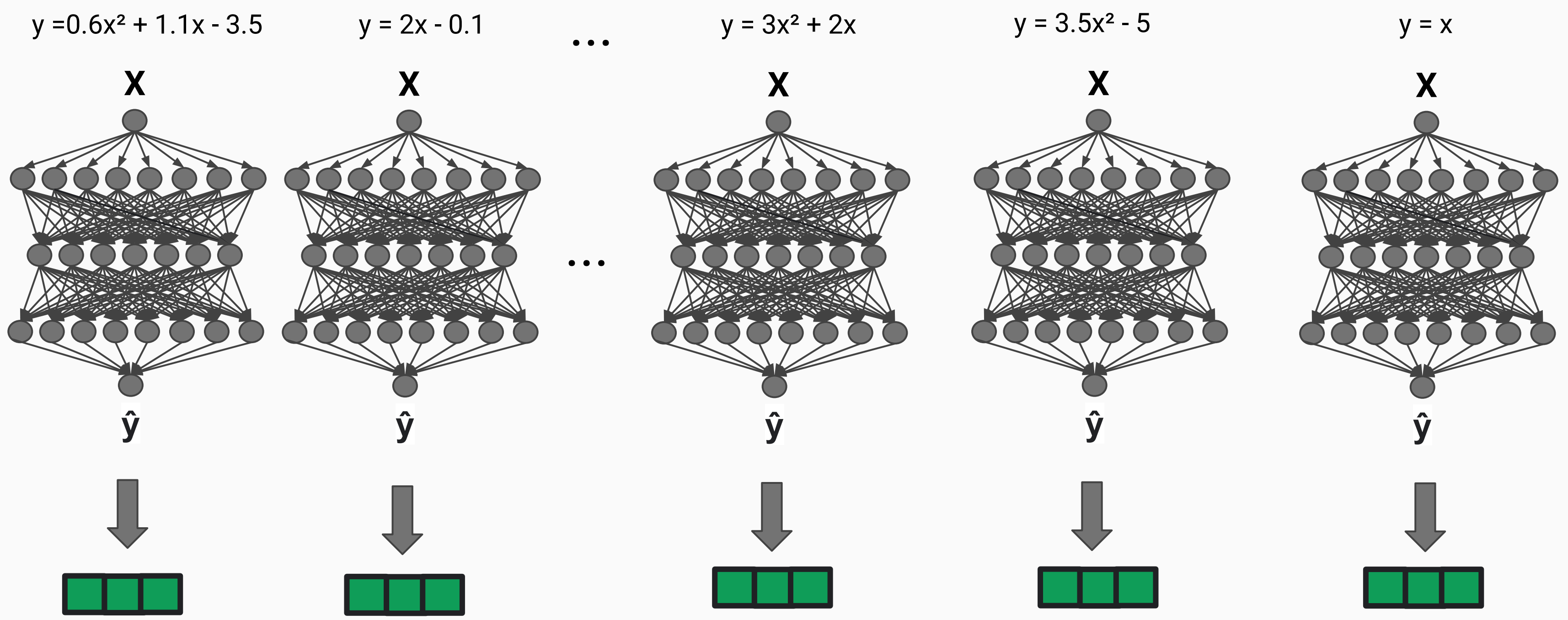


Illustration - Phase 1



Methodology - Phase 1

Step 1: Randomly generate many equations(~100,000) and sample 5,000 (x, y) pairs for each equation

Step 2: Come up with the simplest possible MLP to perform regression on these (x, y) pairs for each generated equation.

Step 3: Train this MLP on the (x, y) pairs for each equation separately with an MSE loss and store the final weights

Illustration - Phase 2

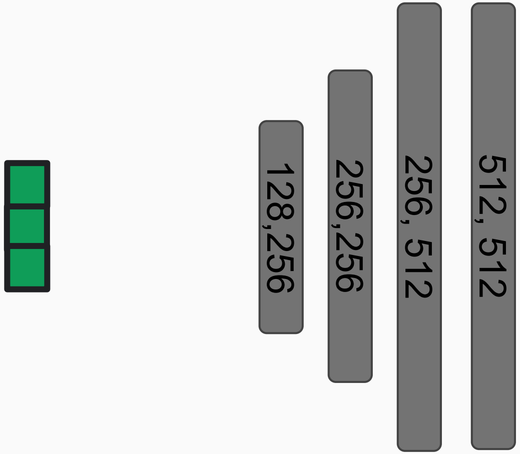


Illustration - Phase 2

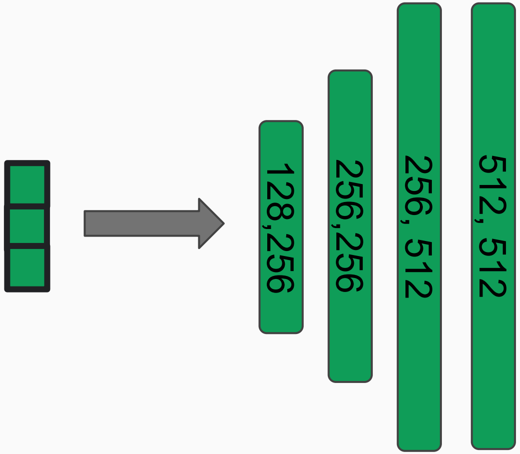


Illustration - Phase 2

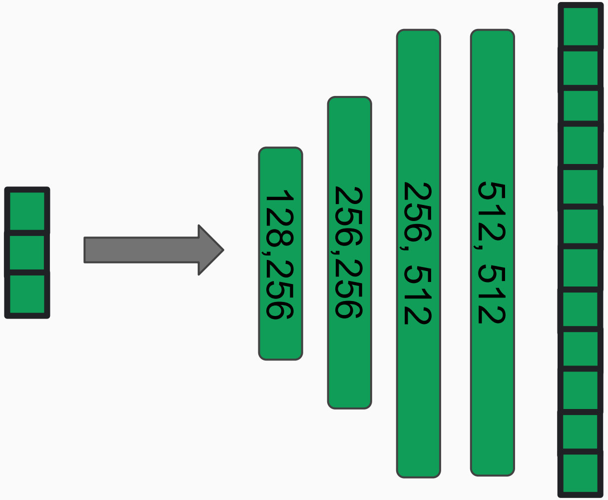
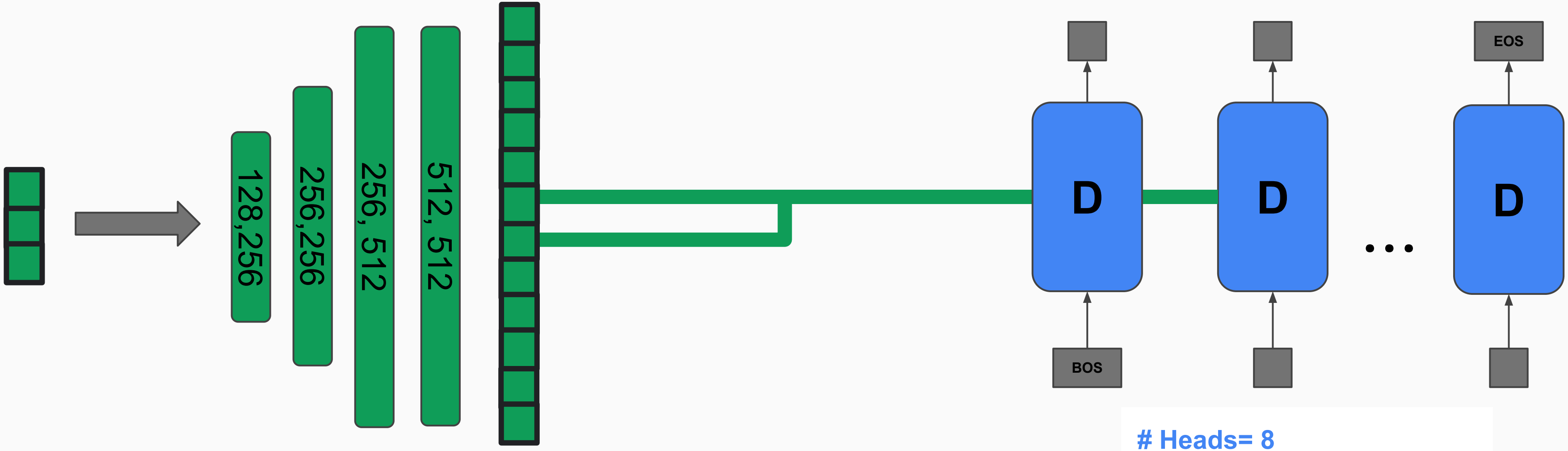


Illustration - Phase 2



 # Heads= 8
Embedding Size = 512
 # Layers = 3-6

Chars = 15 + 4 =19
Max sequence length=20

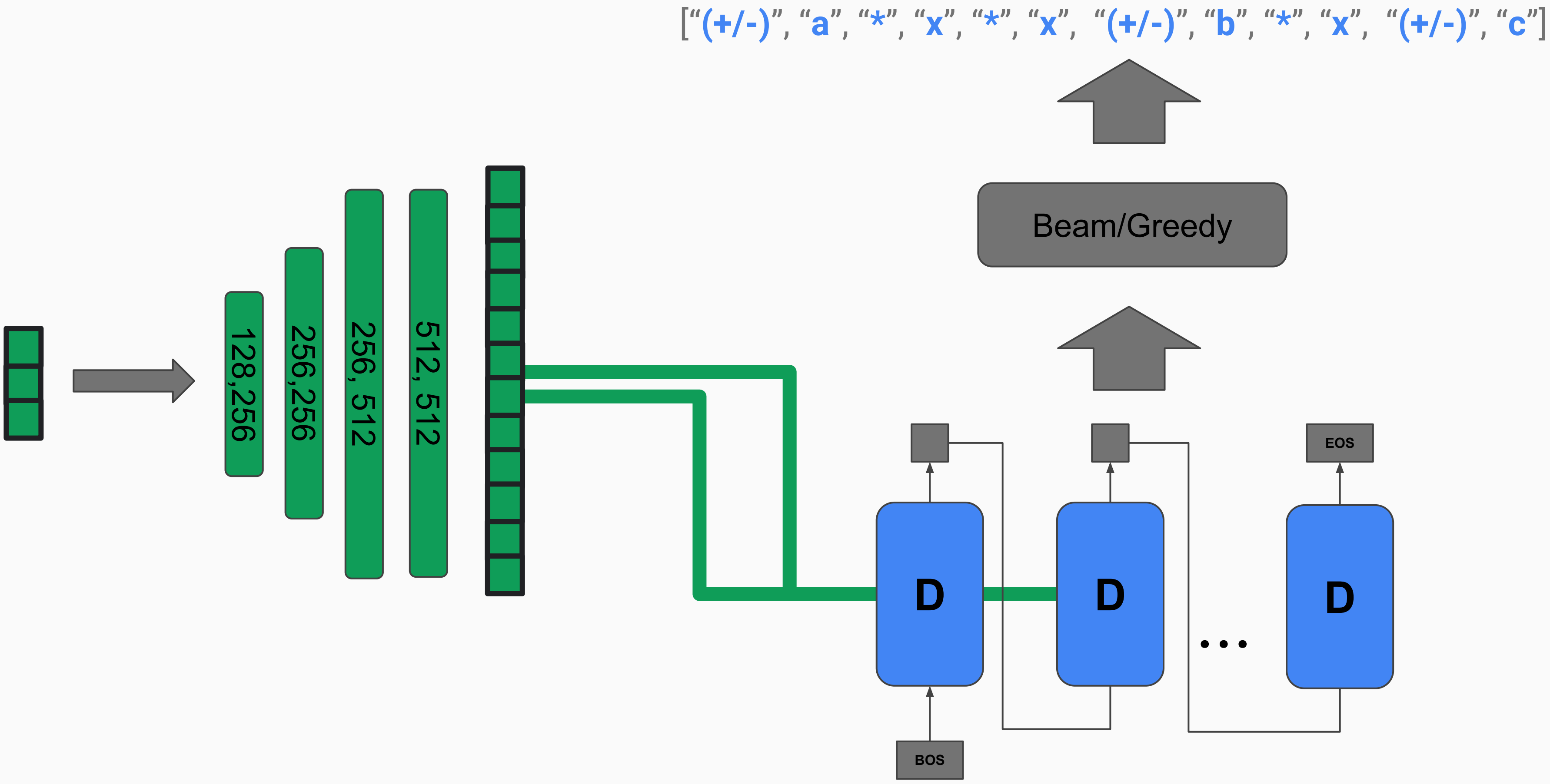
Methodology - Phase 2

Step 1: The weight vectors from the previous phase act as inputs to an MLP which expands the weight vector from size 128 to 512. This new vector is given as memory to the transformer decoder.

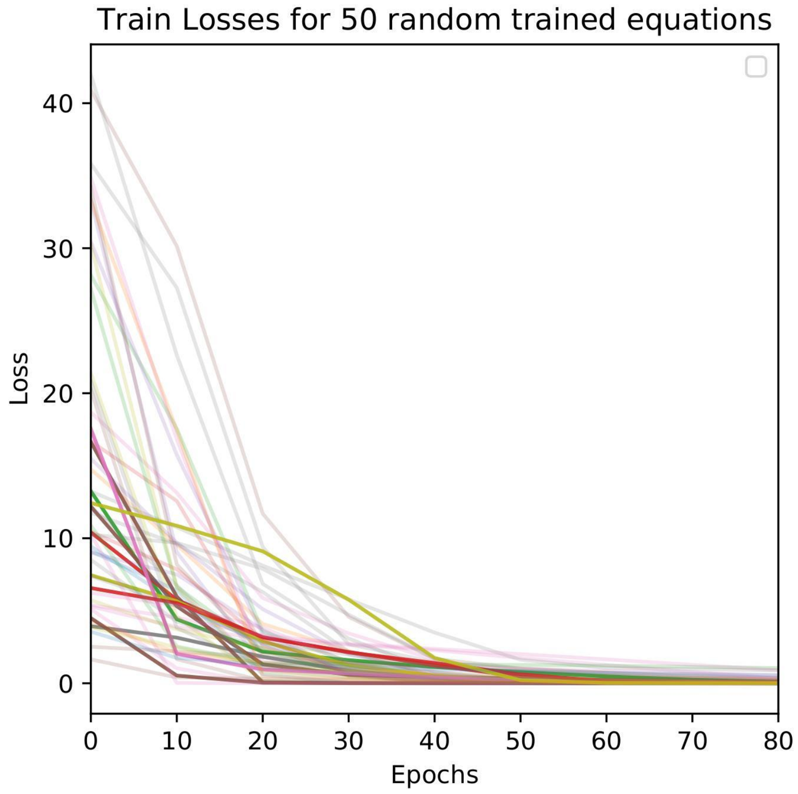
Step 2: The transformer decoder output is given to a generator which outputs a class vector with length equal to vocabulary size which in our case is 19. i.e. Vocab = {0-9, +, *, -, x, .}

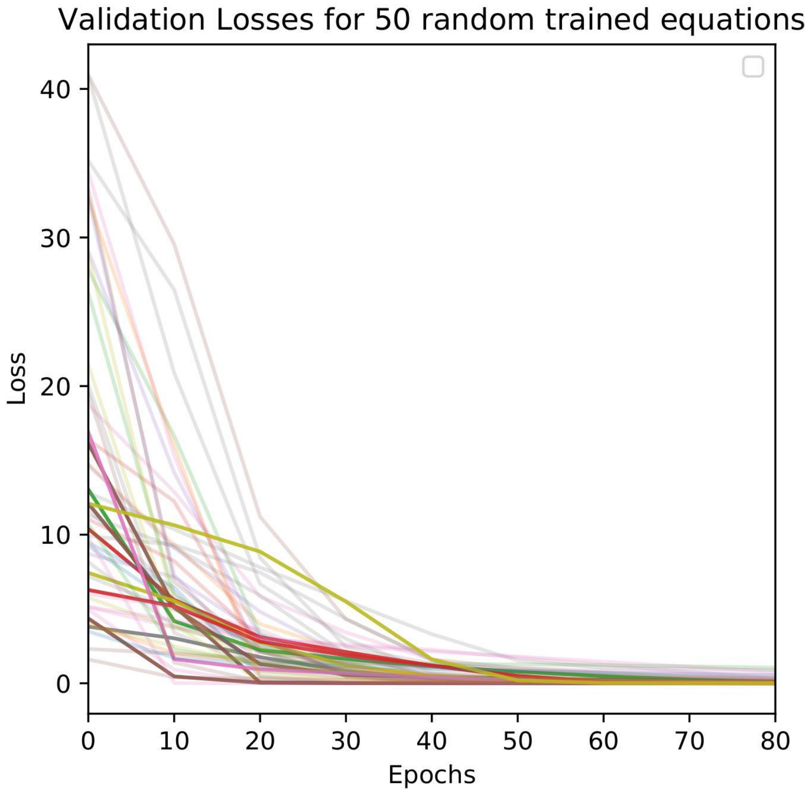
Step 3: We use cross entropy loss during training-validation

Illustration - Inference



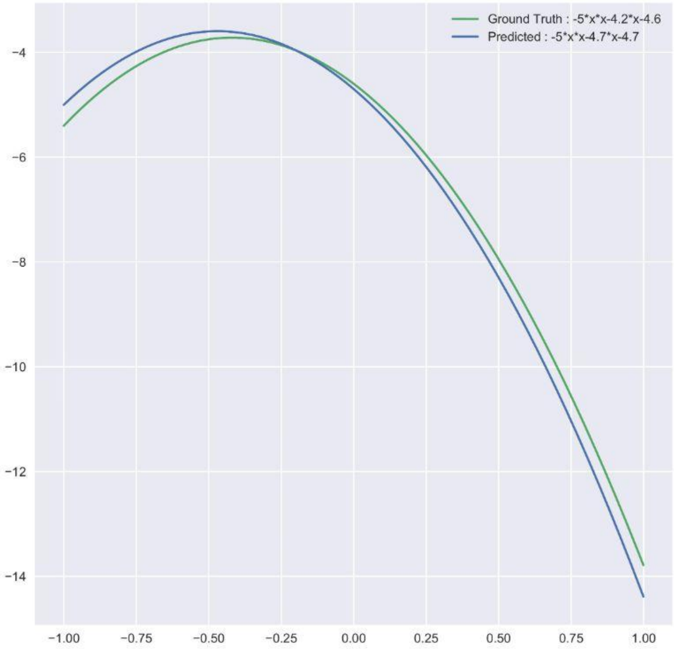
Training Validation Curves - Phase 1

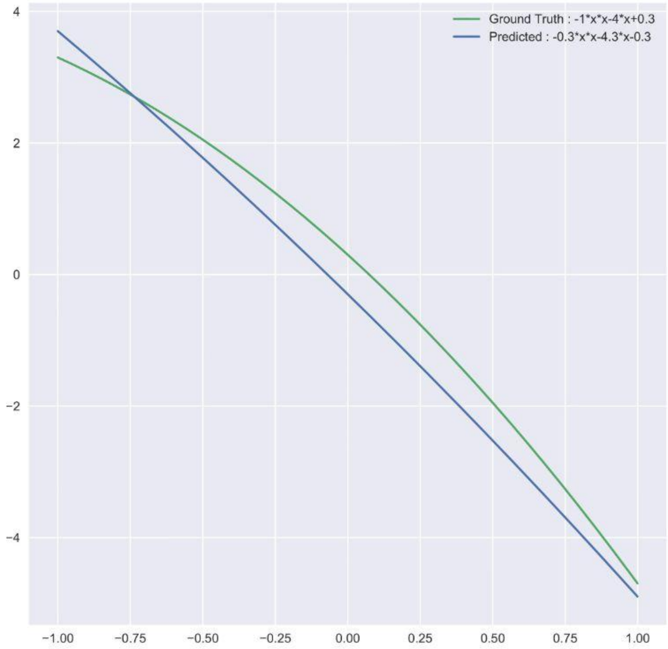




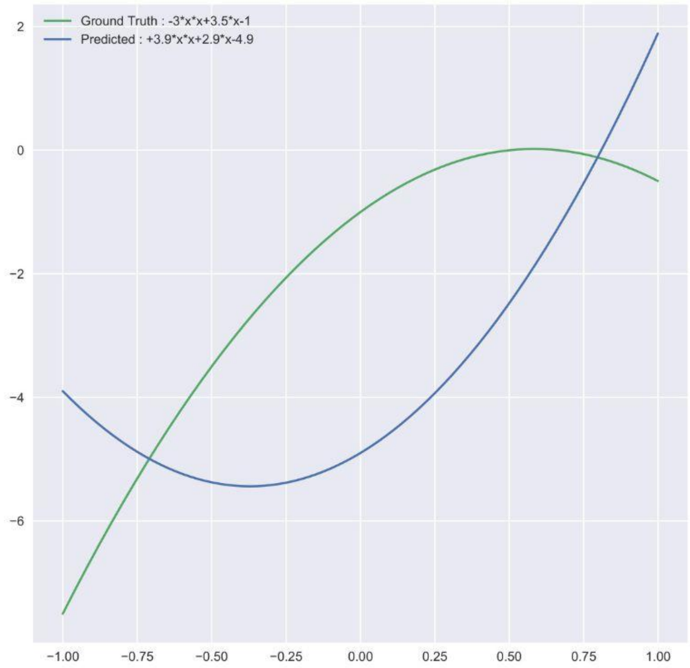
Equation Plots - Ground truth vs Predicted

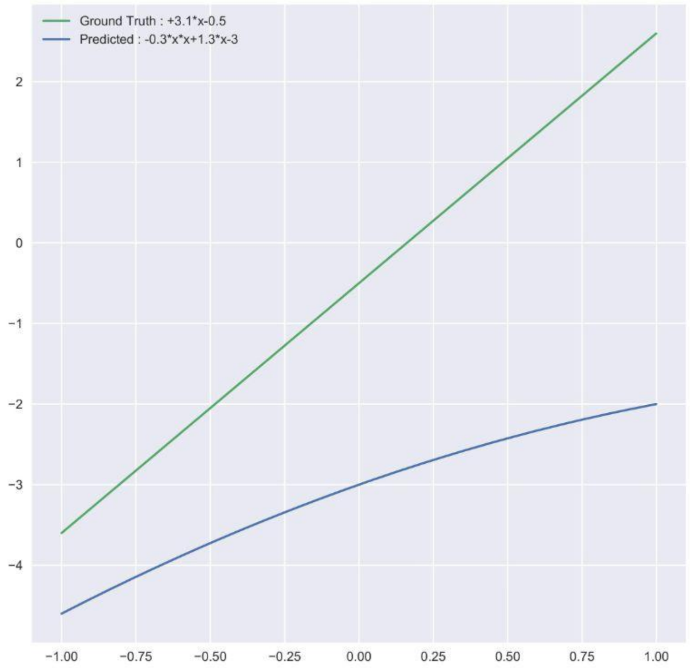
Non-uniform Dataset with sequences lengths of 6, 8, 10, 12, 14, 16 characters



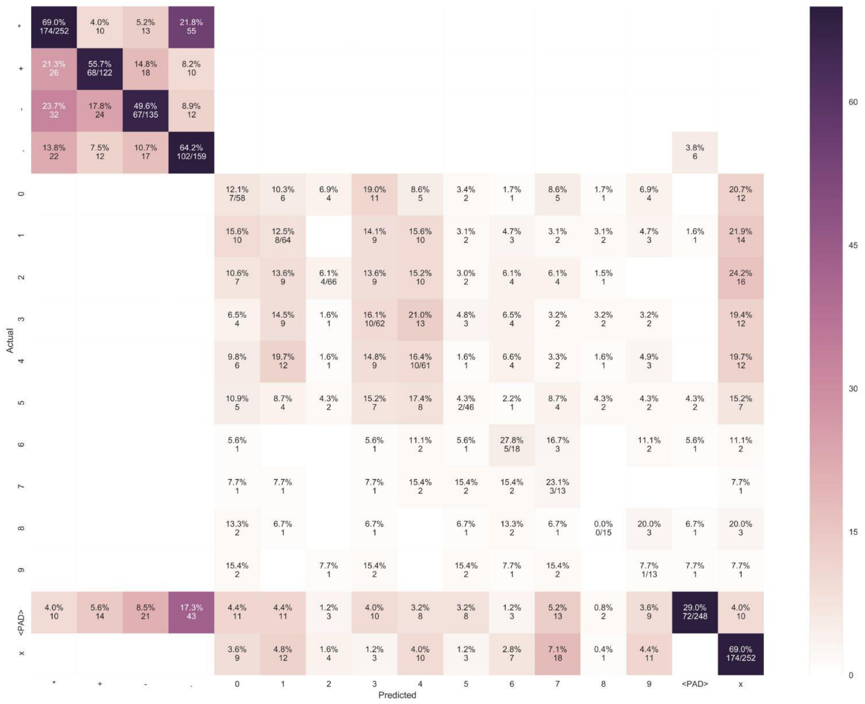


Equation Plots - Ground truth vs Predicted





So how well did the pipeline perform?



Observations and Inferences

1. The model fails to learn the **exact numeric values** of coefficients of the equation.
2. In certain cases, as shown above it predicts numerical values **close to the ground truth**.
3. The model correctly learns **the positions** of each of the individual “types” of tokens relative to each other.
4. In most cases, the model also **predicts the sign** of the numerical values **correctly** as seen above.
5. The 100k dataset is heavily skewed towards sequences with higher number of tokens i.e., 14, 16. This causes an **inherent bias** in the network towards longer sequence lengths and results in incorrect predictions for targets with shorter length.
6. The solution to all the above problems is **More uniform data and more data**

Future Scope

1. The equation weights in a particular order do not mean anything. Augment the dataset by manufacturing more data points by permuting the elements of the weight vectors OR using positional encoding before feeding the weight vector into the feature expanding MLP.
2. Since x lies in [-1, 1] and equation coefficients lie in [-5, 5], it is possible that in certain (x, y) pairs the c term dominates because it's not being multiplied by a value less than 1. Retrying the exercise with larger values of x and coefficients might give further insights.