Optimizing Input Sequences for Maximum Output in a Multi-Layer Perceptron Neural Network.

Md Rakibul Ahasan
Miami University

***Abstract*—In this project, we explore the capabilities of a pre-trained Multi-Layer Perceptron (MLP) model implemented in PyTorch. The model parameters are applied to a neural network (NN) consisting of 10 sequential layers. Each layer is followed by a ReLU activation function, except the final layer. We analyze the architecture of the NN, including its input and output constraints, and apply different optimization process that gives the maximum linear sum of the output. The input sequence has two scenarios, In the first scenario the NN layer input is bound to -10 to 10 and in the second scenario the linear sum of the NN layer input is bound to -10 to 10. In summary, it is a constraint satisfaction problem that produces input sequences and ensures maximum linear sum output for both scenarios.**

***Index Terms*—Input Sequence, Optimizer, and MLP.**

I. Introduction

The most common type of neural network is a multi-layer perception (MLP) network where the signal traverses in the direction from the input to the output [1]. This MLP network consists of the input layer, the output layer, and the hidden layers. However, MLP does not offer computational power over a single-layer network without a non-linear activation function. The activation function is tagged with neurons placed in the different layers of the network and it is useful to cater to the non-linear relationship of the input. The most used activation function is the rectified linear unit (ReLU) that produces the maximum from a given set [2]. MLP with activation functions is considered a deep neural network focused on either execution speed or application usability. PyTorch is a machine learning library that focuses on merging speed and usability and supports a Python programming style converting the code as a model and easier debugging [3]. This project includes a pre-trained MLP network using the ReLU activation function along with the PyTorch machine learning library to produce a set of constrained input sequences and populate the maximum linear sum output. A PyTorch model file is the prerequisite of this project and detailed instructions are:

1) Load the model into Python using ”torch. load”. Then, apply the model parameters (via ”load_stat_dict”) to an NN with 10 sequential layers, of appropriate size, with a ReLU activation function between each layer (no ReLU on the last layer).
2) Tell us about the NN. Inputs, outputs, and architecture/etc. What does it do?
3) Assume all inputs can range from -10 to 10. What input sequence maximizes the linear sum of all outputs? What's the output?
4) Assume the linear sum of all inputs can range from -10 to 10. What input sequence maximizes the linear sum of all outputs? What's the output?

II. Methodology

To achieve the outcome of this project there are four steps. In the initial steps, we will deep dive into the parameter details of the provided MLP model. These include the number of layers, the input and output size of each layer, and the gradient status of the parameters in the layers. In this next step, we will create a neural network model to support the provided model architecture. In terms of the activation functions, we are putting the ReLU activation function in each layer, except the last layer. Fig 1 has these details.

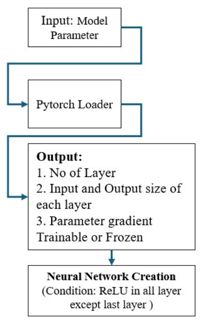
Input: Model Parameter
Pytorch Loader
Output:
1. No of Layer
2. Input and Output size of each layer
3. Parameter gradient Trainable or Frozen
Neural Network Creation
(Condition: ReLU in all layer except last layer)

Fig. 1. Neural network creation from the provided model.

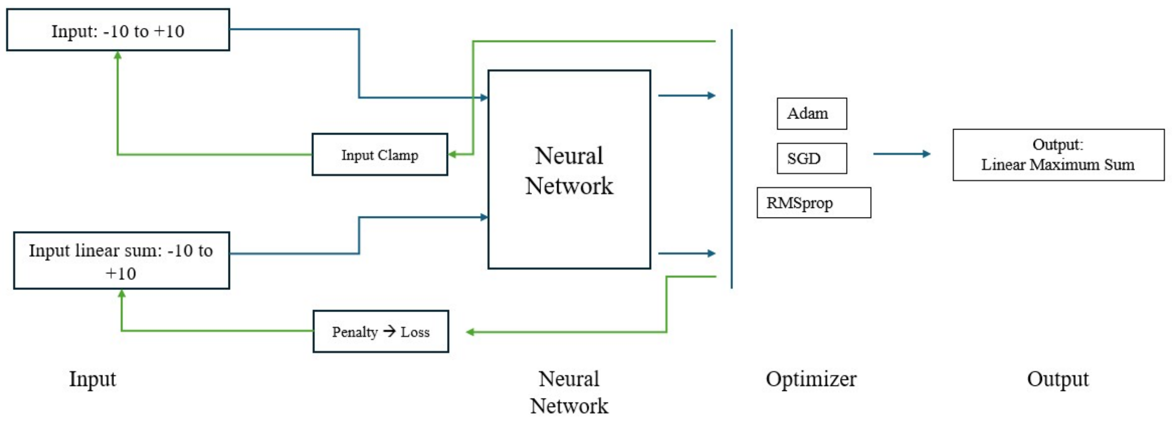

Fig. 2. Output maximization for different optimization technique

Once the neural network is created we pass two input types through the neural network and the optimizer. The optimizer helps to identify the input sequence that gives the maximum linear sum of the outputs for a given number of iterations. In the first input type, each input is bound to -10 to 10. During the optimization process, we introduced a clamping step, that ensures the input is bound to the -10 to 10. In the second input type, the linear sum of all input is bound to the -10 to 10. During this optimization process, we introduced a penalty to the loss so that the linear sum of the input is bound the -10 to 10. Fig 2 has these details. In the end, The output for both input types should produce the maximum linear sum. The bounded condition for both input and output types tells us that it is a constraint satisfaction problem where the objective is to maximize the output linear sum, subject to different input types.

III. Result Discussion and Findings.

In this section, we are discussing the project outcomes. The first outcome is to understand the provided model parameter and load it into a 10-layer neural network. We have used the 'torch.load' command to load the parameter and a for loop in the parameter items to visualize the input/output size of each layer and the gradient status of the parameters. We got a 10-layer neural network, with an input size of 299 neurons, an output size of 68 neurons, and a hidden layer size of 100 neurons each. Moreover, the weight and bias parameters are not trainable, all are frozen. That means during the input optimization process we can not update the weight and bias.

The second outcome, The architecture is a feed-forward neural network, that appears to identify the input data sequence for the maximum linear sum of the outputs. The architecture of the neural network is as follows:

1) An input layer with 299 neurons followed by a ReLU activation function.
2) Eight hidden layers, each with 100 neurons followed by a ReLU activation function.
3) An output layer with 68 neurons.

In the third outcome, we consider three optimizers to optimize the input sequence. They are adaptive moment estimation (Adam), root mean square propagation (RMSprop), and stochastic gradient descent (SGD). In SGD updates input by a fraction of the gradient of the loss function concerning the input with a constant learning rate, the RMSprop considers the moving average of the squared gradients for each input and divides the gradient by the square root of this average to adjust the learning rate and the Adam optimizer considers a moving average of both the gradients (momentum) and the squared gradients (similar to RMSprop) for each input with an adaptive learning rate [4].

The above Fig 3 gives a visualization of our third outcome. The x-axis represents the number of iterations and the y-axis represents the maximum linear sum of the output. Our goal is to find the input sequence that provides the maximum linear sum output after a given number of iterations. It shows SGD optimizer performs better than the RMSprop, and Adam optimizer and reaches convergence in 50 iterations. The findings are:

1) We started the learning rate from .01 to .1. For the learning rate of .01 the Adam and the RMSprop optimizer output are not close to the SGD optimizer, which reason probably because SGD has a favorable learning rate value. However, when we update the learning rate to the .1 value all three optimizers' maximum output is closer.
2) We considered a timing analysis for the optimizers that provide the fastest and converged maximum linear sum of the outputs. The SGD

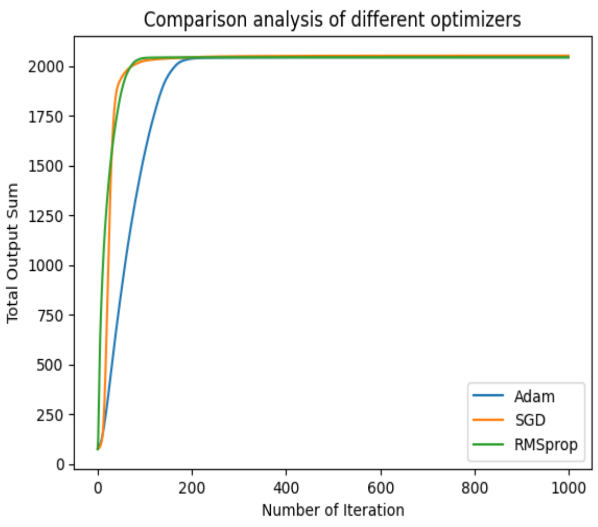

Fig. 3. Comparison of different optimizers for input -10 to 10

optimizer took 1.44 s, The RMSprop optimizer took 1.58 s, and the Adam optimizer took 3.09 s.

3) We configured SGD with a momentum of 0.9, which helps accelerate the convergence in the right direction.

In summary, We are getting the fastest convergence using the SGD optimizer, and the maximum linear sum of output is 2044.54. The corresponding input sequence is available in the shared ipynb file.

The last outcome corresponds to finding the maximum linear sum of output whose input linear sum is bound to the -10 to +10. In this optimization process, we are using the Adam optimizer.

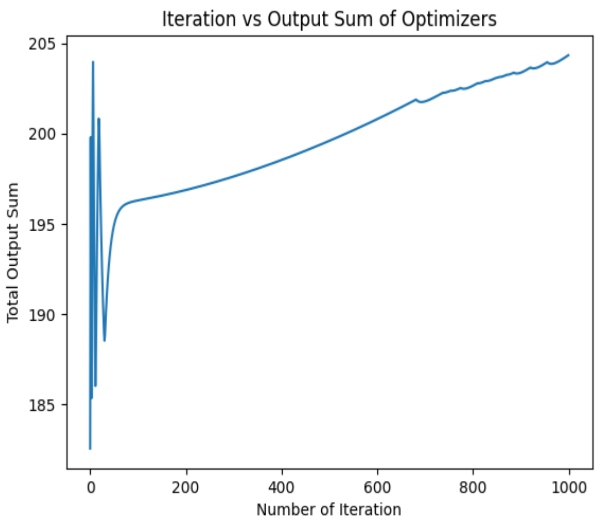

Fig. 4. Input linear sum -10 to 10, Optimizer: Adam

The above Fig 4 shows the maximum linear sum output obtained from the constrained input. Here we use the learning rate .1 which leads to the maximum linear sum output of 204.27. The corresponding input sequence is available in the shared ipynb file. The findings are:

1) We use Adam optimizer for the optimization process because it adjusts the learning rate based on the first and second moments of the gradients.
2) The Adam optimizers incorporate the concept of momentum by using a moving average of the gradients.

These help to generate a faster convergence time of 1.54 s for the maximum linear sum output.

IV. Conclusion and Future Work

This project broadly signifies a constraint satisfaction problem where the input and output are constrained. The inputs are constrained to -10 to 10 or the inputs linear sum is constrained to -10 to 10. In both input scenarios, the outputs have to produce the maximum linear sum output. This input and output are part of an MLP network where each layer weight and bias are not trainable. In that case, we imposed an optimization process on the inputs and tried to find the input sequence that produced the maximum output. We look for the fastest convergence time of the optimization process for a given iteration that satisfies both input and output constraints. We concluded the SGD optimization process that satisfies the input sequence from -10 to 10 and the maximum linear sum output. The Adam optimization process satisfies the linear sum of the input sequence from -10 to 10 and the maximum linear sum output.

In future work, we will explore a more robust optimization framework. Reinforcement learning (RL) is a type of machine learning that takes insight from the environment, imposes appropriate policy action, and custom reward mechanisms to find the optimal solution. Additionally, reinforcement learning (RL) does not require external input, making it interesting to observe how an RL-dependent policy network meets both input and output constraints.

References

[1] Marius-Constantin Popescu, Valentina E. Balas, Liliana Perescu-Popescu, and Nikos Mastorakis. 2009. Multilayer perceptron and neural networks. WSEAS Trans. Cir. and Sys. 8, 7 (July 2009), 579–588.
[2] Agarap, A.F. (2018). Deep Learning using Rectified Linear Units (ReLU). ArXiv, abs/1803.08375.
[3] Adam Paszke et. al. 2019. PyTorch: an imperative style, high-performance deep learning library. Proceedings of the 33rd International Conference on Neural Information Processing Systems. Curran Associates Inc., Red Hook, NY, USA, Article 721, 8026–8037.
[4] Abdolrasol, M.G.M.; Hussain, S.M.S.; Ustun, T.S.; Sarker, M.R.; Hannan, M.A.; Mohamed, R.; Ali, J.A.; Mekhilef, S.; Milad, A. Artificial Neural Networks Based Optimization Techniques: A Review. Electronics 2021, 10, 2689. https://doi.org/10.3390/electronics10212689.