EE-559 Deep Learing Project 2

Damien Gengler, Vincent Yuan

May 28, 2021

1 Introduction

The goal of this project is to design a mini "deep learning framework" using only pytorch's tensor operations and the standard math library.

Using a data set that we generated ourselves with corresponding labels, we should use the framework to create a simple model. The model is composed of 2 input, 1 output and 3 hidden layers of 25 nodes each and is using SGD optimizer in order to learn how to classify data to solve our problem.

In this report, we'll go through the different aspect of our framework to explain what we did.

2 Implementation

2.1 A high level representation: the Module class

As suggested in the specifications, we decided to implement a class Module which is a high-level representation of a module.

It has 3 functions:

- **forward** : returns the data after it passes the module
- **backward** : returns the derivative of the data
- **param** : returns a list of pairs of tensors representing a parameters and its corresponding gradients.

All our modules extend this class and implement override its functions.

2.1.1 Linear layer

We created our linear layer module by first instantiating the weights, the biases Tensors according to the input and ouput sizes and filling them with values following a normal distribution with a given mean and standard deviation. The derivatives are also initialized as zeros tensors.

During forward pass we multiply the matrix weight matrix with the input and add the bias and output it for the next layer. We also keep the input value in order to use it to compute the derivatives during the backward pass.

During backward pass we compute the derivatives for the weights and the biases with the ouput of the backward pass of following layer.

Params returns the list of tuple of the weights and its derivatives and of bias and its derivative.

2.1.2 Activation functions

We implemented three activation functions. The requested ReLU, Tanh and as a bonus the Sigmoid. All the functions forward pass apply the function to the input and return it. During the backward pass we simply compute the derivative of the functions evaluated for the input. They don't have any parameters.

2.1.3 Loss functions

We implemented 3 loss function: Mean Squared Error (MSE), Mean Absolute Error (MAE), and Binary Cross Entropy (BCE). The three of them compute the loss of the output w.r.t the target values in their forward function and and compute the derivative, that is then fed to the network's backward pass in their backwards functions.

2.1.4 Sequentials

We have previously created different modules that will be used to create our network. We now need to combine them to create our neural network model.

The Sequential class enables the possibility of combining different modules sequentially, thus creating a neural network model abstraction. It consists in a list of module. During the forward pass the we apply the forward function of the first module on the input and then feed the output to the next module until the end of the list which is the end of the network. During the backward pass we do the same but call backward in a reverse order with first the derivative of the loss function.

The param function returns the list of all the parameters of the layers of the model.

2.2 Data generation

To generate our dataset, we have sampled points in the space [0, 1] X [0, 1] from an uniform distribution. Then, we have label them depending if they are in the circled centered in (0.5, 0.5) of radius $\frac{1}{\sqrt{2\pi}}$.

We have created helper functions to split our dataset between train and test, or train and validation according to if we wanted to hypertune our SGD learning rate.

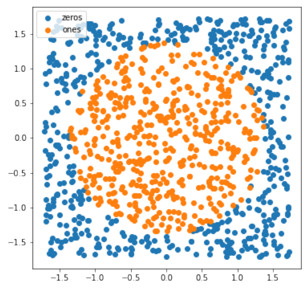

Figure 1: Plot of generated data

2.3 Optimizers

We decided to implement our update step in the form of an Optimizer. They have two functions : Step, that performs a parameter update w.r.t the gradient computed. And $zero_grad$ that sets the gradients of the parameters to zero.

The training loop runs for a given number of epoch. For each epoch we go through all the samples in the training set and do the following :

- Zero out the gradient of the parameters
- Forward pass the sample in the model.
- Evaluate the loss of the output with respect to the target.
- Compute the derivative of the loss function.
- Feed the derivative of the loss to the backward pass to compute the parameters derivatives
- update the parameters by calling optimizer.step()

2.3.1 Stochastic Gradient Descent Optimizer

The SGD Module is initialized with a list of the Model's parameters and a learning rate γ. The step functions here simply performs the SGD update step for each parameter : $\omega := \omega - \gamma d\omega$

2.3.2 Adam Optimizer

Adam is a very popular optimizer that makes use of the average of the second moment of the gradient to adapt the parameter learning rates in a Neural Network. As it is quite efficient and popular we decided to implement it alongside SGD. The Adam module is initialized with a list of the Model's parameters and a learning rate γ and takes three other parameters β_1, β_2 and ϵ (that are almost never changed from their initial values. During initialization it creates a list of values needed to store the moments V_{dp} and S_{dp} (for a given parameter p). We then compute the corrected value of these values $V_{dp}^{corr} = \frac{V_{dp}}{1-\beta_1^t}$ and $S_{dp}^{corr} = \frac{S_{dp}}{1-\beta_2^t}$ and use these to compute the update of the parameter $p := p - \gamma \frac{V_{dp}^{corr}}{\sqrt{S_{dp}^{corr}}+\epsilon}$

3 Results

Using the different modules presented here, we created the model consisting of 2 input, 1 output and three hidden layers of 25 units. We use ReLU as activation functions, MSE as Loss function and SGD as an optimizer. We generate 1000 training datapoints and 1000 test datapoints and evaluate our model on this. Although only the SGD model was asked for we also decided to evaluate using Adam to compare.

Our SGD model was trained with learning rate 5e-5, 500 epochs and MSE loss. Our Adam model was tuned with a learning rate of 5e-3, 100 epochs and MSE Loss.

After training our models on the train dataset, we have assessed their performance on the test set. Our 2 models had very good results, and we had an accuracy of 97.7% with the SGD optimizer and an accuracy of 98.3% with the Adam optimizer. Although both of them give very good results, the Model using Adam optimizer has the advantage to require far less epochs and provide a better accuracy than .

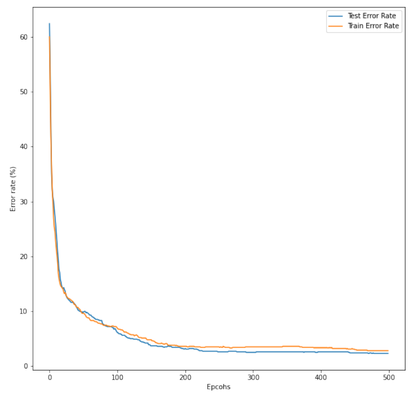

Figure 2: Test and Train loss for SGD over the iterations

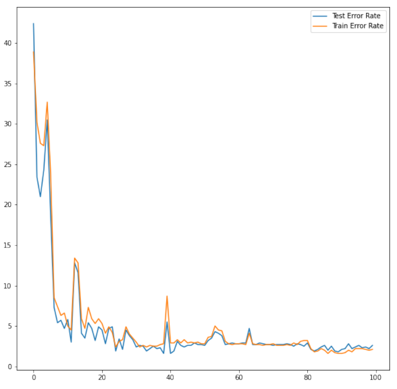

Figure 3: Test and Train loss for Adam over the iterations

4 Conclusion

Our 2 model performed very well on the task presented. The power of our deep learning framework allow us to have a very strong non linear classification power. Even though it is quite small, our framework is already able to reach really good performances on classification task. It is also quite flexible. The addition of Adam allows us to have an even more efficient framework, training faster than SGD.

This project allowed us to take a deep look of what's under the hood of pytorch's machinery. This allowed us to get a better grasp of what is happening when we train a deep neural network using the framework. It was very pleasant to use the powerful tools ato ur disposal to build our own framework. A next improvement step would be to build new layers such as convolutional layers.