Project 2 - Mini deep-learning framework

Michael Allemann, Camille Elleaume, Laure Vancauwenberghe
EPFL, Switzerland, May 2019

I. Introduction

This mini-project is part of the Spring 2019 EE-559 Deep Learning course. The goal of this project is to design, implement and evaluate a mini deep-learning framework using PyTorch tensor operations. Users can build sequential networks, choosing from modules. The implemented modules are the fully connected layer, the dropout layer, and nonlinear activation functions ReLU and Tanh. The implemented optimization is mini-batch Stochastic Gradient Descent using the mean squared error as loss. In this report and in the code, the mini-framework is often annotated as diy/DIY (do it yourself).

II. Architecture

The central component of the mini-framework architecture is the abstract class Module containing the abstract methods forward(input) and backward(gradient). To implement this, ABC was imported and an abstract base class was created. Sequential, Linear, Dropout, ReLU and Tanh are the modules that inherit from the abstract base class Module. Figure 1 shows the UML of Module. A Sequential object takes a list of modules and stores them. When forward(input) is called on a Sequential object, it traverses its list of modules and iteratively calls forward at each module, feeding the next module with the output of the last module. The same thing happens for backward(gradient). The algebra performed during forward and backward passes through modules was inferred from page 8/11 of the class handout on back-propagation [1] and will not be discussed further in this report. The Dropout layer was implemented as described on page 7/12 of the class handout on dropout [2].

III. Initialization

In order to leverage vanishing gradients, much care must be taken during initialization of the weight and bias parameters in the Linear module. In this mini-framework the weight parameters get initialized using the Xavier initialization recommended on page 13/22 of the class handout on initialization [3]. The bias is simply initialized at 0.01.

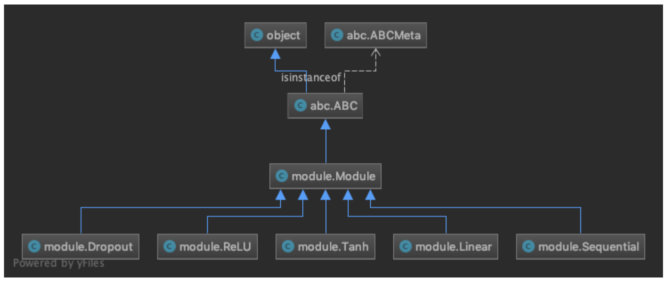

Fig. 1. UML of abstract base class Module and inheritance.

IV. Loss

For this mini-framework mean squared error loss was implemented. The loss is the criterion which is minimized during training, whereby its derivative is propagated backwards through the network in order to change the weights and biases. MSE loss is defined as the sum of the component-wise squared difference, divided by the total number of components in the tensor [4].

V. Optimizer

For this mini-framework mini-batch stochastic gradient descent (SGD) was implemented. The definition of mini-batch SGD can be found on page 12/17 of the class handout on SGD [5].

VI. Models chosen for evaluation

In order to evaluate the mini-framework, 4 models were built and compared. 2 built from the mini-framework and 2 containing PyTorch models. The following are the common parameters:

- Learning rate $\rightarrow$ 0.001
- Number of rounds $\rightarrow$ 10
- Mini-batch size $\rightarrow$ 100
- Number of epochs $\rightarrow$ 100
- Number of training samples $\rightarrow$ 1000
- Number of test samples $\rightarrow$ 1000
- 3 hidden linear layers with 25 nodes each
- Activations $\rightarrow$ ReLU, ReLU, ReLU, Tanh
- MSE loss

The first mini-framework model has no additional parameters, than the ones listed above. The second

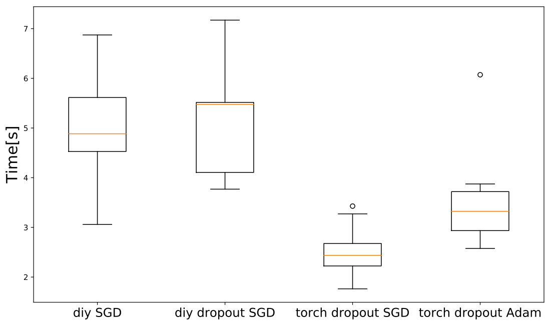

Fig. 2. Boxplots of the time used for training over 10 rounds for 100 epochs and 1000 samples for the four models.

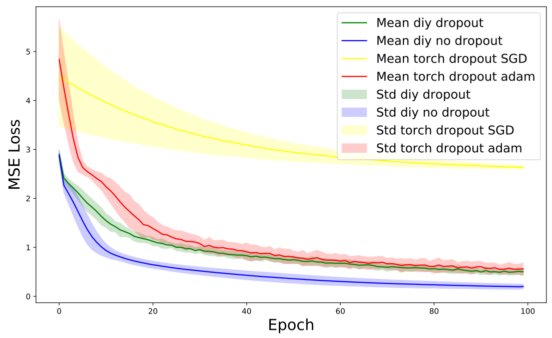

Fig. 3. MSE Loss for the four models over 10 rounds, estimating the mean and std in function of the number of epochs.

mini-framework model additionally has a dropout layer after every activation, dropping 20% of weights during training. The first PyTorch model has exactly the same parameters as the second mini-framework model. The second PyTorch model uses Adam optimization instead of mini-batch SGD, because, for some reason, the PyTorch model with SGD doesn't converge to meaningful results. We guess that the PyTorch SGD implementation does additional transformations in the background and that the learning rate of 0.001 is not adequate for the problem at hand. We did not perform hyperparameter tuning. It is to keep in mind that these parameters are all chosen rather arbitrarily, based on recommendations and past experience.

VII. Experimental Results

To measure performance the mean and standard deviation, over 10 independent rounds, of the time and error rate were recorded for the 4 models, generating new data and reinitializing the models at each round. Additionally the loss per epoch was registered for the 10 independent rounds. Figure 2 shows boxplots of the time used for training. The mini-framework is clearly slower than the out-of-the-box pytorch version, but not by orders of magnitude. This is not surprising, because by using torch.empty() to initialize PyTorch tensor objects, algebraic operations like '+', '*', etc. are overriden by the optimized PyTorch operations.

Figure 3 shows the gradual minimization of the MSE Loss. Rather surprisingly the mini-framework SGD optimization performs equally well as the much more sophisticated PyTorch Adam optimizer. Maybe this is the case, because the cirle function we are trying to learn is not a very complex function. Notable again is the highly suboptimal behavior of the PyTorch model with SGD. Upon inspection we found out that the PyTorch SGD model assigns the same label to all inputs. This is however not the focus of the project, so we did not dig deeper.

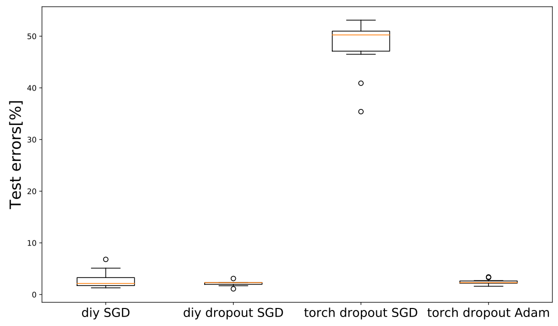

Fig. 4. Boxplot of the test errors over 10 rounds for 100 epochs and 1000 samples for the four models.

Figure 4 shows boxplots of the test error of the 4 models. What could be anticipated by looking at the convergence of the MSE loss is confirmed. Both mini-framework models perform extremely well with average errors around 4% and very tight estimations of standard deviation. The same is true for the PyTorch model using Adam optimization.

VIII. Discussion

Figure 5 confirms, that we have successfully built a mini deep-learning framework capable of learning simple functions. What is shown in Figure 5 are the predictions for the $10 * 1000$ generated test samples over the 10 rounds with 10 different instances of the mini-framework model with dropout. The points with prediction 1 get are blue and the points with prediciton 0 are red. It would

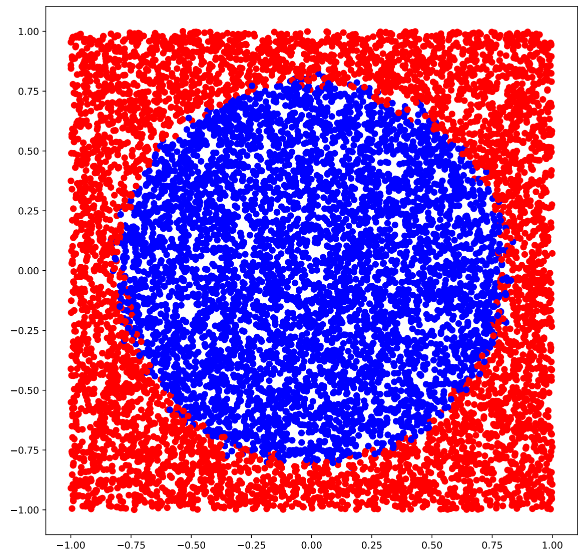

Fig. 5. Label predictions in the 2d-plane for 10 independent rounds with each 1000 test samples. We can observe that the mini-framework has successfully learnt the desired function.

be interesting to try to further implement convolutional layers in order to see, if the mini-framework is capable of learning more interesting functions, like the classification of MNIST data.

REFERENCES

[1] Franois Fleuret. Back-propagation. https://fleuret.org/ee559/ee559-handout-3-6-backprop.pdf, 2019.
[2] Franois Fleuret. Dropout. https://fleuret.org/ee559/ee559-handout-6-3-dropout.pdf, 2019.
[3] Franois Fleuret. Initialization. https://fleuret.org/ee559/ee559-handout-5-5-initialization.pdf, 2019.
[4] Franois Fleuret. Modules and batch processing. https://fleuret.org/ee559/ee559-handout-4-3-modules-and-batch-processing.pdf, 2019.
[5] Franois Fleuret. Sgd. https://fleuret.org/ee559/ee559-handout-5-2-SGD.pdf, 2019.