Implementing a deep learning framework

Andrea Oliveri, Célina Chkroun, Stéphane Nilsson
EPFL, Switzerland

I. Introduction

Deep learning libraries such as Pytorch are extremely complex and sophisticated. To better understand the underlying functioning of such complex libraries, we implemented a mini deep-learning framework using only Pytorch basic tensor operations and disabling Autograd, hence requiring to re-implement the gradient computation by hand.

Our small deep-learning framework implements several types of weight and bias initialization, activation functions and learning rate schedulers. The Linear layer, Sequential model, the Mean Squared Error (MSE) loss, as well as the Early Stopping training callback have also been implemented in this framework. Several abstract classes were used in order to keep a more generalizable structure of the framework and allow to easily expand it with more modules in future.

To test the correct functioning of our designed framework, a simple dataset was generated by uniformly sampling datapoints in $[0,1]^2$ and assigning a label 0 to those outside the disk centered at $(0.5, 0.5)$ and radius $\frac{1}{\sqrt{2\pi}}$, and assigning the label 1 to those inside. A simple Fully Connected Network has been trained using ReLU and Tanh activation functions, SGD optimizer and MSE Loss.

II. Models and Methods

The weight initializers implemented are Xavier Normal and Uniform, He Normal and Uniform, Uniform distribution and Normal distribution. Initializers to allow the user to put a constant value for the whole weight or bias were also implemented. The initializer that should be used for a particular linear layer depends on the activation function following said layer, as some initializers are more suitable for different activation functions [1].

The learning rate schedulers implemented are constant, time decay, exponential decay and step decay. The scheduler is chosen to best adapt the learning rate during the training as a function of the number of epochs. This will help avoiding oscillating around a local minimum, which would cause slow training or possibly divergence [2].

Early stop has also been implemented to avoid overtraining the model, hence preventing overfitting and saving computational resources and time. This training callback allows to stop the training when the validation loss stops improving (end of training) or degrades (overfitting) [3]. The weights of the network at the epoch with best validation loss are stored, and are restored once the training is stopped. Additionally, the use of this callback allowed us to avoid having to fine tune the number of epochs, which can instead be set to a large value as training will be interrupted by early stopping [3].

An abstract class has been created, from which different modules such as the Sequential model, the different layers and the MSE loss will inherit from. This allows for a specific redefinition of the forward and backward pass computation for each subclass.

The only implemented loss is the Mean Squared Error which has the advantage to be very simple to compute and derive, even if it is generally preferably used for regression problems rather than for classification problems as the one treated here. For binary classification problems such as ours, binary crossentropy is generally the commonly used loss function [4].

Several layers are implemented: the Linear fully connected layer, which has weights and bias parameters, the ReLU activation layer and the Tanh activation layer.

The model implemented is a simple Sequential model which will execute the forward and backward pass on the modules added to sequentially, one after the other.

The architecture of the sequential model used within this mini-framework is shown in Figure 1. The final activation layer is Tanh, to saturate the output of the net to a maximum value of one.

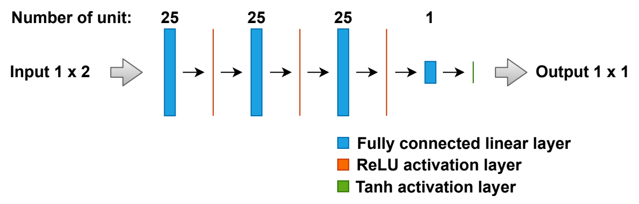


Figure 1: Architecture used for the sequential model that will be trained to classify the dataset. The linear layers are displayed as blue rectangles while the ReLU activation layers are displayed as orange lines and the Tanh activation layer as a green line. The number of units in each linear layer is displayed on top.

For the training, Stochastic Gradient Descent (SGD) was used for parameters update. The linear layers before each ReLU activation layer have been initialized with He Normal for the weights and with zeros for the bias as it is known to be a good initialization for ReLU [1]. For the linear layer before the Tanh activation layer, the Xavier Normal initialization has been used for the weights and the bias has been initialized with zeroes as it is known to be a good initialization for Tanh [1]. For the learning rate scheduler, the time decaying learning rate has been chosen as our tests showed we could easily achieve fast convergence and good results with it. Finally, all the hyper-parameters were fine tuned using cross-validation. After early stops kicks in, the model with the lowest test loss is kept as the final model (the weights stored by early stopping are restored) and one final measure for the training and test accuracy is performed with this model. These results are reported as final accuracy values.

III. Results

Figure 2 (A), the loss of the model evaluated on the training and on the test set is shown and globally it seems the model successfully converges to a local minimum. Figure 2 (B) shows the accuracy of the model on the training and on the test set. The model converges to a relatively good accuracy for both of them. Early stop has stopped the training after 80 epochs. The final model is 99.5% accurate on the training set and 99.1% accurate on the test set. It is interesting to observe from the trend of the training and test loss as well as the achieved final accuracies, that our final model does not seem to have overfit the training set.

A visualisation of the final model decision region compared to the ground truth is presented on Figure 3. From this Figure, we observe that the training leads to a nearly perfect decision boundary compared to the ground truth, with small imperfections mainly linked to the polygonal shape of the decision boundary.

IV. Discussion

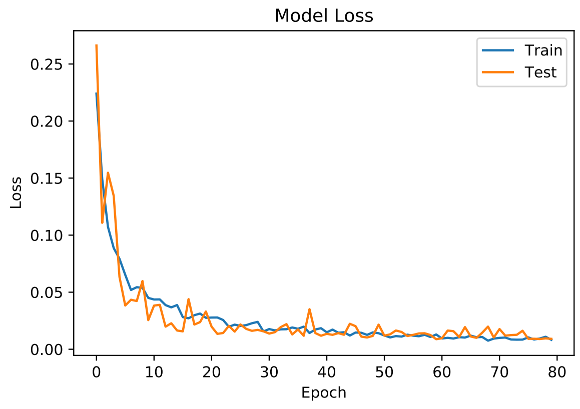
In the results section, we observed that the training seems to have worked well as a good accuracy is achieved and the decision boundary found by the final model is very close to the ground truth. It is worth reminding, however, that these are artificially generated training and test sets which contain no noisy points, and hence such a good performance of our model is not surprising.

To further improve the model, a Sigmoid activation function instead of the Tanh could have been used instead. The target labels used are 0 and 1 but Tanh scales the output values of the net to the range $[-1, 1]$. Additionally, as previously mentioned, the MSE loss was used to train the model, but this loss is generally not considered as well-suited for classification problems. Cross entropy loss is generally considered more appropriate for classification.

V. Conclusion

To conclude, a mini deep learning framework including some common deep learning modules such as Linear layer, Relu and Tanh activation, as well as Initializers and Losses has been implemented. After that, a model has been built using the implemented deep learning framework. The model was trained, and the quality of the training was assessed by logging the loss and accuracy evolution as a function of the number of epochs. Finally, the trained decision region is compared to the ground truth. Results showed that the training evolution was successful and that the trained decision region is close to the ground truth one, which indicates correct implementation of the framework.

(A)



(B)

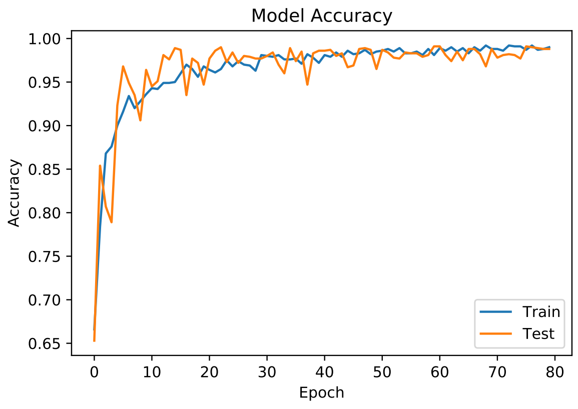


Figure 2: Accuracy and loss of the model on the train and test set measured for each epoch of the training to asses the quality of the training done on the designed model. For both graphs, the measures taken on the train set are displayed in blue and the ones taken on the test set are displayed in orange. (A) MSE (Mean Squared Error) loss of the model during the training evaluated on the train and on the test set. (B) Accuracy of the model during the training evaluated on the train and test set.

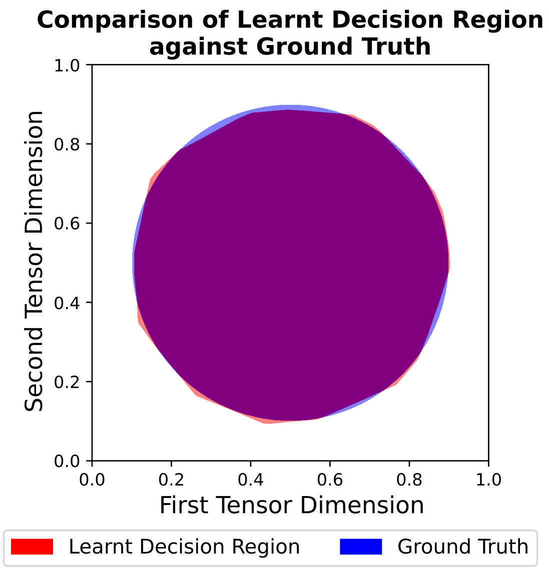


Figure 3: Visual result of the model decision region after the training displayed in red and ground truth decision region displayed in blue.

References

[1] K. Vishnu, "Xavier and He Normal (He-et-al) Initialization | by Vishnu Kakaraparthi | Medium," Oct. 2018. [Online]. Available: https://prateekvishnu.medium.com/xavier-and-he-normal-he-et-al-initialization-8e3d7a087528

[2] J. Brownlee, "How to Configure the Learning Rate When Training Deep Learning Neural Networks," Jan. 2019. [Online]. Available: https://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks/

[3] J. Brownlee, "A Gentle Introduction to Early Stopping to Avoid Overtraining Neural Networks," Dec. 2018. [Online]. Available: https://machinelearningmastery.com/early-stopping-to-avoid-overtraining-neural-network-models/

[4] J. Brownlee, "How to choose loss functions when training deep learning neural networks," Aug 2020. [Online]. Available: https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/