Ferrari Optimizer

Fundamentals of Machine Learning

Edoardo Caproni
Tommaso Quintabà

Goal of the project

We aim to test an optimization algorithm - hereafter referred as Ferrari Optimizer - on a simple use case. We decided on a classifier trained on the Optical Recognition of Handwritten Digits dataset[1]. In the end we show the circumstances in which the Ferrari Optimizer represents an advantage, and the pitfalls that come with it.

Implementation

The dataset works by taking 32x32 bitmaps and dividing them into non-overlapping 4x4 blocks; then the number of pixels in each block are counted and saved in an 8x8 matrix, which we use as input for the MLP.

After loading labels and inputs in the script, we shuffle and split the dataset into training and validation sets. The labels are both available in plain-text and 1-hot encoding.

The network (a Multi-Layer Perceptron) is built as a 64 neuron input layer, a single hidden layer of 30 neurons and an output layer of 10 neurons in 1-hot representation. In any case the network can be modularly built to satisfy any requirement in shape or number of layers.

Before starting the training it is possible to choose whether to use the Ferrari Optimizer or switch to a static learning rate.

The training process is a forward sweep of the network, followed by the loss calculation (using Mean-Squared-Error), and then the backward step. At this point we update weights and biases using the gradients computed with back-propagation and the current learning step.

[1] https://archive.ics.uci.edu/dataset/80/optical+recognition+of+handwritten+digits

Then, if the optimizer is activated, the learning rate is updated as the old learning rate divided by the squared norm of the loss gradient (with respect to the weights of the output layer).

After the training process is complete, the predictions based on the validation set are collected and organized in a full Confusion Matrix. This is instrumental to measure the number of True Positives, False Positives, True Negatives and False Negatives for each class. Finally, we summarize these results as performance metrics: Accuracy, Precision, Recall and the F1 score; both for each class and the comprehensive average.

Observations regarding the Ferrari Optimizer

The main issue was preventing the training process from becoming unstable: we soon realized that the learning rate skyrockets very fast. This is because, as we get close to a stationary point, the gradient will approach 0, and the squared norm of the gradient is the denominator in the new learning rate calculation.

At first we thought about doubling the loss whenever it got below half of the starting value, and dividing by the same amount to keep track of the actual error. This approach bore no fruit since the instability is caused by the gradient of the loss, not the loss itself. Still we decided to keep the method as it could prove useful in other scenarios.

The other contingency in place checks the directly the gradient loss against a threshold: when it is below that, we stop using the optimizer as another iteration would most probably break the training. Then it is trivial to set back the learning rate to its starting value, and continue the warmed-up training with a static learning step.

Conclusion

After through experimentation with various sets of hyper parameters, we conclude that the Ferrari Optimizer is useful to kickstart the training process, especially when the starting loss value is very high and we find ourselves far from stationary points. Still, just using a static learning rate outperformed the former approach in our specific environment, as can be seen in the attached log files.

Furthermore we noticed that using the Ferrari Optimizer can hinder the ability of the MLP to recognize some classes (in the attached logs it was the case for numbers "1" and "3"). We believe this is because when the learning rate gets too high some weights reach very high (absolute) values, which then prove impossible to reel back with a static learning step. It is possible that some neurons get stuck in an "active" state, making it impossible to learn certain features, while the others keep training trying to compensate.