TP1

Timothé MORVAL,Thomas DELLIAUX, Jean-Frédéric DIEBOLD

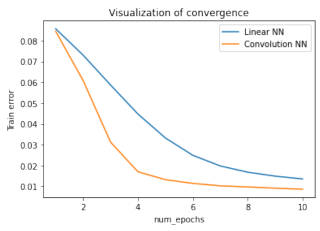
February 7, 2023

1 Models presentation

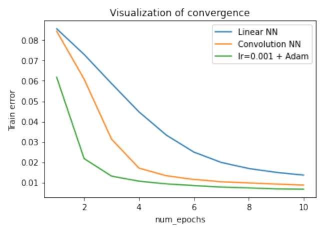
1.1 Multi-classification problem

In the first problem, we tried 3 different approaches:

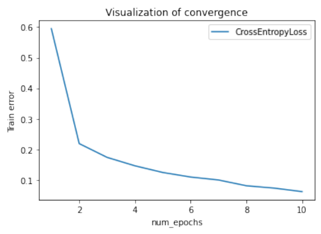
- Model structure: we chose to add a one dimension convolution layers on the top of our network. We tried adding more convolutionnal layers, but it made training longer for no better perfomance. We supposed is due to the small size of the picture.
- Then, we changed the learning rate from 0.1 to 0.001 and switched the optimizer from SGD to Adam.
- Finally, we changed the loss function. Cross-entropy loss is better sutted for multi-classifying pictures than the mean square loss.



Using a CNN



Changing learning rate and optimizer



Changing the loss

We reached an accuracy of 92.68 pourcent on the test dataset, against 88.79 for the original network.

1.2 Regression Problem: House Prices Prediction

The final model is a fully connected MLP with relu activation between each layers, it has following characteristics : (1st) 4*32 (2nd) 32*64 (3rd) 64*64 (4th) 64*1. We use the Adam optimizer with a learning rate of 0.01 with 400 epochs. We normalized the data with mean and std deviation of the train set to get all the features in the same range between -1 and 1. In order to quantify the accuracy, we introduced an MAE indicator ($|\hat{y} - y|/y$). The accuracy was improved from a baseline of 83.4 (2 layers, no relu) to 86.8. We did not manage to run the loss with variance and mean set up. During training process, each 2 epochs, we observed the performance on both training and validation set, so that we could verify if overfitting occured. We decided to save the model each time the validation dataset accuracy improved. Test data proved the performance achieved was robust, with small variation compared to validation data performance.

2 Global discussion

2.1 Tradeoff between performance and efficiency

For testing the different hyperparameters, we followed train error and validation error, changing each hyper parameter one by one. We observed that depending on the parameter, optimal results were achieved shortly and that small finetuning was rarely efficient. Most of the time, we kept parameters corresponding to the best validation error, but sometimes we also took into account the stability and the convergence speed of the training phase. Once we have chosen the number of epochs and the learning rate, we tested again all previous parameters to ensure optimality.

2.2 Impact of parameters

- number of neurons

  We saw that, without activation, adding neurons to the single layer made the model converge faster. However, when adding relu activation, we observed that for a too high number of neurons, there was less stability for poor accuracy improvement and a bit of overfitting. Optimal number of neurons is greater than what a global minimization task would expect in regard of avaliable data, but a too high number slows the process and can bring lower robustness.

- number of layers

  We saw that the more layer we have, the faster the model converges. But, this time, the validation error decrease with the number of layer. Unfortunally, the models with more layers are more computationally expensive, so again we had to chose when to stop to increase the number of layer. We decided therefore to use 4 layers. With the depth, some problems of gradient vanishing or exploding may occur, but is not the case here since our models are relativly small.

- Activation function

  We saw that ReLU beat tanh and sigmoid by far for regression problem. Sigmoid's learning rate is very small. We could explain this by the fact that the loss error satures the sigmoid during the training, making the gradient not meaningful. Activation (ReLU) was one of the most efficient parameter for improving the performance. Adding a non-linearity to the model, made it improve in expressivity, it can approximate more complicated functions.

- batch size

  Increasing the batch size made the convergence slower, and the epoch calculation faster. The algorithm seemed less stable with big batch size (near 400). We had a trade-off between speed of convergence and time for computing an epoch.

- learning rate

  The convergence becomes faster when the learning rate increases, but if it is too high the algorithm becomes less stable as well and may diverge. We had to finetune this parameter for a tradeoff between convergence speed, robustness and performance.

- number of epochs

  With the number of epoch we need to ensure that the model has converged but we need also to take care about overfitting. Indeed, we have seen during the course that an early stop can be a good way to regularize our model. Therefore, we introduced a save management for keeping an ideal training duration.