TP1 Deep Learning: Linear model implementation

Leloup Corentin and Deschamps-Berger Théo

***Abstract*—The aim of the Tp1 was to implement a neural network from scratch without using any neural network libraries.**

I. Data

The dataset is from the famous MNIST database, it's 50,000 images of handwritten digits. Each image has a size of 28*28 pixels.

II. Description of the task

From the MNIST database we had to implement a neural network and trained a linear classifier. We used the following model:

$$o = \mathrm{softmax}(Wx + b)$$

where:
* x is an input image that is represented as a column vector, each value being the "color" of a pixel
* W and b are the parameters of the classifier
* softmax transforms the output weight (logits) into probabilities
* o is column vector that contains the probability of each category

We want to train this model via stochastic gradient descent by minimizing the negative log-likelihood of the data:

$$\mathcal{L}(x, gold) = -\log \frac{\exp(x[gold])}{\sum_j \exp(x[j])} = -x[gold] + \log \sum_j \exp(x[j])$$

This negative log-likelihood is also called the cross-entropy loss.

III. Initialisation

For an affine transformation, it is common to initialise our parameters as the following:

- The bias to 0
- And the weights (projection matrix) with Glorot initialization given by this formula:

$$\mathcal{W} \sim U[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}]$$

IV. Implementations

A. First naive implementation

With 20 epochs:

- We initialized the step size to 0.01
- The final result of the accuracy on the test was 0.9203

Fig. 1. The loss evolution on 20 epochs

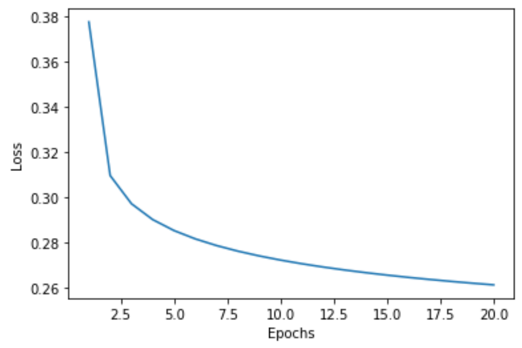
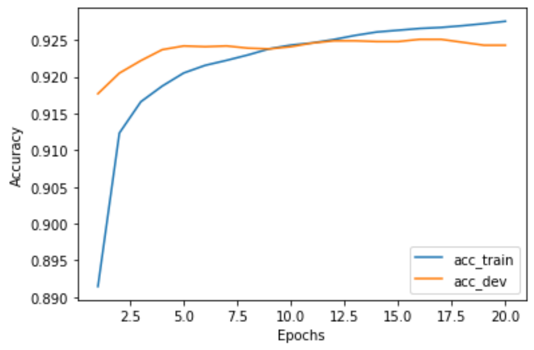


Fig. 2. The train and dev accuracy evolution on 20 epochs



We noticed that:

- The best score of the dev set is on the 16th epoch, but we still used the parameters of the last epoch to obtain the accuracy on the test.
- The model seems to overfit after the 16th epoch, the accuracy of the dev data decrease instead of the accuracy of the train which increase.

B. Optimized implementation

In a second implementation we tried to improve our model introducing the following points:

- Instead of memorizing the parameters of the last epoch only, we kept the parameters that produced the best value on dev data during training and evaluate it on test
- If the score on the dev data during training didn't increase we reduced the step size by 2. It allowed us to not be trapped in a local minimum and learn more.
- We shuffled the data before each epoch to distribute the training set in a homogeneous way. It reduced the chance to find local minimum instead of the global minimum.

With 20 epochs and a step initialized to 0.01 we obtained an accuracy of 0.9261 on the test set.

Fig. 3. The loss evolution on 20 epochs

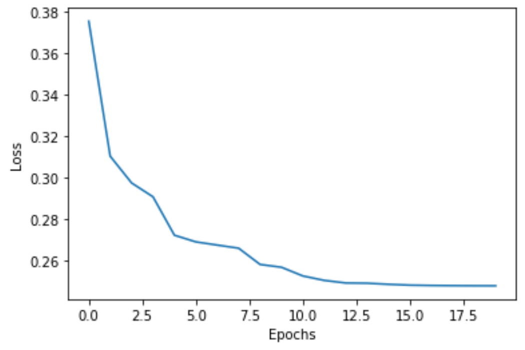
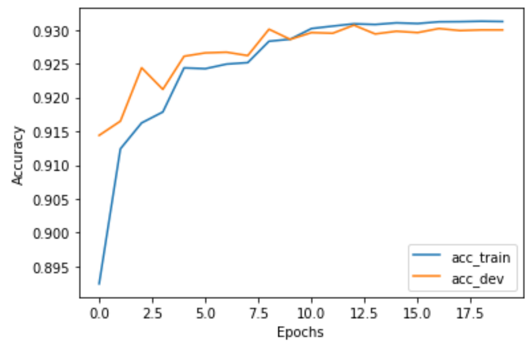


Fig. 4. The train and dev accuracy evolution on 20 epochs



V. Conclusion

With the neural network optimized, we succeed to improve our final results on the test set.