Homework 2: Fully Connected NN for CIFAR10 Classification

Elisa Ferrara

1 Softmax manual implementation

The code in the Softmax notebook implements a complete pipeline for training, tuning, evaluating, and saving a softmax classifier for image classification tasks. Here I summarize some of the choices made:

- *Softmax Loss Function*: vectorized version of the softmax loss function for computing loss and gradient. Here, in order to prevent numerical instability, I decided to subtract the maximum value from the scores before computing the softmax function.
- *Training the Linear Classifier and hyperparameters choice*:
  1. Chosen and designed optimizer: Stochastic Gradient Descent (SGD);
  2. Learning rate: I tried different learning rates (in between linspace(4e-6, 5e-3, 20) using the validation set to tune it. I started with small learning rates (around 5e-7 and 5e-6) and then I noticed performances improved increasing it. I found out that a good one is lr = 3.159368e-03 for which the accuracy on train set is 42.1735 % while the one on the validation set is 41.6000 %. The weights related to the best validation accuracy are saved.

2 Pytorch implementation

The first NN I implemented was a very simple one: one single layer with 512 neurons. The only aspect I was careful was to choose a power of 2 as dimension of the hidden layer in order to have better performances on my CPU. CPUs are optimized for handling data sizes that are powers of 2 and can align well with the size of SIMD vectors, allowing for more efficient parallelization of computations. The results show an accuracy on the validation set with 10 epocs of training of 52 % and 51% on the test set in the Evaluator notebook.

	Single Layer	Deeper Network	Layer Normalization	Dropout	Data Augmentation
Validation Set	54%	54%	54%	54%	65%
Test Set	52%	54%	52%	53.5%	56%

Table 1: Performances of the various NN used and tests on various sources of improvement

2.1 Deepen the network

First general way to improve performances is to deepen the network: the more it is deep the more details it catches and the more non linear it can become. I added two more hidden layers and used a decreasing dimension architecture [512 –> 256 –> 128 neurons].The accuracy on the test set improved of 2%.

2.2 Layer normalization

Next attempt to improve performance is to add Layer normalization, after Fully Connected layer and before nonlinearity.This improves generally gradient flow through the network and in a way acts as a form of regularization. Since there was no significant improvement (actually accuracy got back to 52%, I removed it).

2.3 Dropout

I also added a random dropout after each nonlinearity, the probability of dropout. I tried probabilities of 25%, 35%, 40% and 50% obtaining on the test set accuracy of 50%, 51%, 53% and 53.5%. I will keep the last implementation

2.4 Data Augmentation

Finally, the performances on the validation set rapidly increased (+5%) thanks to data augmentation. I added to the original dataset, the same data but transformed (cropped and horizontally flipped). Also accuracy on test set increased a bit (+1%). Seen the good results achieved with this new dataset, I finally added a third modified copy of the original dataset in which I transformed the images in color and rotation. The result is impressive: 65% accuracy on validation (+ 6% than with just one data augmentation) and 56% of accuracy on test set. I will keep this as final result for the weights.

Final Comment

Increasing the depth of the network or performing even more data augmentation (especially keeping a low dropout rate) can lead to overfit to the training model so I stopped with the above values. Finally, note that I also tested other optimizers such as ADAM but I found that the results were poorer.