Practical work 03 – 05/03/2024
Shallow Networks

Objectives

The main objectives of this Practical Work for Week 3 are the following :

a) Implement Softmax and use MBGD to learn what it means to choose hyper-parameters such as learning rate, batch size or number of epochs.

b) Further deeping your understanding concerning the representational capacity of a MLP .

Submission

— **Deadline** : Tuesday 12 March, 3pm

— **Format** :

  — Exercise 1 (Softmax Classifier)
    — Jupyter notebook `softmax_classifier_stud.ipynb` completed with your solution and results/comments.

  — Exercise 2 (Decision Boundaries)
    — Jupyter notebook `decision_boundaries_stud.ipynb` completed with your solution.

Submission of all files in a single zip-file using the naming convention (for team of two students #1, #2) :

`family name_given name #1- family name_given name #2.zip`

Exercise 1 Softmax Classifier

Implement the Softmax Classifier for the generalised Perceptron. Do this on the basis of the Jupyter notebook `softmax_classifier_stud.ipynb`. As before, do this by only using numpy functionality. Look out for the suitably marked sections that you need to implement.

The notebook also provides the possibility for SGD (Stochastic Gradient Descent) and MBGD (Mini-Batch Gradient Descent). The implementation is already done, you just have to configure it. Therefore, in the call to the method `optimise` of the class `NeuralNetwork` you have a new (optional) argument `batchsize`. Note that a value of `batchsize=0` corresponds to BGD, i.e. to using the full batch.

You will train softmax for the full FashionMNIST dataset. Since this will take more CPU and RAM, you need to be more careful in efficiently implementing the code. Make sure to properly use numpy array arithmetics ! All the training runs should complete in at most a few minutes.

Note that (new) in the Cell [13] (*Sample execution of Neural Network*) you can choose the categories for the training (variable `classes`). When starting to test your implementation, and for parts (a) and (b), you may want to use a small number of categories to speed up the training proceedure.

Proceed as follows :

(a) Implement the update rules for a softmax layer and cross-entropy cost. Keep an eye on the shapes of the numpy arrays defined in the input and to be provided as output.
Use the unit tests at the end of the notebook to test your implementation of the individual methods !

(b) Run the training by using MBGD and observe the learning curves : Cost, Error Rate - both for the training dataset and the validation dataset. Play with the hyper-parameters

- learning rate
- number of epochs
- batch size

Try to reproduce (qualitatively) the results for BGD, MBGD and SGD from Table 4 in the lecture notes (end of chapter 3.7.2).

(c) Now switch to the full set of 10 categories (variable `classes`) and perform the trainig. The overall error rate should be around 15%. Specify your choice of parameters and justify briefly why your choice is well suited.

(d) Finally execute the Cell [15], which shows the weight vectors reorganised as image-tiles. Verify that the result is similar to the one shown in the following figure.
Remark : For a "nice" representation of the weigth vectors choose `random_std = 0` in the constructor of the class `NeuralNetwork`.

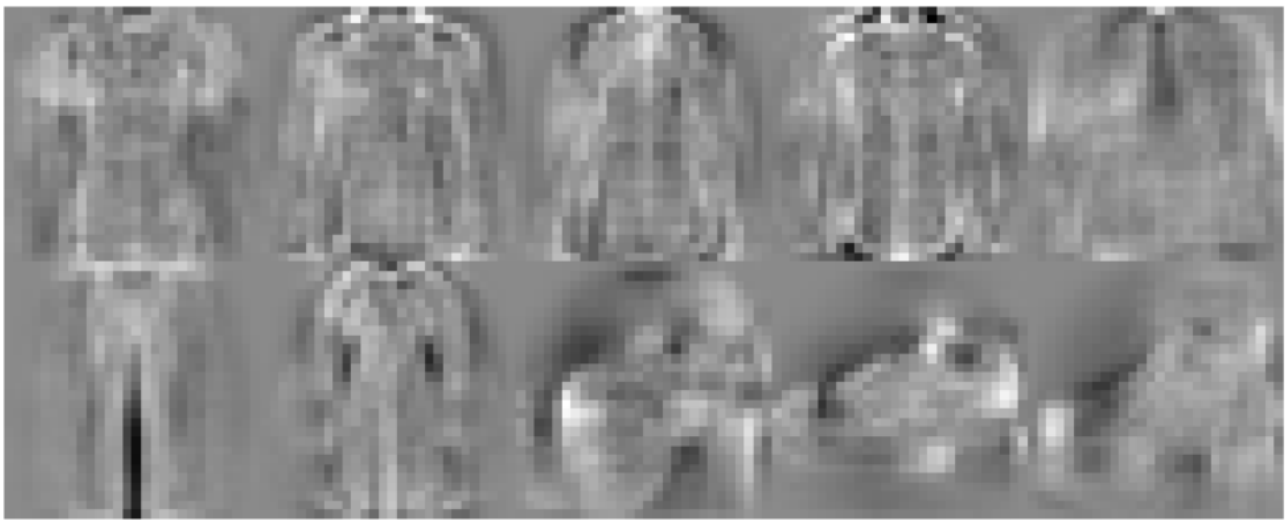

FIGURE 1 – Weight vectors of the Neural Network class for FashionMNIST data after training.

Exercise 2 Decision Boundaries

Implement (manually) suitable decision boundaries to perform a multi-class classification problem. Do this on the basis of the Jupyter notebook `decision_boundaries_stud.ipynb`. It is an extension to the exercise at the end of chapter 4.3.1.3 of the lecture notes from a binary to a 4-class problem.

Proceed as follows :

(a) First choose a *new* random seed on top of Cell [3] to change the configuration of the point sets. Maybe avoid a too large overlap of the sets by trying different values. Once the value fixed the sets should always remain the same.

(b) In Cell [5] extend the weights of the hidden `w1,b1` and the output layer `w_out,b_out`. Note that the number of output neurons should be 4 (for the four point sets i.e. classes). You can solve the problem with three decision boundaries (= hidden neurons) by combining the respective results in the output layer and taking the `argmax` value. For the random seed value of 10 (as given in the template) a good solution would look like in the following figure.
Remark : It is not important to obtain the perfect minimum ; an approximate solution is fully acceptable.

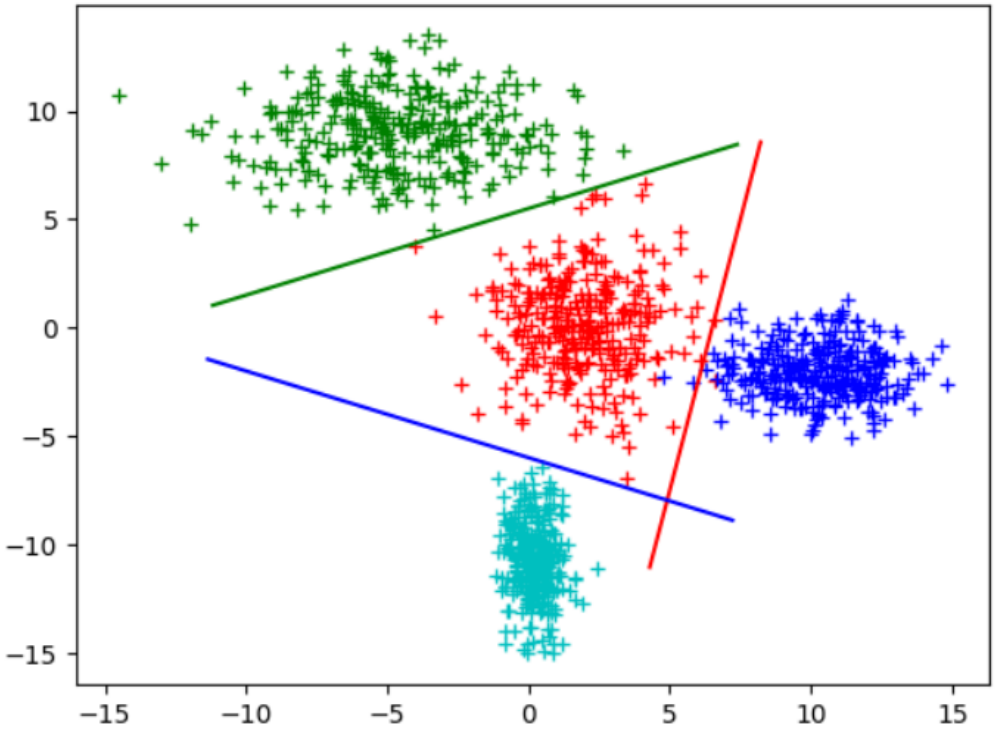

FIGURE 2 – Close to optimal decision boundaries for a value of the random seed equal to 10.