Practical work 04 – 12/03/2024 Back-Propagation

Objectives

Main objective is to implement the forward and back propagation algorithm for an arbitrary (fully connected) MLP with Softmax. A suitable structure is provided in form of a Jupyter Notebook that will allow to easily specify MLP models of arbitrary depth and layer sizes. Its structure, where layers are added in a sequential manner, resembles closely the way (sequential) Neural Networks are constructed in ML frameworks e.g., PyTorch or Tensorflow.

The exercise `4.1.compact-mlp_stud.ipynb` - discussed during the lecture - provides the blue print for the solution. Note however the footnote 84 on page 99 of the lecture notes (v.0.95) : In the notebooks used for the PW the first matrix index of the data is the *sample* index (within the batch) and not the *feature* index, as used for the formulation of the equations in the lecture notes and also for exercise `4.1.compact-mlp_stud.ipynb`.

The implementation of the MLP should then be used to train FashionMNIST data with one or more hidden layers.

Submission

- **Deadline** : Tuesday 19th March, 15pm
- **Format** :
  - Completed Jupyter notebook with the blanks filled in (the sections to be completed marked as usual).
  - Comments and results (plot with learning curve showing the results for different model complexities) either in the notebook or in a pdf-report.

  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 Implement Forward Propagation

Implement the forward propagation through an arbitrary MLP. Use the Jupyter notebook `mlp_stud.ipynb`. Complete the implementation of the method

— propagate

in the classes 'DenseLayer' and the 'SoftmaxLayer'.

Check your implementation with the corresponding unit tests at the end of the notebook.

Exercise 2 Implement Backpropagation

Now, implement backpropagation. Complete the implementation of the method

— backpropagate

in the classes 'DenseLayer' and the 'SoftmaxLayer'.

Again, check your implementation with the corresponding unit tests at the end of the notebook.

Further check the proper implementation of the gradient of the MLP by running the gradient checking (in section 'Test' subsection 'Test analytical value of derivative using backpropagation ...'). This check iterates through all weights, computes the numeric approximation and tests for discrepancies larger than a given limit accuracy ($4.0 \cdot 10^{-7}$). Numeric output is provided only if this threshold is exceeded. Inspect how the numeric approximation of the gradient is computed.

Remark : To speed up processing for the first tests you can reduce the number of classes used for training e.g., `classes = [0,1,2,3]` instead of `classes = [0,1,2,3,4,5,6,7,8,9]`. This should speed up processing on a state of the art CPU (Intel Core i7@1.8 GHz) for one training epoch (one single hidden layer with 100 neurons and batchsize=16) from from 13 s (10 classes) to 5 s (4 classes). The final error rate for 10 classes is around 10%.

Exercise 3 Train FashionMNIST

Now, use your implementation to instantiate and train an MLP for FashionMNIST with mini-batch gradient descent.

Study two different architectures :

a) Shallow Network : Single hidden layer layer with 150 units.
b) Deeper Network : Three hidden layers with 250, 150, 50 hidden layers.

What are suitable learning rates, batch sizes, number of epochs ? Describe your findings including the achieved test error rates and the learning curves.

Finally execute the Cell [18] ("Visualisation of the weight vector of the first layer", right above the "Tests"), which shows the weight vectors of the first hidden layer reorganised as image-tiles. Verify that the result is similar to the one shown in the following figure.

Remark : This kind of image representation can only be done for the first layer. For higher layers these correlations will be with respect to features generated by the previous layer. Thus, correlations in the second layer are with respect to the features of the first layer i.e., those represented in the image patches of Cell [18].
For CNNs will will see, that for all convolutional layers this type of representation is useful and is one of the strengths of this NN-type.

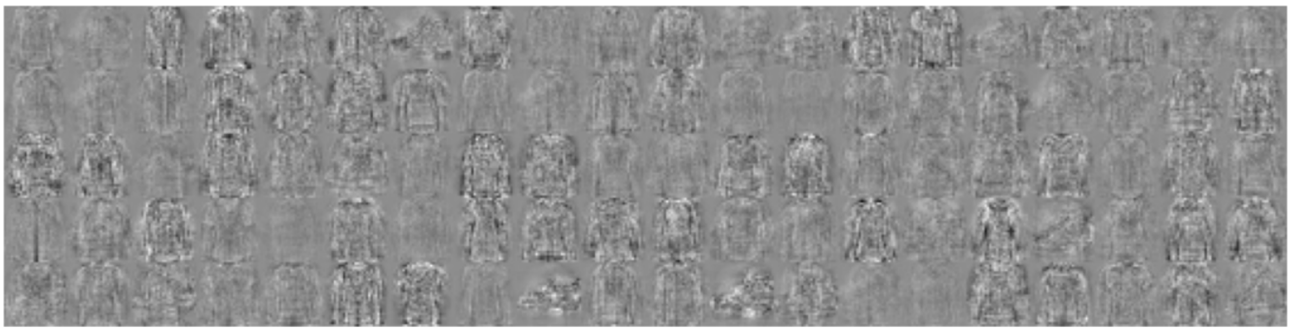

FIGURE 1 – Weight vectors of the first hidden layer of the MLP for FashionMNIST data after training.