Practical work 05 – 19/03/2023
Regularisation/Batch-Normalization

Objectives

Main objectives are the implementation and testing of regularisation techniques (L^2 and Dropout) and the application of batch normalisation in relation with the findings of X. Glorot and J. Bengio.

Submission

- **Deadline** : Tuesday 26th March, 15pm
- **Format** :
  - Completed Jupyter notebook with the blanks filled in (the sections to be completed marked as usual, `### START YOUR CODE ###, ### END YOUR CODE ###`).
  - Comments and results (plot with learning curves and/or weight histograms showing the results for the different parameter settings) 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 Regularisation

Starting point for the following work is the Jupyter notebook `mlp_regularise_stud.ipynb`. First implement the missing sections.

- Extend the `DenseLayer` methods `activation_function` and `d_activation_function` with the missing functions `tanh` and `relu`.
- Extend the `DenseLayer` with with the proper weight initialization scheme (c.f. Fig. 131 in lecture notes v0.97) for each activation function, e.g. $\sqrt{6/(n_{in} + n_{out})}$ for `tanh`. Note that we use the weight initialization schemes for the uniform distribution $U[-r, r]$ (left column in Fig. 131 of the lecture notes).
- Extend the method `gradient_descend` of the `DenseLayer` and the `SoftmaxLayer` with L^2-regularisation scheme (chapter 5.3.1.1 in lecture notes). The variable `self.regularize` plays the role of the parameter λ in the L^2-regularisation formula. You can define the value of `self.regularize` in the constructor of the class `NeuralNetwork`.
- Extend the methods `set_dropout_vector` and `back_propagate` of class `DropoutLayer` for proper dropout (applied to the input vector of each fully connected layer). Use the Unit Test at the end of the script to verify your implementation.
  When inspecting the constructor of class `NeuralNetwork` you will notice that for settings of `drop_p > 0` before each layer (including input layer) a `DropoutLayer` is added.

Now perform training using different settings to observe the effect of the regularisation schemes :

- As basesline use a MLP with two hidden layers (e.g. 200 and 100 neurons), which should show considerable overfitting as illustrated in Fig. 138 of the lecture notes. Use hyperparameter settings (learning rate, epochs, ...) you found to be appropriate in PW-04 from last week.
- Now switch on L^2-regularisation by choosing a value of `regularize = 0.0002` in the constructor of the class `NeuralNet`. Note that this value was chosen with a batch size of 16 (c.f. Fig. 138 of lecture notes) and a different batch size may require a rescaling of the regularisation parameter (why ?).
  Observe the behaviour of the learning curves and weight histograms and compare to the baseline without regularisation (as well as to Figs.139 and 140 of the lecture notes).
- Switch off L^2-regularisation (by choosing `regularize = 0.0`) and switch on dropout. A value of `drop_p` $\approx$ `0.2` will already be sufficient. Again observe the behaviour of the learning curves in comparison with the curves obtained without regularisation. Compare to the Fig.144 of the lecture notes.

Discuss briefly your findings.

Exercise 2 Batch Normalization

Starting point is the Jupyter notebook `activation_variance_batchnorm_stud.ipynb`. It is an extension of the script used in the exercise at the end of chapter 5.1.4 on page 112 of the lecture notes. We want to illustrate that batch normalization can compensate for bad weigth initialisation.

- Study the script and observe the main differences with respect to the script used in the exercise - cited above - during the lecture :
  when setting the parameter `batch_norm = True` in the constructor of `NeuralNetwork` each `DenseLayer` is followed by a `BatchNormLayer`. We implement the batch normalization as an independent layer for reasons of clarity. Leave `batch_norm = False` for the moment.
- Execute the cell named `Propagation step` (requires all cells above). One propagation step is performed and the activation histograms are shown. Due to the poor weight initialisation scheme of parameter `random_std = 1*np.sqrt(1/num_feat)` (corresponding to $\sqrt{2/(n_{in}+n_{out})}$) the activation variances change during the forward and backward path (as shown in Fig.128 of the lecture notes).
- Implement propagation and backpropagation of the batch normalization layer in the notebook. The relevant formulas are given in chapter 8.1 of the lecture notes. Check your implementation with the corresponding unit tests at the end of the notebook.
- Activate the batch normalization layers in the constructor of class `NeuralNetwork` using parameter `batch_norm = True` . Then reexecute the cell named `Propagation step` and observe the histogram of activations. As shown in the following figure these should now be constant except for the first layer (or last in backpropagation). This is due to the fact that we plot the activation right after the output of the `DenseLayer` and for the first layer no batch normalization was present yet.

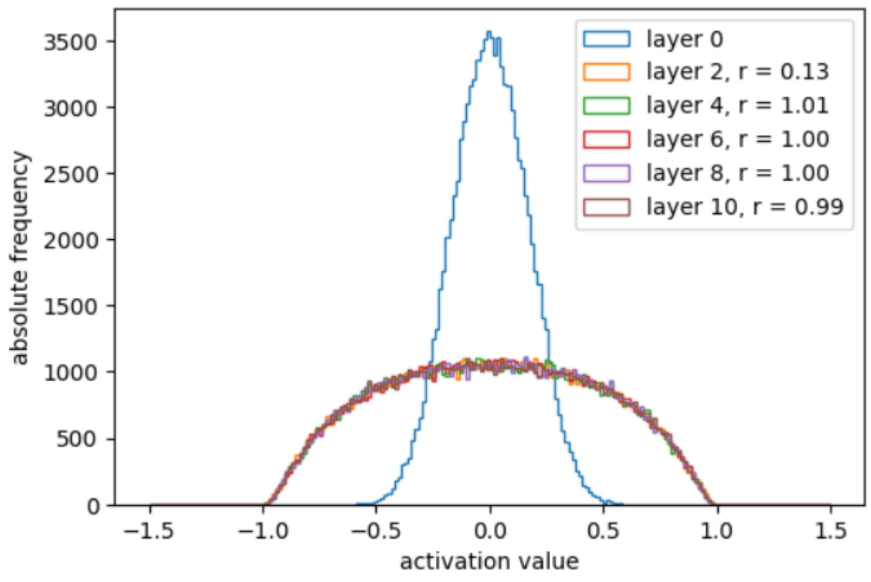

FIGURE 1 – Histogram of activations after applying batch normalization.