Sapienza Università di Roma

ADVANCED MACHINE LEARNING

REPORT HW 1

Students

Mert Yildiz
1951070
Ali Reza Seifi Mojaddar
1900547
Gianmarco Ursini
1635956
Mohammadreza Mowlai
1917906

Professor

Prof. Fabio Galasso

Accademic Year 2021/2022

Question 1: Implementing the feedforward model

In this question we have implemented a two-layered neural network architecture as well as the loss function to train it.The ReLU non-linear activation ϕ, applied element-wise on each unit, resulting in the activations $a^{(2)} = \phi(z^{(2)})$. The ReLU function has the following form:

$$\Phi(u) = \begin{cases} u, & \text{if } u \geq 0 \\ 1, & \text{if } u < 0 \end{cases} \tag{1}$$

The Softmax is defined as:

$$\psi(u_i) = \frac{e^{u_i}}{\sum_j e^{u_j}} \tag{2}$$

So, we have implemented the functions above in the code in `two_layernet.py` for the feedforward model. We have generated the scores for the toy inputs that match the correct scores given in the `ex2_FCnet.py` and we got 2.9173411603133914e-08 as the difference between our scores and correct scores.
To be able to train the above model on large datasets, with larger layer widths, the code has to be very efficient. To do this we avoided using any python for loops in the forward pass and instead we have used matrix and vector multiplication routines in the Numpy library.

Question 2: Backpropagation

2.a)

Given the shape of our unregularized cross-entropy loss function given N samples of the training set:

$$J(\theta, \{x_i, y_i\}_{i=1}^N) = -log(\psi(z_i^{(3)})_{y_i}) \tag{3}$$

$$J(\theta, \{x_i, y_i\}_{i=1}^N) = -\frac{1}{N}\sum_{i=1}^{N} log\left(\frac{exp(z_i^{(3)})_{y_i}}{\sum_{j=1}^{K} exp(z_i^{(3)})_j}\right) = -\frac{1}{N}\sum_{i=1}^{N}\left((z_i^{(3)})_{y_i} - log\left(\sum_{j=1}^{K} exp(z_i^{(3)})_j\right)\right) \tag{4}$$

We can derivate it with respecto to $z_i^{(3)}$:

$$\frac{\partial J}{\partial z_i^{(3)}} = -\frac{1}{N}\left(\frac{\partial (z_i^{(3)})_{y_i}}{\partial z_i^{(3)}} - \frac{\partial}{\partial z_i^{(3)}} log\left(\sum_{j=1}^{K} exp(z_i^{(3)})_j\right)\right) = \frac{1}{N}\left(\frac{1}{\sum_{j=1}^{K} exp(z_i^{(3)})_j}\frac{\partial \sum_{j=1}^{K} exp(z_i^{(3)})_j}{\partial z_i^{(3)}} - \delta_{j,y_i}\right) \tag{5}$$

$$\frac{\partial J}{\partial z_i^{(3)}} = \frac{1}{N}\left(\frac{z_i^{(3)}}{\sum_{j=1}^{K} exp(z_i^{(3)})_j} - \delta_{j,y_i}\right) = \frac{1}{N}\left(\psi(z_i^{(3)}) - \delta_{j,y_i}\right) \in \mathbb{R}^3 \tag{6}$$

Notice that in (5) we dropped the summatory because of the derivation only respect to i_{th} realization of $z^{(3)}$, and we introduced the vector δ_{j,y_i} (remembering the Kroenecker delta function) that is a vector of size 3 (coherently with the size of $z^{(3)}$) which contains all 0 except for the location indexed by $j = y_i$, containing 1.

2.b)

We are aiming now to compute the derivative of the unregularized loss in (28) w.r.t. the weight matrix $W^{(2)}$. Leveraging the chain rule we have:

$$\frac{\partial J}{\partial W^{(2)}} = \sum_{i=1}^{N} \frac{\partial J}{\partial z_i^{(3)}}\frac{\partial z_i^{(3)}}{\partial W^{(2)}} \tag{7}$$

Reminding us that the first factor has been already obtained, we compute now:

$$\frac{\partial z_i^{(3)}}{\partial W^{(2)}} = \frac{\partial (w^{(2)} a_i^{(2)} + b^{(2)})}{\partial W^{(2)}} = (a_i^{(2)})^T \in \mathbb{R}^{1\times 10} \tag{8}$$

So that :

$$\frac{\partial J}{\partial W^{(2)}} = \sum_{i=1}^{N} \frac{1}{N}\left(\psi(z_i^{(3)}) - \delta_{j,y_i}\right) \times (a_i^{(2)})^T \in \mathbb{R}^{3\times 10} \tag{9}$$

We can confirm and assess that the shape of $\frac{\partial J}{\partial W^{(2)}}$ is coherent with the shape of $W^{(2)}$.

We can see that the difference induced by the regularization therm to the partial derivative is just (by the definition of the Frobenius norm, the analogue of L_2 norm for vectors applied to matrixes, where $Tr()$ resemble the *trace*):

$$\frac{\partial \lambda \left(||W^{(1)}||_2^2 + ||W^{(2)}||_2^2\right)}{\partial W^{(2)}} = \frac{\partial \lambda Tr(W^{(2)} \cdot W^{(2)T})}{\partial W^{(2)}} = 2\lambda W^{(2)} \in \mathbb{R}^{3\times 10} \tag{10}$$

So that deriving the regularized loss we obtain:

$$\frac{\partial \tilde{J}}{\partial W^{(2)}} = \sum_{i=1}^{N} \frac{1}{N}\left(\psi(z_i^{(3)}) - \delta_{j,y_i}\right) \times (a_i^{(2)})^T + 2\lambda W^{(2)} \in \mathbb{R}^{3\times 10} \tag{11}$$

2.c)
We want now to compute, leveraging the chain rule, all the partial derivatives of the regularized loss $\tilde{J}$ w.r.t. all the others model's parameters, i.e.:

$$\frac{\partial \tilde{J}}{\partial W^{(1)}} \qquad \frac{\partial \tilde{J}}{\partial b^{(2)}} \qquad \frac{\partial \tilde{J}}{\partial b^{(1)}} \tag{12}$$

Given a general vector $x \in \mathbb{R}^k$, we know that the application of the *ReLu()* (that we will recall as function $\phi()$) to it is elementwise, i.e. $\phi(x) \in \mathbb{R}^k$. Even if $\phi()$ is a subdifferentiable function in the 0 (the limit of the incremental ratios are different if we approach from left or right), we assume that:

$$\frac{d\phi(x)}{dx} = \Theta(x) \in \mathbb{R}^{k \times k} \tag{13}$$

Where $\Theta()$ is the theta Heaviside function containing the non-zero elements only on the diagonal by the criteria:

$$\Theta(x)_{i,i} = \begin{cases} 0, & \text{if } x_i < 0 \\ 1, & \text{if } x_i \geq 0 \end{cases} \qquad \forall i \in [1, k] \tag{14}$$

We recall that $a_i^{(2)} \in \mathbb{R}^{10}$, $z_i^{(2)} \in \mathbb{R}^{10}$, $b_i^{(2)} \in \mathbb{R}^3$, $W^{(1)} \in \mathbb{R}^{10 \times 4}$ and $b_i^{(1)} \in \mathbb{R}^{10}$
2.c.1) $\frac{\partial \tilde{J}}{\partial W^{(1)}}$
By the chain rule:

$$\frac{\partial \tilde{J}}{\partial W^{(1)}} = \sum_{i=1}^{N} \frac{\partial \tilde{J}}{\partial z_i^{(3)}} \frac{\partial z_i^{(3)}}{\partial a_i^{(2)}} \frac{\partial a_i^{(2)}}{\partial z_i^{(2)}} \frac{\partial z_i^{(2)}}{\partial W^{(1)}} \tag{15}$$

Being:

$$\frac{\partial z_i^{(3)}}{\partial a_i^{(2)}} = \frac{\partial (W^{(2)} a_i^{(2)} + b^{(2)})}{\partial a_i^{(2)}} = W^{(2)} \in \mathbb{R}^{3 \times 10} \tag{16}$$

$$\frac{\partial a_i^{(2)}}{\partial z_i^{(2)}} = \frac{\partial \phi(z_i^{(2)})}{\partial z_i^{(2)}} = \Theta(z_i^{(2)}) \in \mathbb{R}^{10 \times 10} \tag{17}$$

$$\frac{\partial z_i^{(2)}}{\partial W^{(1)}} = F \in \mathbb{R}^{10 \times 10 \times 4} \tag{18}$$

Recalling that the k_{th} element of $z^{(2)}$ will be computed as $z_k^{(2)} = \sum_j W_{k,j}^{(1)} a_j^{(1)}$,
we have that $\frac{\partial (z^{(2)})_k}{\partial W_{y,j}^{(1)}} = F_{k,y,j} = a_j^{(1)} \cdot \delta_{y,k}$ indicating as δ the Kroenecker function. So adding the extra regularization derived therm and using the (6):

$$\frac{\partial \tilde{J}}{\partial W^{(1)}} = \sum_{i=1}^{N} \frac{1}{N} \left(\psi(z_i^{(3)}) - \delta_{j,y_i} \right) \times W^{(2)} \times \Theta(z_i^{(2)}) \times F + 2\lambda W^{(1)} \in \mathbb{R}^{10 \times 4} \tag{19}$$

We can confirm the dimensionality of $\frac{\partial \tilde{j}}{\partial W^{(1)}}$ assessing that:

$$\mathbb{R}^3 \times \mathbb{R}^{3 \times 10} \times \mathbb{R}^{10 \times 10} \times \mathbb{R}^{10 \times 10 \times 4} + \mathbb{R}^{10 \times 4} = \mathbb{R}^{10 \times 4} \tag{20}$$

2.c.2) $\frac{\partial \tilde{J}}{\partial b^{(2)}}$
By the chain rule:

$$\frac{\partial \tilde{J}}{\partial b^{(2)}} = \sum_{i=1}^{N} \frac{\partial \tilde{J}}{\partial z_i^{(3)}} \frac{\partial z_i^{(3)}}{\partial b^{(2)}} \tag{21}$$

Being:

$$\frac{\partial z_i^{(3)}}{\partial b^{(2)}} = \mathbb{I} \in \mathbb{R}^{3\times 3} \qquad \mathbb{I}_{i,j} = \begin{cases} 0, & \text{if } i \neq j \\ 1, & \text{if } i = j \end{cases} \tag{22}$$

We obtain using the therm computed at (6):

$$\frac{\partial \tilde{J}}{\partial b^{(2)}} = \sum_{i=1}^{N} \frac{1}{N} \left(\psi(z_i^{(3)}) - \delta_{j,y_i} \right) \times \mathbb{I} \in \mathbb{R}^{3} \tag{23}$$

2.c.3) $\frac{\partial \tilde{J}}{\partial b^{(1)}}$
By the chain rule:

$$\frac{\partial \tilde{J}}{\partial b^{(1)}} = \sum_{i=1}^{N} \frac{\partial \tilde{J}}{\partial z_i^{(3)}} \frac{\partial z_i^{(3)}}{\partial a_i^{(2)}} \frac{\partial a_i^{(2)}}{\partial z_i^{(2)}} \frac{\partial z_i^{(2)}}{\partial b^{(1)}} \tag{24}$$

Being:

$$\frac{\partial z_i^{(2)}}{\partial b^{(1)}} = \mathbb{X} \in \mathbb{R}^{10\times 10} \qquad \mathbb{X}_{i,j} = \begin{cases} 0, & \text{if } i \neq j \\ 1, & \text{if } i = j \end{cases} \tag{25}$$

And using the therms computed at (6), (16) and (17) we obtain:

$$\frac{\partial \tilde{J}}{\partial b^{(1)}} = \sum_{i=1}^{N} \frac{1}{N} \left(\psi(z_i^{(3)}) - \delta_{j,y_i} \right) \times W^{(2)} \times \Theta(z_i^{(2)}) \times \mathbb{X} \in \mathbb{R}^{10} \tag{26}$$

We can confirm the dimensionality of $\frac{\partial \tilde{j}}{\partial b^{(1)}}$ assessing that:

$$\mathbb{R}^3 \times \mathbb{R}^{3\times 10} \times \mathbb{R}^{10\times 10} \times \mathbb{R}^{10\times 10} = \mathbb{R}^{10} \tag{27}$$

Question 3: Stochastic Gradient Descent

3.a)

The following equation needs to be solved using Stochastic Gradient Descent

$$\theta^* = \underset{\theta}{\operatorname{argmin}}\, \tilde{J}(\theta, \{x_i, y_i\}_{i=1}^{N}) \tag{28}$$

Stochastic Gradient Descent accumulates the gradient over a small random subset of the training samples which is called mini-batch at each iteration. The output of the implementation is depicted in the following graph.

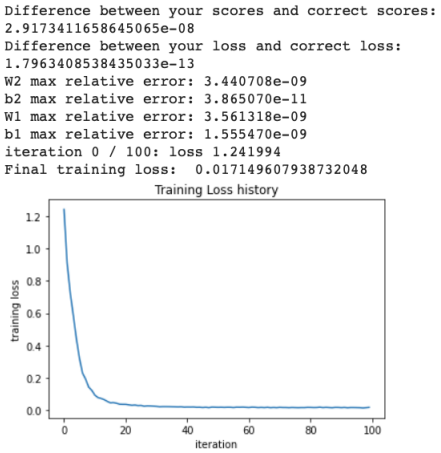
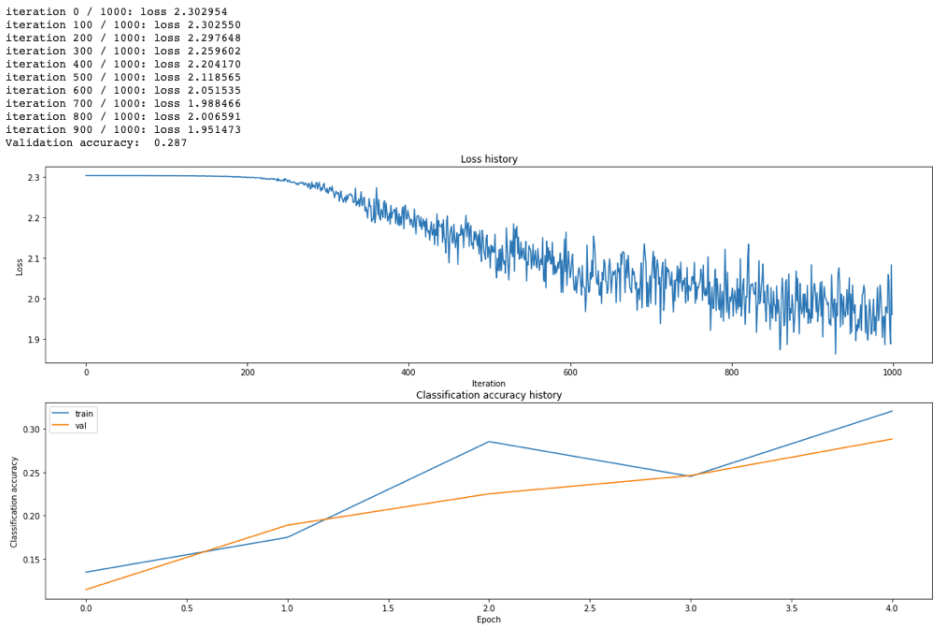


Figure 1: It's clear that final training loss is around 0.017 and the training curve agrees with the HW guidelines.

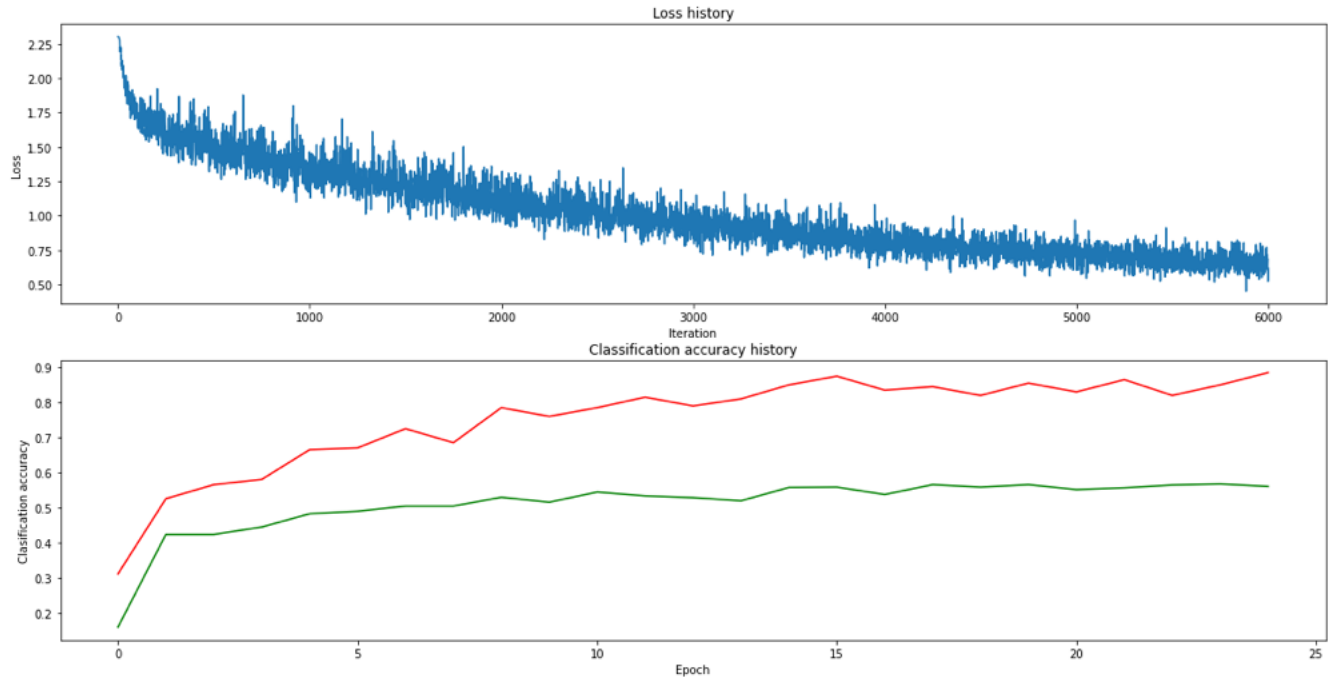
3.b) Improving model training with better hyper-parameters.

As it is depicted in the graph, the validation accuracy is about 0.29, which is not satisfactory.



In this section PCA is applied to reduce the dimensionality of the original images. PCA tries to preserve the essential parts that have more variation of the data and remove the non-essential parts with fewer variation. Our input size is 32 * 32 * 3 which has 3072 features that together represent the image. Here as the parameter, the value of 510 is passed to the PCA model. After applying PCA, the dimension is reduced to 510 principal components from the actual 3072 dimensions.
The next step is to analyze model training. After some investigations the results are as follows.
Learning rate: It controls how quickly the model is adapting to the problem, smaller learning rates require more training epochs but could make the model stuck, whereas larger learning rates could result the model to converge quickly to a sub-optimal solution. The challenge here is to select carefully the learning rate as it is one of the main hyper-parameters of the model. We have permuted different values for learning rate in combination with other hyper-parameters to come up with the learning rate value which is 1e-3.
Number of Neurons: The other important hyper-parameter is the number of neurons such that using few neurons in the hidden layers will result in underfitting where there are few neurons in the hidden layer to adequately detect the signal in a dataset, while using too many neurons may result in overfitting. In this part we also choose the corresponding value based on trying many values and came up with the number 600.
Other hyper-parameters are set as follows, num-iters equals 6000, batch-size equals 200, regularization strength is 0.0001 and finally, learning-rate-decay is set to 0.90. As it it clear in the final result the validation accuracy is above 0.57 and test accuracy is 0.566.

```
best validation accuracy achieved during cross-validation:0.575000
```




```
Test accuracy:  0.566
```

Question 4: Implement MLP using PyTorch library

In this question we are assigned to implement the multi-layer perceptron by PyTorch library.

4.a) implementing a two-layer perceptron network in the class of MultiLayerPerceptron
This part has been implemented on **ex2-pytorch.py**

4.b) Validation accuracy with two-layer network

In the light of the Q3, the used hidden layer has 550 perceptron, learning rate of 0.001, ReLU as the activation function, batch size of 200 and 20 epochs, which resulted a Validation accuracy of 56.5 % and Test accuracy of 54.5 %.

4.c) Increasing the layers and comparing the results.

By increasing the number of layers, we observed the validation accuracy increased except the last 4 hidden layer network which slightly dropped.
The accuracies were as follows:

2-layer network with the size of 550 as the hidden layer: Val acc.= **56.7%**

3-layer network with the size of 550 and 128 as the hidden layers: Val acc.= **56.6%**

4-layer network with the size of 550, 128, 100 as the hidden layers: Val acc.= **57.4%**

5-layer network with the size of 550, 250, 180, 120 as the hidden layers: Val acc.= **55.6%**

These numbers resulted from the model with learning rate of 0.001, ReLU as the activation function and 20 epochs.

In this part we implemented different parameters and different models in terms of architecture.
For instance, at the end of each layer a dropout layer implemented, activation functions changed, used different learning rates and epochs. But the dropout layers slightly reduced the validation accuracy to around 55%, On the other hand using **LeakyReLU** as our activation function had the most impact.

finally our best model was the 4-layer network with 3 hidden layers of 550, 128, 100, lr = 0.001, LeakyReLU as activation function and 25 epochs which resulted of **57.5%** of **Validation** accuracy and **56%** of **Test** accuracy