Advanced Machine Learning - Assignment 2

Avitabile Luca 1707378, Broglio Matteo 1200739,
Colombi Marco 1713520, Loretucci Lorenzo 1903794
Github repository:
https://github.com/lorenzoloretucci/Deep-Neural-Network-and-Backpropagation

19th November 2020

1 Implementing the feedforward model

Nothing to report.

2 Backpropagation

2.1

Given:

$$\psi(z_i^{(3)}) = \frac{e^{z_i^3}}{\sum_{j=1}^{K} e^{(z_i^{(3)})_j}} \tag{1}$$

and:

$$\psi'(z_i^{(3)}) = \psi(z_i^{(3)})(\Delta_{i,j} - \psi(z_i^{(3)})) \tag{2}$$

and:

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

then:

$$\begin{aligned}\frac{\partial J}{\partial z_i^{(3)}}(\theta, \{x_i, y_i\}_{i=1}^N) &= \frac{\partial J}{\partial z_i^{(3)}} - \frac{1}{N}\sum_{i=1}^{N} log(\psi(z_i^{(3)})) = -\frac{1}{N}\sum_{i=1}^{N}\frac{\partial J}{\partial z_i^{(3)}} log(\psi(z_i^{(3)})) = \\ &= -\frac{1}{N}\sum_{i=1}^{N}\frac{1}{\psi(z_i^{(3)})}\psi'(z_i^{(3)})\mathbb{1}_i = -\frac{1}{N}\sum_{i=1}^{N}\frac{1}{\psi(z_i^{(3)})}\psi(z_i^{(3)})(\Delta_{i,j} - \psi(z_i^{(3)}))\mathbb{1}_i = \\ &= -\frac{1}{N}\sum_{i=1}^{N}(\Delta_{i,j} - \psi(z_i^{(3)}))\mathbb{1}_i = \frac{1}{N}\sum_{i=1}^{N}(\psi(z_i^{(3)}) - \Delta_{i,j})\mathbb{1}_i = \frac{1}{N}(\psi(z_i^{(3)}) - \Delta_{i,j})\end{aligned} \tag{4}$$

2.2

Given:

$$\frac{\partial z_i^{(3)}}{\partial W^{(2)}} = \frac{\partial z_i^{(3)}}{\partial W^{(2)}} W^{(2)} a^{(2)} + b^{(2)} = {a_i^{(2)}}^T \tag{5}$$

from (4) and (5):

$$\begin{aligned}\frac{\partial J}{\partial W^{(2)}}(\theta, \{x_i, y_i\}_{i=1}^N) = \sum_{i=1}^{N}\frac{\partial J}{\partial z_i^{(3)}}\cdot\frac{\partial z_i^{(3)}}{\partial W^{(2)}} &= \sum_{i=1}^{N}\frac{1}{N}(\psi(z_i^{(3)}) - \Delta_{i,j})\cdot\frac{\partial z_i^{(3)}}{\partial W^{(2)}} = \\ &= \sum_{i=1}^{N}\frac{1}{N}(\psi(z_i^{(3)}) - \Delta_{i,j})\, {a_i^{(2)}}^T\end{aligned} \tag{6}$$

Then, given:

$$\frac{\partial}{\partial W^{(2)}}\lambda\left(\left\|W^{(1)}\right\|_2^2\left\|W^{(2)}\right\|_2^2\right)=\frac{\partial}{\partial W^{(2)}}\lambda\left(\sum_{p=1}^{P}\sum_{q=1}^{Q}\left(W_{p,q}^{(1)}\right)^2+\sum_{p=1}^{P}\sum_{q=1}^{Q}\left(W_{p,q}^{(2)}\right)^2\right)= \\ =\lambda\left(\sum_{p=1}^{P}\sum_{q=1}^{Q}2\left(W_{p,q}^{(2)}\right)\right)=2\lambda\left(\sum_{p=1}^{P}\sum_{q=1}^{Q}\left(W_{p,q}^{(2)}\right)\right)=2\lambda W^{(2)} \tag{7}$$

and:

$$\tilde{J}(\theta,\{x_i,y_i\}_{i=1}^N)=\frac{1}{N}\sum_{i=1}^{N}-log\left[\frac{e^{z_i^3}}{\sum_{j=1}^{K}e^{(z_i^{(3)})_j}}\right]+\lambda\left(\left\|W^{(1)}\right\|_2^2\left\|W^{(2)}\right\|_2^2\right) \tag{8}$$

from (7) and (8):

$$\frac{\partial\tilde{J}}{\partial W^{(2)}}=\frac{\partial J}{\partial W^{(2)}}+\frac{\partial}{\partial W^{(2)}}\lambda\left(\left\|W^{(1)}\right\|_2^2\left\|W^{(2)}\right\|_2^2\right)=\sum_{i=1}^{N}\frac{1}{N}(\psi(z_i^{(3)})-\Delta_{i,j})\,{a_i^{(2)}}^T+2\lambda W^{(2)} \tag{9}$$

2.3

$$\frac{\partial z^{(3)}}{\partial b^{(2)}}=1\Rightarrow\frac{\partial J}{\partial b^{(2)}}=\frac{\partial J}{\partial z^{(3)}}\cdot\frac{\partial z^{(3)}}{\partial b^{(2)}}=\frac{1}{N}(\psi(z_i^{(3)})-\Delta_{i,j}) \tag{10}$$

$$\frac{\partial z^{(3)}}{\partial a^{(2)}}={W^{(2)}}^T\Rightarrow\frac{\partial J}{\partial a^{(2)}}=\frac{\partial J}{\partial z^{(3)}}\cdot\frac{\partial z^{(3)}}{\partial a^{(2)}}=\frac{1}{N}(\psi(z_i^{(3)})-\Delta_{i,j})\cdot{W^{(2)}}^T \tag{11}$$

$$\frac{\partial a^{(2)}}{\partial z^{(2)}}=\begin{cases}1\ if\ z_i^{(2)}>0\\0\ otherwise\end{cases}=\mathbb{1}_{z_i^{(2)}>0}\Rightarrow\frac{\partial J}{\partial z^{(2)}}=\frac{\partial J}{\partial a^{(2)}}\cdot\frac{\partial a^{(2)}}{\partial z^{(2)}}= \\ =\frac{1}{N}(\psi(z_i^{(3)})-\Delta_{i,j})\cdot{W^{(2)}}^T\mathbb{1}_{z_i^{(2)}>0} \tag{12}$$

$$\frac{\partial z^{(2)}}{\partial b^{(1)}}=1\Rightarrow\frac{\partial J}{\partial b^{(1)}}=\frac{\partial J}{\partial z^{(2)}}\cdot\frac{\partial z^{(2)}}{\partial b^{(1)}}=\frac{1}{N}(\psi(z_i^{(3)})-\Delta_{i,j})\cdot{W^{(2)}}^T\mathbb{1}_{z_i^{(2)}>0} \tag{13}$$

$$\frac{\partial z^{(2)}}{\partial W^{(1)}}={a^{(1)}}^T\Rightarrow\frac{\partial J}{\partial W^{(1)}}=\frac{\partial J}{\partial z^{(2)}}\cdot\frac{\partial z^{(2)}}{\partial W^{(1)}}= \\ =\frac{1}{N}(\psi(z_i^{(3)})-\Delta_{i,j})\cdot{W^{(2)}}^T\mathbb{1}_{z_i^{(2)}>0}\cdot{a^{(1)}}^T \tag{14}$$

3 Stochastic gradient descent training

After a first training where we obtained 28% on validation set accuracy, we experimented with different hyper-parameters and techniques. The best way among the ones that we tried was standardizing the data using the following formula:

$$X\_train\_standardized=\frac{X\_data-\mu}{\sigma}.$$

Then we applied the PCA on the standardized data, to further improving the performance of our model, maintaining the 90% of the variance of our data. From an hyper-parameters point of view, we decided to implement a Grid-Search from scratch looking for the best combination of *hidden size*, *learning rate*, *regularization term* and *number of iterations*. The best combination of hyper-parameters, that gave us a validation accuracy of **56,3%**, was:

Best performance = {hidden_size': 256, 'num_iters': 5000, 'learning_rate': 0.1, 'reg': 0.001}.

Using this configuration, we plot the following picture:

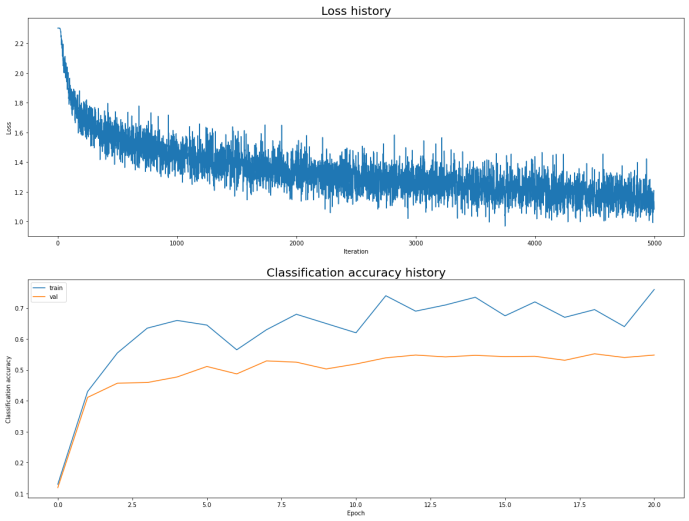

Figure 1: History of the best model

From Figure(1) we can see how our best model was trained. Indeed after several runs and experiments we observed that with a fixed *batch size* = 200 the best way to improve Accuracy was increasing the *number of iterations* (from 1000 to 5000) and *learning rate* (from 0.001 to 0.1). In fact the Loss history in Figure(1) looks like a curve (smooth in range 4000-5000) and not a simple line as before (default hyper-parameters).
In the end, the accuracy that we got on the test set was **54,8%**.

4 Implement multi-layer perceptron using PyTorch library

Point 4b)

With a simple two-layer network (only ReLU as activation layer) and the default option: Hidden_size = [50], num_classes = 10, num_epochs = 10, batch_size = 200, learning_rate = 1e-3, learning_rate_decay = 0.95 and reg = 0.001.

We obtained a final validation accuracy of **51.2%**.

Point 4c)

After the performance above we did different experiment with networks of 2 to 6 layers and default hyper-parameters combination. For the best model the parameters that we changed were *hidden_size*= [[256], [256,128], [256,128,64], [256,128,64,32], [256,128,64,32,16]] and the *number of layers*. Another important improvement was adding *nn.BatchNorm1d()* between each layers with *nn.ReLU()* as activation function.

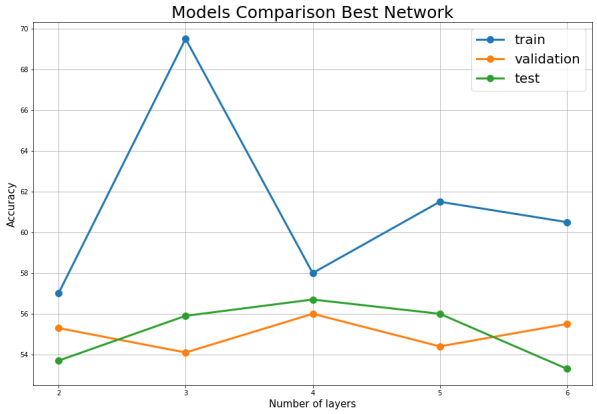

Figure 2: Accuracy for each layer setup after 10 epochs.

In Figure(2) are reported all our Accuracies for each layers on the last epoch. The best set-up is the 4 layer model with a test accuracy of **56,7%** (training accuracy: **58%**, validation accuracy: **56%**). Indeed this setup doesn't show overfitting compared to others, like the one with 3 layers.

For a further analysis on the behavior of the accuracy and loss while changing the depth of the neural network, we can see the plot below (Figure 3).

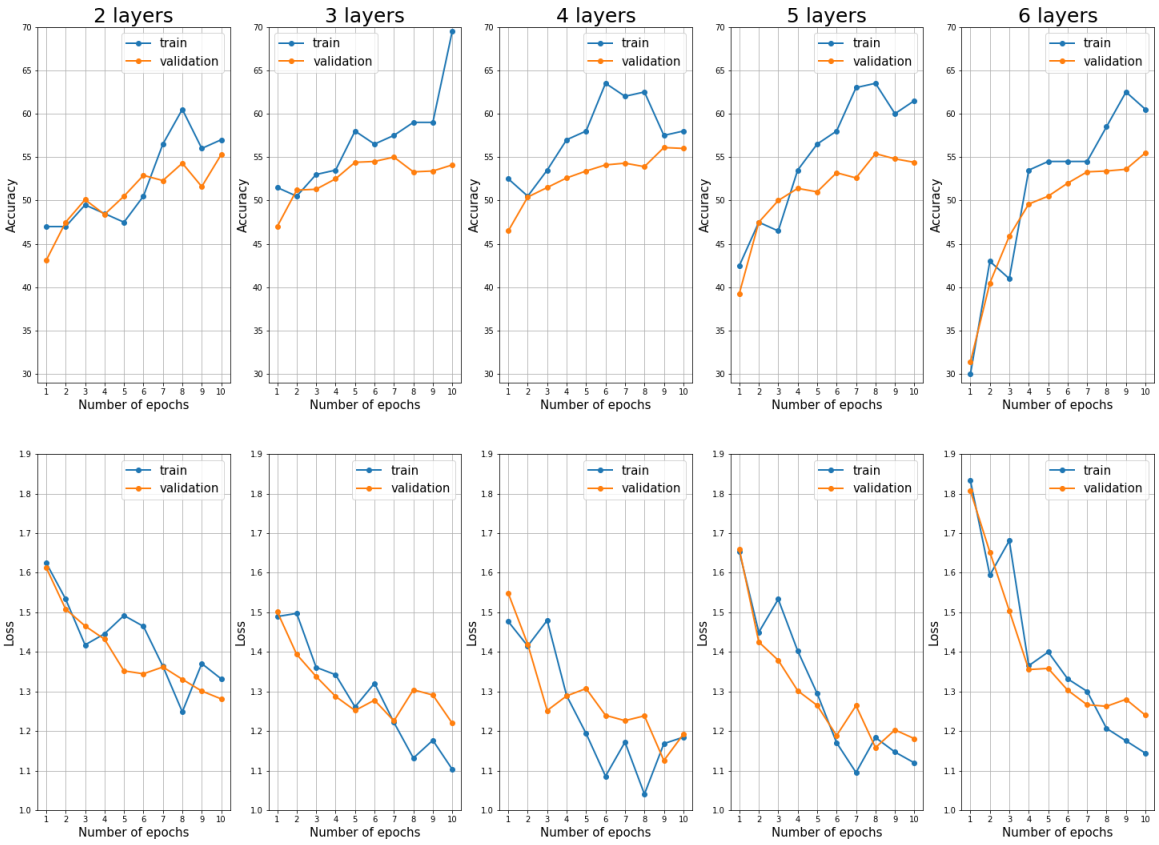

Figure 3: History of the best network

We can observe when the number of layers increases, the accuracy on both train and validation start from a lower value and increase their positive trend (and the delta of the accuracy from the first and last epoch is higher). In this specific problem this is due to the complexity of the model, as an higher complexity needs more "time"(epochs) to converge to a reasonable value (accuracy/loss) and starts from a worst condition.