



SAPIENZA
UNIVERSITÀ DI ROMA

Exercise 2: Deep Neural Network and Backpropagation

Feola Luigi 1772197, Caciccia Fabrizio 1869364, Yann
Leterrier 1924402

April 2020

Question 1

Question 2: Backpropagation

Question 2a-b

In the following, for simplicity in the notation, we will write J and \tilde{J} avoiding their own parameters.

Remembering that:

$$\frac{d \log(f(x))}{dx} = \frac{1}{f(x)} \frac{df(x)}{dx} \quad (1)$$

and evaluating the softmax function and its derivative:

$$\psi(z^{(3)})_{y_i} = \frac{e^{z_{y_i}^{(3)}}}{\sum_{j=1}^K e^{z_j^{(3)}}}$$
$$\frac{\partial \left(\frac{e^{z_{y_i}^{(3)}}}{\sum_{j=1}^K e^{z_j^{(3)}}} \right)}{\partial z_j^{(3)}} = \begin{cases} \psi(z^{(3)})_{y_i} (1 - \psi(z^{(3)})_j), & \text{if } y_i = j \\ -\psi(z^{(3)})_{y_i} \psi(z^{(3)})_j, & \text{otherwise} \end{cases}$$

or equivalently:

$$\psi(z^{(3)})_{y_i} \left(\Delta - \psi(z^{(3)})_j \right) \quad (2)$$

where:

$$\delta_{i,j} \in \Delta = \begin{cases} 1, & \text{if } y_i = j \\ 0, & \text{otherwise} \end{cases}$$

We verify that the partial derivative of the loss w.r.t. $W^{(2)}$ is:

$$\frac{\partial \tilde{J}}{\partial W^{(2)}} = \frac{\partial J}{\partial z^{(3)}} \frac{\partial z^{(3)}}{\partial W^{(2)}} + 2\lambda W^{(2)} = \frac{1}{N} \left(\psi(z^{(3)}) - \Delta \right) a^{(2)'} + 2\lambda W^{(2)}$$

since:

- using (1)-(2) we get:

$$\frac{\partial J}{\partial z^{(3)}} = \frac{1}{N} \sum_{i=1}^N \frac{-1}{\psi(z^{(3)})_{y_i}} \frac{\partial \psi(z^{(3)})_{y_i}}{\partial z^{(3)}} = \frac{1}{N} \sum_{i=1}^N \frac{\psi(z^{(3)})_{y_i}}{\psi(z^{(3)})_{y_i}} (\psi(z^{(3)}) - \Delta)$$

- $\frac{\partial z^{(3)}}{\partial W^{(2)}} = \frac{\partial (W^{(2)} a^{(2)})}{\partial W^{(2)}} = a^{(2)'}$ (since $b^{(2)}$ does not depend on $W^{(2)}$)

- $2\lambda W^{(2)}$ is the derivative of the regularization part of the loss $\lambda \left\| \begin{pmatrix} W^{(1)} \end{pmatrix} + \begin{pmatrix} W^{(2)} \end{pmatrix} \right\|_2^2$ w.r.t. $W^{(2)}$.

2c

Applying recursively the Chain-rule applied to derivatives, we get the following Upstream/Local gradients.

Upstream gradient:

$$\frac{\partial J}{\partial z^{(2)}} = \frac{\partial J}{\partial \psi(z^{(3)})} \frac{\partial \psi(z^{(3)})}{\partial z^{(3)}} \frac{\partial z^{(3)}}{\partial a^{(2)}} \frac{\partial a^{(2)}}{\partial z^{(2)}}$$

Gradient w.r.t. $W^{(1)}$ and $b^{(1)}$:

$$\begin{aligned} \frac{\partial \tilde{J}}{\partial W^{(1)}} &= \frac{\partial J}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial W^{(1)}} + 2\lambda W^{(1)} \\ &= a^{(1)T} \frac{1}{N} \left(\psi(z^{(3)}) - \Delta \right) W_2^T I_{a_2 > 0} + 2\lambda W^{(1)} \end{aligned}$$

$$\begin{aligned} \frac{\partial \tilde{J}}{\partial b^{(1)}} &= \frac{\partial J}{\partial z^{(2)}} \\ &= \frac{1}{N} \left(\psi(z^{(3)}) - \Delta \right) W_2^T I_{a_2 > 0} \end{aligned}$$

Upstream gradient:

$$\frac{\partial J}{\partial z^{(3)}} = \frac{\partial J}{\partial \psi(z^{(3)})} \frac{\partial \psi(z^{(3)})}{\partial z^{(3)}}$$

Gradient w.r.t. $W^{(2)}$ and $b^{(2)}$:

$$\begin{aligned} \frac{\partial \tilde{J}}{\partial W^{(2)}} &= \frac{\partial J}{\partial z^{(3)}} \frac{\partial z^{(3)}}{\partial W^{(2)}} + 2\lambda W^{(2)} \\ &= a^{(2)T} \frac{1}{N} \left(\psi(z^{(3)}) - \Delta \right) + 2\lambda W^{(2)} \end{aligned}$$

$$\begin{aligned} \frac{\partial \tilde{J}}{\partial b^{(2)}} &= \frac{\partial J}{\partial z^{(3)}} \\ &= \frac{1}{N} \left(\psi(z^{(3)}) - \Delta \right) \end{aligned}$$

Question 3: Stochastic gradient descent training

Question 3b

Our hyper-parameters tuning gives us a 55.4% on the test set.

We first searched the best hidden size since we thought that the performances was low because the network was not powerful enough.

So we trained the network with different hidden sizes, going from 10 to 340 obtaining the following results:

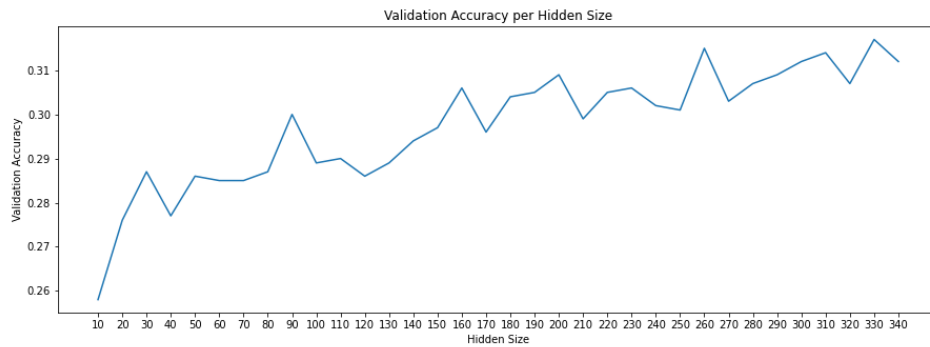


Figure 1: Validation Accuracy per Hidden Size.

From this search we noticed that the bigger is the size of the hidden layer the better the network performs.

We then thought that another issue could be a wrong learning rate. So, we search among a set of different learning rates as shown in Figure 2

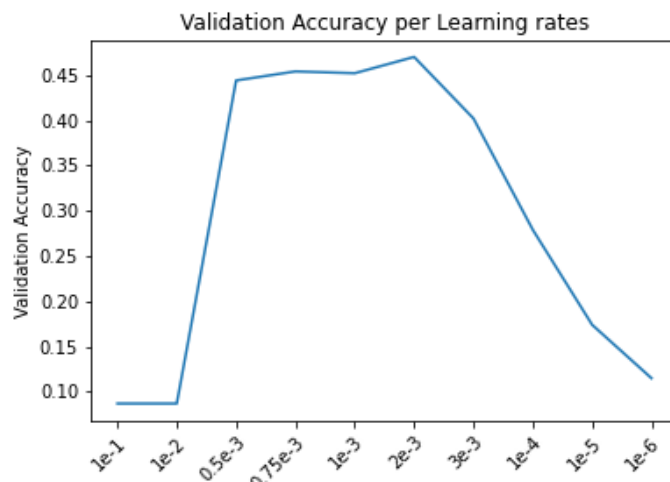


Figure 2: Validation Accuracy per Learning rates.

Here we observed that too high and too low values bring the network to achieve low accuracy. The learning rates that performed better was the one from $0.5e-3$ to $2e-3$.

With these values we ran a grid search over the hidden sizes and the best learning rates we found from the previous two research. The results of the search are shown in Figure 6.

What we found is that the best learning rate overall is $2e-3$, which performs almost always better. We also noticed that, for that particular learning rate, the best values for the size of the hidden layer are between 160 and 230 or above 300. But since the best overall was 180, we chose this particular value.

Next, with these two values, we investigate the effect of longer training sessions by increasing the number of iterations per trials as show in Figure 3.

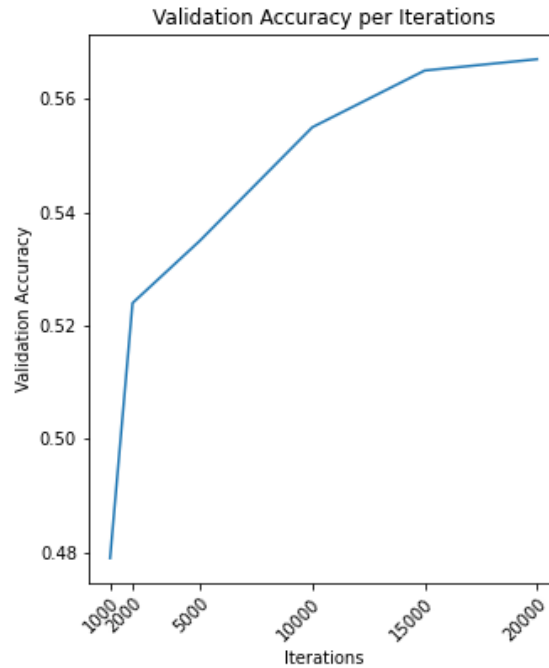


Figure 3: Validation Accuracy per iterations.

These results would suggest that increasing the number of iterations further could improve the validation accuracy more. But since the growth trend seems to slowdown, we decided to stop there and use 20000 as number of training iterations to use for the final test.

In Figure 5 we show the details of the training process with the final setup. The model trained with this configuration obtained a score of 55.4% accuracy on the test set.

The model could definitely be further improved. The plot clearly shows that the

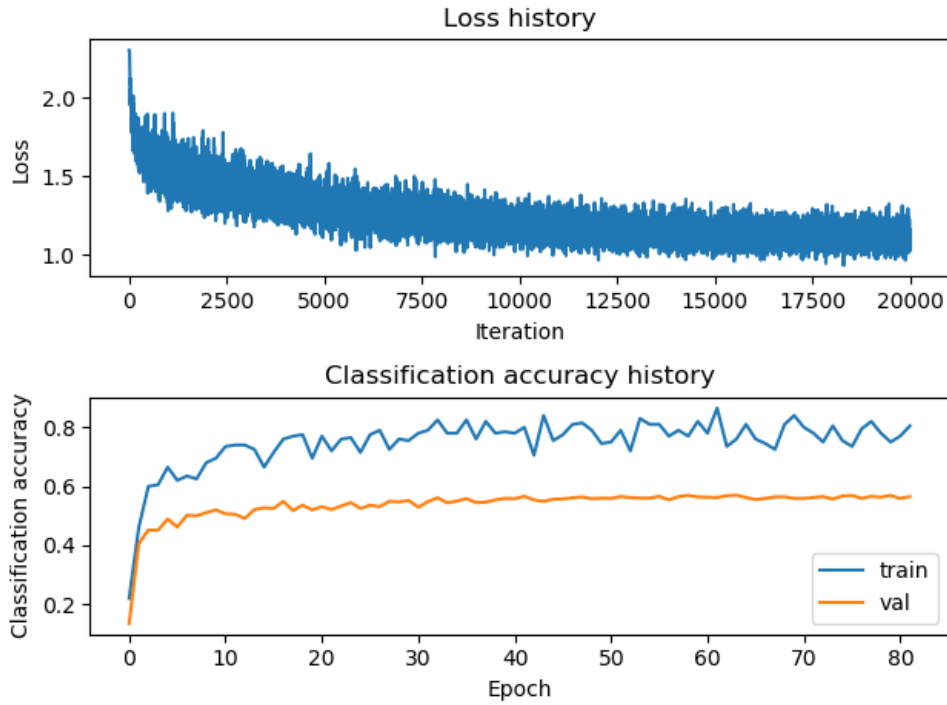


Figure 4: Training process using the best founded model.

training loss oscillates too much, symptom that some sort of regularization could be needed. But the result seems enough to us for a manually build neural network.

Question 4

Without initialising a seed of course, we get different results each time we execute the training (even if the model is not changed), but the final results are closer to that in the following tabular. Actually, these poor results using more than 3 layers are explained by the fact that the initial parameters are no longer relevant for the neural network. Changing learning rate, batch size and initialisation, number of epochs, and so on, we achieve better results. We also decided to add some batch normalisation and dropout layers to strengthen the network. We did not other activation layers than ReLU because the accuracy was not better with them.

Finally, our best model is (see the code to see the parameters value) :

```
nn.Linear(input_size , hidden_layers [0])
nn.BatchNorm1d(num_features=hidden_layers [0])
nn.ReLU()
nn.Dropout(p=0.4)

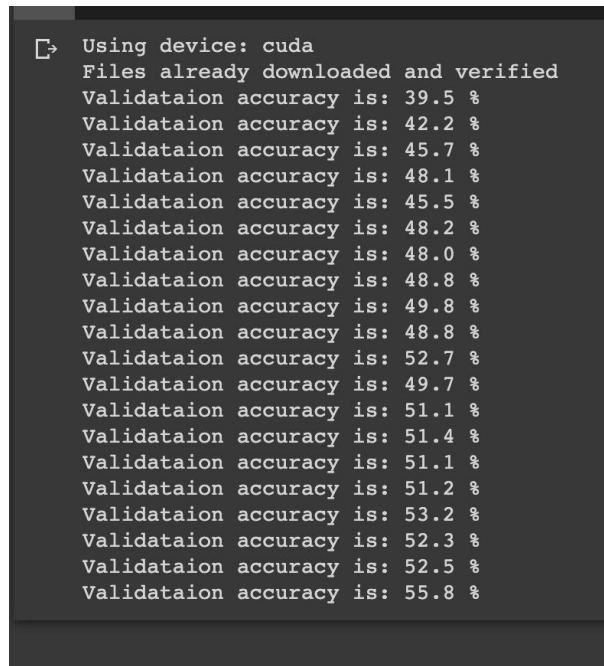
for hidden_layer in hidden_layers:
    nn.Linear(hidden_layer , hidden_layer)
    nn.BatchNorm1d(num_features=hidden_layer)
    nn.ReLU()
    nn.Dropout(p=0.4)

nn.Linear(hidden_layers [-1], num_classes)
```

Actually, we expected better results than question 3. We would have use me specific layers for that kind of classification, such that convolutional and pooling layers but we thought that is was beyond what was required for this exercise.

# nn.Linear layers	Neurons in the last nn.Linear layer	Validation Accuracy (%)
1	50	50.8
2	40	50.6
3	30	8.7
4	20	7.8

These were the results of the best model:



```
Using device: cuda
Files already downloaded and verified
Validataion accuracy is: 39.5 %
Validataion accuracy is: 42.2 %
Validataion accuracy is: 45.7 %
Validataion accuracy is: 48.1 %
Validataion accuracy is: 45.5 %
Validataion accuracy is: 48.2 %
Validataion accuracy is: 48.0 %
Validataion accuracy is: 48.8 %
Validataion accuracy is: 49.8 %
Validataion accuracy is: 48.8 %
Validataion accuracy is: 52.7 %
Validataion accuracy is: 49.7 %
Validataion accuracy is: 51.1 %
Validataion accuracy is: 51.4 %
Validataion accuracy is: 51.1 %
Validataion accuracy is: 51.2 %
Validataion accuracy is: 53.2 %
Validataion accuracy is: 52.3 %
Validataion accuracy is: 52.5 %
Validataion accuracy is: 55.8 %
```

Figure 5

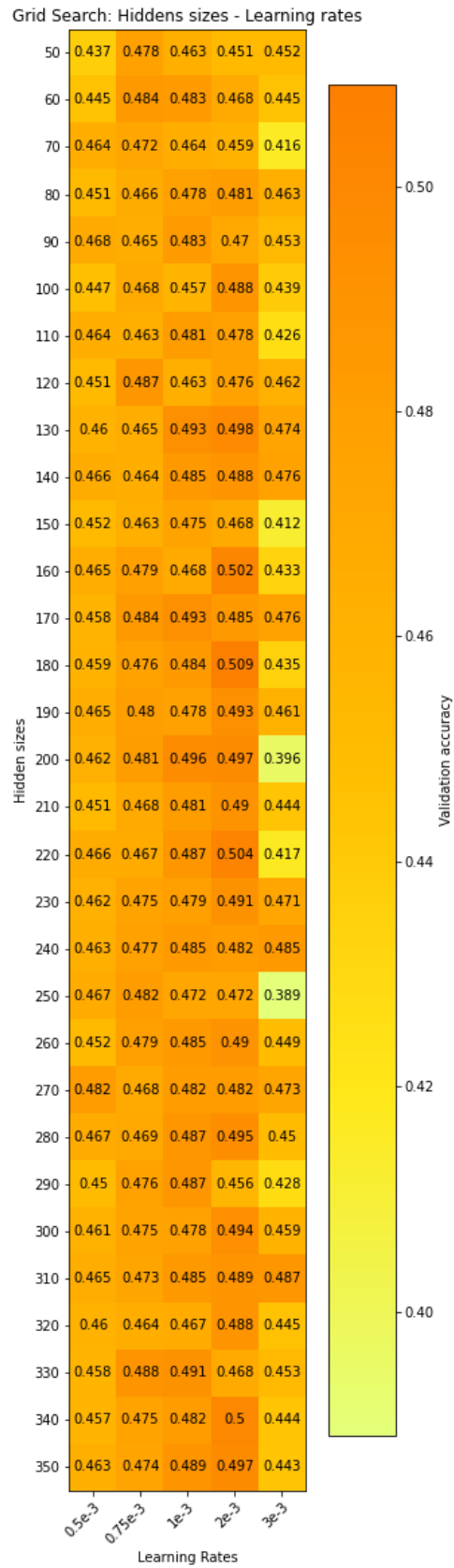


Figure 6: Grid Search over different Hidden sizes and Learning rates