Advanced Machine Learning
Deep Neural Network and Backpropagation

Theodoros Sofianos (1867968) Katsiaryna Zavadskaya (1847985)

April 2020

1 Backpropagation

1.1 $\frac{\partial J}{\partial z^{(3)}}$

We want to verify that the loss function defined in equation 1 has the gradient w.r.t. $z^{(3)}$ as in equation 2.

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

$$\frac{\partial J}{\partial z^{(3)}}(\theta, \{x_i, y_i\}_{i=1}^{N}) = \frac{1}{N}\left(\psi(z^{(3)}) - \Delta\right), \quad \text{where} \quad \Delta = \begin{cases} 1 & \text{if } y_i = j \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

Derivative of Softmax for 1 input with respect to 1 neuron:

$$\frac{\partial a_i^{(3)}}{\partial z_j^{(3)}} = \frac{\partial \frac{e^{z_i^{(3)}}}{\sum_k^C e^{z_k^{(3)}}}}{\partial z_j^{(3)}}$$

Using the quotient rule of derivatives we can rewrite this as:

1. for $i = j$:

$$\frac{\partial a_i^{(3)}}{\partial z_j^{(3)}} = \frac{e^{z_i}\cdot\sum_{k=1}^{C} e^{z_k^{(3)}} - e^{z_j^{(3)}} e^{z_i^{(3)}}}{\left(\sum_{k=1}^{C} e^{z_k^{(3)}}\right)^2} = \frac{e^{z_i}}{\sum_{k=1}^{C} e^{z_k}}\cdot\frac{\sum_{k=1}^{C} e^{z_k^{(3)}} - e^{z_j^{(3)}}}{\sum_{k=1}^{C} e^{z_k^{(3)}}} = a_i^{(3)}\cdot(1 - a_i^{(3)}), \quad \text{since} \quad i = j$$

2. for $i \neq j$:

$$\frac{\partial a_i^{(3)}}{\partial z_j^{(3)}} = \frac{\partial \frac{e^{z_i^{(3)}}}{\sum_k^C e^{z_k^{(3)}}}}{\partial z_j^{(3)}} = \frac{0 - e^{z_j^{(3)}} e^{z_i^{(3)}}}{\left(\sum_{k=1}^{C} e^{z_k^{(3)}}\right)^2} = -\frac{e^{z_j^{(3)}}}{\sum_{k=1}^{C} e^{z_k^{(3)}}}\cdot\frac{e^{z_i^{(3)}}}{\sum_{k=1}^{C} e^{z_k^{(3)}}} = -a_j^{(3)}\cdot a_i^{(3)}$$

$$\text{so} \quad \frac{\partial a_i^{(3)}}{\partial z_j^{(3)}} = \begin{cases} a_i^{(3)}\cdot(1 - a_i^{(3)}), & \text{if} \quad i = j \\ -a_j^{(3)}\cdot a_i^{(3)}, & \text{if} \quad i \neq j \end{cases}$$

Derivative of Cross Entropy for 1 input w.r.t last layer:

$$\frac{\partial J(\theta, X, Y)}{\partial a^{(3)}} = \frac{\partial - Y\cdot log(a^{(3)})}{\partial a^{(3)}} = -Y\cdot\frac{1}{a^{(3)}} = -\frac{Y}{a^{(3)}}$$

,where Y is the one hot encoded vector of our target label.

Derivative of Cross Entropy for 1 input of the Softmax layer:

$$\frac{\partial J(\theta)}{\partial z_i^{(3)}} = \frac{-\sum_{k=1}^{C} y_k\cdot log(a_k^{(3)})}{z_i^{(3)}} = \quad \text{from the chain rule} \quad = -\sum_{k=1}^{C} y_k\cdot\frac{\partial log(a_k^{(3)})}{\partial a_k^{(3)}}\cdot\frac{\partial a_k^{(3)}}{\partial z_i^{(3)}} = -\sum_{k=1}^{C} y_k\cdot\frac{1}{a_k^{(3)}}\cdot\frac{\partial a_k^{(3)}}{\partial z_i^{(3)}} =$$

$$= \text{as we have seen, Softmax takes different values for } i = j \text{ and } i \neq j =$$

$$= \left[\frac{y_i}{a_i^{(3)}} \cdot \frac{\partial a_i^{(3)}}{\partial z_i^{(3)}} + \sum_{k=1,k\neq i}^{C} \frac{y_k}{a_k^{(3)}} \cdot \frac{a_k^{(3)}}{z_i^{(3)}} \right] = -\frac{y_i}{a_i^{(3)}} \cdot a_i^{(3)}(1-a_i^{(3)}) - \sum_{k=1,k\neq i}^{C} \frac{y_k}{a_k^{(3)}} \cdot (-a_k^{(3)} a_i^{(3)}) =$$

$$= -y_i(1-a_i^{(3)}) + \sum_{k=1,k\neq i}^{C} y_k \cdot a_i^{(3)} = -y_i + y_i a_i^{(3)} + \sum_{k=1,k\neq i}^{C} y_k \cdot a_i^{(3)} =$$

$$= a_i^{(3)} \cdot (y_i + \sum_{k=1,k\neq i}^{C} y_k) - y_i = a_i^{(3)} \cdot \sum_{k=1}^{C} y_- y_i$$

But since y is one-hot, its sums is 1.

$$\Rightarrow \frac{\partial J(\theta)}{\partial z_i^{(3)}} = a_i^{(3)} - y_i$$

We can write this in terms of vectors:

$$\Rightarrow \frac{\partial J(\theta)}{\partial z^{(3)}} = a^{(3)} - Y, \quad \text{where } Y \text{ is one-hot vector of the labels} \tag{3}$$

Now for multiple inputs, the Cross Entropy loss become

$$J(\theta) = \frac{1}{N} \cdot Y \cdot (-log(a^{(3)}), \quad \text{where } Y \text{ is a matrix, as well as } a^{(3)}$$

$$\frac{\partial J(\theta)}{\partial a^{(3)}} = -\frac{1}{N} \cdot \frac{Y}{a^{(3)}}, \quad \text{same as before, just averaged}$$

If we follow the chain rule, we will have to multiply the gradient matrix with the Jacobian of the Softmax. Since Softmax layer is a matrix for multiple inputs, its Jacobian will be a Tensor. We can skip this hard computations by using the formula 3 from before. Since $\frac{\partial J(\theta)}{\partial a^{(3)}}$ is now averaged for multiple inputs the gradient will be $\frac{1}{N} \cdot (a^{(3)} - Y) = \frac{1}{N} \cdot (\psi(z^{(3)}) - \Delta)$, where Δ is one-hot of the labels.

1.2 $\frac{\partial J}{\partial W^{(2)}}$ and $\frac{\partial \tilde{J}}{\partial W^{(2)}}$

We need to compute the effect of the weight matrix $W^{(2)}$ on the loss in equation 1. This is done by applying the chain rule.

$$\frac{\partial J}{\partial W^{(2)}} = \frac{\partial J}{\partial z^{(3)}} \cdot \frac{\partial z^{(3)}}{\partial W^{(2)}} = \frac{1}{N} \cdot (\psi(z^{(3)}) - \Delta) \cdot \frac{W^{(2)} a^{(2)} + b^{(2)}}{\partial W^{(2)}} = \frac{1}{N} \cdot (\psi(z^{(3)}) - \Delta) \cdot a^{(2)^T}$$

Note that in the above formula,we have the 2nd matrix transposed, since the shape of the gradients must be the same as the shape of their respective layers.Also note that in this homework we have used the forward pass of z=xW+b rather than z=Wx+b.That means that when calculating the derivatives w.r.t. the weight matrices and biases, the order of the chain rule must be inversed,and consequently, the matrices' multiplications. The report was written before the correction of the homework, so all the derivatives are calculated based on the forward pass of z=Wx+b,although in our code we have used z=xW+b, as per the instructions. Similarly, we need to compute the derivatives of the regularized loss from the equation 4:

$$\tilde{J}(\theta, \{x_i, y_i\}_{i=1}^{N}) = \frac{1}{N} \sum_{i=1}^{N} -log \left[\frac{e^{z_{y_i}^{(3)}}}{\sum_{j=1}^{K} e^{z_j^{(3)}}} \right] + \lambda \left(\left\| W^{(1)} \right\|_2^2 + \left\| W^{(2)} \right\|_2^2 \right) \tag{4}$$

So here we have:

$$\frac{\partial \tilde{J}}{\partial W^{(2)}} = \frac{\partial J}{\partial W^{(2)}} + \frac{\partial J_{l2}}{\partial W^{(2)}} =$$

$$= \frac{1}{N} \cdot (\psi(z^{(3)}) - \Delta) \cdot a^{(2)^T} + \frac{\lambda \left(\|W^{(1)}\|_2^2 + \|W^{(2)}\|_2^2 \right)}{\partial W^{(2)}} = \frac{1}{N} \cdot (\psi(z^{(3)}) - \Delta) \cdot a^{(2)^T} + 2 \cdot \lambda \cdot W^{(2)}$$

1.3 $\frac{\partial \tilde{J}}{\partial W^{(1)}}$, $\frac{\partial \tilde{J}}{\partial b^{(2)}}$ and $\frac{\partial \tilde{J}}{\partial b^{(1)}}$

Now we want to find expressions for the derivatives of the regularized loss in equation 4 w.r.t. $W^{(1)}, b^{(1)}, b^{(2)}$.

$$\frac{\partial \tilde{J}}{\partial W^{(1)}} = \frac{\partial J}{\partial W^{(1)}} + \frac{\partial J_{L2}}{\partial W^{(1)}} =$$
$$= \frac{\partial J}{\partial z^{(3)}} = \frac{\partial z^{(3)}}{\partial a^{(2)}} \cdot \frac{\partial a^{(2)}}{\partial z^{(2)}} \cdot \frac{\partial z^{(2)}}{\partial W^{(1)}} + \frac{\partial J_{L2}}{\partial W^{(1)}} =$$

$$= \frac{1}{N} \cdot (\psi(z^{(3)}) - \Delta) \cdot \frac{W^{(2)}a^{(2)} + b^{(2)}}{\partial a^{(2)}} \cdot \frac{\varphi(z^{(2)})}{\partial z^{(2)}} \cdot \frac{W^{(1)}a^{(1)} + b^{(1)}}{\partial W^{(1)}} + \frac{\partial \lambda \left(\left\| W^{(1)} \right\|_2^2 + \left\| W^{(2)} \right\|_2^2 \right)}{\partial W^{(1)}} =$$

$$= \frac{1}{N} \cdot (\psi(z^{(3)}) - \Delta) \cdot W^{(2)^T} \cdot \varphi^{'}(z^{(2)}) \cdot a^{(1)^T} + 2\lambda W^{(1)}$$

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

$$\frac{\partial \tilde{J}}{\partial b^{(2)}} = \frac{\partial J}{\partial b^{(2)}} = \frac{\partial J}{\partial z^{(3)}} \cdot \frac{\partial z^{(3)}}{\partial b^{(2)}} = \frac{1}{N} \cdot (\psi(z^{(3)}) - \Delta) \cdot \frac{W^{(2)}a^{(2)} + b^{(2)}}{\partial b^{(2)}} = \frac{1}{N} \cdot (\psi(z^{(3)}) - \Delta) \times 1$$

where 1 is a vector of ones of size $b^{(2)}$.

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

$$\frac{\partial \tilde{J}}{\partial b^{(1)}} = \frac{\partial J}{\partial b^{(1)}} = \frac{\partial J}{\partial z^{(3)}} \cdot \frac{\partial z^{(3)}}{\partial a^{(2)}} \cdot \frac{\partial a^{(2)}}{\partial z^{(2)}} \cdot \frac{\partial z^{(2)}}{\partial b^{(1)}} =$$
$$= \frac{1}{N} \cdot (\psi(z^{(3)}) - \Delta) \cdot \frac{W^{(2)}a^{(2)} + b^{(2)}}{\partial a^{(2)}} \cdot \frac{\varphi(z^{(2)})}{\partial z^{(2)}} \cdot \frac{W^{(1)}a^{(1)} + b^{(1)}}{\partial b^{(1)}} = \frac{1}{N} \cdot (\psi(z^{(3)}) - \Delta) \cdot W^{(2)^T} \cdot \varphi^{'}(z^{(2)}) \times 1$$

where 1 is a vector of ones of size $b^{(1)}$.

2 Stochastic Gradient Descent Training

In this section, we tried to adjust the hyperparameters of the original model. Since the model has low capacity,or underfits, in other words, we will increase vastly the number of neurons in the hidden layer, from 50 to 512. Since this gives us many more weights and biases to learn, we will increase the learning rate from 1e-04 to 1e-03. We also increase the batch size from 200 to 350, for running time purposes.The regularizer value of 0.25 is way too high, we try with 0.01. In this way, we try to increase the capacity of the model, and if it had started to overfit, then we would have used techniques such as dropouts, batch normalization, higher regularizer value or even Early Stoppings.

However, after running the model we observed that the training loss and accuracy are by a small margin better than the validation ones (53.4% for training and 51.4% for validation set). That means that the model is not overfitting on the training set, in fact it is still underfitting, since without using regularization techniques and with a very high number of neurons and iterations, the training accuracy is just marginally over 50%. This observation is compatible with the theory that Feedforward Neural Networks are not ideal for Image classification tasks. Since our model has accuracy higher than 48% on the validation set, we will accept it as our best model. Some more experiments for the hyperparameters can be found in the last part, which run faster since it is implemented using Pytorch. Our model on the test set gives us 50.9 % accuracy.

3 Multi-layer perceptron using PyTorch library

3.1

In this section we implemented same two-layer fully connected network using PyTorch library. After training our network for 10 epochs, we achieved final validation accuracy equal to 50.9 %.

Layers	Epochs	Learning Rate	Hidden Layers' Sizes	Validation Accuracy
2	10	1e-03	[64]	52.3%
2	10	1e-03	[128]	52%
2	30	1e-03	[64]	56.3%
2	30	1e-03	[128]	56.3%
3	10	1e-03	[128,64]	51%
3	10	1e-03	[128,256]	54.3%
3	30	1e-03	[128,64]	53.9%
3	30	1e-03	[128,256]	56.6%
4	10	1e-03	[128,128,256]	54.1%
4	10	1e-02	[128,128,64]	44.5%
4	30	1e-02	[128,128,64]	53%
4	30	1e-03	[128,128,256]	54.3%
5	10	1e-02	[256,128,64,64]	44.2%
5	30	1e-02	[256,128,64,64]	52%
5	10	1e-03	[256,128,64,64]	7.8%

Table 1: Network configurations for part 4.c

3.2

Observations:

We have experimented by changing only the parameters that are listed in the table above. All the rest have their default values. Based on the observations of part 3.b, we saw that the training loss is not so different from the validation loss,in other words our models don't suffer from overfitting. On the contrary, we could say that our models suffer more from underfitting, something that makes sense, since feedforward Neural Networks are regarded to have limited power for Image Classification. This is why we decided to not use dropout layers,batch normalization,increase the regularizer value lambda or introduce Early Stopping, since all of these techniques are used to prevent the model from overfitting, which is not the case in this occasion.

We can see that most of our experimental models don't have very big differences in the validation accuracy, ranging from 52-56%.

The 2-layers networks perform better for the 30 epochs, no matter their hidden size. Actually, they give us some of the best results for 30 epochs and they have the advantage that they are trained faster than other models, because they have less neurons and consequently less weights to learn.

For 3-layer models,we see that the 30 epochs help the models achieve better results, but even more important is the addition of more neurons in the last layer, something that really help the models improve.

For 4 and 5 layer networks, the models fail to improve further and actually produce worse results. For 5 layers, there are many weights to be learnt, so we will have to increase the learning rate and also the number of epochs to get some decent results.

Overall, we can claim that adding more than 3 layers, doesn't improve the accuracy and also makes the training slower, so it is a bad idea to do so. If we look only at the accuracies, our best model has 2 hidden layers of sizes [128,256] and is trained for 30 epochs. However, someone could train this model for just 10 epochs, with a small decline in performance and a faster training, or even train a 2 layer model for 30 epochs, with just 64 neurons, which performs almost the same as our best model, but it is much better in terms of training time. More concretely, we can "sacrifise" just 0.3% accuracy by picking a model with 64 neurons rather than have an accuracy of +0.3%, but having to train a network with 128+256 neurons and with 2 hidden layers.

We chose to pick the best model in terms of accuracy, so the one with the 3 layers, and its accuracy on the test set is 56.2%, similar to 56.6% in the validation.