AML Assignment 2 - report

April 11, 2020

Fabio Montello (1834411), Francesco Russo (1449025), Michele Cernigliaro (1869097)

1 Overview

In this report we proceed to answer as requested the questions 2 (a, b, c), 3 (a,b) and 4 (a,b,c). For each point we are going to write a brief description using figures and formulas whenever needed.

2 Question 2a

Verify that the loss function defined in Eq. (5) has the gradient w.r.t. $z^{(3)}$ as below:

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

We have the loss function

$$J = \frac{1}{N}\left[\sum_{i=1}^{N} -log\psi(z_{yi})\right] = \frac{1}{N}\left[\sum_{i=1}^{N} -log\frac{e^{z_{y_i}}}{\sum_j e^{z_j}}\right] = \frac{1}{N}\left[\sum_{i=1,j=k_i}^{N} -log\frac{e^{z_{i,j}}}{\sum_j e^{z_j}}\right]$$

Where we use the notation $z = z^{(3)}$ and the softmax activation function is
$\psi(z_{yi}) = \frac{e^{z_{y_i}}}{\sum_j e^{z_j}} = \frac{e^{z_{ik_i}}}{\sum_j e^{z_j}}$

The matrix ∇_J of the derivatives of J with respect to z_{ij} for every $i = 1...N$ and for every $j = 1...K$ has the general element

$$\nabla_{J_{i',j'}} = \frac{\partial J}{\partial z_{i'j'}} = \frac{\partial}{\partial z_{i'j'}}\frac{1}{N}\left[\sum_{i=1}^{N} -log\psi(z_{yi})\right] = -\frac{\partial}{\partial z_{i'j'}}\frac{1}{N}\left[log\psi(z_{yi'})\right] = -\frac{1}{N}\frac{1}{\psi(z_{yi'})}\frac{\partial}{\partial z_{i'j'}}\psi(z_{yi'}) =$$

$$-\frac{1}{N}\frac{1}{\psi(z_{yi'})}\frac{e^{z_{i'j'}}\cdot\delta(j',k_{i'})\sum_j e^{z_{i'j}} - e^{z_{i'k'_i}}e^{z_{i'j'}}}{(\sum_j e^{z_{i'j}})^2} =$$

$$-\frac{1}{N}\frac{\sum_j e^{z_{i'j}}}{e^{z_{i'k'_i}}}\frac{e^{z_{i'j'}}\cdot\delta(j',k_{i'})\sum_j e^{z_{i'j}} - e^{z_{i'k'_i}}e^{z_{i'j'}}}{(\sum_j e^{z_{i'j}})^2} =$$

$$\frac{1}{N}\left[\frac{e^{z_{i'j'}}}{\sum_j e^{z_{i'j}}} - \delta(j', k_i')\right]$$

where $\delta(j', k_i')$ is the Kronecker Delta function

In matrix form this is exactly:

$$\nabla_J = \frac{1}{N}\left[\psi(z) - \Delta_{j,ki}\right]$$

3 Question 2b

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

$$\begin{aligned}\frac{\partial J}{\partial W^{(2)}}(\{x_i, y_i\}_{i=1}^N) &= \frac{\partial J}{\partial z^{(3)}} \cdot \frac{\partial z^{(3)}}{\partial W^{(2)}} \\ &= \frac{1}{N}(\psi(z^{(3)}) - \Delta)a^{(2)}\end{aligned}$$

As seen before, we compute the chain rule as follow:

$$\frac{\partial J}{\partial W^{(2)}}(\{x_i, y_i\}_{i=1}^N) = \frac{\partial J}{\partial z^{(3)}} \cdot \frac{\partial z^{(3)}}{\partial W^{(2)}}$$

We already computed:

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

We know that

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

And so we have that:

$$\frac{\partial J}{\partial W^{(2)}}(\{x_i, y_i\}_{i=1}^N) = \frac{1}{N}(\psi(z^{(3)}) - \Delta)a^{(2)}$$

Similarly, verify that the regularized loss in Eq. (6) has the derivatives

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

We can now find the partial derivative of the loss w.r.t the regularization term, which is:

$$\frac{\partial\left[\lambda\left(\left\|W^{(1)}\right\|_2^2 + \left\|W^{(2)}\right\|_2^2\right)\right]}{\partial W^{(2)}} = 2\lambda W^{(2)}$$

So we obtain:

$$\frac{d\tilde{J}}{dW^{(2)}} = \frac{dJ}{dz^{(3)}} \cdot \frac{dz^{(3)}}{dW^{(2)}} = \frac{1}{N}\left[\psi(z^{(3)}) - \Delta\right] \cdot a^{(2)} + 2\lambda W^{(2)}$$

4 Question 2c

We can repeatedly apply chain rule as discussed above to obtain the derivatives of the loss with respect to all the parameters of the model $\theta = \left(W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)}\right)$. Derive the expressions for the derivatives of the regularized loss in Eq. (6) w.r.t. $W^{(1)}, b^{(1)}, b^{(2)}$ now.

We start deriving the loss J w.r.t $W^{(1)}$, by applying the chain rule. We'll had then the regularization term. Applying the chain rule we get for $W^{(1)}$ the following:

$$\frac{\partial J}{\partial W^{(1)}} = \frac{\partial J}{\partial z^{(2)}} \cdot \frac{\partial z^{(2)}}{\partial W^{(1)}}$$

For $\frac{\partial J}{\partial z^{(2)}}$ we apply iteratively the chain rule obtaining:

$$\begin{aligned}\frac{\partial J}{\partial z^{(2)}} &= \frac{\partial J}{\partial a^{(2)}} \odot \frac{\partial a^{(2)}}{\partial z^{(2)}} \\ &= \left(\frac{\partial J}{\partial z^{(3)}} \cdot \frac{\partial z^{(3)}}{\partial a^{(2)}}\right) \odot \frac{\partial a^{(2)}}{\partial z^{(2)}}\end{aligned}$$

Where $\odot$ indicates the element-wise product (Hadamard product) and $\frac{\partial a^{(2)}}{\partial z^{(2)}} \in \mathbb{R}^{10\times 5}$ is the the derivative of the ReLu activation w.r.t. $z^{(2)}$, which is also the heaviside step function:

$$\frac{\partial a_{ij}^{(2)}}{\partial z^{(2)}} = \begin{cases} 1 & if \quad z_{ij}^{(2)} > 0 \\ 0 & otherwise \end{cases}$$

We do already know $\frac{\partial J}{\partial z^{(3)}}$, now the calculation of the other partial derivatives is straightforward:

$$\begin{aligned}\frac{\partial z^{(3)}}{\partial a^{(2)}} &= \frac{\partial\left(W^{(2)}a^{(2)} + b^{(2)}\right)}{\partial a^{(2)}} = W^{(2)} \\ \frac{\partial z^{(2)}}{\partial W^{(1)}} &= \frac{\partial\left(W^{(1)}a^{(1)} + b^{(1)}\right)}{\partial W^{(1)}} = a^{(1)}\end{aligned}$$

Now we finally obtain the loss derivatives w.r.t. $W^{(1)}$ as it follows:

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

We can now find the gradient of $W^{(1)}$ w.r.t the regularization term, which is:

$$\frac{\partial \left[\lambda \left(\left\|W^{(1)}\right\|_2^2 + \left\|W^{(2)}\right\|_2^2\right)\right]}{\partial W^{(1)}} = 2\lambda W^{(1)}$$

Hence we end up having:

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

Which can be also implemented in vectorized form using numpy.

We now derive the loss J w.r.t $b^{(1)}$. Applying the chain rule we obtain similarly:

$$\begin{aligned}\frac{\partial J}{\partial b^{(1)}} &= \frac{\partial J}{\partial z^{(2)}} \cdot \frac{\partial z^{(2)}}{\partial b^{(1)}} \\ &= \left[\left(\frac{\partial J}{\partial z^{(3)}} \cdot \frac{\partial z^{(3)}}{\partial a^{(2)}}\right) \odot \frac{\partial a^{(2)}}{\partial z^{(2)}}\right] \cdot \frac{\partial z^{(2)}}{\partial b^{(1)}} \\ &= \left[\left(\frac{\partial J}{\partial z^{(3)}} \cdot \frac{\partial z^{(3)}}{\partial a^{(2)}}\right) \odot \frac{\partial a^{(2)}}{\partial z^{(2)}}\right] \cdot \mathbb{I}\end{aligned}$$

Where $\mathbb{I}$ in this case denotes a matrix of ones (not the identity matrix).

For the loss J w.r.t. $b^{(2)}$, similarly we have the following:

$$\frac{dJ}{db^{(2)}} = \frac{dJ}{dz^{(3)}} \cdot \frac{dz^{(3)}}{db^{(2)}} = \frac{1}{N}\left[\psi(z^{(3)}) - \Delta\right] \cdot \mathbb{I}$$

Where $\mathbb{I}$ in this case denotes a matrix of ones (not the identity matrix).

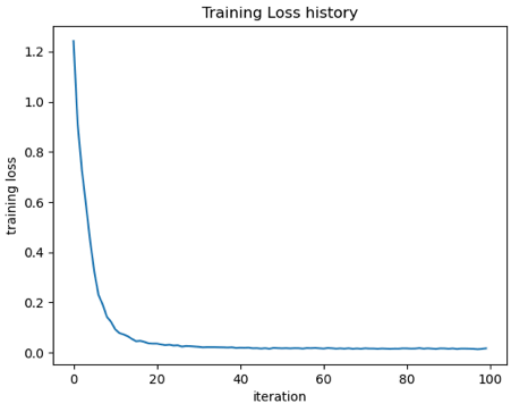
5 Question 3a

Implement the stochastic gradient descent algorithm in two_layernet.py and run the training on the toy data. Your model should be able to obtain loss = 0.02 on the training set and the training curve should look similar to the one shown in Fig. 2.

We report here the code in which we implemented the stochastic gradient descent, sampling (pseudo-)randomly batches of size **batch_size** from the training set. Computing loss and gradients (the latters with backpropagation) using the function loss() , and then updating the parameters.

```
215            ###########################################################################
216            # TODO: Create a random minibatch of training data and labels, storing  #
217            # them in X_batch and y_batch respectively.                             #
218            ###########################################################################
219
220            # *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
221            idx = np.random.randint(low = 0, high = num_train, size = batch_size)
222            X_batch = X[idx]
223            y_batch = y[idx]
224
225            pass
226
227            # *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
228
229            # Compute loss and gradients using the current minibatch
230            loss, grads = self.loss(X_batch, y=y_batch, reg=reg)
231            loss_history.append(loss)
232
233            ###########################################################################
234            # TODO: Use the gradients in the grads dictionary to update the         #
235            # parameters of the network (stored in the dictionary self.params)      #
236            # using stochastic gradient descent. You'll need to use the gradients   #
237            # stored in the grads dictionary defined above.                         #
238            ###########################################################################
239
240            # *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
241            self.params['W1'] = self.params['W1'] - learning_rate*grads["W1"]
242            self.params['b1'] = self.params['b1'] - learning_rate*grads["b1"]
243            self.params['W2'] = self.params['W2'] - learning_rate*grads["W2"]
244            self.params['b2'] = self.params['b2'] - learning_rate*grads["b2"]
245
246            pass
247            # *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
```

As you can see from the results below, the training loss history curve of our toy data shows a similar training curve as the one shown in question 3a. Also the final training loss is ~ 0.02 as suggested.




```
Difference between your scores and correct scores:
2.9173411630889 49e-08
Difference between your loss and correct loss:
1.794120407794253e-13
W1 max relative error: 3.669858e-09
W2 max relative error: 3.440708e-09
b1 max relative error: 2.738421e-09
b2 max relative error: 3.865049e-11
iteration 0 / 100: loss 1.241994
Final training loss: 0.01714960793873202
```

6 Question 3b

[...] Your task is to debug the model training and come up with better hyper-parameters to improve the performance on the validation set. Visualize the training and validation performance curves to help with this analysis. [...] Once you have tuned your hyper-parameters, and get validation accuracy greater than 48% run your best model on the test set once and report the performance. (report, 5 points)

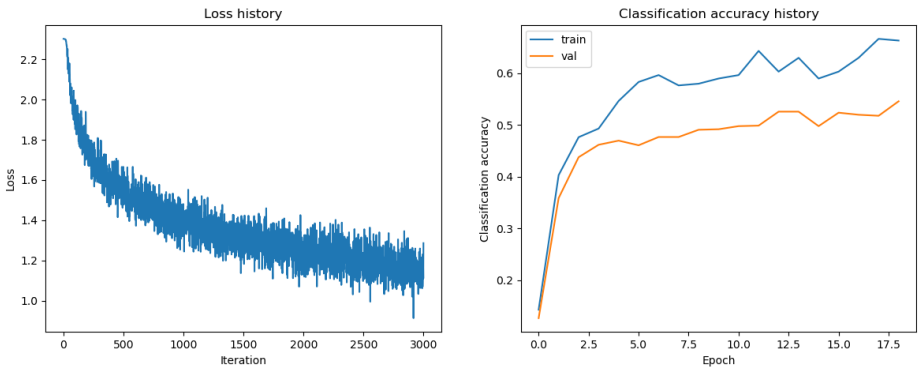
In order to find the optimal combination of hyperparameters, we performed an heuristic procedure (grid search). The final grid on which we trained the models is:

- hidden_size = [75, 100]
- num_iters = [2000, 3000]
- batch_size = [200, 300]
- learning_rate = [0.001]
- reg = [0.01 , 0.0001, 0.001 , 0.1]

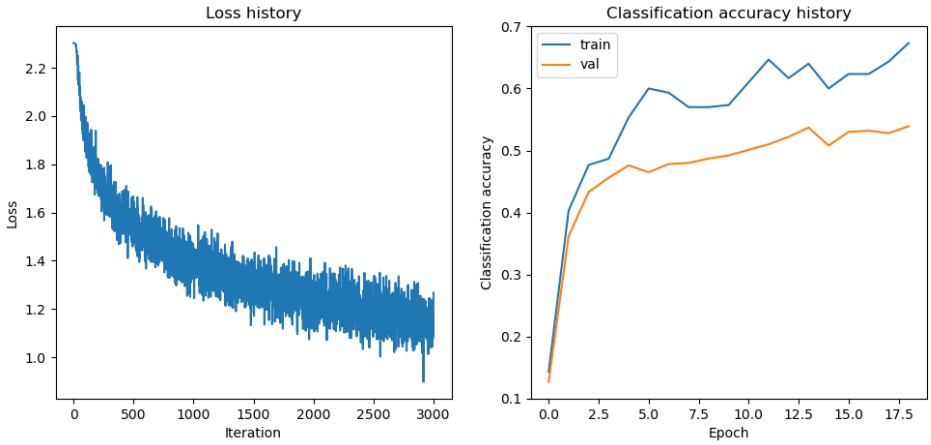
Which yielded to 32 configurations to be tried. We report here the best 15 results sorted by accuracy on the validation set:

	hidden_size	num_iters	batch_size	learning_rate	reg	train_loss	val_loss	train_acc	val_acc
0	100	3000	300	0.001	0.0100	1.147523	1.331869	0.604347	0.539
1	100	3000	300	0.001	0.0001	1.134403	1.340498	0.609735	0.536
2	100	3000	200	0.001	0.0010	1.153017	1.343609	0.602490	0.533
3	100	3000	200	0.001	0.1000	1.183392	1.370682	0.588980	0.525
4	75	3000	300	0.001	0.0001	1.179377	1.370900	0.596204	0.525
5	75	3000	300	0.001	0.0100	1.176405	1.361328	0.594592	0.522
6	100	3000	200	0.001	0.0001	1.145084	1.362785	0.603265	0.522
7	100	2000	300	0.001	0.0001	1.245536	1.369817	0.565898	0.521
8	75	2000	300	0.001	0.0010	1.282174	1.393147	0.554673	0.521
9	100	2000	300	0.001	0.1000	1.266654	1.384809	0.561796	0.519
10	100	2000	200	0.001	0.0100	1.256129	1.369715	0.559224	0.519
11	75	3000	300	0.001	0.0010	1.177504	1.382315	0.591612	0.516
12	75	2000	300	0.001	0.1000	1.275136	1.388821	0.557367	0.513
13	75	3000	200	0.001	0.1000	1.213762	1.381529	0.580714	0.512
14	100	2000	200	0.001	0.0001	1.257539	1.394553	0.563449	0.512

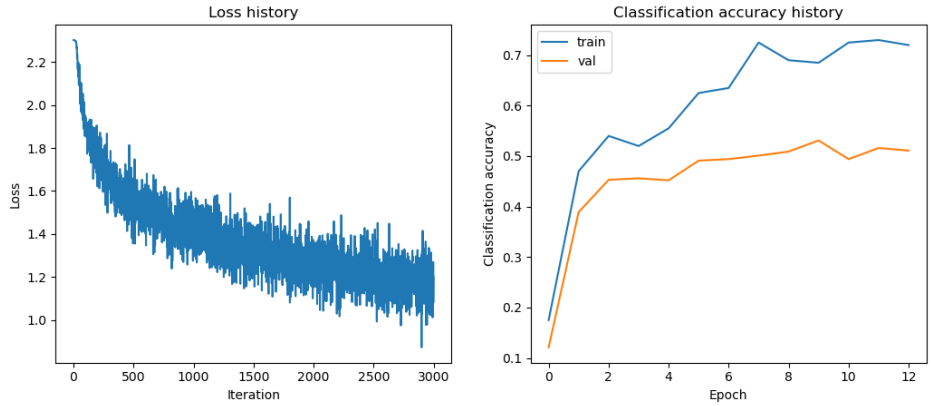
Notice that all of our first 15 "best models" have a greater accuracy than the one proposed in the question 3b ($> 48\%$). In order to help us with the best model choice, we plot the curves of the loss trend and the accuracy history during training of the first four top models:



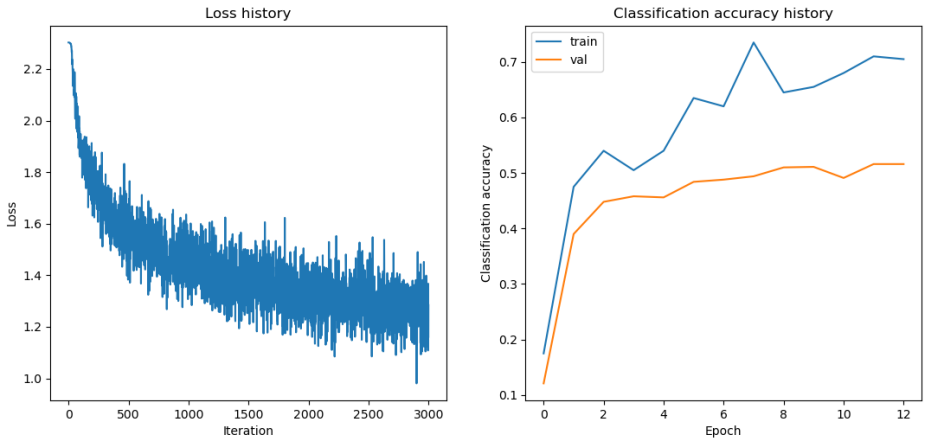
Parameters: hidden_size = 100, num_iters = 3000, batch_size = 300, learning_rate= 0.001, reg = 0.01



Parameters: hidden_size = 100, num_iters = 3000, batch_size = 300, learning_rate= 0.001, reg = 0.0001



Parameters: hidden_size = 100, num_iters = 3000, batch_size = 200, learning_rate= 0.001, reg = 0.001



Parameters: hidden_size = 100, num_iters = 3000, batch_size = 200, learning_rate= 0.001, reg = 0.1

The training seems similar for all the models, with the loss which fluctuates along the iterations due to its mini-batch approximations. The first two models seem to be the slightly more stable both in terms of loss history and in terms of overfitting.

7 Question 4a

Complete the code to implement a multi-layer perceptron network in the class MultiLayerPerceptron in ex2 pytorch.py. This includes instantiating the required layers from torch.nn and writing the code for forward pass. Initially you should write the code for the same two-layer network we have seen before.

8 Question 4c

Now that you can train the two layer network to achieve reasonable performance, try increasing the network depth to see if you can improve the performance. Experiment with networks of at least 2, 3, 4, and 5 layers, of your chosen configuration. Report the training and validation accuracies for these models and discuss your observations. Run the evaluation on the test set with your best model and report the test accuracy.