AML Assignment 2 - report

April 14, 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:

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

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:

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

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:

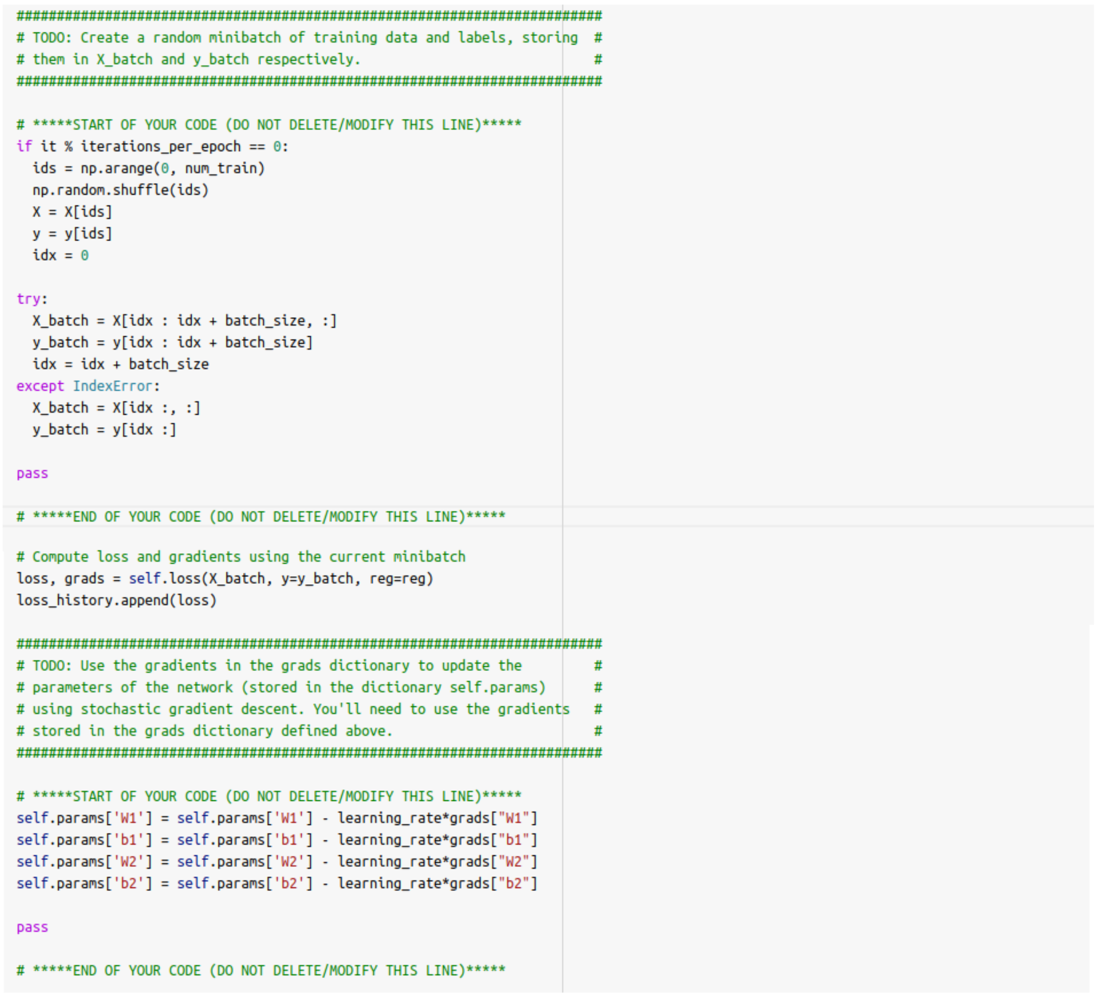
$$\frac{dJ}{db^{(2)}} = \frac{dJ}{dz^{(3)}} \cdot \frac{dz^{(3)}}{db^{(2)}} = \frac{1}{N}\left[\psi(z^{(3)}) - \Delta\right] \cdot \mathbb{I}$$

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

5 Question 3a

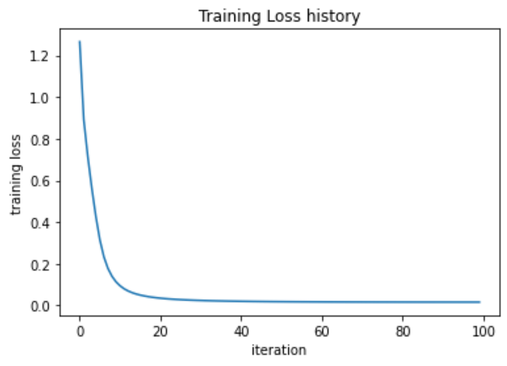
Implement the stochastic gradient descent algorithm in two_layernet.py and run the training on the toy data. Your model should be able to obtain loss = 0.02 on the training set and the training curve should look similar to the one shown in Fig. 2.

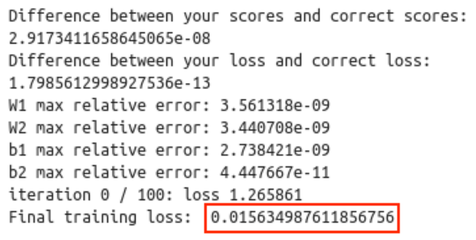
We report here the code in which we implemented the stochastic gradient descent, shuffling the full training set at the beginning of each epoch and then picking consecutive batches of size **batch_size**. Computing loss and gradients (the latters with backpropagation) using the function loss() , and then updating the parameters.



```
#########################################################################
# TODO: Create a random minibatch of training data and labels, storing  #
# them in X_batch and y_batch respectively.                             #
#########################################################################

# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
if it % iterations_per_epoch == 0:
  ids = np.arange(0, num_train)
  np.random.shuffle(ids)
  X = X[ids]
  y = y[ids]
  idx = 0

try:
  X_batch = X[idx : idx + batch_size, :]
  y_batch = y[idx : idx + batch_size]
  idx = idx + batch_size
except IndexError:
  X_batch = X[idx :, :]
  y_batch = y[idx :]

pass

# *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

# Compute loss and gradients using the current minibatch
loss, grads = self.loss(X_batch, y=y_batch, reg=reg)
loss_history.append(loss)

#########################################################################
# TODO: Use the gradients in the grads dictionary to update the         #
# parameters of the network (stored in the dictionary self.params)      #
# using stochastic gradient descent. You'll need to use the gradients   #
# stored in the grads dictionary defined above.                         #
#########################################################################

# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
self.params['W1'] = self.params['W1'] - learning_rate*grads["W1"]
self.params['b1'] = self.params['b1'] - learning_rate*grads["b1"]
self.params['W2'] = self.params['W2'] - learning_rate*grads["W2"]
self.params['b2'] = self.params['b2'] - learning_rate*grads["b2"]

pass

# *****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:
2.9173411658645065e-08
Difference between your loss and correct loss:
1.7985612998927536e-13
W1 max relative error: 3.561318e-09
W2 max relative error: 3.440708e-09
b1 max relative error: 2.738421e-09
b2 max relative error: 4.447667e-11
iteration 0 / 100: loss 1.265861
Final training loss:  0.015634987611856756
```

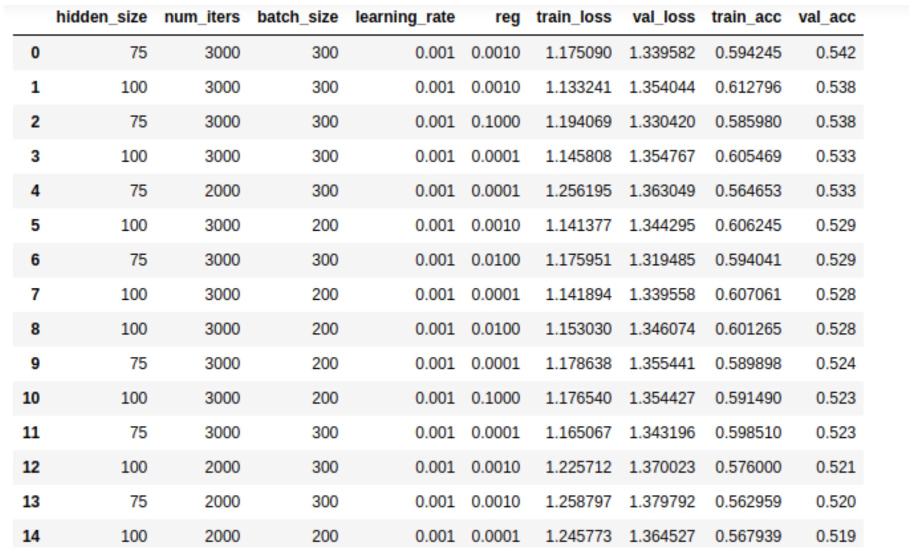
6 Question 3b

[...] 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:

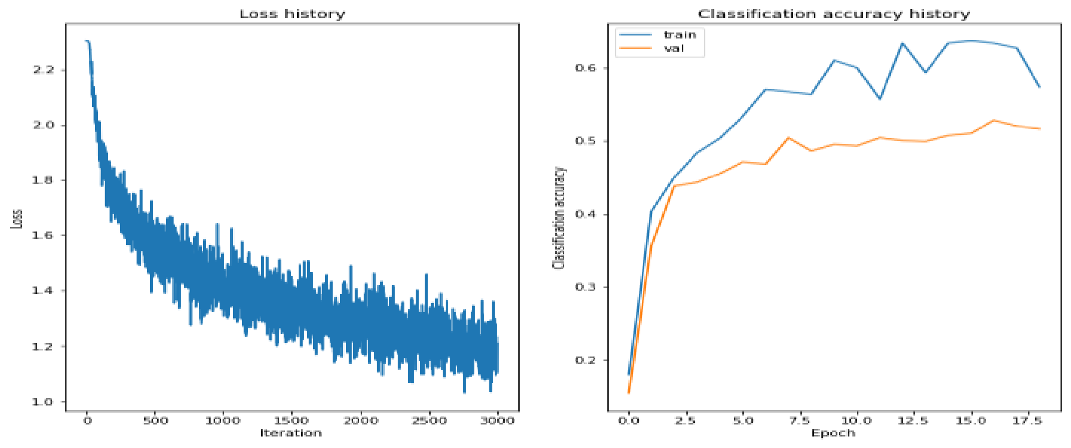
- hidden_size = [75, 100]
- num_iters = [2000, 3000]
- batch_size = [200, 300]
- learning_rate = [0.001]
- reg = [0.01 , 0.0001, 0.001, 0.1]

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

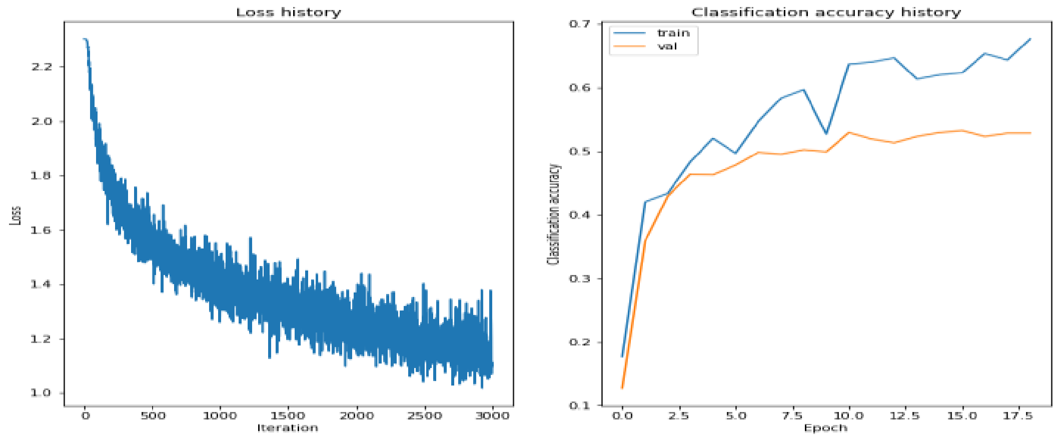


	hidden_size	num_iters	batch_size	learning_rate	reg	train_loss	val_loss	train_acc	val_acc
0	75	3000	300	0.001	0.0010	1.175090	1.339582	0.594245	0.542
1	100	3000	300	0.001	0.0010	1.133241	1.354044	0.612796	0.538
2	75	3000	300	0.001	0.1000	1.194069	1.330420	0.585980	0.538
3	100	3000	300	0.001	0.0001	1.145808	1.354767	0.605469	0.533
4	75	2000	300	0.001	0.0001	1.256195	1.363049	0.564653	0.533
5	100	3000	200	0.001	0.0010	1.141377	1.344295	0.606245	0.529
6	75	3000	300	0.001	0.0100	1.175951	1.319485	0.594041	0.529
7	100	3000	200	0.001	0.0001	1.141894	1.339558	0.607061	0.528
8	100	3000	200	0.001	0.0100	1.153030	1.346074	0.601265	0.528
9	75	3000	200	0.001	0.0001	1.178638	1.355441	0.589898	0.524
10	100	3000	200	0.001	0.1000	1.176540	1.354427	0.591490	0.523
11	75	3000	300	0.001	0.0001	1.165067	1.343196	0.598510	0.523
12	100	2000	300	0.001	0.0010	1.225712	1.370023	0.576000	0.521
13	75	2000	300	0.001	0.0010	1.258797	1.379792	0.562959	0.520
14	100	2000	200	0.001	0.0001	1.245773	1.364527	0.567939	0.519

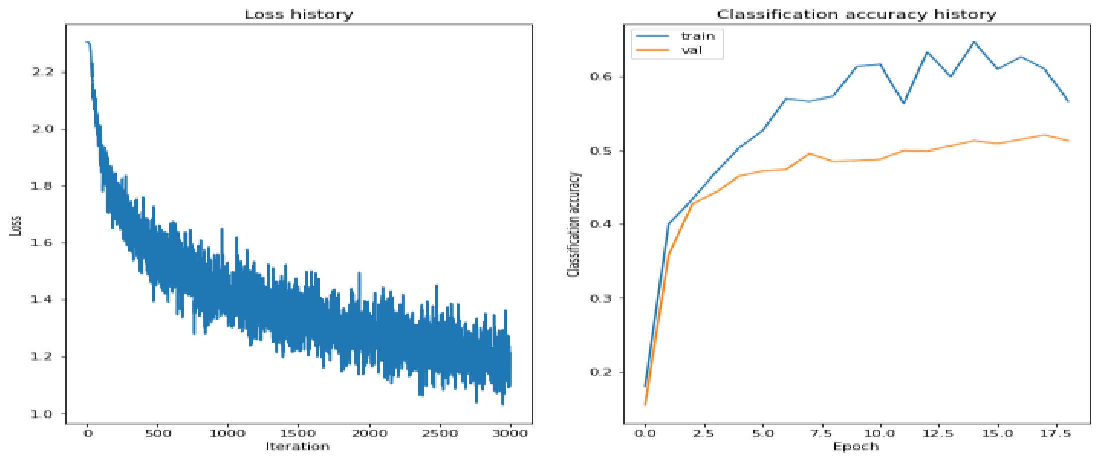
Notice that all of our first 15 “best models” have a greater accuracy than the one proposed in the question 3b ($> 48\%$). In order to help us with the best model choice, we plot the curves of the loss trend and the accuracy history during training of the first four top models:



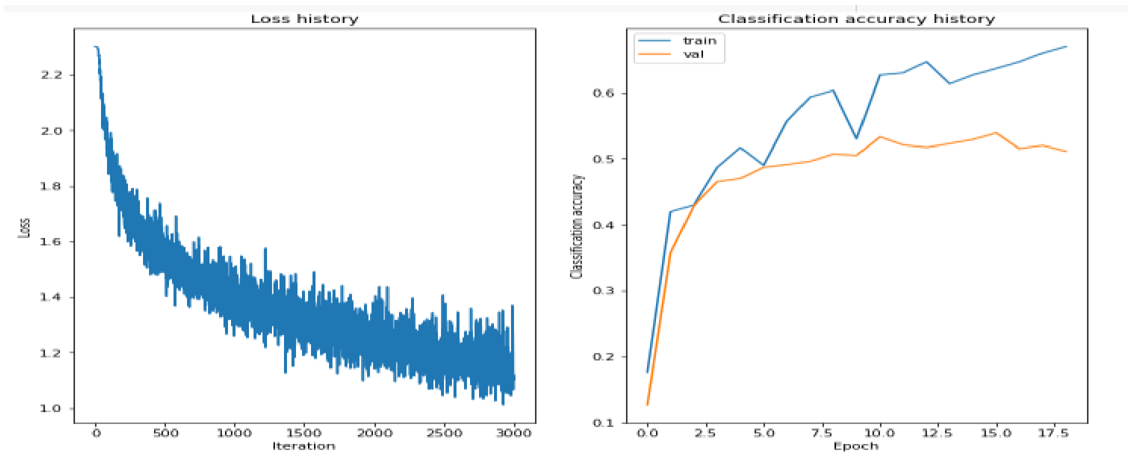
Parameters: hidden_size = 75, num_iters = 3000, batch_size = 300, learning_rate= 0.001, reg = 0.001



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



Parameters: hidden_size = 75, num_iters = 3000, batch_size = 300, learning_rate= 0.001, reg = 0.0001



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

The training seems similar for all the models, with the loss which fluctuates along the iterations due to its mini-batch approximations. However we notice that all of the models exhibit mild overfitting tendencies after the 10th epoch.

7 Question 4a

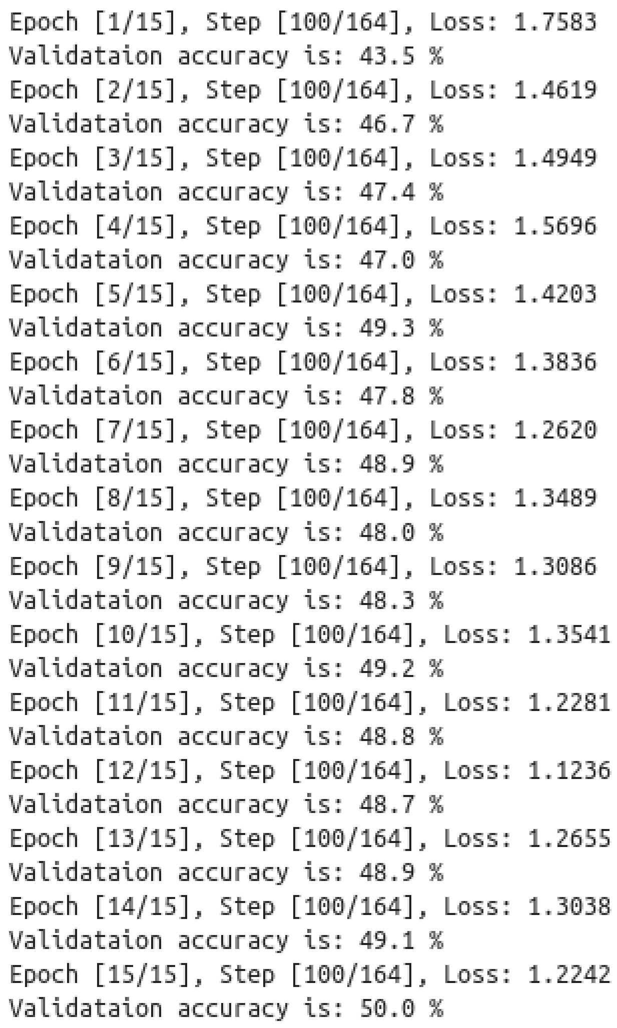
Complete the code to implement a multi-layer perceptron network in the class MultiLayerPerceptron in ex2 pytorch.py. This includes instantiating the required layers from torch.nn and writing the code for forward pass. Initially you should write the code for the same two-layer network we have seen before.

The implementation was quite straightforward using nn.Sequential and the torch layers nn.Linear, nn.ReLU, etc.

(for further information see the code)

8 Question 4b

Executing our pytorch code with similar parameters w.r.t to our best implementation in the previous grid-search of Exercise 3b) we got a validation accuracy of 50%. Here we can see the training process:



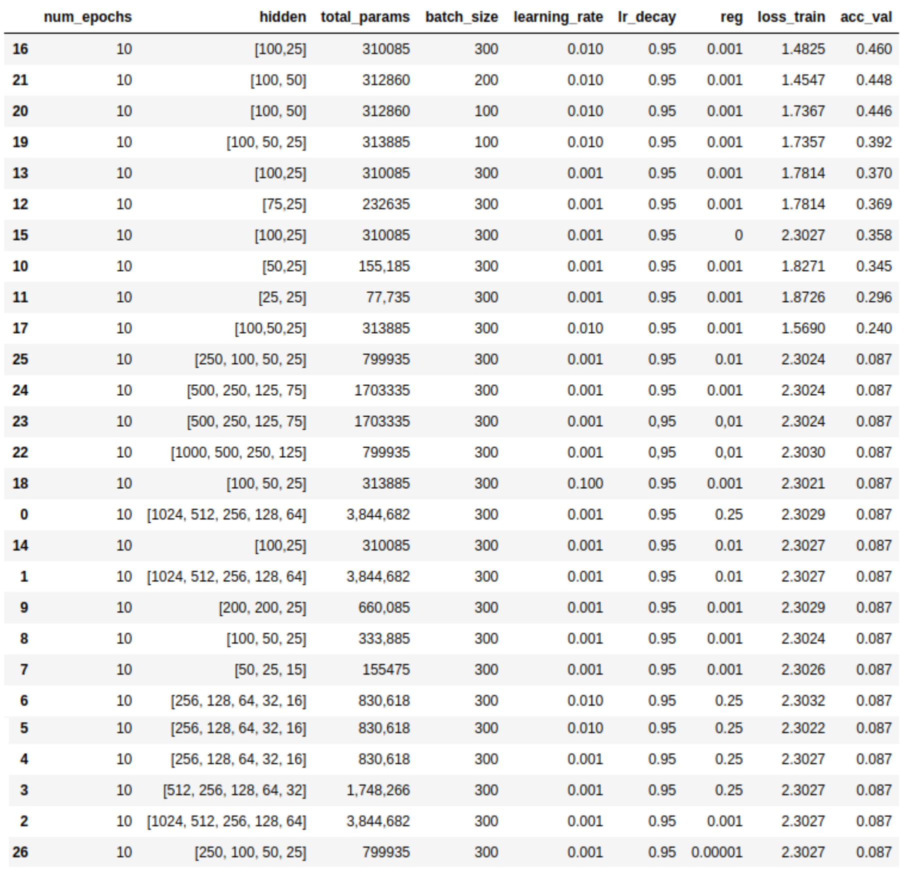
```
Epoch [1/15], Step [100/164], Loss: 1.7583
Validataion accuracy is: 43.5 %
Epoch [2/15], Step [100/164], Loss: 1.4619
Validataion accuracy is: 46.7 %
Epoch [3/15], Step [100/164], Loss: 1.4949
Validataion accuracy is: 47.4 %
Epoch [4/15], Step [100/164], Loss: 1.5696
Validataion accuracy is: 47.0 %
Epoch [5/15], Step [100/164], Loss: 1.4203
Validataion accuracy is: 49.3 %
Epoch [6/15], Step [100/164], Loss: 1.3836
Validataion accuracy is: 47.8 %
Epoch [7/15], Step [100/164], Loss: 1.2620
Validataion accuracy is: 48.9 %
Epoch [8/15], Step [100/164], Loss: 1.3489
Validataion accuracy is: 48.0 %
Epoch [9/15], Step [100/164], Loss: 1.3086
Validataion accuracy is: 48.3 %
Epoch [10/15], Step [100/164], Loss: 1.3541
Validataion accuracy is: 49.2 %
Epoch [11/15], Step [100/164], Loss: 1.2281
Validataion accuracy is: 48.8 %
Epoch [12/15], Step [100/164], Loss: 1.1236
Validataion accuracy is: 48.7 %
Epoch [13/15], Step [100/164], Loss: 1.2655
Validataion accuracy is: 48.9 %
Epoch [14/15], Step [100/164], Loss: 1.3038
Validataion accuracy is: 49.1 %
Epoch [15/15], Step [100/164], Loss: 1.2242
Validataion accuracy is: 50.0 %
```

9 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.

These are the results for the loss train and validation accuracy, obtained by trial and error with several runs, using different numbers of hidden layers (up to 5) as well as different combinations of hyperparameters (batch_size, learning_rate, lr_decay, reg). We observe that the accuracy value drastically decreases by increasing the number of hidden layers, highlighting a saturation of the network, which is thus unable to behave as an effective classifier (getting stuck around 8% accuracy).

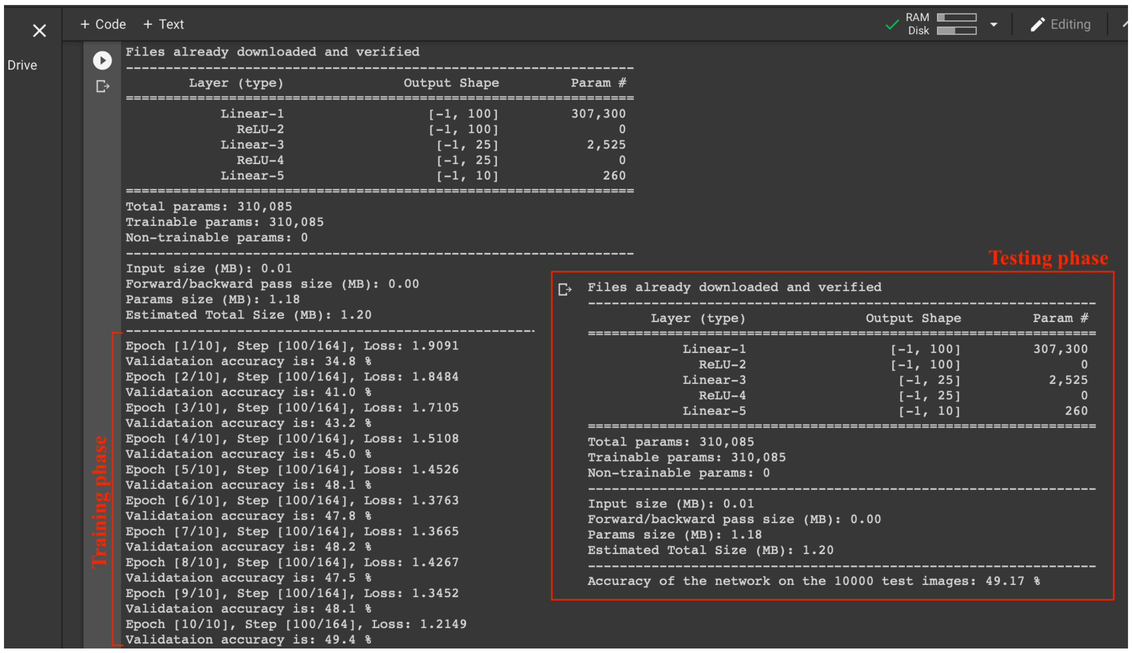
NB: note that we are not including the output layer among the hidden layers.



	num_epochs	hidden	total_params	batch_size	learning_rate	lr_decay	reg	loss_train	acc_val
16	10	[100,25]	310085	300	0.010	0.95	0.001	1.4825	0.460
21	10	[100, 50]	312860	200	0.010	0.95	0.001	1.4547	0.448
20	10	[100, 50]	312860	100	0.010	0.95	0.001	1.7367	0.446
19	10	[100, 50, 25]	313885	100	0.010	0.95	0.001	1.7357	0.392
13	10	[100,25]	310085	300	0.001	0.95	0.001	1.7814	0.370
12	10	[75,25]	232635	300	0.001	0.95	0.001	1.7814	0.369
15	10	[100,25]	310085	300	0.001	0.95	0	2.3027	0.358
10	10	[50,25]	155,185	300	0.001	0.95	0.001	1.8271	0.345
11	10	[25, 25]	77,735	300	0.001	0.95	0.001	1.8726	0.296
17	10	[100,50,25]	313885	300	0.010	0.95	0.001	1.5690	0.240
25	10	[250, 100, 50, 25]	799935	300	0.001	0.95	0.01	2.3024	0.087
24	10	[500, 250, 125, 75]	1703335	300	0.001	0.95	0.001	2.3024	0.087
23	10	[500, 250, 125, 75]	1703335	300	0.001	0,95	0,01	2.3024	0.087
22	10	[1000, 500, 250, 125]	799935	300	0.001	0,95	0,01	2.3030	0.087
18	10	[100, 50, 25]	313885	300	0.100	0.95	0.001	2.3021	0.087
0	10	[1024, 512, 256, 128, 64]	3,844,682	300	0.001	0.95	0.25	2.3029	0.087
14	10	[100,25]	310085	300	0.001	0.95	0.01	2.3027	0.087
1	10	[1024, 512, 256, 128, 64]	3,844,682	300	0.001	0.95	0.01	2.3027	0.087
9	10	[200, 200, 25]	660,085	300	0.001	0.95	0.001	2.3029	0.087
8	10	[100, 50, 25]	333,885	300	0.001	0.95	0.001	2.3024	0.087
7	10	[50, 25, 15]	155475	300	0.001	0.95	0.001	2.3026	0.087
6	10	[256, 128, 64, 32, 16]	830,618	300	0.010	0.95	0.25	2.3032	0.087
5	10	[256, 128, 64, 32, 16]	830,618	300	0.010	0.95	0.25	2.3022	0.087
4	10	[256, 128, 64, 32, 16]	830,618	300	0.001	0.95	0.25	2.3027	0.087
3	10	[512, 256, 128, 64, 32]	1,748,266	300	0.001	0.95	0.25	2.3027	0.087
2	10	[1024, 512, 256, 128, 64]	3,844,682	300	0.001	0.95	0.001	2.3027	0.087
26	10	[250, 100, 50, 25]	799935	300	0.001	0.95	0.00001	2.3027	0.087

Run the evaluation on the test set with your best model and report the test accuracy.

We report here the training and testing phases using the best model, obtaining in accuracy 49.4% and 49.17% respectively on the training and on the test set. We ran the model on Colaboratory in order to speed up the computation using GPU's:



```
Files already downloaded and verified
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                  [-1, 100]         307,300
              ReLU-2                  [-1, 100]               0
            Linear-3                   [-1, 25]           2,525
              ReLU-4                   [-1, 25]               0
            Linear-5                   [-1, 10]             260
================================================================
Total params: 310,085
Trainable params: 310,085
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 0.00
Params size (MB): 1.18
Estimated Total Size (MB): 1.20
----------------------------------------------------
Epoch [1/10], Step [100/164], Loss: 1.9091
Validataion accuracy is: 34.8 %
Epoch [2/10], Step [100/164], Loss: 1.8484
Validataion accuracy is: 41.0 %
Epoch [3/10], Step [100/164], Loss: 1.7105
Validataion accuracy is: 43.2 %
Epoch [4/10], Step [100/164], Loss: 1.5108
Validataion accuracy is: 45.0 %
Epoch [5/10], Step [100/164], Loss: 1.4526
Validataion accuracy is: 48.1 %
Epoch [6/10], Step [100/164], Loss: 1.3763
Validataion accuracy is: 47.8 %
Epoch [7/10], Step [100/164], Loss: 1.3665
Validataion accuracy is: 48.2 %
Epoch [8/10], Step [100/164], Loss: 1.4267
Validataion accuracy is: 47.5 %
Epoch [9/10], Step [100/164], Loss: 1.3452
Validataion accuracy is: 48.1 %
Epoch [10/10], Step [100/164], Loss: 1.2149
Validataion accuracy is: 49.4 %
```

```
Files already downloaded and verified
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                  [-1, 100]         307,300
              ReLU-2                  [-1, 100]               0
            Linear-3                   [-1, 25]           2,525
              ReLU-4                   [-1, 25]               0
            Linear-5                   [-1, 10]             260
================================================================
Total params: 310,085
Trainable params: 310,085
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 0.00
Params size (MB): 1.18
Estimated Total Size (MB): 1.20
----------------------------------------------------------------
Accuracy of the network on the 10000 test images: 49.17 %
```