Course: DD2424 - Assignment 3

Álvaro Orgaz Expósito

April 29, 2019

In this assignment, we will use a k-layer neural network with one input layer (as many nodes as images pixels), k-1 hidden layer (with *ReLu* or rectified linear units as activation function), and one output layer (as many nodes as possible image labels and *SoftMax* as activation function) with the loss function cross entropy (classification metric) with L2 regularization. Then, the predicted class corresponds to the label with the highest predicted probability (since the sum of the output activations for a sample is 1 as a consequence of using *SoftMax*). The shape of the weights matrix is as many rows as output nodes and as many columns as input nodes for each layer. And for the bias, gammas and betas only 1 column with as many rows as layer nodes (the last 2 parameters for the *Batch Normalization* implemented).

Then, for computing a forward pass through the network or prediction, given some parameters and *Batch Normalization* in hidden layers, we use from the initial input image X^0:

for $l = 1, ..., k-1$

$$S^l = W^l \times X^{l-1} + b^l \tag{1}$$

$$\hat{S}^l = BatchNormalize(S^l, \mu^l, \sigma^l) \tag{2}$$

$$\tilde{S}^l = \gamma^l \odot \hat{S}^l + \beta^l \tag{3}$$

$$X^l = ReLu(\hat{S}^l) \quad where \quad ReLu(\hat{S}^l_{ij}) = max(0, \hat{S}^l_{ij}) \tag{4}$$

and then finally

$$S = W^k \times X^{k-1} + b^k \tag{5}$$

$$P = SoftMax(S) \quad where \quad SoftMax(S_{ji}) = \frac{exp(S_{ji})}{\sum_{c=1}^{C} exp(S_{ci})} \tag{6}$$

where

$$BatchNormalize(S^l, \mu^l, \sigma^l) = diag(\sigma^l + \epsilon)^{-1}(S^l - \mu^l) \tag{7}$$

and ϵ is a small number, and S^l and $\tilde{S}^l$, which are the un/normalized scores, have as many rows as layer nodes (as well as μ^l and σ^l).

For computing the mean and standard deviation of each dimension or layer node in the batch of size n we use

$$\mu^l = \frac{\sum_{i=1}^{n} S_i^l}{n} \tag{8}$$

$$\sigma_j^l = \sqrt{\frac{\sum_{i=1}^{n}(S_{ji}^l - \mu_j^l)^2}{n}} \quad for \quad j = 1, ..., \# \, layer\, nodes \tag{9}$$

but note that for validating and testing we use the means and standard deviations calculated during the training.

Also, the loss function (cross entropy) to minimise respect to parameters is:

$$J = L + \lambda \sum_{l=1}^{k} ||W^l||^2 = \frac{1}{N} \sum_{i=1}^{N} -log\Big(\underset{[1\times C]}{Y_{:i}^T} \times \underset{[C\times 1]}{P_{:i}}\Big) + \lambda \sum_{l=1}^{k} ||W^l||^2 \tag{10}$$

Then, for computing the gradients of the cross-entropy loss function and an L2 regularization term in mini-batch mode (given some images and labels, the weights and the regularization parameter) I use the equations mentioned in the assignment instructions by data batches and then update the parameters of each layer (W^l, b^l, γ^l and β^l) with updates equal to $-\eta_{update}$ times the respective gradients.

$$W_{t+1}^k = W_t^k - \eta_{t+1} \frac{\partial J\big(Batch^{t+1}\big)}{\partial W^k} \quad where \quad \frac{\partial J\big(Batch^{t+1}\big)}{\partial W^k} = \frac{1}{N_{Batch^{t+1}}} \sum_{i=1}^{N_{Batch^{t+1}}} \frac{\partial J_i}{\partial W^k} \tag{11}$$

$$b_{t+1}^k = b_t^k - \eta_{t+1} \frac{\partial J\big(Batch^{t+1}\big)}{\partial b^k} \quad where \quad \frac{\partial J\big(Batch^{t+1}\big)}{\partial b^k} = \frac{1}{N_{Batch^{t+1}}} \sum_{i=1}^{N_{Batch^{t+1}}} \frac{\partial J_i}{\partial b^k} \tag{12}$$

$$\gamma_{t+1}^k = \gamma_t^k - \eta_{t+1} \frac{\partial J\big(Batch^{t+1}\big)}{\partial \gamma^k} \quad where \quad \frac{\partial J\big(Batch^{t+1}\big)}{\partial \gamma^k} = \frac{1}{N_{Batch^{t+1}}} \sum_{i=1}^{N_{Batch^{t+1}}} \frac{\partial J_i}{\partial \gamma^k} \tag{13}$$

$$\beta_{t+1}^k = \beta_t^k - \eta_{t+1} \frac{\partial J\big(Batch^{t+1}\big)}{\partial \beta^k} \quad where \quad \frac{\partial J\big(Batch^{t+1}\big)}{\partial \beta^k} = \frac{1}{N_{Batch^{t+1}}} \sum_{i=1}^{N_{Batch^{t+1}}} \frac{\partial J_i}{\partial \beta^k} \tag{14}$$

1 Data used & initialization of the parameters of the network

For this assignment I will just use data in the CIFAR 10, concretely 45.000 images from all 5 batch files *data batch #* for training, the remaining 5.000 images for validation and the file *test batch* for testing. Each batch contains 10000 images (columns) of 3072 pixels (rows) and 10 possible labels (targets). In figure 1 we can find the distribution of labels by created sets of CIFAR 10. Importantly the distribution in training validation and test is approximately uniform.

I will normalize (minus the mean and divided by standard deviation) the images by pixel (which are between 0 and 255) using the mean and standard deviation of the training data, since normalization helps when training the networks.

I will initialize the network parameters with the weights matrix with a Gaussian random distribution (zero mean and standard deviation $2/\sqrt{input\,dimension}$ following the *He initialization*, the bias and β matrix with zeros, and γ with ones.

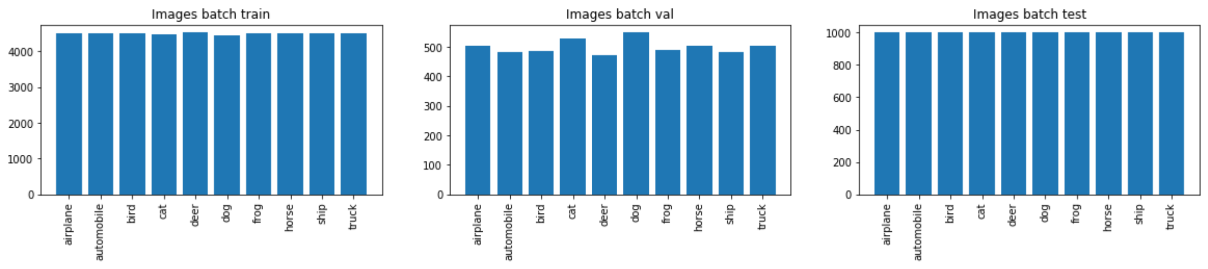


Figure 1: Distribution of labels by images sets of CIFAR 10.

2 Checking the computed gradients for the network parameters (WITHOUT *Batch Normalization*)

I want to check that the gradients computed in the gradients function without *Batch Normalization* correspond to the correct gradients. To do this, I compare the gradient obtained by the network (analytical method) with the gradient computed with the finite difference method (numerical method), after initializing the parameters as mentioned. As error measures I will apply the absolute error in 3 different scenarios:

- Firstly, for 2-layer network with 50 hidden nodes, I study the gradients of the 10 first dimensions of the first 5 images of the training set without regularization. The result is:
  - For weights W^1: 100.0% of absolute errors below 1e-6 and the maximum is 2.9144e-08.
  - For bias b^1: 100.0% of absolute errors below 1e-6 and the maximum is 1.9984e-08.
  - For weights W^2: 100.0% of absolute errors below 1e-6 and the maximum is 1.0063e-06.
  - For bias b^2: 100.0% of absolute errors below 1e-6 and the maximum is 8.0699e-08.
- Secondly, for 3-layer network with 50/50 hidden nodes, I study the gradients of the 10 first dimensions of the first 5 images of the training set without regularization. The result is:
  - For weights W^1: 100.0% of absolute errors below 1e-6 and the maximum is 2.0314e-08.
  - For bias b^1: 100.0% of absolute errors below 1e-6 and the maximum is 1.7173e-08.
  - For weights W^2: 100.0% of absolute errors below 1e-6 and the maximum is 3.6173e-07.
  - For bias b^2: 100.0% of absolute errors below 1e-6 and the maximum is 2.5908e-08.
  - For weights W^3: 100.0% of absolute errors below 1e-6 and the maximum is 6.4744e-07.
  - For bias b^3: 100.0% of absolute errors below 1e-6 and the maximum is 6.0997e-08.

- Thirdly, for 4-layer network with 50/50/25 hidden nodes, I study the gradients of the 10 first dimensions of the first 5 images of the training set without regularization. The result is:
  - For weights W^1: 100.0% of absolute errors below 1e-6 and the maximum is 2.237e-08.
  - For bias b^1: 100.0% of absolute errors below 1e-6 and the maximum is 1.7213e-08
  - For weights W^2: 100.0% of absolute errors below 1e-6 and the maximum is 2.4880e-07.
  - For bias b^2: 100.0% of absolute errors below 1e-6 and the maximum is 1.8120e-08.
  - For weights W^3: 100.0% of absolute errors below 1e-6 and the maximum is 2.6750e-07.
  - For bias b^3: 100.0% of absolute errors below 1e-6 and the maximum is 4.6682e-08.
  - For weights W^4: 100.0% of absolute errors below 1e-6 and the maximum is 1.8688e-07.
  - For bias b^4: 100.0% of absolute errors below 1e-6 and the maximum is 7.9988e-08.

3 Checking the computed gradients for the network parameters (WITH *Batch Normalization*)

I want to check that the gradients computed in the gradients function with *Batch Normalization* correspond to the correct gradients. To do this, I compare the gradient obtained by the network (analytical method) with the gradient computed with the finite difference method (numerical method), after initializing the parameters as mentioned. As error measures I will apply the absolute error in 2 different scenarios:

- Firstly, for 2-layer network with 50 hidden nodes, I study the gradients of the 10 first dimensions of the first 5 images of the training set without regularization. The result is:
  - For weights W^1: 97.8% of absolute errors below 1e-6 and the maximum is 1.7628e-06.
  - For bias b^1: 100.0% of absolute errors below 1e-6 and the maximum is 1.3322e-16.
  - For weights γ^1: 100.0% of absolute errors below 1e-6 and the maximum is 1.4987e-08.
  - For bias β^1: 100.0% of absolute errors below 1e-6 and the maximum is 2.0604e-08.
  - For weights W^2: 100.0% of absolute errors below 1e-6 and the maximum is 6.9411e-08.
  - For bias b^2: 100.0% of absolute errors below 1e-6 and the maximum is 6.6373e-08.
- Secondly, for 3-layer network with 50/50 hidden nodes, I study the gradients of the 10 first dimensions of the first 5 images of the training set without regularization. The result is:
  - For weights W^1: 99.6% of absolute errors below 1e-6 and the maximum is 1.3681e-06.
  - For bias b^1: 100.0% of absolute errors below 1e-6 and the maximum is 8.8817e-17.
  - For weights γ^1: 100.0% of absolute errors below 1e-6 and the maximum is 5.179e-08.
  - For bias β^1: 100.0% of absolute errors below 1e-6 and the maximum is 6.1067e-08.
  - For weights W^2: 100.0% of absolute errors below 1e-6 and the maximum is 3.9499e-07.
  - For bias b^2: 100.0% of absolute errors below 1e-6 and the maximum is 4.9960e-17.
  - For weights γ^2: 100.0% of absolute errors below 1e-6 and the maximum is 2.7055e-08.
  - For bias β^2: 100.0% of absolute errors below 1e-6 and the maximum is 1.3721e-08.
  - For weights W^3: 100.0% of absolute errors below 1e-6 and the maximum is 8.375e-08].
  - For bias b^3: 100.0% of absolute errors below 1e-6 and the maximum is 6.9580e-08.

4 Train k-layer network with cyclical learning rates

There is not really one optimal learning rate when training a neural network with vanilla (fixed η) mini-batch gradient descent. Choose a too small learning rate and training will take too long and too large a learning rate may result in training diverging. For this assignment, I will explore the rather recent idea of exploiting cyclical learning rates *Smith, 2015* [1] as this approach eliminates much of the trial-and-error associated with finding a good learning rate and some of the costly hyperparameter optimization over multiple parameters associated with training with momentum. The main idea of cyclical learning rates is that during training the learning rate is periodically changed in a systematic fashion from a small value to a large one and then from this large value back to the small value. And this process is then repeated again and again until training is stopped. This assignment uses triangular CLR and the training is run for a set number of complete cycles and is stopped when the learning rate is at its smallest.

To evaluate the learning process I use the metrics loss and accuracy (fraction of well-predicted images or images with the highest predicted class equal to the true) after each epoch of the mini-batch gradient descent algorithm with cyclical learning rate (CLR). Note that I do not save the metrics after each update since evaluating the metrics for each batch makes the plots not smooth.

4.1 3-layer network (50 hidden nodes)

In figures 2 and 3 we can find the learning curves for the default parametrization case with and without *Batch Normalization*: batch size 100, 2 cycles, η_{min} 1e-5, η_{max} 1e-1, step size 5*45000/100=2250, λ 0.005. Note that it is equivalent to 20 epochs equal to 2 cycles $\cdot$ 2 $\cdot$ step size 2250 / (45000 images / batch size 100).

Also, in figures 4 and 5 we can find its test accuracy and confusion matrix which achieves an accuracy of 53.11% in the case without and 53.64% in the case with *Batch Normalization.*

4.2 9-layer network (50, 30, 20, 20, 10, 10, 10, and 10 hidden nodes)

In figures 6 and 7 we can find the learning curves for the default parametrization case with and without *Batch Normalization*: batch size 100, 2 cycles, η_{min} 1e-5, η_{max} 1e-1, step size 5*45000/100=2250, λ 0.005. Note that it is equivalent to 20 epochs equal to 2 cycles $\cdot$ 2 $\cdot$ step size 2250 / (45000 images / batch size 100).

Also, in figures 8 and 9 we can find its test accuracy and confusion matrix which achieves an accuracy of 50.32% in the case without and 52.97% in the case with *Batch Normalization.*

The conclusion after these results is that, at least with the default parametrization, increasing the number of hidden layer to 9 does not achieve better results. Also, the *Batch Normalization* technique helps the deeper network to improve the results, and this difference is not noticed in the network with only 3 hidden layers.

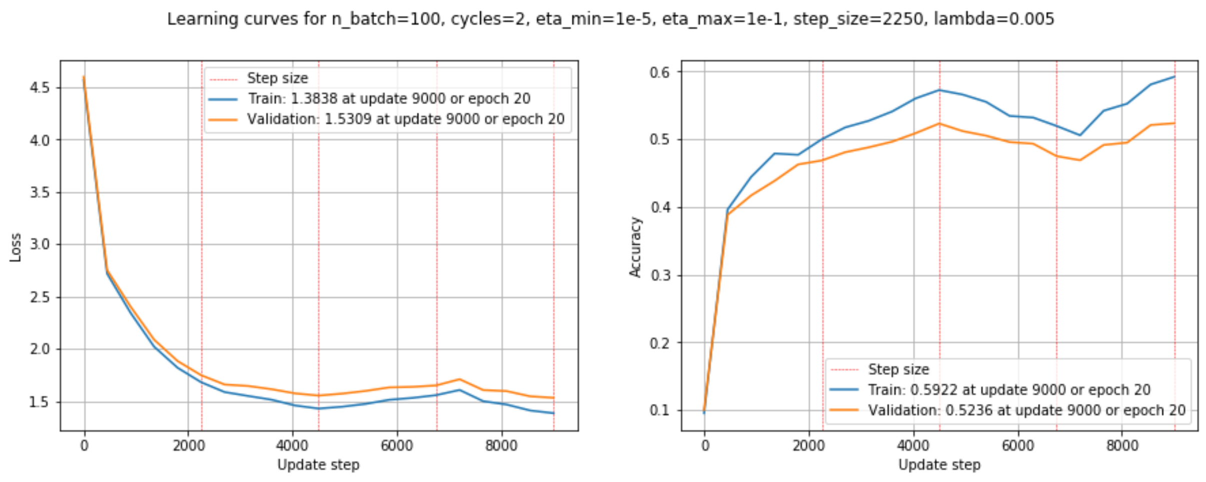


Figure 2: Learning curves for the default parametrization without *Batch Normalization* and 3 layers.

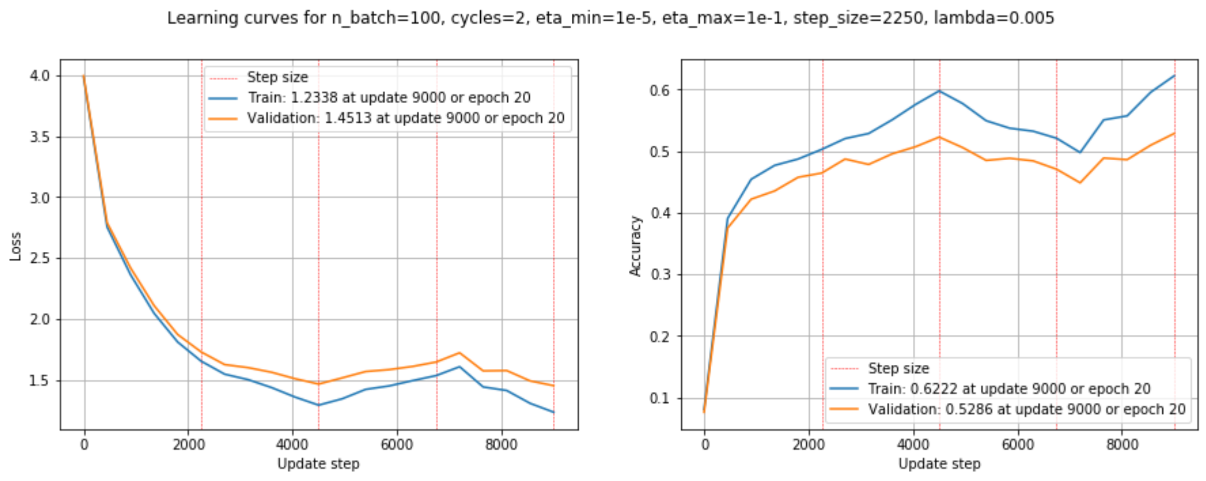


Figure 3: Learning curves for the default parametrization with *Batch Normalization* and 3 layers.

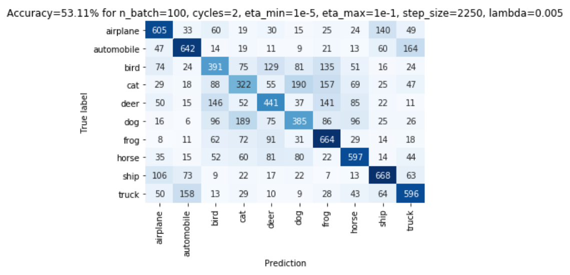


Figure 4: Test accuracy and confusion matrix for the default parametrization without *Batch Normalization* and 3 layers.

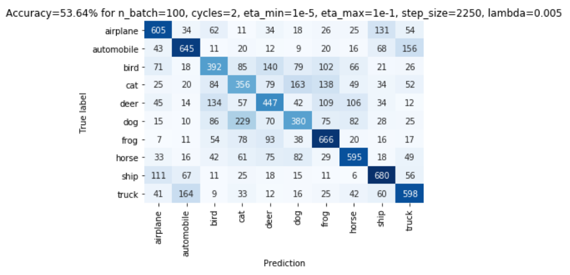


Figure 5: Test accuracy and confusion matrix for the default parametrization with *Batch Normalization* and 3 layers.

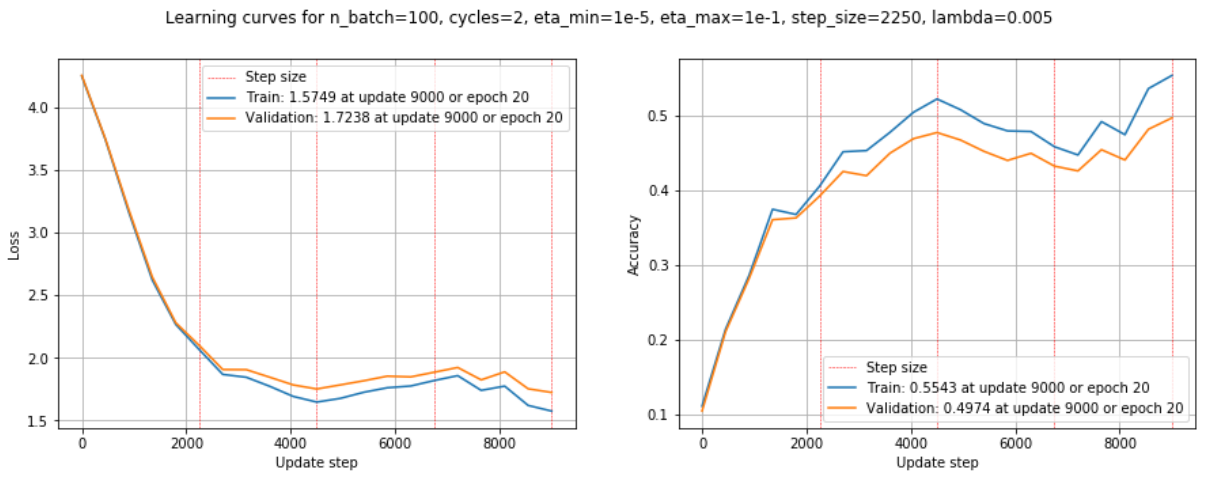


Figure 6: Learning curves for the default parametrization without *Batch Normalization* and 9 layers.

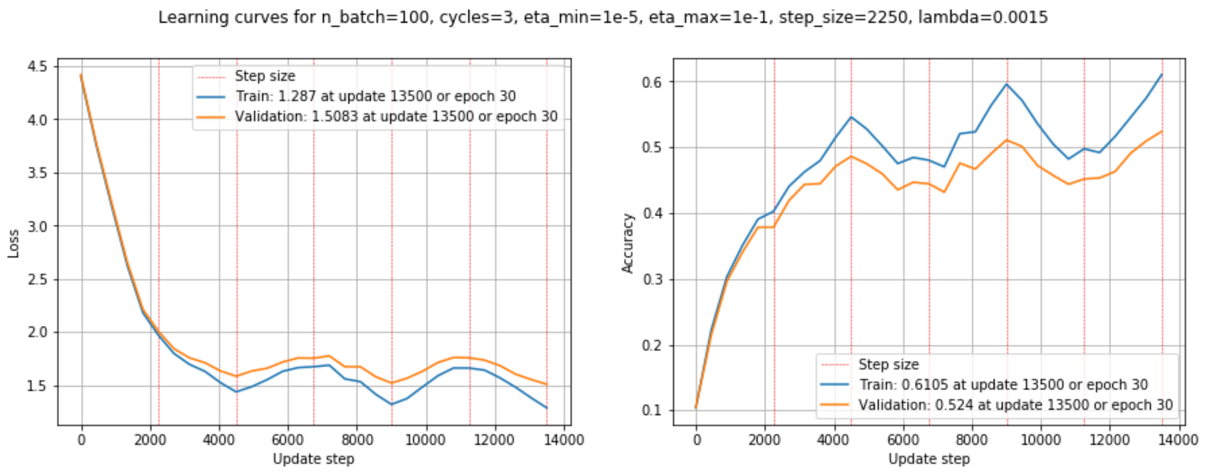
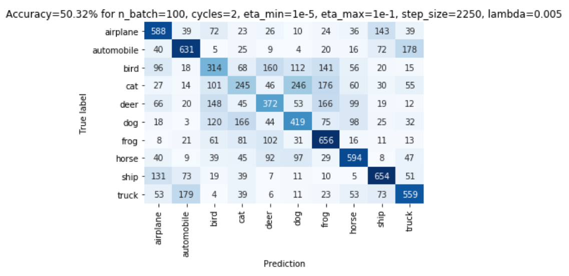
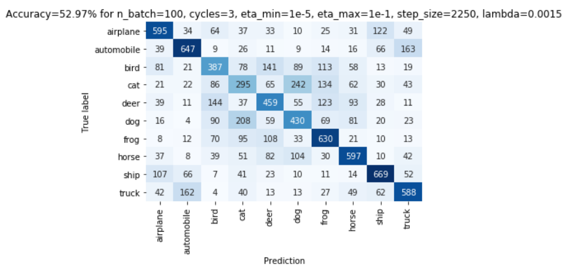


Figure 7: Learning curves for the default parametrization with *Batch Normalization* and 9 layers.



	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	588	39	72	23	26	10	24	36	143	39
automobile	40	631	5	25	9	4	20	16	72	178
bird	96	18	314	68	160	112	141	56	20	15
cat	27	14	101	245	46	246	176	60	30	55
deer	66	20	148	45	372	53	166	99	19	12
dog	18	3	120	166	44	419	75	98	25	32
frog	8	21	61	81	102	31	656	16	11	13
horse	40	9	39	45	92	97	29	594	8	47
ship	131	73	19	39	7	11	10	5	654	51
truck	53	179	4	39	6	11	23	53	73	559

Figure 8: Test accuracy and confusion matrix for the default parametrization without *Batch Normalization* and 9 layers.



	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	595	34	64	37	33	10	25	31	122	49
automobile	39	647	9	26	11	9	14	16	66	163
bird	81	21	387	78	141	89	113	58	13	19
cat	21	22	86	295	65	242	134	62	30	43
deer	39	11	144	37	459	55	123	93	28	11
dog	16	4	90	208	59	430	69	81	20	23
frog	8	12	70	95	108	33	630	21	10	13
horse	37	8	39	51	82	104	30	597	10	42
ship	107	66	7	41	23	10	11	14	669	52
truck	42	162	4	40	13	13	27	49	62	588

Figure 9: Test accuracy and confusion matrix for the default parametrization with *Batch Normalization* and 9 layers.

5 Coarse random search to set lambda

Firstly, I perform a coarse search over a very broad range of values for lambda to optimize the performance of the 3-layer network trained with *Batch Normalization*. About the range of λ values, I have tried a list of 10 values coming from $10^{random\,value}$ with uniform sampling $random\,value \sim Uniform(-5, -1)$. This is the ordered list of λ for the coarse search: [0.00034, 0.00049, 0.00056, 0.00151, 0.00156, 0.00257, 0.00383, 0.00725, 0.03690, 0.07155].

In figure 10 we can find the output of the coarse search for λ regularization parameter. Here there is the performance of each λ value for the default parametrization case: batch size 100, 2 cycles, η_{min} 1e-5, η_{max} 1e-1, step size 5 epochs or 5 times number of images divided by batch size (in our case 5·45000/100=2250). The conclusion is that within the range of values searched, we find a region of optimal values for the validation accuracy but not for the loss since the lower regularization value the better for this range searched (though not a big difference). Thus, choosing the optimal $\lambda = 0.0015$, we can see in figure 11 that the test accuracy achieved by this optimized 3-layer network with *Batch Normalization* is 53% which is not an improvement with respect to the previous test accuracy with default $\lambda = 0.005$.

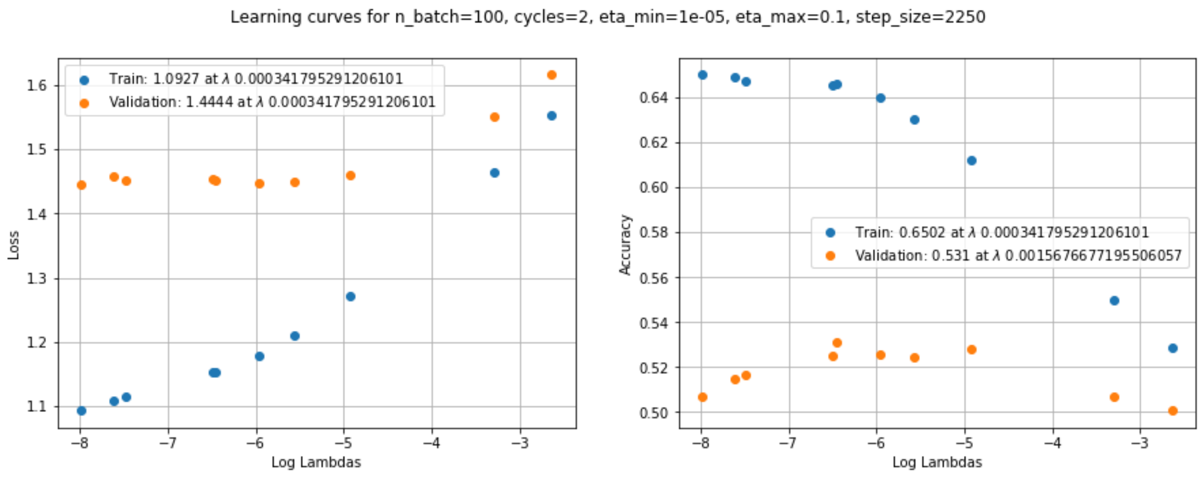
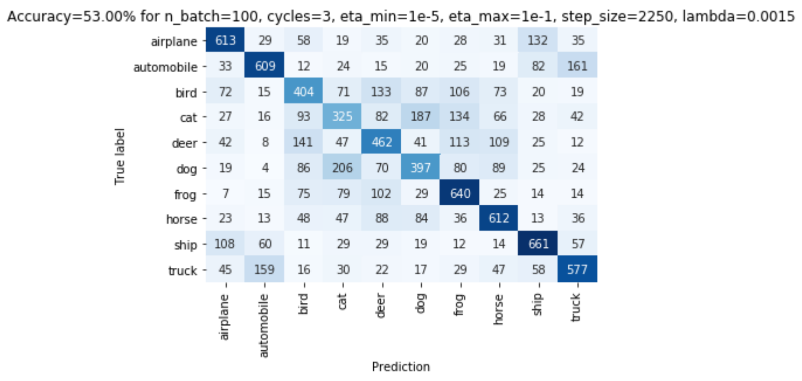


Figure 10: Coarse search for λ regularization parameter.


Accuracy=53.00% for n_batch=100, cycles=3, eta_min=1e-5, eta_max=1e-1, step_size=2250, lambda=0.0015
True label
Prediction

	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	613	29	58	19	35	20	28	31	132	35
automobile	33	609	12	24	15	20	25	19	82	161
bird	72	15	404	71	133	87	106	73	20	19
cat	27	16	93	325	82	187	134	66	28	42
deer	42	8	141	47	462	41	113	109	25	12
dog	19	4	86	206	70	397	80	89	25	24
frog	7	15	75	79	102	29	640	25	14	14
horse	23	13	48	47	88	84	36	612	13	36
ship	108	60	11	29	29	19	12	14	661	57
truck	45	159	16	30	22	17	29	47	58	577

Figure 11: Test accuracy and confusion matrix for the default parametrization with *Batch Normalization* and optimal regularization.

6 Sensitivity to initialization

The previous results seem to indicate that the batch normalization is a good thing! The frequently stated pros of using it are that training becomes more stable, higher learning rates can be used and it acts as a form of regularization. Then, let's try different training regimes for initializing the parameters, instead of using *He initialization*, by using normally distributed initialization with 0 mean and σ equal to the same value at each layer. For three runs I set σ as 1e-1, 1e-3 and 1e-4 respectively and train the network with and without *Batch Normalization.*

In table 1 we can find the test accuracy for each experiment. It is very interesting how using *Batch Normalization* helps a lot to increase the performance or training with stability the networks with lower regularization term.

Sigma	Batch Normalization	Test Accuracy
0.1	0.0	0.5296
0.1	1.0	0.5378
0.001	0.0	0.1
0.001	1.0	0.536
0.0001	0.0	0.1
0.0001	1.0	0.536

Table 1: Table of test accuracies by sensitivity to initialization experiments.

References

[1] Smith, L. N. (2015). Cyclical learning rates for training neural networks. *arXiv:1506.01186.*