Course: DD2424 - Assignment 2

Álvaro Orgaz Expósito

April 14, 2019

In this assignment, I will use a neural network with one input layer (as many nodes as image pixels), one hidden layer of 50 hidden nodes the major part of time (with *ReLu* or rectified linear units as activation function), and one output layer (as many nodes as possible image labels) with the loss function cross entropy (classification metric) and an L2 regularization term on the weights matrix. Then, the predicted class corresponds to the label with the highest predicted probability (since the output activation function is the *Softmax*). Then, the shape of the weights matrix is as many rows as output nodes and as many columns as input nodes. And for the bias only 1 column with as many rows as output nodes.

Then, for computing the gradients of the cross entropy loss function and an L2 regularization term in mini-batch mode of gradient descent (back propagation) I have used the following equations by data batches, where D is the number of input features, M is the number of hidden nodes, N is the number of training samples and C is the number of possible output labels.

- Initial input and output data:

$$\underset{[D\times N]}{X} \qquad \underset{[C\times N]}{Y} \tag{1}$$

- Forward pass (predictions):

$$\underset{[M\times N]}{S^1} = \underset{[M\times D]}{W^1} \times \underset{[D\times N]}{X} + \underset{[M\times 1]}{b^1} \tag{2}$$

$$\underset{[M\times N]}{H} = ReLu\left(\underset{[M\times N]}{S^1}\right) \quad where \quad ReLu(S^1_{mi}) = max(0, S_{mi}) \tag{3}$$

$$\underset{[C\times N]}{S} = \underset{[C\times M]}{W^2} \times \underset{[M\times N]}{H} + \underset{[C\times 1]}{b^2} \tag{4}$$

$$\underset{[C\times N]}{P} = SoftMax\left(\underset{[C\times N]}{S}\right) \quad where \quad SoftMax(S_{ji}) = \frac{exp(S_{ji})}{\sum_{c=1}^{C} exp(S_{ci})} \tag{5}$$

- Loss function (cross entropy) to minimise respect to weights:

$$J = L + \lambda(||W^1||^2 + ||W^2||^2) = \frac{1}{N}\sum_{i=1}^{N} -log\left(\underset{[1\times C]}{Y^T_{:i}} \times \underset{[C\times 1]}{P_{:i}}\right) + \lambda\left(\sum_{m,d}^{M,D} {W^1_{md}}^2 + \sum_{c,m}^{C,M} {W^2_{cm}}^2\right) \tag{6}$$

- Backward pass (updates equal to $-\eta_{update}$ times the following gradients):

+ Respect to weights and bias of output layer:

$$\frac{\partial \lambda ||W^2||^2}{\partial W^2} = 2\lambda \underset{[C\times M]}{W^2} \quad and \quad \frac{\partial L}{\partial W^2} = \frac{1}{N}\left(\underset{[C\times N]}{P} - \underset{[C\times N]}{Y}\right) \times \underset{[N\times M]}{H^T} \tag{7}$$

$$then \quad \frac{\partial J}{\partial W^2} = \frac{\partial L}{\partial W^2} + \frac{\partial \lambda ||W^2||^2}{\partial W^2} \quad and \quad \frac{\partial J}{\partial b^2} = \frac{\partial L}{\partial b^2} = \frac{1}{N}\left(\underset{[C\times N]}{P} - \underset{[C\times N]}{Y}\right) \times \underset{[N\times 1]}{1} \tag{8}$$

+ Respect to weights and bias of hidden layer:

$$\frac{\partial \lambda ||W^1||^2}{\partial W^1} = 2\lambda \underset{[M\times D]}{W^1} \quad and \quad \frac{\partial L}{\partial W^1} = \frac{1}{N}\underset{[M\times N]}{G} \times \underset{[N\times D]}{X^T} \tag{9}$$

$$with \quad \underset{[M\times N]}{G} = \left(\left(\underset{[M\times C]}{W^{2^T}} \times \left(\underset{[C\times N]}{P} - \underset{[C\times N]}{Y}\right)\right) \cdot \left(Ind(\underset{[M\times N]}{H} > 0)\right)\right) \tag{10}$$

$$then \quad \frac{\partial J}{\partial W^1} = \frac{\partial L}{\partial W^1} + \frac{\partial \lambda ||W^1||^2}{\partial W^1} \quad and \quad \frac{\partial L}{\partial b^1} = \frac{1}{N}\underset{[M\times N]}{G} \times \underset{[N\times 1]}{1} \tag{11}$$

1 Read in the data & initialize the parameters of the network

For this assignment I will just use data in the CIFAR 10 file *data batch 1* for training, the file *data batch 2* for validation and the file *test batch* for testing. But in the end, I will also use the batches 3, 4, and 5 for training and validation. 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 images batch of CIFAR 10. Importantly the class 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.

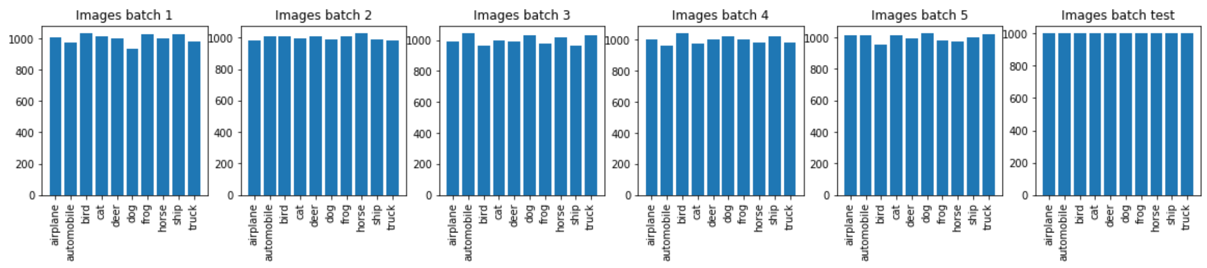


Figure 1: Distribution of labels by images batch of CIFAR 10.

I will initialize the network parameters with the two weights matrix with a Gaussian random distribution (zero mean and standard deviation $1/\sqrt{input\,dimension}$ following the Xavier[1] initialization) and the two bias matrix with zeros.

2 Compute the gradients for the network parameters

I want to check that the gradients computed in the gradients function 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 weight and bias. As error measures I will apply the absolute error and relative error in two different scenarios:

- Firstly, I study the gradients of the 20 first dimensions of the first image of *data_batch_1* without regularization. The result is:
  - For weights W^1: 100.0% of absolute errors below 1e-6, the maximum absolute error is 2.3969e-08, 99.0% of relative errors below 1e-6 and the maximum relative error is 1.7505e-05.
  - For bias b^1: 100.0% of absolute errors below 1e-6, the maximum absolute error is 1.6308e-08, 100.0% of relative errors below 1e-6 and the maximum relative error is 7.8017e-07.
  - For weights W^2: 100.0% of absolute errors below 1e-6, the maximum absolute error is 6.3421e-10, 98.0% of relative errors below 1e-6 and the maximum relative error is 4.6667e-06.
  - For bias b^2: 100.0% of absolute errors below 1e-6, the maximum absolute error is 4.6503e-08, 100.0% of relative errors below 1e-6 and the maximum relative error is 2.2656e-07.
- Secondly, I study the gradients of all dimensions of the first 5 images of *data_batch_1* with regularization. The result is:
  - For weights W^1: 100.0% of absolute errors below 1e-6, the maximum absolute error is 5.0681e-08, 80.54% of relative errors below 1e-6 and the maximum relative error is 0.0569.
  - For bias b^1: 100.0% of absolute errors below 1e-6, the maximum absolute error is 1.3667e-08, 96% of relative errors below 1e-6 and the maximum relative error is 9.0247e-06.
  - For weights W^2: 100.0% of absolute errors below 1e-6, the maximum absolute error is 1.7868e-07, 86.0% of relative errors below 1e-6 and the maximum relative error is 3.3871e-05.

[1] *Xavier initialization:* $\sigma(W) = 1/\sqrt{D}$. In general, the non-activated output of a neuron m for an input i is the matrix product between its weight vector and the input values $W_{m:} \times X_{:i}$ and it is helpful for training that, initially with random Gaussian weights, all nodes have the same output variance. Then, assuming that $X \sim Normal(\mu = 0, \sigma^2 = 1)$ and $W \sim Normal(\mu = 0, \sigma^2 = 1/D)$ are two random uncorrelated variables $cov(X, W) = 0$, the node variance is $Var(\sum_{d=1}^{D} W_{md} \cdot X_{di}) = \sum_{d=1}^{D} Var(W_{md} \cdot X_{di}) = \sum_{d=1}^{D} Var(W_{md}) \cdot Var(X_{di}) = \sum_{d=1}^{D} 1/D \cdot 1 = 1$. Note that the assumption of non-correlation provides us with: $Var(\sum_{d=1}^{D} Y_d) = \sum_{d=1}^{D} Var(Y_d)$ and $Var(WX) = E[W^2] \cdot E[X^2] - E[W]^2 \cdot E[X]^2 = (Var(W) - E[W]^2) \cdot (Var(X) - E[X]^2) - E[W]^2 \cdot E[X]^2$ where in our case with $E[W] = E[X] = 0$ we get $Var(WX) = Var(W) \cdot Var(X)$.

- For bias b^2: 100.0% of absolute errors below 1e-6, the maximum absolute error is 7.7565e-08, 90.0% of relative errors below 1e-6 and the maximum relative error is 2.2522e-05.

3 Train your 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.

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

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

In figure 2, I do a sanity check of the mini-batch coded function and we can find the learning curves for training set when using only 100 images and the parametrization: batch size 10, 2 cycles, η_{min} 1e-5, η_{max} 1e-1, step size 500, λ 0. Note that it is equivalent to 200 epochs equal to 2 cycles $\cdot$ 2 $\cdot$ step size 500 / (100 images / batch size 10). Then, we can see how the accuracy is 100% after epoch 13 and the cost is practically 0 after update step number 250 or epoch 25, which means that the training set is completely learnt (overfitting). It means that my mini-batch gradient descent algorithm with CLR is working properly.

In figure 3 we can find the triangular shape of the learning rate function by update step for the sanity check training. Starting in η_{min}, the η_{max} is achieved after one step size or 500 updates and after 500 more it goes down to η_{min} again completing 1 entire cycle.

In figure 4 we can find the learning curves for the parametrization case of assignment figure 3: batch size 100, 1 cycle, η_{min} 1e-5, η_{max} 1e-1, step size 500, λ 0.01. Note that it is equivalent to 10 epochs equal to 1 cycle $\cdot$ 2 $\cdot$ step size 500 / (10000 images / batch size 100).

In figure 5 we can find the learning curves for the parametrization case of assignment figure 4: batch size 100, 3 cycle, η_{min} 1e-5, η_{max} 1e-1, step size 800, λ 0.01. Note that it is equivalent

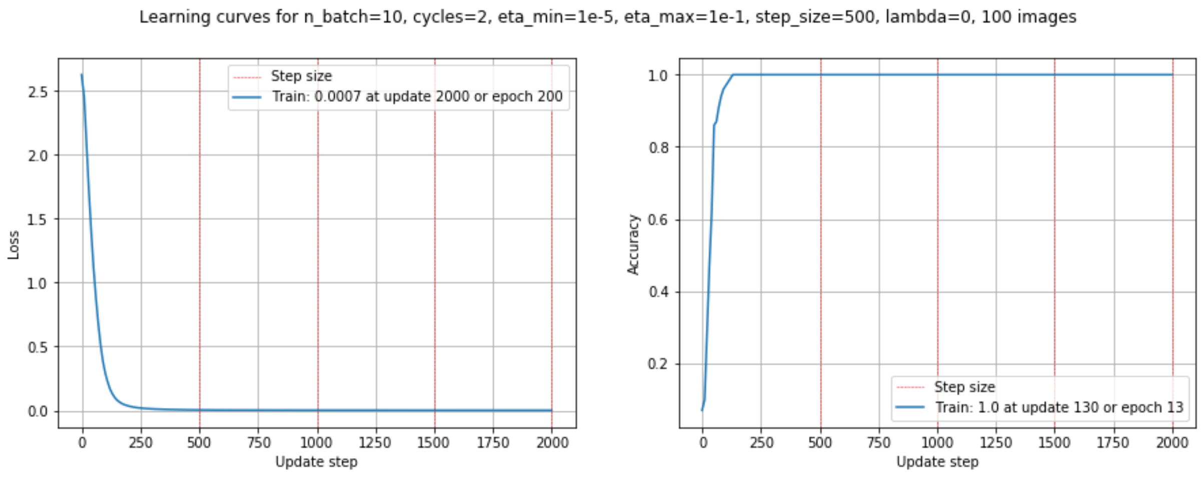


Figure 2: Learning curves of sanity check.

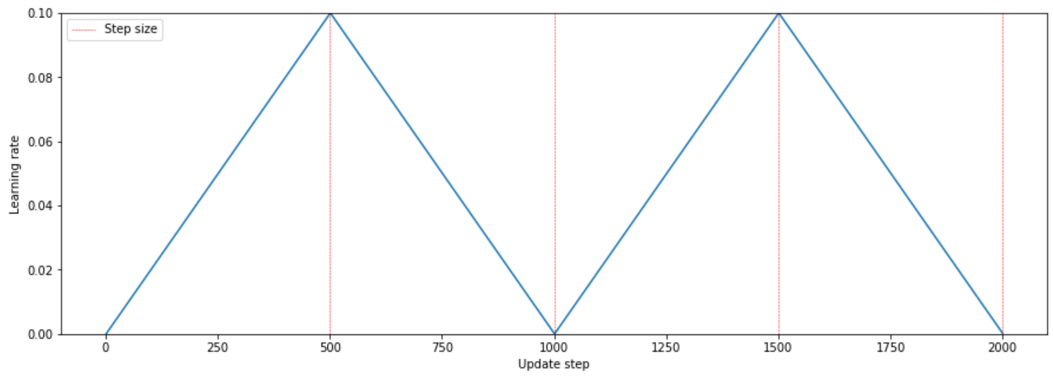


Figure 3: Learning rate by update step in CLR of sanity check.

to 48 epochs equal to 3 cycle · 2 · step size 800 / (10000 images / batch size 100). In the learning curves, we can clearly see how the loss and accuracy vary as the η_t varies at each update step, improving when approximating to η_{min} or ending cycles, and worsening when approximating to η_{max} or in the middle of cycles.

In figure 6 and 7 we can find the test accuracy and confusion matrix for each parametrization case. Firstly, we can see how after only 1 cycle with step size 500, at the end of training a test accuracy of 46% is achieved. Secondly, after only 3 cycles with step size 800 an accuracy of 47.29% is achieved.

3.1 Coarse random search to set lambda

To perform the random search I have trained for each λ several times the network from different random initializations and then I have measured the λ performance via the mean and standard deviation of accuracy for the last update step (last weights and bias network parameters) of all random initializations. That is why in the plots we can find for each λ a dot with the mean and an error bar representing $\pm$ 3 standard deviations.

Firstly, I perform a coarse search over a very broad range of values for lambda. To perform

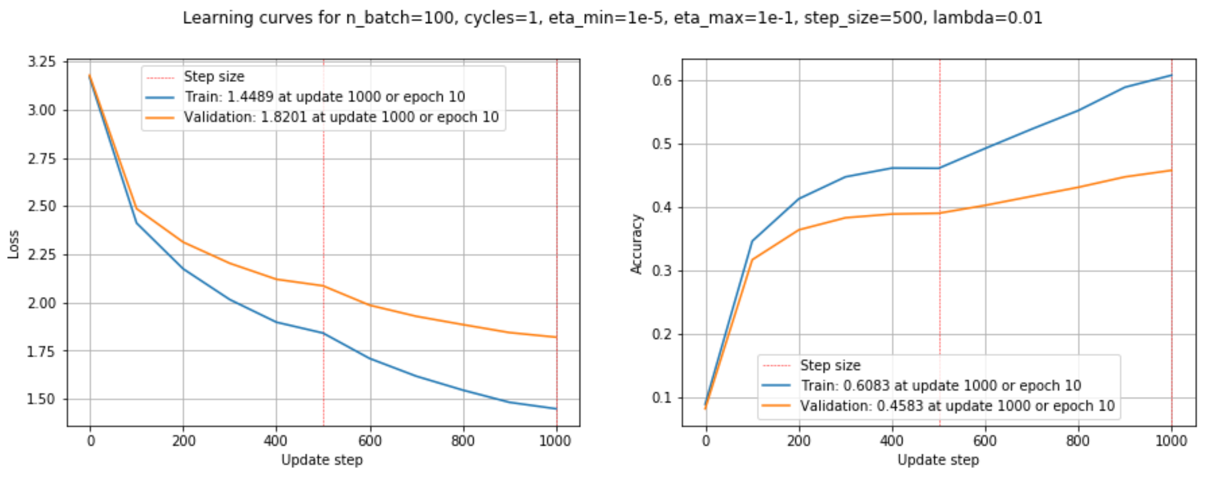


Figure 4: Learning curves for the parametrization case of assignment figure 3.

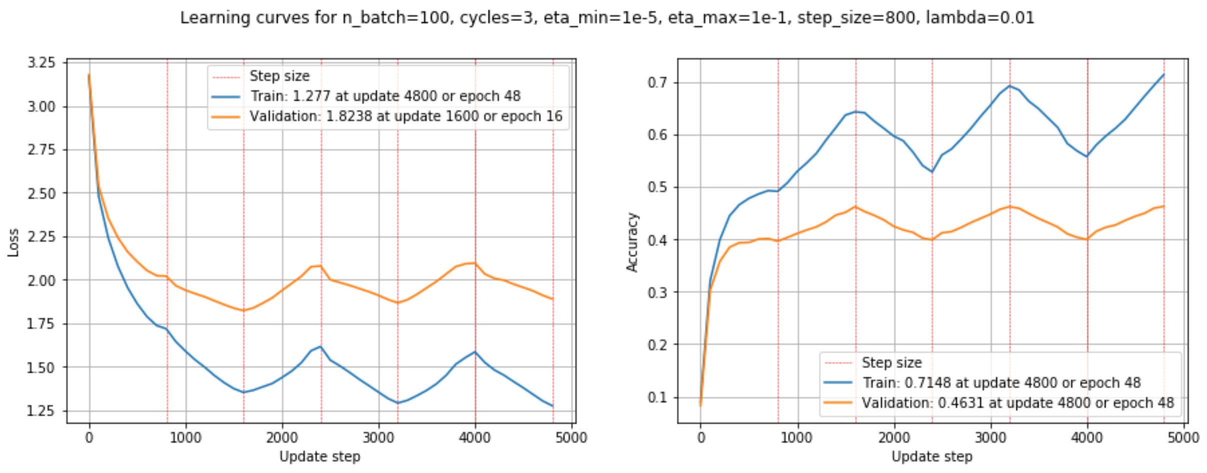
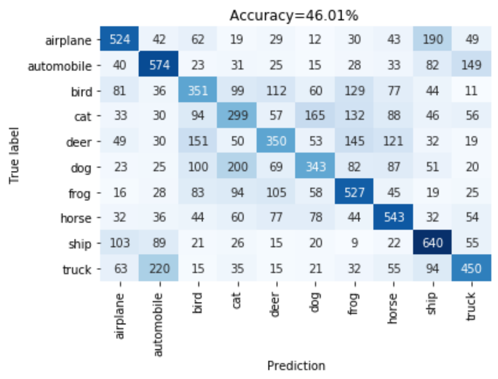


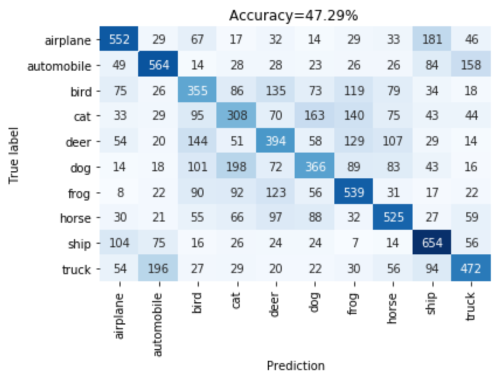
Figure 5: Learning curves for the parametrization case of assignment figure 3.



Accuracy=46.01%

True label \ Prediction	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	524	42	62	19	29	12	30	43	190	49
automobile	40	574	23	31	25	15	28	33	82	149
bird	81	36	351	99	112	60	129	77	44	11
cat	33	30	94	299	57	165	132	88	46	56
deer	49	30	151	50	350	53	145	121	32	19
dog	23	25	100	200	69	343	82	87	51	20
frog	16	28	83	94	105	58	527	45	19	25
horse	32	36	44	60	77	78	44	543	32	54
ship	103	89	21	26	15	20	9	22	640	55
truck	63	220	15	35	15	21	32	55	94	450

Figure 6: Test accuracy and confusion matrix for the parametrization case of assignment figure 3.



Accuracy=47.29%

True label \ Prediction	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	552	29	67	17	32	14	29	33	181	46
automobile	49	564	14	28	28	23	26	26	84	158
bird	75	26	355	86	135	73	119	79	34	18
cat	33	29	95	308	70	163	140	75	43	44
deer	54	20	144	51	394	58	129	107	29	14
dog	14	18	101	198	72	366	89	83	43	16
frog	8	22	90	92	123	56	539	31	17	22
horse	30	21	55	66	97	88	32	525	27	59
ship	104	75	16	26	24	24	7	14	654	56
truck	54	196	27	29	20	22	30	56	94	472

Figure 7: Test accuracy and confusion matrix for the parametrization case of assignment figure 4.

this search I use most of the training data available (45000 images) and the rest for the validation (5000 randomly selected) since it is convenient to use the same ballpark amount of data than when training the network in the final phase with all training data before implementing.

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: [0.000341795291206101, 0.0004950159553733192, 0.0005627932047415164, 0.0015119336467640998, 0.0015676677195506057, 0.002576638574613588, 0.00383333215615666, 0.007257005721594274, 0.03690557729213758, 0.07155682161754859]

In figure 8 we can find the output of the coarse search for λ regularization parameter. Here there is the performance of each λ value for the parametrization case: batch size 100, 2 cycles, η_{min} 1e-5, η_{max} 1e-1, step size 2 epochs or 2 times number of images divided by batch size (in our case 2·45000/100=900). Note that it is equivalent to 8 epochs equal to 2 cycles · 2 · step size 900 / (45000 images / batch size 100). The conclusion is the lower λ the better validation performance.

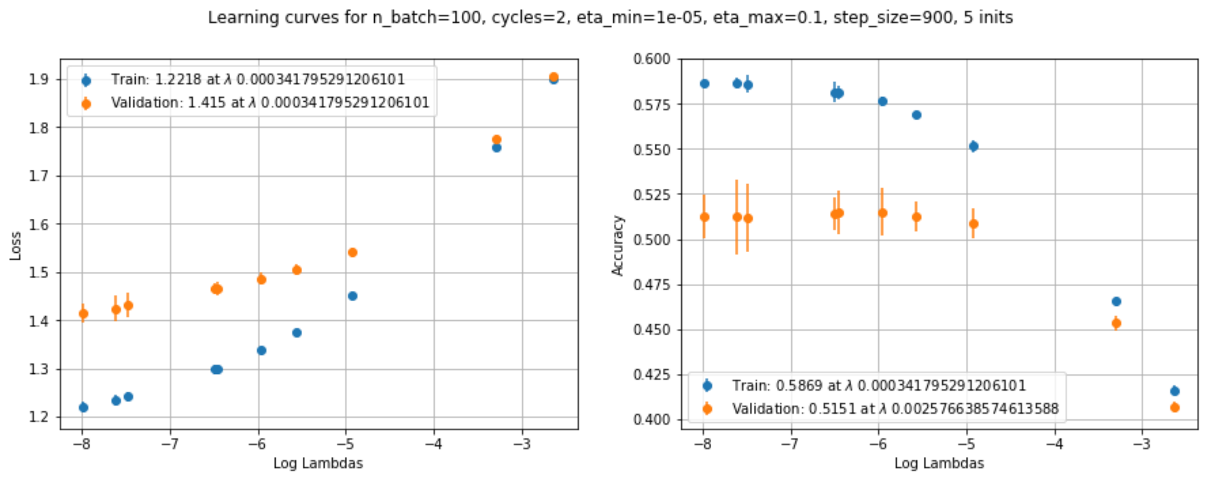


Figure 8: Coarse search for λ regularization parameter.

3.2 Fine random search to set lambda

Secondly, I perform a fine search over a defined range of values for lambda using the coarse search as a reference. About the range of λ values, I have tried a list of 8 values between the range [0, 0.007] with equal width 0.001. For defining the range of the fine search I have used approximately the value of the 8th λ value since I get similar validation results within this range. This is the ordered list of λ for the fine search: $[0, 0.001, 0.002, 0.003, 0.004, 0.005, 0.006, 0.007]$.

In figure 9 we can find the output of the fine search for λ regularization parameter. Here there is the performance of each λ value for the same parametrization case than coarse search. The conclusion is that various values get similar performance in validation set but I will choose $\lambda = 0.001$ (higher validation accuracy) as the optimal value for the following sections.

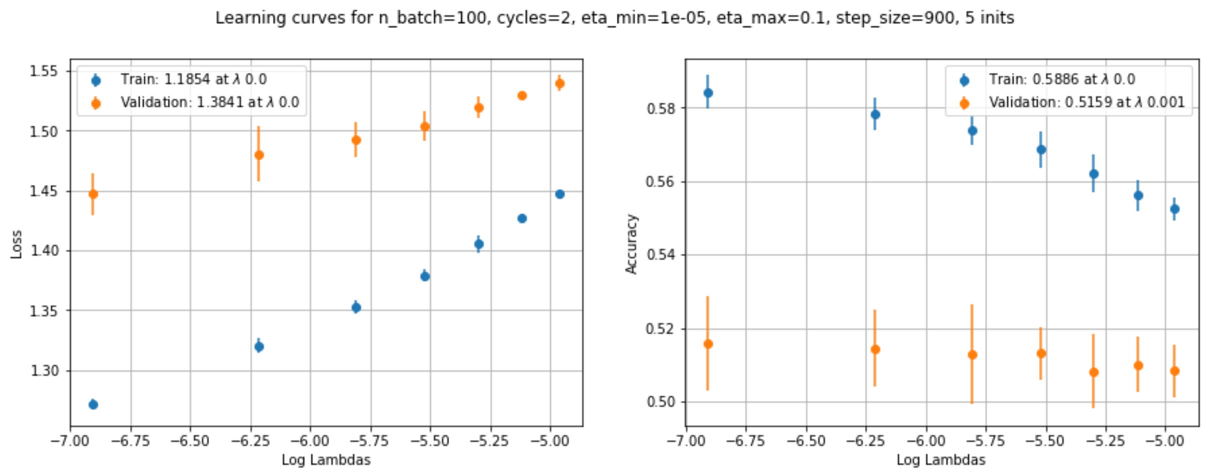


Figure 9: Fine search for λ regularization parameter.

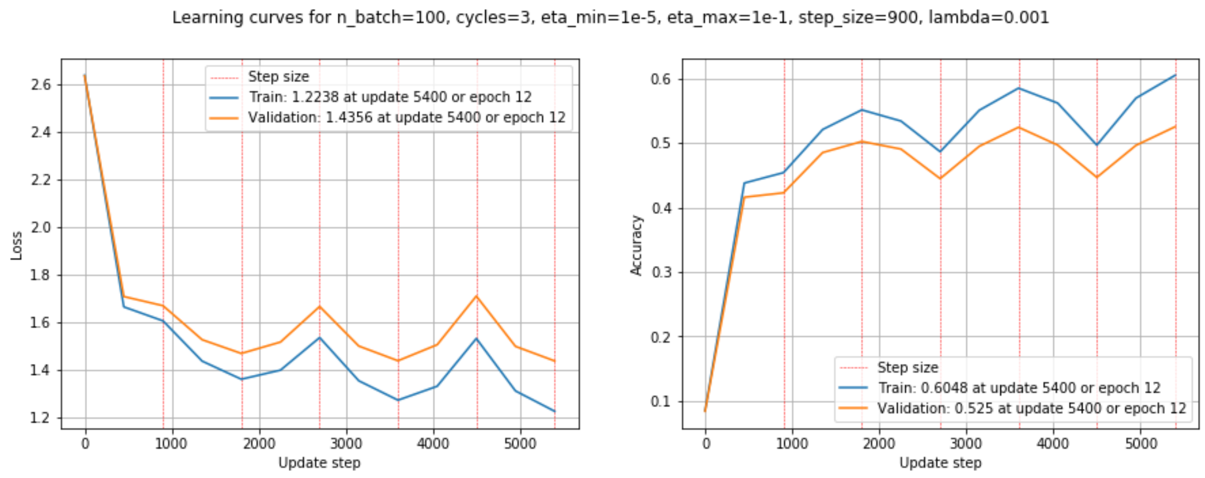
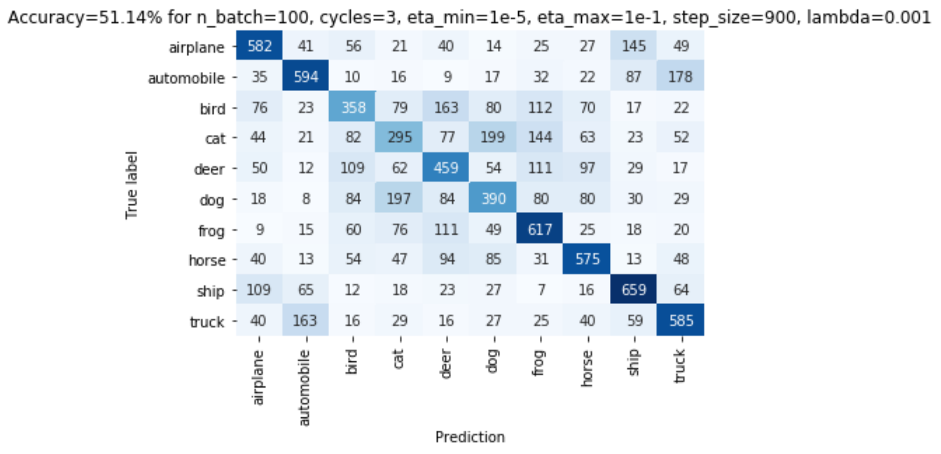


Figure 10: Learning curves for the best parametrization.



	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	582	41	56	21	40	14	25	27	145	49
automobile	35	594	10	16	9	17	32	22	87	178
bird	76	23	358	79	163	80	112	70	17	22
cat	44	21	82	295	77	199	144	63	23	52
deer	50	12	109	62	459	54	111	97	29	17
dog	18	8	84	197	84	390	80	80	30	29
frog	9	15	60	76	111	49	617	25	18	20
horse	40	13	54	47	94	85	31	575	13	48
ship	109	65	12	18	23	27	7	16	659	64
truck	40	163	16	29	16	27	25	40	59	585

Figure 11: Test accuracy and confusion matrix for the best parametrization.

3.3 Best lambda

Once the best regularization parameter is found, I train the same network parametrization but with 1 more cycle to see how much we can increase the performance in validation and test sets. In figure 10 we can find the learning curves for the trained network with the best parametrization case (same than coarse search with λ 0.001) where we can see how accuracy of 52.5% in the validation set is achieved. In figure 11 we can find its test accuracy and confusion matrix which achieves an accuracy of 51.14%.

References

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