DD2424 Deep Learning in Data Science
Assignment 3

Option 1

"K-layer Neural Network with Multiple Outputs to Classify Images from CIFAR-10 Dataset"

Prashant Yadava

1 Gradient Computation Check

To verify the accuracy of the analytical gradient computation, it was compared with the corresponding numerical gradient, which was obtained by translating the provided Matlab code into Python. The main training proceeded only after confirming that the absolute difference between the analytical and numerical gradients was within an acceptable threshold of **1e-6**.

To ensure computational efficiency, only the first 10 training examples and the first five features from the training dataset X_train were utilized, and regularization was turned off.

The relevant code and functions are included in the submitted code package; thus, only the results are presented here. The tables below illustrate that the absolute differences between the gradients were predominantly within the threshold of 1e-6. The parameters Gamma (γ) and Beta (β) refer to those used in the batch normalization layers.

2-layer network, 50 hidden nodes

```
Layer 0:
 Weight
    % of absolute errors below 1e-6: 99.40%
    Maximum absolute error: 1.40e-06
 Bias
  % of absolute errors below 1e-6: 100.00%
  Maximum absolute error: 8.88e-10

 Gamma
  % of absolute errors below 1e-6: 100.00%
  Maximum absolute error: 1.91e-08

 Beta
  % of absolute errors below 1e-6: 100.00%
  Maximum absolute error: 2.48e-08
-------------------------------------------
Layer 1:
 Weight
  % of absolute errors below 1e-6: 100.00%
  Maximum absolute error: 8.29e-08

 Bias
  % of absolute errors below 1e-6: 100.00%
  Maximum absolute error: 7.18e-08
```

3-layer network, 50 & 50 hidden nodes

```
Layer 0:
 Weights
  % of absolute errors below 1e-6: 99.60%
  Maximum absolute error: 1.14e-06
 Bias
  % of absolute errors below 1e-6: 100.00%
  Maximum absolute error: 4.44e-10
 Gamma
  % of absolute errors below 1e-6: 100.00%
  Maximum absolute error: 4.48e-08
 Beta
  % of absolute errors below 1e-6: 100.00%
  Maximum absolute error: 6.08e-08
-----------------------------------------
Layer 1:
 Weights
  % of absolute errors below 1e-6: 100.00%
  Maximum absolute error: 2.52e-07
 Bias
  % of absolute errors below 1e-6: 100.00%
  Maximum absolute error: 3.89e-17
 Gamma
  % of absolute errors below 1e-6: 100.00%
  Maximum absolute error: 2.67e-08
 Beta
  % of absolute errors below 1e-6: 100.00%
  Maximum absolute error: 2.92e-08
-----------------------------------------
Layer 2:
 Weights
  % of absolute errors below 1e-6: 100.00%
  Maximum absolute error: 8.74e-08
 Bias
  % of absolute errors below 1e-6: 100.00%
  Maximum absolute error: 6.77e-08
```

The absolute differences consistently being under the reasonable threshold was seen as a confirmation of the correctness of the analytical gradient computation implementation.

2 Training Neural Networks

3-layer network

In this section, two execution runs of a 3-layer neural network were performed: one without batch normalization and the other with batch normalization. The network hyper-parameters used were set to the default values specified in the assignment description.
These are:

Table 1: Hyperparameters used for training the neural network.

Parameter	**Value**
Cyclic Learning Rate (min)	1×10^{-5}
Cyclic Learning Rate (max)	1×10^{-1}
Number of Cycles	2
Batch Size	100
Step Size	$5 \times \frac{45000}{\text{batch\_size}}$
Regularization Parameter (λ)	0.005

Without Batch Normalization

Figure 1 shows that the the 3-layer neural network achieved a test accuracy of 52% without batch normalization.

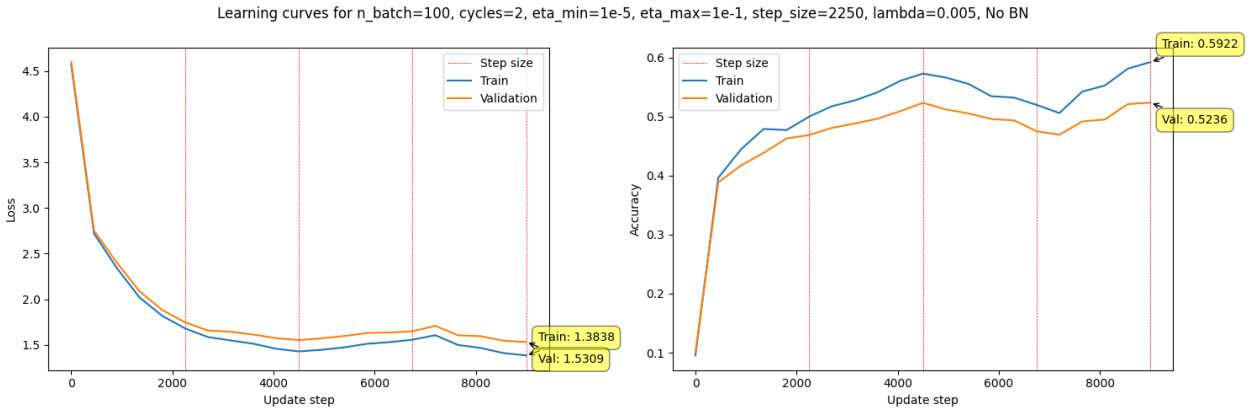

Figure 1: Loss and Accuracy Plots for 3 layer neural network with default values, without batch normalization

With Batch Normalization

Figure 2 shows the learning curves for the 3-layer neural network with batch normalization turned on. The test accuracy was 53%.

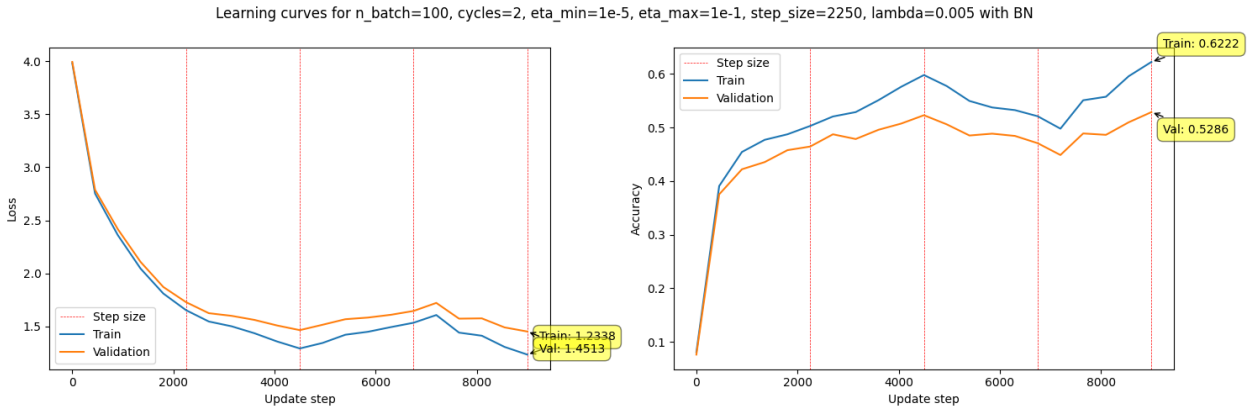

Figure 2: Loss and Accuracy Plots for 3 layer neural network with default values, with batch normalization

Training a 9-layer network

We train a 9 layer neural network dimensions with `[50,30,20,20,10,10,10,10]` as the dimensions of the hidden layers.

The default parameters listed in Table 1 earlier for the 3-layer network were used for this network as well.

Without batch normalization

Figure 3 shows the test accuracy of the 9-layer network to be `49.8%` when batch normalization wasn't used.

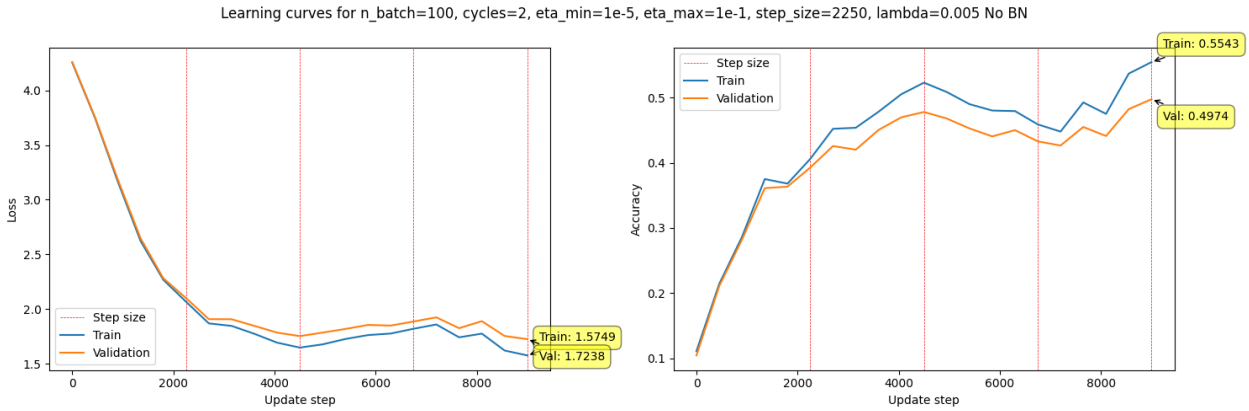

Figure 3: Loss and Accuracy Plots for 9 layer neural network with default values, without batch normalization

With batch normalization

Figure 4 shows the test accuracy of the 9-layer network to be `51.1%` when batch normalization was used.

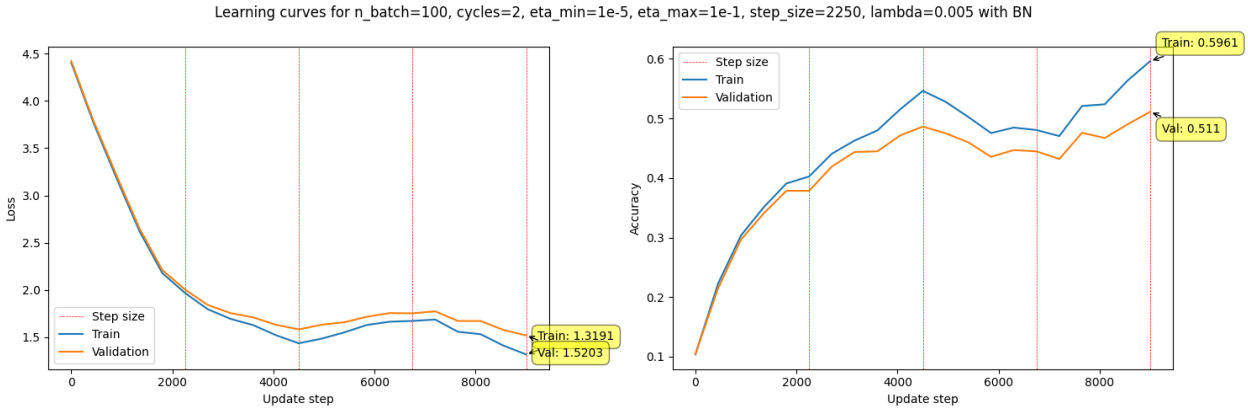

Figure 4: Loss and Accuracy Plots for 9 layer neural network with default values, without batch normalization

The conclusion from these results is that increasing the complexity of the neural network, e.g., the case of moving from 3 to 9 layers, does not necessarily lead to increase in better performing networks. Although, the technique of Batch Normalization can networks increase accuracy, its effects are more obvious in deeper neural networks than shallow ones.

3 Coarse random search to find best lambda

Table 2 summarizes the candidate values of the regularization parameter λ, uniformly sampled within the range [0, 1], for optimizing a three-layer neural network with batch normalization. The table provides the corresponding training and validation losses, along with their respective accuracies. The results indicate that λ=0.0034 yields the highest performance, achieving a training accuracy of `59.2%` and a validation accuracy of `51.1%`.

The default hyper-parameters presented earlier in Table 1 were selected for this run as well.

Table 2: Comparison of Lambda Values, Training Loss, Accuracy, Validation Loss, and Validation Accuracy

Lambda	**Train Loss**	**Train Accuracy**	**Validation Loss**	**Validation Accuracy**
1.2106×10^{-5}	1.2298	0.5671	1.4737	0.4762
2.2561×10^{-5}	1.2185	0.5745	1.4700	0.4756
4.7473×10^{-5}	1.2531	0.5673	1.4928	0.4722
6.1980×10^{-5}	1.2372	0.5748	1.4952	0.4816
7.9284×10^{-5}	1.2264	0.5777	1.4604	0.4894
9.8634×10^{-4}	1.3091	0.5852	1.5421	0.4958
3.4245×10^{-3}	1.3077	0.5920	1.5182	0.5110
9.8904×10^{-3}	1.3938	0.5694	1.5489	0.5110
1.1019×10^{-2}	1.4061	0.5653	1.5409	0.5100
1.2170×10^{-2}	1.4252	0.5565	1.5532	0.5004

4 Sensitivity to initialization

Table 3 reports the test accuracies obtained from various experimental runs using different initialization parameters sampled from a normal distribution with zero mean and varying values of the σ parameter. It is important to note that this initialization method differs from the *He* initialization employed in previous experiments.

Each row in the table represents a distinct combination of σ values and whether batch normalization was applied, alongside the corresponding test accuracy.

The results demonstrate that batch normalization proves to be particularly effective when σ is small (0.001 and 0.0001). In these cases, batch normalization dramatically improves test accuracy from 0.1 to 0.536, highlighting its ability to stabilize training and enable the network to perform significantly better. Additionally, even for larger σ values (e.g., $\sigma = 0.1$), batch normalization provides a moderate increase in accuracy, indicating its utility across different initialization scenarios.

Sigma	**Batch Normalization**	**Test Accuracy**
0.1	No	0.5296
0.1	Yes	0.5378
0.001	No	0.1
0.001	Yes	0.536
0.0001	No	0.1
0.0001	Yes	0.536

Table 3: Comparison of Test Accuracy across different Sigma values with and without Batch Normalization.

Next, the plots for these combination of values are presented.

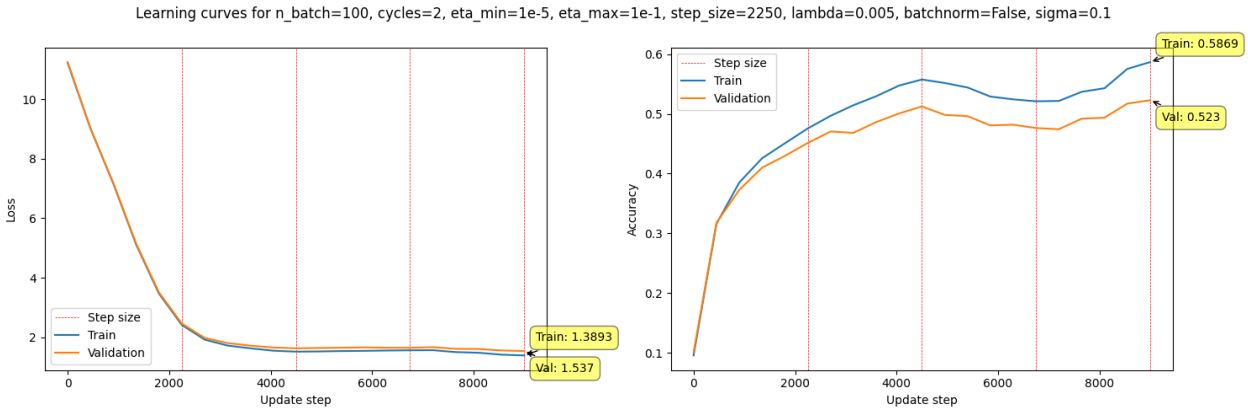

Figure 5: Loss and Accuracy curve for $\sigma = 0.1$ and batch normalization = False

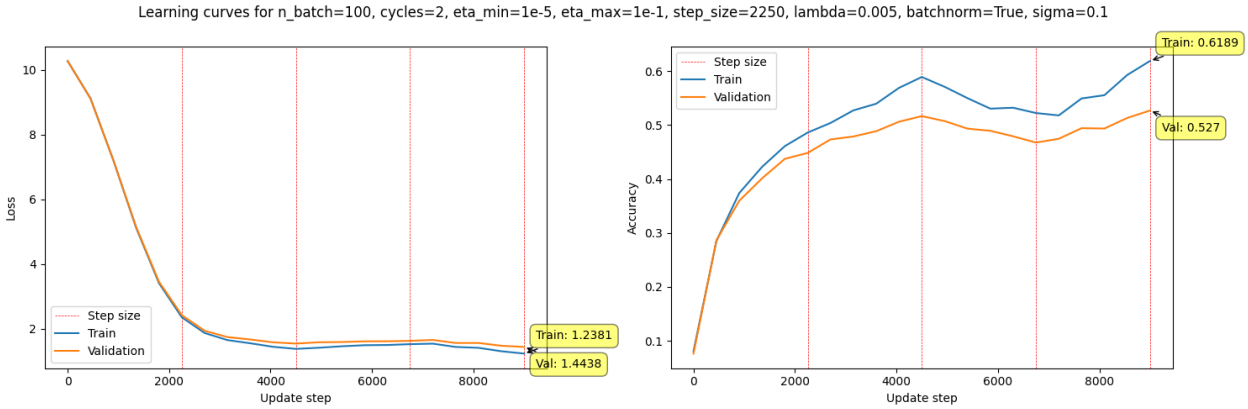

Figure 6: Loss and Accuracy curve for $\sigma = 0.1$ and batch normalization = True

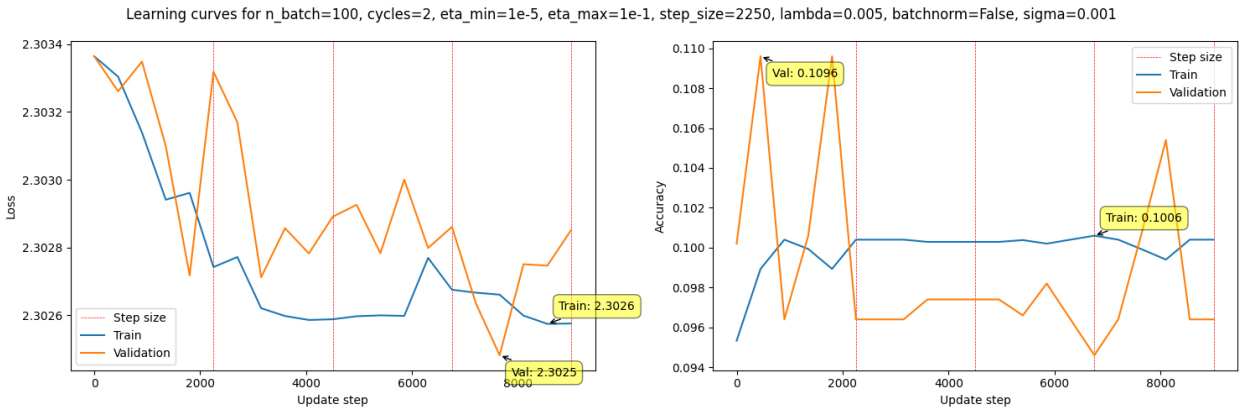

Figure 7: Loss and Accuracy curve for $\sigma = 0.001$ and batch normalization = False

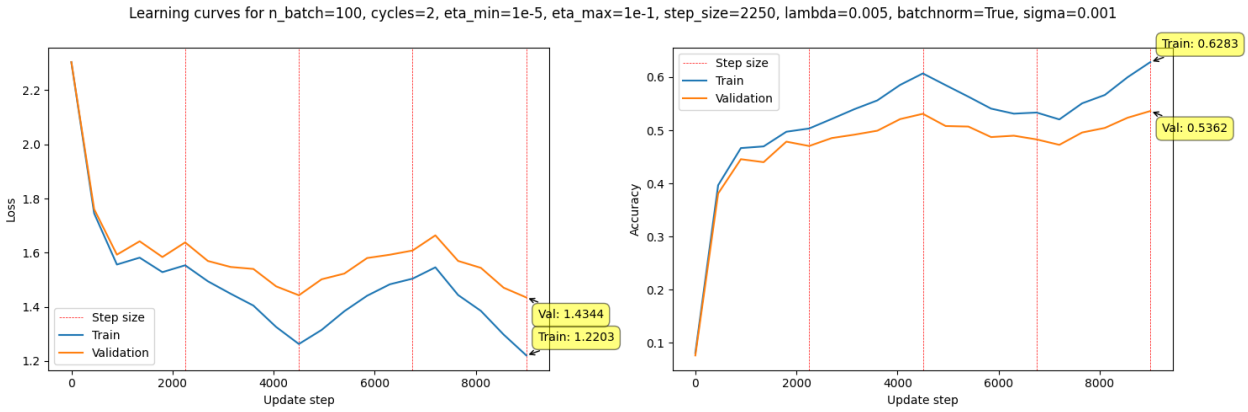

Figure 8: Loss and Accuracy curve for $\sigma = 0.001$ and batch normalization = True

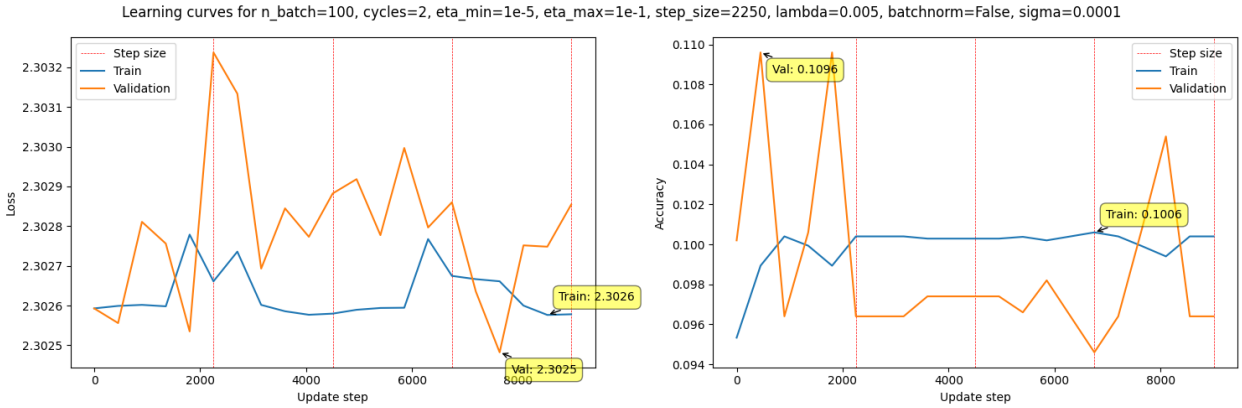

Figure 9: Loss and Accuracy curve for $\sigma = 0.0001$ and batch normalization = False

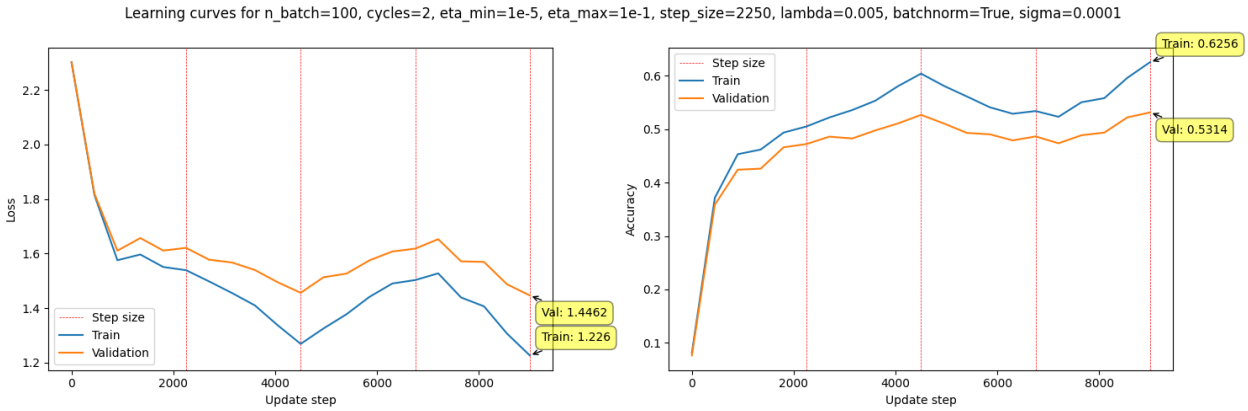

Figure 10: Loss and Accuracy curve for $\sigma = 0.0001$ and batch normalization = True

A noteworthy observation among the plots with lower sigma, σ values (0.001 and 0.0001) is the significant fluctuation in both loss and accuracy for both training and validation sets when batch normalization is not employed. The learning curves in these instances exhibit a lack of smoothness and demonstrate erratic behavior, as the network presents increased sensitivity to the scale of inputs and gradients. This sensitivity can potentially lead to more dramatic changes in the loss landscape from one batch to another.

Batch normalization serves to normalize the inputs to each layer, thereby potentially reducing the internal covariate shift. Furthermore, it contributes to a smoother and more predictable optimization landscape.

When the weight initialization (directly influenced by the σ parameter) and other hyperparameters remain constant, batch normalization alters the interaction of these parameters with the learning process. Batch normalization plays a crucial role in enhancing the modelś robustness to sub-optimal initialization or learning rates.

In conclusion, the observed smoothness in the graphs with batch normalization can be primarily attributed to its stabilizing effect. This normalization technique facilitates more consistent and efficient learning, resulting in smoother learning curves and generally improved performance, as evidenced by the higher accuracy achieved in the batch-normalized version of the model.