Name(s): Sharvi Tomar, Robert Bookland
NetID(s): stomar2, rmb5
Team name on Kaggle leaderboard: Team RS

For each of the sections below, your reported test accuracy should approximately match the accuracy reported on Kaggle.

Briefly describe the hyperparameter tuning strategies you used in this assignment. Then record your optimal hyperparameters and test/val performance for the four different network types.

Initially, we were searching/tuning the hyperparameters by hand. We briefly experimented with changing batch size, but we didn't notice any significant improvements to either accuracy or training time. Additionally, we also experimented with the number of epochs, and ended up settling on 50 epochs because it offered a relatively short training time and the accuracy vs epochs charts seemed to tell a tale of convergence.
Onto the hyperparameters, we started out by changing the learning rate by powers of ten, and increasing the hidden layer size by steps of 10. We also played with the regularization constant and kept it under 0.1 until we realized that our implementation included a divide by m (sample size) when calculating both the cost and gradients. We then allowed regularization to take values of up to 20 which becomes functionally 0.1 when divided by 200.
After much frustration, we ended up writing some nested for loops that would search over the hyperparameter space, and output candidates whenever the validation accuracy approached the kaggle benchmarks. On my hardware it took roughly 1.5 minutes to train an individual model, and I wrote the nested for loops s.t. it would take ~8.5 hours to complete assuming all of the models worked. Occasionally, the net construction would break due to some overflow issue often related to a large learning rate. In these instances the validation accuracy would be very low, and I used this check as a test to break from sections of the loop to avoid training presumably broken models. Under these conditions each loop usually terminated in around 3hrs. These loops showed us that a learning rate around 0.0041 for both 2- and 3-layer SGD gave us the most candidate models. Using the output from these for loops as starting points and guidance we started fine tuning models to get our final results.
This general story of hand search, for-loop search, and then fine tune applied to all of the SGD/Adam models.

Two-layer Network Trained with SGD

Best hyperparameters (if you changed any of the other default hyperparameters like initialization method, etc. please note that as well):

Batch size:	epochs = 50 batch_size = 200
Learning rate:	learning_rate = 0.0035, learning_rate_decay =

	0.99
Hidden layer size:	hidden_size = 45
Regularization coefficient:	regularization = 15

Record the results for your best hyperparameter setting below:

Validation accuracy:	87.83
Test accuracy:	87.55

Three-layer Network Trained with SGD

We ended up taking a very similar approach to the 2-layer SGD. We briefly experimented with differing the hidden sizes from each other, which lead us to fixing a bug in the underlying code. We accidentally had a -1 instead of a -i when updating the gradient of the bias of layers. This didn't break dimensionally when we had the only 2 layers or matching hidden sizes, but we still fixed the code and ensured that the problem was resolved. Our for loops only searched across matching hidden sizes because this allowed to essentially copy the 2-layer loop, and not add extensive computation searching across another hyperparameter.

Batch size:	epochs = 50 batch_size = 200
Learning rate:	learning_rate = 0.0035, learning_rate_decay = 0.99
Hidden layer size:	hidden_size = 45
Regularization coefficient:	regularization = 15

Record the results for your best hyperparameter setting below:

Validation accuracy:	88.12
Test accuracy:	87.34

Two-layer Network Trained with Adam

Best hyperparameters (if you changed any of the other default hyperparameters like initialization method, etc. please note that as well):

Batch size:	batch_size = 200

	epochs = 50
Learning rate:	learning_rate =0.00031
Hidden layer size:	hidden_size = 35
Regularization coefficient:	regularization = 50
β_1	b1: float = 0.9
β_2	b2: float = 0.999

Record the results for your best hyperparameter setting below:

Validation accuracy:	88.01
Test accuracy:	87.1

Again a very similar story of using for-loops. Our Adam implementation seemed very sensitive to the learning rate so we ended up searching/using smaller values. Unfortunately, after tuning and submitting our current best Kaggle results we discovered a bug in our Adam code. When updating m_hat and v_hat we were not raising b1 & b2 to the power of t. We then fixed the bug to be a truer Adam update, and retrained/researched across the hyperparameters. Unfortunately, we discovered the bug closer to the due date of the assignment. Since the discovery and retraining/re-searching our models, we have resubmitted results obtained with corrected code. We also looked under submissions, and checked "Use for Final Score" for the appropriate models if it wasn't already the best.

Three-layer Network Trained with Adam

Essentially, the same story as the 3-layer SGD search with the smaller learning rates of the Adam model. We never submitted a buggy Adam 3-layer, so the results on the leaderboard should only be correct results.

Best hyperparameters (if you changed any of the other default hyperparameters like initialization method, etc. please note that as well):

Batch size:	batch_size = 200 epochs = 85
Learning rate:	learning_rate = 1.5e-3
Hidden layer size:	hidden_size = [65,65]
Regularization coefficient:	regularization = 17
β_1	b1: float = 0.9
β_2	b2: float = 0.999

Record the results for your best hyperparameter setting below:

Validation accuracy:	88.34
Test accuracy:	87.60

Comparison of SGD and Adam

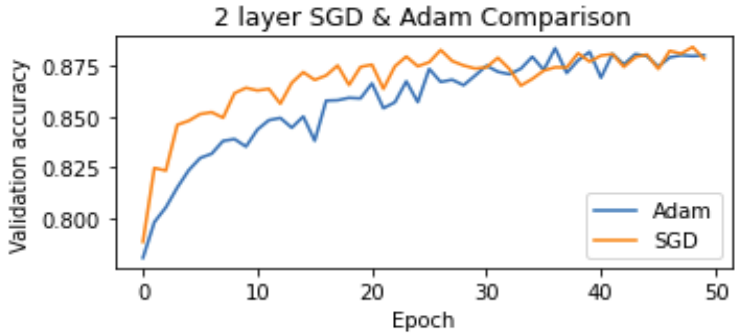
Attach two plots, one of the training loss for each epoch and one of the validation accuracy for each epoch. Both plots should have a line for SGD and Adam. Be sure to add a title, axis labels, and a legend.

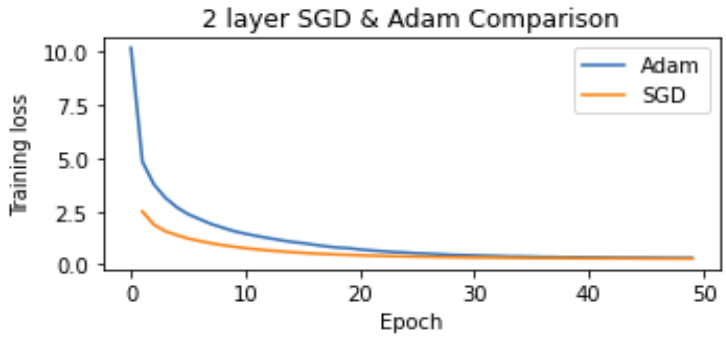
Compare the performance of SGD and Adam on training times and convergence rates. Do you notice any difference? Note any other interesting behavior you observed as well.

Please find the comparison plots on the next page.

What we notice when comparing the performance of SGD vs Adam for a given learning rate is that Adam's training loss appears to converge much faster than SGD. However Adam's validation accuracy also appears to vary much more greatly between epochs than we observe in SGD. Part of this peculiarity could be that we hesitated to apply a learning decay in Adam as we felt that part of Adam's purpose was to define the training schedule similar to how the decay helps define the training schedule in SGD.

1. 2 layer SGD and 2 layer Adam Comparison Plot





2. 3 layer SGD and 3 layer Adam Comparison Plot