DA6401 - Assignment 1

Write your own backpropagation code and keep track of your experiments using wandb.ai.

Deadline: 10th March 2025, Hard deadline. No extensions will be provided.

Mohit Singh da24m010

Created on February 27 | Last edited on March 17

Instructions

- The goal of this assignment is twofold: (i) implement and use gradient descent (and its variants) with backpropagation for a classification task (ii) get familiar with Wandb which is a cool tool for running and keeping track of a large number of experiments
- This is a **individual assignment** and no groups are allowed.
- Collaborations and discussions with other students is strictly prohibited.
- You must use Python (NumPy and Pandas) for your implementation.
- You cannot use the following packages from Keras, PyTorch, Tensorflow: optimizers, layers
- If you are using any packages from Keras, PyTorch, Tensorflow then post on Moodle first to check with the instructor.
- You have to generate the report in the same format as shown below using wandb.ai. You can start by cloning this report using the clone option above. Most of the plots that we have asked for below can be (automatically) generated using the

APIs provided by `wandb.ai`. You will upload a link to this report on Gradescope.

- You also need to provide a link to your GitHub code as shown below. Follow good software engineering practices and set up a GitHub repo for the project on Day 1. Please do not write all code on your local machine and push everything to GitHub on the last day. The commits in GitHub should reflect how the code has evolved during the course of the assignment.
- You have to check Moodle regularly for updates regarding the assignment.

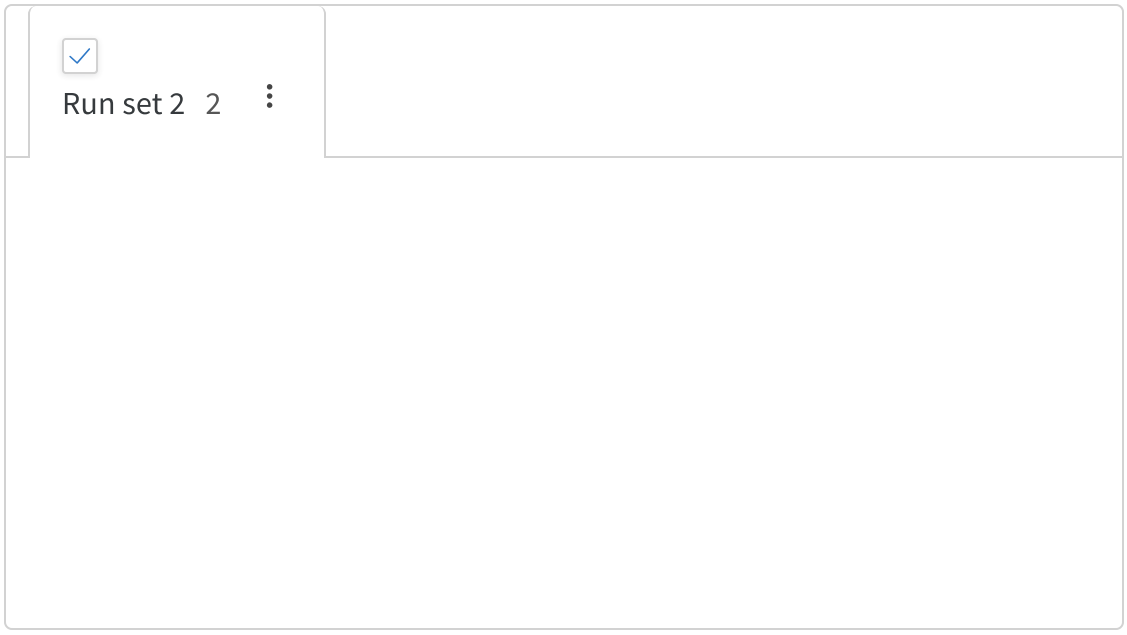
Problem Statement

In this assignment you need to implement a feedforward neural network and write the backpropagation code for training the network. We strongly recommend using numpy for all matrix/vector operations. You are not allowed to use any automatic differentiation packages. This network will be trained and tested using the Fashion-MNIST dataset. Specifically, given an input image (28 x 28 = 784 pixels) from the Fashion-MNIST dataset, the network will be trained to classify the image into 1 of 10 classes.

Your code will have to follow the format specified in the `Code Specifications` section.

Question 1 (2 Marks)

Download the fashion-MNIST dataset and plot 1 sample image for each class as shown in the grid below. Use `from keras.datasets import fashion_mnist` for getting the fashion mnist dataset.



Question 2 (10 Marks)

Implement a feedforward neural network which takes images from the fashion-mnist data as input and outputs a probability

distribution over the 10 classes.

Your code should be flexible such that it is easy to change the number of hidden layers and the number of neurons in each hidden layer.

Question 3 (24 Marks)

Implement the backpropagation algorithm with support for the following optimisation functions

- sgd
- momentum based gradient descent
- nesterov accelerated gradient descent
- rmsprop
- adam
- nadam

(12 marks for the backpropagation framework and 2 marks for each of the optimisation algorithms above)

We will check the code for implementation and ease of use (e.g., how easy it is to add a new optimisation algorithm such as Eve). Note that the code should be flexible enough to work with different batch sizes.

Question 4 (10 Marks)

Use the sweep functionality provided by wandb to find the best values for the hyperparameters listed below. Use the standard train/test split of fashion_mnist (use `(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()`). Keep 10% of the training data aside as validation data for this hyperparameter search. Here are some suggestions for different values to try for hyperparameters. As you can quickly see that this leads to an exponential number of combinations. You will

have to think about strategies to do this hyperparameter search efficiently. Check out the options provided by wandb.sweep and write down what strategy you chose and why.

- number of epochs: 5, 10
- number of hidden layers: 3, 4, 5
- size of every hidden layer: 32, 64, 128
- weight decay (L2 regularisation): 0, 0.0005, 0.5
- learning rate: 1e-3, 1 e-4
- optimizer: sgd, momentum, nesterov, rmsprop, adam, nadam
- batch size: 16, 32, 64
- weight initialisation: random, Xavier
- activation functions: sigmoid, tanh, ReLU

wandb will automatically generate the following plots. Paste these plots below using the "Add Panel to Report" feature. Make sure you use meaningful names for each sweep (e.g. hl_3_bs_16_ac_tanh to indicate that there were 3 hidden layers, batch size was 16 and activation function was ReLU) instead of using the default names (whole-sweep, kind-sweep) given by wandb.

To address the exponential number of hyperparameter combinations, Bayesian optimization was chosen as the sweep method in WandB. Unlike grid search, which exhaustively tests all possible combinations, or random search, which selects parameters randomly, Bayesian optimization intelligently selects the next set of hyperparameters based on previous results.

Question 5 (5 marks)

We would like to see the best accuracy on the validation set across all the models that you train.

wandb automatically generates this plot which summarises the test accuracy of all the models that you tested. Please paste this plot below using the "Add Panel to Report" feature

Question 6 (20 Marks)

Based on the different experiments that you have run we want you to make some inferences about which configurations worked and which did not.

Here again, wandb automatically generates a "Parallel co-ordinates plot" and a "correlation summary" as shown below. Learn about a "Parallel co-ordinates plot" and how to read it.

By looking at the plots that you get, write down some interesting observations (simple bullet points but should be insightful). You can also refer to the plot in Question 5 while writing these insights. For example, in the above sample plot there are many configurations which give less than 65% accuracy. I would like to zoom into those and see what is happening.

I would also like to see a recommendation for what configuration to use to get close to 95% accuracy.

- Many configurations result in less than 65% validation accuracy, particularly when using very low learning rates (e.g., 0.0001) or a high number of hidden layers (e.g., 5) or a low value for hidden size (eg. 32).

- The performance of models using random initialization and Xavier initialization is quite similar.
- Almost all configurations that achieved high validation accuracy had a weight_decay value of 0. This suggests that L2 regularization (weight decay) might not be beneficial for this specific dataset and model architecture.
- Configurations with epoch value of 10 consistently resulted in higher validation accuracies compared to lower epoch values. This suggests that the model benefits from extended training, allowing it to converge to better weights.
- Most of the best-performing configurations used either Tanh or ReLU as the activation function, Activation functions like Sigmoid were generally associated with lower validation accuracy, likely due to saturation effects and gradient issues.
- Optimizers like NADAM, Adam, and RMSprop consistently resulted in high validation accuracy. These optimizers adaptively adjust learning rates, making them more effective in converging to optimal solutions.

Based on the observations from hyperparameter sweeps and key insights, a recommended configuration for reaching ~95% accuracy on MNIST would be:

Hidden Layers: 3

Hidden Size: 64 (Increasing the hidden size can help learn more complex features)

Optimizer: Adam or NADAM

Learning Rate: 0.01

Weight Decay: 0

Activation Function: Tanh or ReLU

Batch_size : 32

Epochs: 10 (Increasing epochs can help increasing the accuracy further)

Question 7 (10 Marks)

For the best model identified above, report the accuracy on the test set of fashion_mnist and plot the confusion matrix as shown below. More marks for creativity (less marks for producing the plot shown below as it is)

Test Accuracy for the best model : 87.6%

Question 8 (5 Marks)

In all the models above you would have used cross entropy loss. Now compare the cross entropy loss with the squared error loss. I would again like to see some automatically generated plots or your own plots to convince me whether one is better than the other.

1. **Best Model Performance**

Cross-Entropy Loss:

-Train Accuracy: 90.83%

-Validation Accuracy: 88.7%

-Best model had 3 hidden layers of size 64

-Some other sweeps even reached around 90% validation accuracy

Mean Squared Error (MSE) Loss:

-Train Accuracy: 89.56%

-Validation Accuracy: 88.06%

-Best model had 5 hidden layers of size 64

2. Difference in Best Accuracies

-The best validation accuracy achieved with cross-entropy loss was slightly higher than with MSE loss.

-This confirms that cross-entropy loss is generally more effective for classification tasks.

3. Distribution of Validation Accuracies

From the val_accuracy vs. created plot, fewer hyperparameter configurations resulted in validation accuracy below 0.6 for cross-entropy loss compared to MSE loss.

This suggests that cross-entropy is more robust across different hyperparameter choices.

4. Performance of Initial Hyperparameters

The initial few hyperparameter configurations for MSE loss resulted in very low accuracies.

In contrast, for cross-entropy loss, the initial hyperparameter configurations consistently achieved around 80% accuracy while running sweeps in WandB using Bayesian optimization.

This indicates that cross-entropy loss provides better stability and faster convergence in hyperparameter searches.

Question 9 (10 Marks)

Paste a link to your github code for this assignment

https://github.com/DA24M010/da6401_Assignment1

- We will check for coding style, clarity in using functions and a `README` file with clear instructions on training and evaluating the model (the 10 marks will be based on this)
- We will also run a plagiarism check to ensure that the code is not copied (0 marks in the assignment if we find that the code is plagiarized)
- We will also check if the training and test data has been split properly and randomly. You will get 0 marks on the assignment if we find any cheating (e.g., adding test data to training data) to get higher accuracy

Question 10 (10 Marks)

Based on your learnings above, give me 3 recommendations for what would work for the MNIST dataset (not Fashion-MNIST). Just to be clear, I am asking you to take your learnings based on extensive experimentation with one dataset and see if these learnings help on another dataset. If I give you a budget of running only 3 hyperparameter configurations as opposed to the large number of experiments you have run above then which 3 would you use and why. Report the accuracies that you obtain using these 3 configurations.

Answer:

Based on the learnings from the Fashion-MNIST experiments, I would choose the following three hyperparameter configurations for MNIST. These choices are based on patterns observed in the best-performing models from Fashion-MNIST, ensuring that we maximize accuracy while minimizing the number of experiments needed. Also according to WandB's "Parameter Importance with Respect to Validation Accuracy" metric, these three hyperparameters were among the most important.

1. num_hidden = 3

- In Fashion-MNIST, models with 3 hidden layers consistently performed well, balancing generalization and complexity.
- MNIST is an easier dataset, and adding more layers (e.g., 5) is unlikely to significantly improve accuracy.
- Average val accuracy was 81.30% for num_hidden = 3 and 72.36% for num_hidden = 5

2. optmizer = nadam or adam

- Adam and NADAM were among the top-performing optimizers in Fashion-MNIST.
- NADAM (Nesterov-accelerated Adam) may provide slight improvements in convergence due to better momentum updates.

- Also using nadam and adam optimizers helps to adjust the learning rate.
- Average val accuracy were around 82.64% for Adam and NADAM vs. 76% for RMSprop and 10% for nesterov and momentum.

3. hidden_size = 64

- The best-performing models in Fashion-MNIST had hidden layer sizes of 64, making it a strong candidate for MNIST as well.
- A smaller size (e.g., 32) might underfit, while a larger size (e.g., 128) may lead to unnecessary complexity.
- Average val accuracy for hidden_size = 64 was 81.77% and for hidden_size = 32 it was 42.8%.

Max val accuracy using these three hyperparameters = 88.7%

Min val accuracy using these three hyperparameters = 74.58%

Average val accuracy using these three hyperparameters = 84%

- These accruacies were obtained by grouping the runs for each hyperparameter using wandb group by function and filters in sweeps dashboard.

Code Specifications

Please ensure you add all the code used to run your experiments in the GitHub repository.

You must also provide a python script called `train.py` in the root directory of your GitHub repository that accepts the following command line arguments with the specified values -

We will check your code for implementation and ease of use. We will also verify your code works by running the following command and checking wandb logs generated -

```
python train.py --wandb_entity myname --wandb_project myprojectr
```

Arguments to be supported

Name	Default Value	Description
`-wp`, `--wandb_project`	myprojectname	Project name used to track experiments in Weights & Biases dashboard
`-we`, `--wandb_entity`	myname	Wandb Entity used to track experiments in the Weights & Biases dashboard.
`-d`, `--dataset`	fashion_mnist	choices: ["mnist", "fashion_mnist"]
`-e`, `--epochs`	1	Number of epochs to train neural network.
`-b`, `--batch_size`	4	Batch size used to train neural network.
`-l`, `--loss`	cross_entropy	choices: ["mean_squared_error", "cross_entropy"]
`-o`, `--optimizer`	sgd	choices: ["sgd", "momentum", "nag", "rmsprop", "adam", "nadam"]
`-lr`, `--learning_rate`	0.1	Learning rate used to optimize model parameters
`-m`, `--momentum`	0.5	Momentum used by momentum and nag optimizers.
`-beta`, `--beta`	0.5	Beta used by rmsprop optimizer
`-beta1`, `--beta1`	0.5	Beta1 used by adam and nadam optimizers.
`-beta2`, `--beta2`	0.5	Beta2 used by adam and nadam optimizers.
`-eps`, `--epsilon`	0.000001	Epsilon used by optimizers.

Name	Default Value	Description
`-w_d`, `--weight_decay`	.0	Weight decay used by optimizers.
`-w_i`, `--weight_init`	random	choices: ["random", "Xavier"]
`-nhl`, `--num_layers`	1	Number of hidden layers used in feedforward neural network.
`-sz`, `--hidden_size`	4	Number of hidden neurons in a feedforward layer.
`-a`, `--activation`	sigmoid	choices: ["identity", "sigmoid", "tanh", "ReLU"]

Please set the default hyperparameters to the values that give you your best validation accuracy. (Hint: Refer to the Wandb sweeps conducted.)

You may also add additional arguments with appropriate default values.

Self Declaration

I, Mohit Singh, swear on my honour that I have written the code and the report by myself and have not copied it from the internet or other students.