

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.

Ujjwal Singh cs23m071

Created on March 10 | 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.

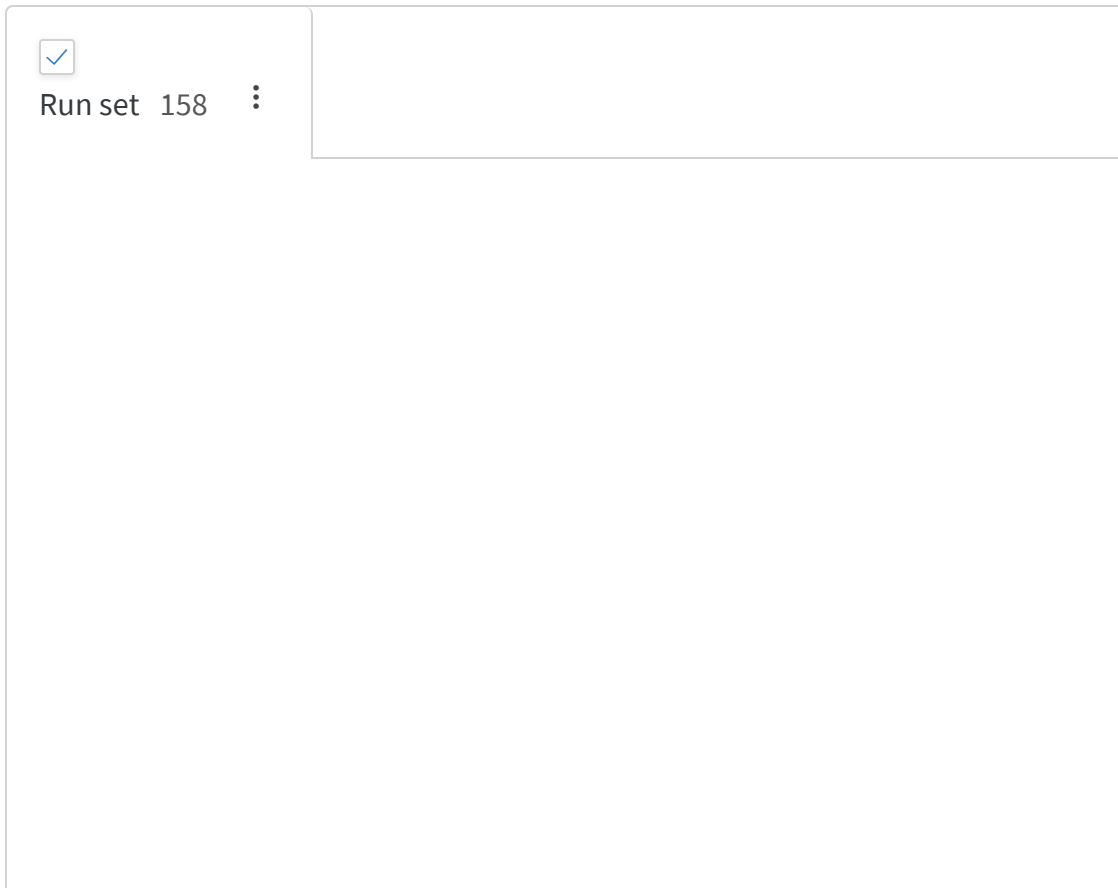
▼ Problem Statement

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 ($28 \times 28 = 784$ 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.

▼ Answer:

I've implemented a feedforward neural network module along with an optimizer. You can find the GitHub repository here:

<https://github.com/ujjwalsingh12/DA6401>

In this implementation, the forwardPropagation method takes an array of layer class objects and an input image as input. It produces a probability distribution over the classes and stores it within the same array of objects. This makes it easy to print and access these values anytime, which are then used in the backpropagation module as well as for loss calculations.

▼ 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.

Answer

I've implemented backpropagation along with all six optimization functions within the Optimizer class. You can find the GitHub repository here:

<https://github.com/ujjwalsingh12/DA6401/blob/main/question2.py>

In the backpropagation framework, I pass an array of layer class objects as an argument. These objects are precomputed in the train.py file based on user input or default values. Additionally, the function takes an input, an output, and a weight decay coefficient.

The backpropagation algorithm calculates the gradient of the weight and bias terms based on the loss function and stores these values within the layer_objects array itself. The updated array is then returned to the caller function, which uses it within the optimization algorithm accordingly.

▼ 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.

▼ Answer

wandb.sweep() offers three main methods for hyperparameter search:

Bayesian Search: This method defines a performance measure and its optimization objective (maximization or minimization). As different hyperparameter configurations are evaluated, the process builds on past results, refining selections iteratively. This approach helps optimize training performance efficiently, leveraging previous insights to guide future choices.

Random Search: Unlike Bayesian search, random search does not rely on prior knowledge. Instead, it samples hyperparameter values from a predefined distribution. While this allows for an unbiased exploration of the hyperparameter space, it lacks the efficiency of Bayesian optimization.

Grid Search: This method constructs a grid of hyperparameter values and evaluates every possible combination. However, due

to its exponential computational cost (m^n combinations for n hyperparameters and m values each), it becomes impractical for large-scale searches.

Given our requirements, Bayesian Search is the most suitable choice. It efficiently navigates the hyperparameter space by leveraging past evaluations, making it an optimal approach for improving model performance over successive iterations.

Below is the configuration I used:

```
sweep_config = {
    "method": "bayes",
}
metric = {
    "name": "val_accuracy",
    "goal": "maximize"
}
sweep_config['metric'] = metric
parameter_dict = {
    'number_of_epochs': {
        'values': [5, 10, 15]
    },
    'number_of_hidden_layers': {
        'values': [3, 4, 5]
    },
    'size_of_every_hidden_layer': {
        'values': [32, 64, 128]
    },
    'weight_decay': {
        'values': [0, 0.0005]
    },
    'learning_rate': {
        'values': [0.1, 0.01, 0.001]
    },
    'optimizer': {
        'values': ['stochastic', 'momentum', 'nesterov accelerated']
    },
    'batch_size': {
        'values': [16, 32, 64]
    },
}
```

```
'weight_initialisation': {  
    'values': ['random', 'xavier']  
},  
'activation_functions': {  
    'values': ['sigmoid', 'tanh', 'relu']  
}  
}
```



Sweep: zwzt6eow 1 158 ⋮



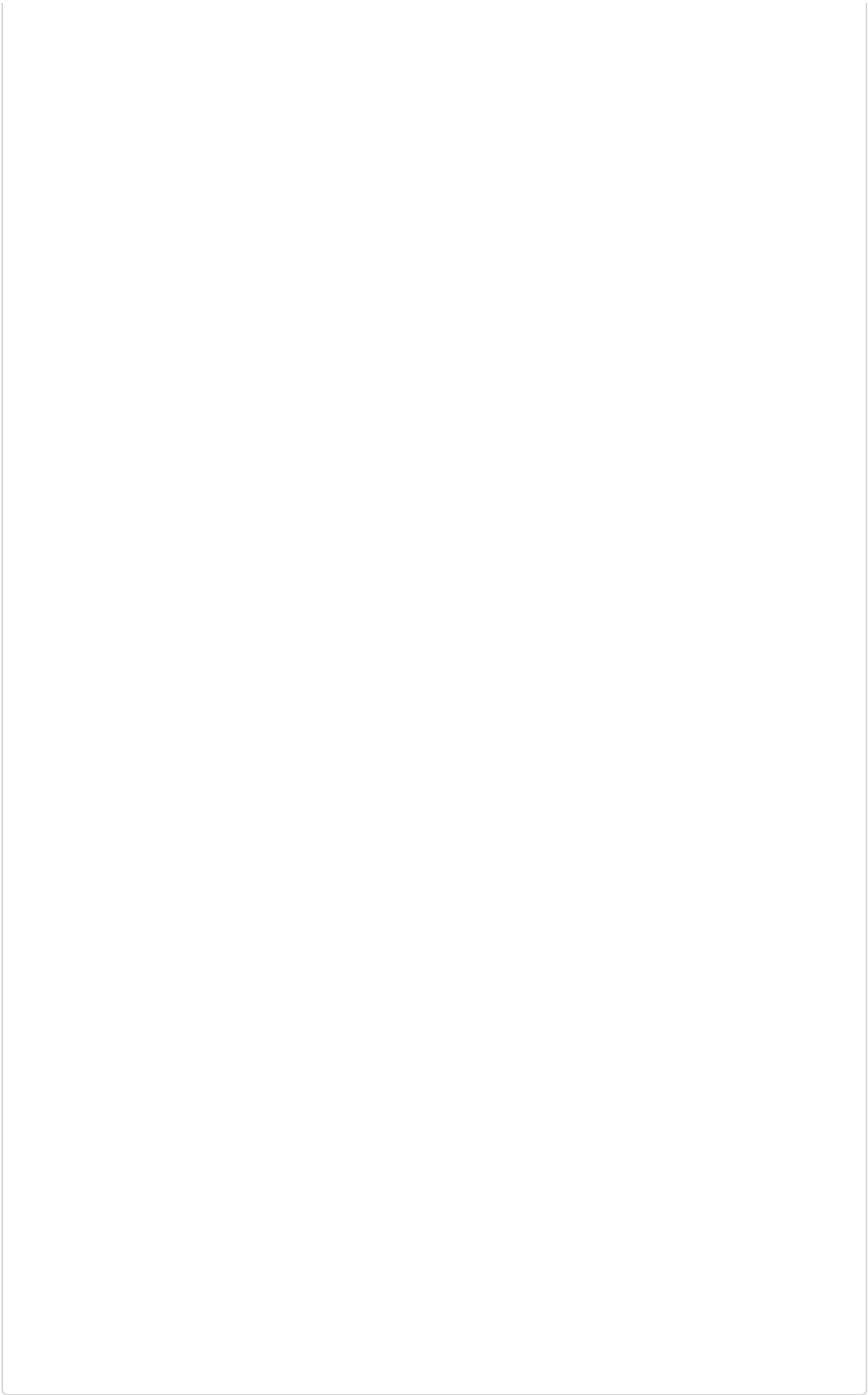
▼ Question 5 (5 marks)

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



Sweep: zwzt6eow 1 158 ⋮



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.

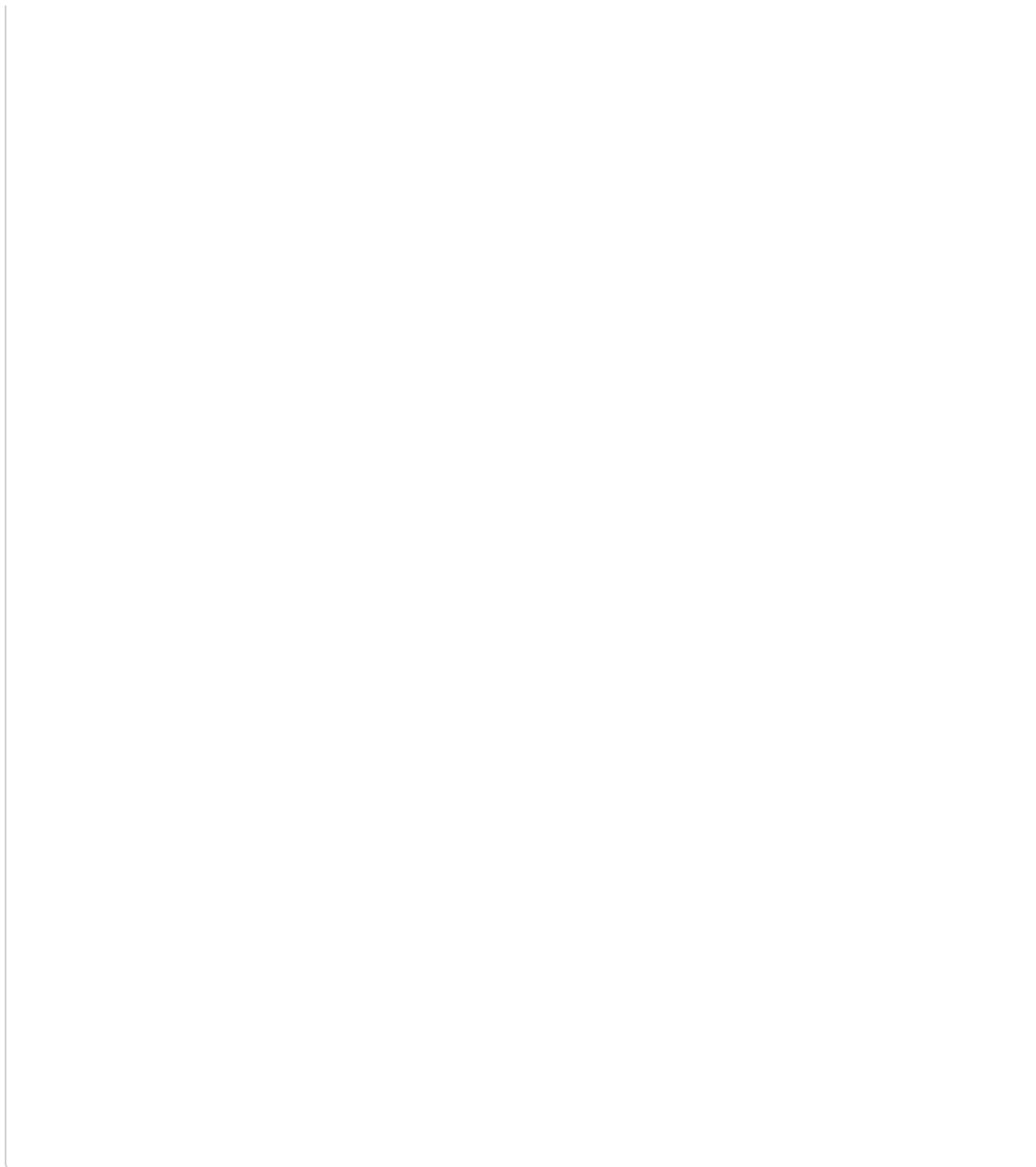
Here again, wandb automatically generates a "Parallel co-ordinates plot" and a "correlation summary" as shown below. Learn about a "Parallel co-ordinates plot" and how to read it.

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.



Sweep: zwzt6eow 1 132 ⋮



▼ 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)



Sweep: 7c0nik6g 1 158 ⋮

▼ 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.



Sweep: 7c0nik6g 1 141 ⋮

▼ Question 9 (10 Marks)

Paste a link to your github code for this assignment

Example: https://github.com/<user-id>/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

Github Link : <https://github.com/ujjwalsingh12/DA6401>

▼ 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.

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
<code>-wp, --wandb_project</code>	myprojectname	Project name used to track experiments in Weights & Biases dashboard
<code>-we, --wandb_entity</code>	myname	Wandb Entity used to track experiments in the Weights & Biases dashboard.
<code>-d, --dataset</code>	fashion_mnist	choices: ["mnist", "fashion_mnist"]

Name	Default Value	Description
<code>-e, --epochs</code>	1	Number of epochs to train neural network.
<code>-b, --batch_size</code>	4	Batch size used to train neural network.
<code>-l, --loss</code>	cross_entropy	choices: ["mean_squared_error", "cross_entropy"]
<code>-o, --optimizer</code>	sgd	choices: ["sgd", "momentum", "nag", "rmsprop", "adam", "nadam"]
<code>-lr, --learning_rate</code>	0.1	Learning rate used to optimize model parameters
<code>-m, --momentum</code>	0.5	Momentum used by momentum and nag optimizers.
<code>-beta, --beta</code>	0.5	Beta used by rmsprop optimizer
<code>-beta1, --beta1</code>	0.5	Beta1 used by adam and nadam optimizers.
<code>-beta2, --beta2</code>	0.5	Beta2 used by adam and nadam optimizers.
<code>-eps, --epsilon</code>	0.000001	Epsilon used by optimizers.
<code>-w_d, --weight_decay</code>	.0	Weight decay used by optimizers.
<code>-w_i, --weight_init</code>	random	choices: ["random", "Xavier"]
<code>-nhl, --num_layers</code>	1	Number of hidden layers used in feedforward neural network.
<code>-sz, --hidden_size</code>	4	Number of hidden neurons in a feedforward layer.
<code>-a, --activation</code>	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.



Sweep: 7c0nik6g 1 158 ⋮

Hyperparameter Recommendations

Best Performing Configuration (Highest Accuracy)

- **Optimizer:** Nadam
 - **Learning Rate:** 0.001
 - **Hidden Layers:** 3
 - **Hidden Layer Size:** 128
 - **Epochs:** 15
 - **Batch Size:** 50
 - **Reasoning:** This setup achieved the highest validation accuracy (88.9%), making it a strong candidate for final deployment or fine-tuning.
-

Alternative with Nearly Identical Performance

- **Optimizer:** Nadam
 - **Learning Rate:** 0.001
 - **Hidden Layers:** 3
 - **Hidden Layer Size:** 128
 - **Epochs:** 15
 - **Batch Size:** 50
 - **Reasoning:** This setup is almost identical to the best-performing one, achieving 88.85% accuracy. It can serve as a backup verification to ensure the top configuration is consistent and not an outlier.
-

Regularization-Focused Configuration

- **Optimizer:** NAGD (Nesterov Accelerated Gradient Descent)
 - **Learning Rate:** 0.0001
 - **Hidden Layers:** 3
 - **Hidden Layer Size:** 128
 - **Epochs:** 10
 - **Batch Size:** 64
 - **Reasoning:** This configuration uses a lower learning rate and NAGD optimizer, which can improve stability and generalization while reducing overfitting. It also uses a larger batch size, which might help with training efficiency.
-

Final Recommendation

- The **first two configurations** (Nadam optimizer, 0.001 LR, batch size 50) should be used for **primary training**.
- The **third configuration** (NAGD optimizer, 0.0001 LR, batch size 64) provides a **generalization-focused alternative** to ensure robustness.



Self Declaration

I, Ujjwal 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.

Created with  on Weights & Biases.

<https://wandb.ai/cs23m071-indian-institute-of-technology-madras/Deep%20Learning%20Course%20DA6401/reports/DA6401-Assignment-1--VmIldzoxMTcyMzE3MA>