GR5241 Spring 2022

Project Milestone 2: Deep Learning Part

Name: Kangshuo Li UNI: kl3259

Code Instructions

Since question 3 / 4 / 5 / 7 follows the same structure, so the coding part can be explained in a same way.

We first import all the packages needed and activate the connection with google drive.

Load the data by torchvision.datasets.MNIST() and preprocess the data by converting to numpy array and converting back.

Move models and data to CUDA GPU.

In question 3 / 4 / 5 / 7, first define neural network structures by create a subclass of torch.nn()

Question 3 / 4 / 5 / 7 (a)

> !!!Note that in the first half of the jupyter notebook, there are some parts not using functions. These parts are just unpackaged parts but strictly follow the main methodology I packaged into function in the later part. !!!

Two main functions are used in (a) and (b):

- train_with_seeds() Input: model_type, random_seed_list, learning_rate, epochs, suffix = ""

  Output: global loss function values dataframes with globals()["different name here"], by using different vairable names

  Process: Automatic create loss function dataframes in the first loop of random seeds and the second loop of epoch by using train_loop() and test_loop(). Inside the first loop of random seeds, instantiate NN model by if-else structure and the model_type, instantiate the optimizer and loss function. Creating different dataframe variable names by combining suffix and str(seed).

  - train_loop()

    Input: dataloader_, model_, loss_fn_, optimizer_, epoch_, seed_(unused)

    Output: avg_cross_entropy_error, avg_mis_clf_error for training set

    Process: Training in each epoch. Load the training dataset from the dataloader with a for loop. In each for loop, compute the prediction(fp) and loss function value and count correct predictions, then calculate back propagation. After the batch iteration finished, compute the evaluation metrics, print result

  - test_loop()

Input: dataloader_, model_, loss_fn_, seed_(unused)

Output: avg_cross_entropy_error, avg_mis_clf_error for the test / validation set

Process: Testing in each epoch. Load the test / validation dataset from the dataloader with a for loop. In each for loop, compute the prediction(fp) with pretrained model. After the batch iteration finished, compute the evaluation metrics, print result

- plot_error()

  Input: fig_title, loss_df, suffix = "", random_seed_list, error_type, lay_out

  Output: learning curves for a single dataframe or multiple dataframe with one specific error_type assigned

  Process: lay_out choose subplot structure, (1,1) for "single", (2,3) for "multi_seeds". Make configurations on matplotlib. "single" uses loss_df to plot, "multi_seeds" uses the global dataframes by their names with suffix and random seeds strings via *globals()[names here]*. Assign title by fig_title. Save the figure.

Use error_type = "Cross_entropy" in this question.

Suffixes can be used: "lenet_", "cnn_", "single_layer_", "alexnet_"

Question 3 / 4 / 5 / 7 (b)

Training part is already finished in (a)

Plot part is same as (a), only change the error_type = "Misclassification".

Question 3 / 4 / 5 / 7 (c)

- model_comparison()

  Input: random_seed_list, suffix = ""

  Output: dataframe with seeds in index and Cross_entropy_error_train/Misclassification_error_train/Cross_entropy_error_test/Misclassification_error_test/Accuracy_test in columns

  Process: read in the original loss function values dataframe by suffix. Append the last row of loss function values dataframe w.r.t different seeds to this df. Calculate accuracy from Misclassification_error_test.

Find the best model from the model_comparison() output.

Load the best model with torch.load()

Then plot the weights or kernels.

- plot_param() / plot_param_lenet() / plot_param_cnn() / plot_param_alexnet()

  Input: fig_name, params, suffix = "", grid_height, grid_width

  Output: Assign title by fig_name.

  Process: if-else structure identify the dimension of params, if the kernel dimension got input channel and output channel, if only one channel then sampling on that dimension to get 6 kernels to display. Then sample the grid_height and grid_width number of kernels to display. For weight in single layer NN, use all the weights. Visualize by subplots. Suffix for assigning the figure file name.

Question 3 / 4 / 5 / 7 (d)

- train_with_lr_mmt() Input: best_random_seed, learning_rate_list, momentum_list, epochs, suffix

  Output: loss function dataframe for grid search

  Process: Add learning rate for loop and momentum for loop to make grid search, remaining part is same as train_with_seed but with a single best random seed.

  - train_loop() same as in (a)
  - test_loop() same as in (a)

- plot_error_alt() Input: fig_title, lr_list, mmt_list, suffix, loss_df, error_type, lay_out

  Output: subplots for learning curves in train_with_lr_mmt()

  Process: Add for loop of learning rate list and momentum list to plot the learning curve on grid. Remaining part is the same as plot_error(). Also can choose error_type and lay_out(1 df or multi_seed)

- model_comparison_alt() Input: learning_rate_list, momentum_list, suffix = ""

  Output: evaluation metrics dataframe for the learning process in train_with_lr_mmt() grid search

  Process: Works the same as model_comparison()

Question 6

matplotlib subplots + random sample rows in array + numpy.reshape()

> For detailed implementation, please read the inline comments.