Project 0: Familiarizing with PyTorch

CSE 849 Deep Learning (Spring 2025)

Gautam Sreekumar
Instructor: Zijun Cui

January 24, 2025

The purpose is to familiarize yourself with PyTorch, so that you can implement a basic PyTorch project from scratch. You can use this page as a reference throughout this assignment.

1 Setting Up The Environment

PyTorch is an open-source library of efficient tools for constructing and training neural network architectures. Since PyTorch and its related libraries are actively maintained, they have frequent upgrade releases, and in turn may have conflicts with other Python packages. Therefore, we will first setup an environment without any package clashes so that we can also avoid bugs due to minor changes between various versions (e.g., changes in the default argument).

Tasks:

1. [**Install Conda**] We will setup our Python environment using `conda`. Go to Miniconda installation page and install the latest version of Miniconda.
2. [**Create Environment**] Create a temporary environment named "temp" by following the instructions here.
3. [**Install Packages**] Install the following packages following the instructions here. Remember to activate the "temp" environment before installing the packages:
   - Python
   - Matplotlib
   - NumPy
   - PyTorch
4. [**GPU**] If you want to use GPU (assuming your system has GPUs), make sure that you are installing a GPU-compatible version of PyTorch. To verify if your installed PyTorch supported GPUs, open a Python console and run the following the code:

   ```
   import torch
   print(torch.cuda.is_available()) # "True" means GPU-compatible
   ```

   You can also check using `conda list` command in a terminal and see the version description next to `torch` package.
5. [**.yml File**] You can also create conda environments using .yml files. For each assignment, we will provide a .yml file to include the required conda environment. Find the .yml file in the current assignment folder and install the environment following the instructions here. Remember to deactivate your "temp" environment and activate the new environment created using the .yml file. We included the CPU-only `torch` package in the shared conda environment since not everyone may have access to GPUs. You can verify using `conda list`.

2 Create Your Pipeline (30 points)

Now you will create your pipeline step by step, from generating a synthetic dataset to training a model on this dataset. Along the way, we will cover some useful practices.

Tasks:

1. **[Randomness]** Random number generators play a very important role in training neural networks, influencing aspects such as initial parameter values, data sampling order, and noise generation. By setting random seeds at the beginning, we can control the randomness, allowing us to evaluate our model under consistent conditions. Set the random seed to the last five digits of your student ID.

2. **[Define Dataset Class]** We will construct a custom map-style `Dataset` that will enumerate over the data samples. Create a Python class inherited from `Dataset` class. Your class must take as arguments (1) the number of samples n, (2) a variable named "split" indicating whether the data is for training, validation, or testing, and (3) pre-defined values, α, β for data generation. A `Dataset` class requires

   - a `__len__` method to return the total number of available samples, which should be n
   - a `__getitem__` method that maps an index $0 \leq i < n$ to the corresponding data sample. It should also apply transformations needed on a per-sample basis.

   Check this example for reference. In `__getitem__`, return the appropriate pair of x and y based on the input index.

3. **[Data Generation]** Generate the data as follows:

   (a) Set a random seed inside the data class according to the value of the "split" variable. This ensures consistency in the dataset across multiple evaluations.

   (b) Generate a random vector x of size n using `torch.randn`. Generate it on the CPU, not the GPU.

   (c) Generate a random noise vector e in the same way as x.

   (d) Calculate y using the formula $y = \alpha x + \beta + e$.

4. **[Creating A Dataloader]** An instance of the `DataLoader` class efficiently loads data samples from a given `Dataset` using parallel threads that run in the background. Create an instance of a `DataLoader` following the instructions here. Use a sufficiently large batch size and set the number of workers appropriately to ensure fast sample loading.

5. **[Create A Model]** PyTorch uses `nn.Module` as the base class for all model components, including loss functions. This page lists various components that PyTorch supports natively.

   (a) Since the original data generation process is a linear function, construct a linear layer using `nn.Linear`.

   (b) To improve reproducibility, explicitly initialize the model. Set the random seed to the last of your student ID. Note that five digits in Section 2.1. is the seed for the overall process which would affect the data generation process. We want to keep the data same and rerun the experiments for various model initializations. The three-digit seeds here are for model initialization. Use `nn.init.xavier_normal_` to initialize the parameters of your model following the instructions here.

   (c) If you have access to a GPU, you can move your model to the GPU using `to` method.

6. **[Setup Loss Function and Optimizer]** `torch.nn` also includes loss functions that can be called after every forward step during training. Since we are handling a regression task, use `nn.MSELoss`. Refer to this page for a list of natively supported optimizers. Construct an optimizer using one of the listed options. We recommend `SGD`, `RMSProp`, or `Adam` due to their popularity and effectiveness.

7. [**Training Loop**] Congratulations! You have all the components ready to train the model. Now, you need to (1) iterate through batches of your data obtained from your dataloader, (2) pass the input through the model (move the data to the GPU if your model is already on the GPU), (3) calculate the loss between the prediction and the true output, (4) run backpropagate using `backward` method of the loss function output, and (5) update your model using `step` method of your optimizer (remember to call `zero_grad` method before passing the input through the model). You can use the example in this page for reference. Additional references: 1, 2.

8. [**Evaluating Your Model On Validation Set**] Prepare validation set following steps 2 - 4, keeping the same α and β. During validation, set the model to `eval` mode and perform only the forward pass within a `torch.no_grad()` block. Collect the training loss and validation loss in separate lists and visualize them using Matplotlib's plot function.

To submit:

1. Completed Python files. **Note:** do not rename the Python files as they will be run using an automated script.

2. Saved checkpoint with the model's state dictionary. Name the file `q2_model.pt`. Refer here on how to save the model.

3. `q2_plot.png` contains training and validation loss curves along with the visualization of the fitted model. The plot is automatically generated by your starter code.

Grading:

1. 25 points for submission completeness and functionality: Python files (10 pts), saved model (5 pts), and plots (10 pts).

2. 5 points for the accuracy of your estimations. Since we know the true values of α and β, you will be evaluated on how close your model estimates are to the true values.

3 Apply Your Pipeline On A Given Dataset (20 points)

In most cases, data comes from external sources and you may not know the nature of the data distribution in advance. To illustrate this, we have provided you a `data.pt` file along with the homework. Load this file using `torch.load` to a Python dictionary. It will contain six keys: `x_train`, `y_train`, `x_val`, `y_val`, `x_test`, and `y_test`.

Tasks:

1. Create train, validation and test datasets from the provided data. Create corresponding dataloaders for each dataset.

2. Create an MLP with two hidden layers and initialize the parameters using `normal_()` method after setting the seed. The MLP should have three linear layers mapping from the 1D input space to 10D space to another 10D space and then finally to the 1D output space. If x is the input scalar, then the output is $\hat{y} = W_3(W_2(W_1x + b_1) + b_2) + b_3$ where $W_3 \in \mathbb{R}^{1\times10}$, $W_2 \in \mathbb{R}^{10\times10}$, $W_1 \in \mathbb{R}^{10\times1}$, $b_1, b_2 \in \mathbb{R}^{10}$ and $b_3 \in \mathbb{R}$.

3. Setup a mean squared error loss function and an AdamW optimizer. The default hyperparameters for the optimizer are provided in the starter file.

4. Train your model using the training dataset with seeds 1 to 5 (both inclusive) Visualize the training samples (x and y) in a scatter plot. On the same plot, visualize the output from your model using a different color for each seed. Sort the predicted data pairs $(x, \hat{y})$ in ascending order of x and plot it as as a continuous as shown here. Codes for this part need to be implemented by yourself. Name your plot `q3_plot.png`.

5. Tune hyperparameters (e.g., batch size, learning rate) based on the model's performance on the validation dataset. At the end, choose the hyperparameter settings that yield the best validation accuracy. You are provided with default values of hyperparameters.

To submit:

1. Completed Python files. **Note:** do not rename the Python files as the files will be run using an automated script.
2. Choose the model that achieved the lowest validation error as your *best* model. Run your best model on the test dataset. Submit the predicted $\hat{y}$ values as a text file `q3_test_output.txt` (code for this part is provided).
3. Save the model state dictionary using `torch.save` and submit the checkpoint file. Name the file `q3_model.pt`. Make sure to match the architecture in the question. Otherwise, it will not be evaluated and marks will be deducted.
4. Submit the plot `q3_plot.png` showing the model's prediction on the training dataset with various seeds. The curves for all seeds must be on the same plot.

Grading:

1. 15 points for submission completeness and functionality. Python files (5 pts), saved model (5 pts), and plots and results (5 pts). Your output is automatically graded using a Python evaluation script. Therefore, make sure to follow the details in the question accurately.
2. 5 points for the accuracy of your estimations. We have a testing error based on our implementation. You will be scored based on how close to/below your prediction error is to ours.

4 Submission Format

Submit a folder named `first name_last name_project_0` using the same subfolders as below:

1. code
   (a) data
   (b) q2
   (c) q3
2. results
   (a) q2_model.pt
   (b) q2_plot.png
   (c) q3_model.pt
   (d) q3_plot.png
   (e) q3_test_output.txt