DeepRLClass

Lydia And John

August 2025

1 A brief tutorial of Linux Commands

- `pwd` Prints the current directory's path
- `cd PATH` Changes the current directory to the path in the argument
- `ls` Displays the files and directories in the current directory
- `mkdir PATH` Creates an empty directory at the path in the argument
- `rm FILE` Permanently deletes the file in the argument
- `rmdir DIRECTORY` Permanently deletes the directory in the argument
- `*` This is the wildcard symbol, for example `name*` would mean all files beginning with the text `name`. Be very careful combining `*` with `rm`, as you may permanently delete more than you intended!

`ssh <username>@<machine_name>` is a command in Linux you may use to connect your terminal to another machine's terminal remotely. For example, if my username on the remote machine were `johnbaxter` and I was trying to connect to the remote machine `moons.cs.unm.edu`, I could use the following command to connect to the remote machine:

```
ssh johnbaxter@moons.cs.unm.edu
```

You may need to render windows generated on the remote machine, in which case you need to use the options `-XY` to turn on window forwarding (for example if you are rendering plots with pyplot):

```
ssh -XY <username>@<machine_name>
```

While ssh-ing into a remote machine, if you get disconnected your programs will halt. To prevent this, use `screen` to create a detachable context that persists after disconnecting. When you are in the context of `screen`, you may detach from that context by typing `ctrl-a` and then `d`. To reattach a detached context, enter on the command prompt: `screen -r`

2 A brief tutorial of using Git

- `git clone URL` where URL is found on a repository's page on GitHub will create a fresh copy of that repository in the current directory
- `git pull` will pull updates from the remote repository into yours
- `git add FILE` will add a file to the current list of changed files to be committed
- `git commit` will bring up an editor which will let you write a note on the changes you made to the files you are committing
- `git push` will push your committed changes from your local repository to the remote repository

3 Installing needed Python libraries in a virtual environment (CPU Version)

- Make sure you are using a Python version between 3.9 and 3.12 on a Linux machine. To check the python version, type: `python3 --version`
- In the terminal, in the directory where you want the virtual environment installed, type: `python3 -m venv venv_cpu`
- Activate the virtual environment: `source venv_cpu/bin/activate`
- If that succeeded, '`(venv_cpu)`' should be at the beginning of the terminal command prompt. All of the following instructions assume your virtual environment is active.
- Update pip, which we use to install libraries in your virtual environment: `python3 -m pip install --upgrade pip`
- Install setuptools and wheel, which are needed to build a few libraries: `python3 -m pip install --upgrade setuptools wheel`
- Install swig: `python3 -m pip install --upgrade swig`
- Install numpy (a library for array math), networkx (a graph library), and the CPU version of pytorch (for neural networks); the following is one line: `python3 -m pip install --upgrade numpy networkx torch --index-url https://download.pytorch.org/whl/cpu`
- Install gymnasium (which implements many RL environments), below is again one line on the command prompt:
  `python3 -m pip install --upgrade gymnasium[atari,box2d,classic-control,other]`
- Install matplotlib (a library for plotting data):
  `python3 -m pip install --upgrade matplotlib`

4 Installing needed Python libraries in a virtual environment (GPU Version)

- Make sure you are using a Python version between 3.9 and 3.12 on a Linux machine. Make sure your GPU supports CUDA. Note the version. Note that only some CS department machines have suitable GPUs.
- In the terminal, in the directory where you want the virtual environment installed, type: `python3 -m venv venv_gpu`
- Activate the virtual environment: `source venv_gpu/bin/activate`
- If that succeeded, '`(venv_gpu)`' should be at the beginning of the terminal command prompt. All of the following instructions assume your virtual environment is active.
- Update pip: `python3 -m pip install --upgrade pip`
- Install setuptools and wheel (these are needed for swig):
  `python3 -m pip install --upgrade setuptools wheel`
- Install swig (needed for box2d):
  `python3 -m pip install --upgrade swig`
- Install numpy and networkx:
  `python3 -m pip install --upgrade numpy networkx`
- Install the right version of torch for your version of CUDA. Check both `https://pytorch.org/` and, if your GPU doesn't support the latest torch, use `https://pytorch.org/get-started/previous-versions/` and find the right version. For example, on my desktop in my office the latest CUDA version supported by my GPU is 12.4, so I used the following (this is all one line): `python3 -m pip install torch==2.6.0`
  `--index-url https://download.pytorch.org/whl/cu124`
- Install gymnasium (which implements many RL environments), below is again one line on the command prompt:

  ```
  python3 -m pip install --upgrade
  gymnasium[atari,box2d,classic-control,other]
  ```
- Install matplotlib (a library for plotting data):
  `python3 -m pip install --upgrade matplotlib`
- Test that CUDA is working in torch. Run the following script; if there are no errors or warnings, you are probably good to go:

  ```
  import torch
  torch.zeros((3,3), dtype=torch.float32, device="cuda")
  ```

Assignment 1: Neural Networks (DUE: 2025-Sept-29)

In this assignment, you will be exploring factors that influence neural network performance and training time of neural networks on a supervised learning task. Supervised learning differs from reinforcement learning in that a set of correct answers are known in advance (as opposed to reinforcement learning, where the correct behavior is not necessarily known, just a reward function).

Learning Objectives

By the end of this assignment, students will have:

- Set up a reproducible machine learning environment,
- Implemented and train a fully-connected neural network in PyTorch,
- Swept network hyperparameters across, with seeded replicates,
- Interpreted loss curves associated with hyperparameter choices,
- Written a short empirical report summarizing methods and findings.

Data Provided: `swept_volume_data.npz`

You will be given a compressed numpy file called `swept_volume_data.npz` that stores several numpy arrays. (It will be posted on Canvas soon.) This file contains features (network inputs) and labels (expected network outputs) generated for a real world robotics problem. The problem involves estimating the volume of space passed through by a robot (composed of rigid bodies connected by rotational joints) moving between two configurations (lists of joint angle values). Predicting this volume (called a *swept volume*) is useful in planning robot motions in cluttered surroundings, as the probability of a motion resulting in a collision with something in the robot's environment is well estimated by the swept volume of that motion.

You may load `swept_volume_data.npz` into a dictionary and convert the numpy arrays to pytorch tensors with the following code:

```
import numpy as np
import torch

device = "cuda" if torch.cuda.is_available() else "cpu"
data = dict(np.load("swept_volume_data.npz"))
for key, value in data.items():
    print(f"{key}: {value.shape} shape, type {value.dtype}.")
    data[key] = torch.tensor(
            value,
            dtype=torch.float32,
            device=device)
```

Running the above code in the directory where `swept_volume_data.npz` is located will reveal the following data:

```
training_features: (100000, 14) shape, type float32.
training_labels: (100000, 1) shape, type float32.
evaluation_features: (10000, 14) shape, type float32.
evaluation_labels: (10000, 1) shape, type float32.
testing_features: (10000, 14) shape, type float32.
testing_labels: (10000, 1) shape, type float32.
```

Here, the first number of a shape is the number of rows of training (100,000), evaluation (10,000), or testing (10,000) data. The second number is the number of columns of floating point numbers for a given feature or label row. Here, features have 14 values, and labels have 1 value. The features here are two configurations. Features `[:,:7]` are the starting joint angles of the robot undergoing motion, while features `[:,7:]` are the ending joint angles. Feature angles are given in radians. The label value associated with a feature is the volume passed through by the seven-jointed robot moving its pose between the starting and ending joint angles. Label values are Liters of swept volume.

You will be tasked with putting together code to train a fully connected neural network minimizing mean square error of output predictions (computed by the network from input features) relative to the associated labels.

Coding a Neural Network

You will need code for a fully connected neural network. Be sure you have VSCode Copilot set up, and be in appropriate virtual environment (see section 3 for CPU or 4 for GPU).

In our testing, this prompt in Copilot with GPT-4o gave code for defining and training a fully connected neural network with randomly generated data:

```
Generate a fully connected neural network with a variable number
of hidden layers and perform supervised training. Have training,
evaluation, and testing data. Use pytorch.
```

We will go over in class in more detail how to get a good network and training code for our problem, and how to modify the code to use the features and labels from `swept_volume_data.npz`.

Training your Neural Network

Training data in supervised learning is used to adjust neural network parameters. Evaluation data tells us how well the network is doing during training; it is a separate set of data so as to be unbiased (it wouldn't be fair to evaluate with the same exact data you train on). Your final trained model will be the one that has the lowest evaluation data loss. Finally, we want to have a measure of performance of the final model selected that is unbiased with respect to the training and evaluation data, so there is a third testing dataset that will measure its performance independent of the data used to train and select.

The results of training a neural network will depend on the initialization of the network's parameters; therefore, seed the random number generator and re-initialize the network layers' parameters for each run.

Note that you may need to modify the code to record the evaluation loss and network model parameters each epoch, and load the model with the lowest evaluation loss for the final testing. Look up `numpy.argmax()`, `model.state_dict()`, and `model.load_state_dict()` for these purposes. Also make sure that the loss function being used is mean square error (`torch.nn.MSELoss`). To plot data, we recommend using `matplotlib.pyplot`. Make sure your network's layer weights and biases are initialized (the code from Copilot may not do this); you should use `torch.nn.init.xavier_uniform_()` for layer weights and `torch.nn.init.zeros_()` for biases.

Assignment Choice: CPU or GPU Version

Choose between the CPU task or the GPU task for the assignment, below; either way, you must document the resources and AI you use, comment your code, and share your code on GitHub Classroom.

CPU: Training Data Size, Learning Rate Analysis

Create a neural network with one hidden layer of 64 neurons, input dimension 14 and output dimension 1. Use a training batch size of 1000. For each of the combination of learning rates in `[1e-4,1e-3,1e-2,1e-1]` and training data sizes `[1000,10000,100000]`, plot the evaluation loss curves and produce a table of testing losses for the model states with the lowest evaluation loss for each. Also make a table of the mean and standard deviation of training time for your setups. Note that you should use the full size of data for evaluation and testing. Be sure to train multiple times (at least 5x per setup) to get standard deviation values. Seed the random generator using `torch.manual_seed()`. Train for at least 1000 epochs in each setup.

GPU: Network Architecture Analysis

Note that the code Copilot gives may not set the device of the neural network layers; if it does not, you will have to set that by adding `device="cuda"` to the end of the constructor calls of your layers.

Create networks with varying number of hidden layers `[1,2,3]` and neurons per hidden layer `[32,64,128,256]` with a learning rate of `1e-3` and the full training data set. Use a training batch size of 10000. For each setup, plot evaluation loss curves and produce a table of testing losses for the model states with the lowest evaluation loss for each. Also make a table of the mean and standard deviation of training time for your setups. Be sure to train multiple times (at least 5x per setup) to get standard deviation values. As in the CPU case, seed the random generator. Train for at least 1000 epochs in each setup.

Both: Self-Directed Investigation

Now that you've investigated what we've told you to look at, come up with a change you could make to the architecture or training that you haven't already done in the assignment, make a hypothesis of what the change will result in, and carry out the experiment. Was your hypothesis correct? Why do you think it did, or did not, help training?

Things you need to do before Turn In

- 491 Only: Form a group of two to work on the project together.
- 591 Only: You will be working individually on this projeet.
- Join the GitHub Classroom assignment with the following link: `https://classroom.github.com/a/akPcXbgY`
- Get VSCode, CoPilot, and a virtual environment (`venv_cpu` or `venv_gpu`) working on Linux (CS Department or your own).
  - Please *do not* commit your virtual environment folder to your GitHub Classroom repository (they are very large and system-dependent).

Rubric: What to Turn In: GitHub Classroom

- Push to your GitHub Classroom assignment repository base directory:
  - A file named `neural_network.py`, containing a class `NeuralNetwork`, implementing a fully connected neural network, which has:
    - An `__init__()` method, which has four integer parameters:
      - `in_dimension`, the feature size,
      - `out_dimension`, the label size,
      - `hidden_layers`, the number of hidden layers,
      - `neurons_per_hidden_layer`, the neurons per hidden layer.
    - `__init__()` should create linear layers, then initialize the layer weights using `torch.nn.init.xavier_uniform_`, and initialize the layer biases using `torch.nn.init.zeros_`.
    - A `assignment()` method, which returns string `"cpu"` or `"gpu"` depending on which assignment you are doing.
    - A `forward()` method, which takes a single parameter `x`, returning the result of `x` passed through the neural network layers, with `torch.nn.ReLU()` activation after every layer except the last.

  - In the same file `neural_network.py`, have a main method which performs supervised learning with training, validation, and testing sets with the data from `swept_volume_data.npz`.
    * CPU Assignment: Your code should train 5 replicates each for the different learning rates and training data sizes listed earlier. You will need to save evaluation plots and test data to files for your report, however you do not need to commit these to your repository.
    * GPU Assignment: Your code should train 5 replicates each for the different hidden layer counts and neurons per hidden layers listed earlier. You will need to save evaluation plots and test data to files for your report, however you do not need to commit these to your repository.
- Important, acknowledge all assistance from AI and other resources!

Rubric: What to Turn In: Canvas

- Submit on Canvas a PDF named `NeuralNetworkReport.pdf`:
  - Write your name(s) and whether you are doing the CPU or GPU assignment.
  - INTRODUCTION SECTION: Describe in this section (a paragraph or two long) the dataset and the questions about supervised learning you are using them to address.
  - METHODOLOGY SECTION: Describe in a section (a paragraph or two) supervised learning and how your code implements it.
  - ASSIGNMENT RESULTS AND DISCUSSION SECTION, CPU Assignment: Have a section on your results for learning rates and training data sizes. This should include two plots (evaluation losses versus epochs) and one table (testing losses). How do the explored variables affect convergence, performance, and training time?
  - ASSIGNMENT RESULTS AND DISCUSSION SECTION, GPU Assignment: Have a section on your results for network architectures. This should include two plots (evaluation losses versus epochs) and one table (testing losses). How do the explored variables affect convergence, performance, and training time?
  - SELF DIRECTED RESULTS AND DISCUSSION SECTION: Present your variant on architecture or training, and your hypothesis about what changes you would see in performance. Give an explanation for why what you changed helped, or hurt, the performance of learning. Present an evaluation loss plot and tessting loss values.
  - ACKNOWLEDGMENT SECTION: Note any AI or other resources you used in coding your experiments and preparing this document.