CPSC 481 Artificial Intelligence
Project 4 – Machine Learning

Mode: **Up to 4 persons.**
Due Date: As shown

Introduction

This project will be an introduction to machine learning.
The code for this project contains the following files, available as a zip archive, ***machinelearning.zip***.

Files you'll edit:

models.py	Perceptron and neural network models for a variety of applications

Files you should read but NOT edit:

nn.py	Neural network mini-library

Files you will not edit:

autograder.py	Project autograder
backend.py	Backend code for various machine learning tasks
data	Datasets for digit classification and language identification
submission_autograder.py	Submission autograder (generates tokens for submission, not used)

Files to Edit and Submit: You will fill in portions of ***models.py*** during the assignment. Please do not change the other files in this distribution.

Evaluation: Your code will be autograded for technical correctness. Please do not change the names of any provided functions or classes within the code, or you will wreak havoc on the autograder. However, the correctness of your implementation – not the autograder's judgements – will be the final judge of your score. If necessary, we will review and grade assignments individually to ensure that you receive due credit for your work.

Academic Dishonesty: We will be checking your code against other submissions in the class for logical redundancy. If you copy someone else's code and submit it with minor changes, we will know. These cheat detectors are quite hard to fool, so please don't try. We trust you all to submit your own work only; please don't let us down. If you do, we will pursue the strongest consequences available to us.

Proper Dataset Use: Part of your score for this project will depend on how well the models you train perform on the test set included with the autograder. We do not provide any APIs for you to access the test set directly. Any attempts to bypass this separation or to use the testing data during training will be considered cheating.

Getting Help: You are not alone! If you find yourself stuck on something, office hours, section, and the discussion forum are there for your support; please use them. If you can't make our office hours, let us know and we will schedule more. We want these projects to be rewarding and instructional, not frustrating and demoralizing. But, we don't know when or how to help unless you ask.

Discussion: Please be careful not to post spoilers.

Installation

For this project, you will need to install the following two libraries:

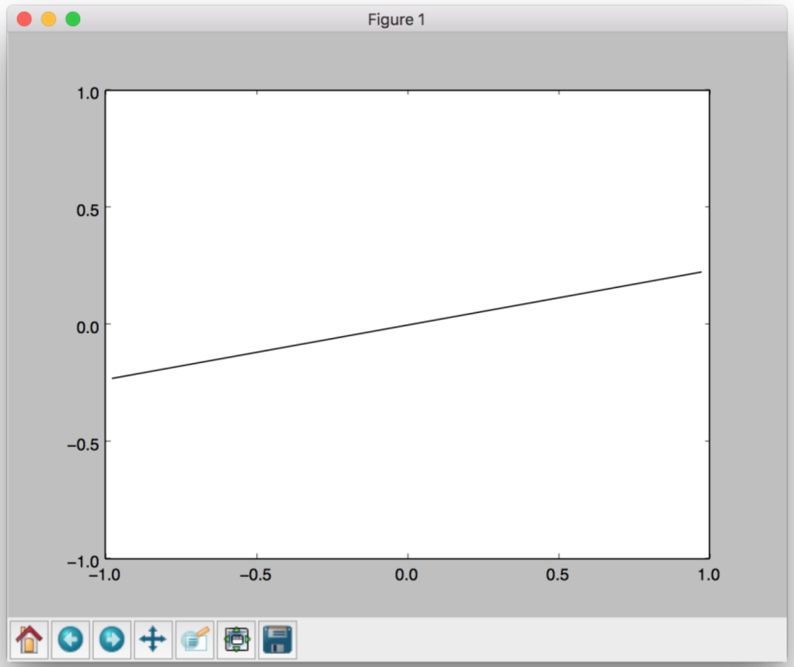
- numpy, which provides support for large multi-dimensional arrays
- matplotlib, a 2D plotting library

You will not be using these libraries directly, but they are required in order to run the provided code and autograder.

To test that everything has been installed, run:

```
python autograder.py --check-dependencies
```

If **numpy** and **matplotlib** are installed correctly, you should see a window pop up where a line segment spins in a circle:



To install the dependencies*:*

pip install numpy
pip install matplotlib

Follow the following instructions for further trouble shooting:
https://numpy.org/install/
https://phoenixnap.com/kb/install-numpy

Provided Code

For this project, you have been provided with a neural network mini-library (nn.py) and a collection of datasets (backend.py).

The library in nn.py defines a collection of node objects. Each node represents a real number or a matrix of real numbers. Operations on node objects are optimized to work faster than using Python's built-in types (such as lists).

Here are a few of the provided node types:

nn.Constant represents a matrix (2D array) of floating point numbers. It is typically used to represent input features or target outputs/labels. Instances of this type will be provided to you by other functions in the API; you will not need to construct them directly.
nn.Parameter represents a trainable parameter of a perceptron or neural network.
nn.DotProduct computes a dot product between its inputs.

Additional provided functions:

nn.as_scalar can extract a Python floating-point number from a node.
When training a perceptron or neural network, you will be passed a dataset object. You can retrieve batches of training examples by calling dataset.iterate_once(batch_size):

```
for x, y in dataset.iterate_once(batch_size):
...
```

For example, let's extract a batch of size 1 (i.e., a single training example) from the perceptron training data:

```
>>> batch_size = 1
>>> for x, y in dataset.iterate_once(batch_size):
...     print(x)
...     print(y)
...     break
...
<Constant shape=1x3 at 0x11a8856a0>
<Constant shape=1x1 at 0x11a89efd0>
```

The input features x and the correct label y are provided in the form of nn.Constant nodes. The shape of x will be batch_size x num_features, and the shape of y is batch_size x num_outputs. Here is an example of computing a dot product of x with itself, first as a node and then as a Python number.

```
>>> nn.DotProduct(x, x)
<DotProduct shape=1x1 at 0x11a89edd8>
>>> nn.as_scalar(nn.DotProduct(x, x))
1.9756581717465536
```

Question 1: Perceptron

Before starting this part, be sure you have **numpy** and **matplotlib** installed!

In this part, you will implement a binary perceptron. Your task will be to complete the implementation of the **PerceptronModel** class in **models.py**.

For the perceptron, the output labels will be either 1 or −1, meaning that data points (x, y) from the dataset will have y be a **nn.Constant** node that contains either 1 or −1 as its entries.

We have already initialized the perceptron weights **self.w** to be a 1 × dimension parameter node. The provided code will include a bias feature inside x when needed, so you will not need a separate parameter for the bias.

Your tasks are to:

- Implement the **run(self, x)** method. This should compute the dot product of the stored weight vector and the given input, returning an nn.DotProduct object.
- Implement **get_prediction(self, x)**, which should return 1 if the dot product is non-negative or −1 otherwise. You should use **nn.as_scalar** to convert a scalar Node into a Python floating-point number.
- Write the **train(self)** method. This should repeatedly loop over the data set and make updates on examples that are misclassified. Use the update method of the nn.Parameter class to update the weights. When an entire pass over the data set is completed without making any mistakes, 100% training accuracy has been achieved, and training can terminate.

In this project, the only way to change the value of a parameter is by calling **parameter.update(direction, multiplier)**, which will perform the update to the weights:

weights←weights + direction × multiplier

The direction argument is a Node with the same shape as the parameter, and the multiplier argument is a Python scalar.

To test your implementation, run the autograder:

```
python autograder.py -q q1
```

Note: the autograder should take at most 20 seconds or so to run for a correct implementation. If the autograder is taking forever to run, your code probably has a bug.

Plz. Ignore questions 2-4 for this project.

Grading for this project:
There is a project 4 rubric attached below project description on Canvas.
Your program needs to pass autograder tests.

Deliverables:
Include team members in a readme file.
Submit models.py and readme to the submission link.
One team only submit ONE copy. Pick a team member who will be responsible for submitting the project.
Other members do not need to submit. Canvas may mark your project as "Missing", but you do not need to worry about it.
I'll contact you if I cannot find your name in any readme files.