

Codelet 4: Multilayer Perceptrons using PyTorch

The due date for this codelet is **Wednesday, Feb 26 at 11:59PM**.

Introduction

The aim of this codelet is to continue building on your PyTorch skills and to give you a concrete example demonstrating the role hidden representations play with neural networks. You should draw on Chapter 5 of the textbook (particularly 5.2.1), the PyTorch documentation, and all of your prior codelets.

Important: Only use PyTorch, numpy, pandas, matplotlib, and in-built Python for this codelet. The use of any other libraries, including the books d2l library results in an automatic unsatisfactory grade for this assignment. A core goal of this class is to build your competencies as a machine learning engineer. I want to minimize abstractions from other libraries so that you build these skills.

Outline

- ☐ Grading
- ☐ Multilayer Perceptrons
 - ☐ Data
 - ☐ Model
 - ☐ Training and Evaluation
 - ☐ Plotting

Your assignment

Your task is to:

1. Download `codelet4.zip` from the course website and open it. You will find these instructions, `xor.csv` which contains the data, and `codelet4.py`, which has scaffolding for you.
2. Complete each of the 4 broad tasks below in `codelet4.py` and include a file called `codelet4.pdf` with your answers to the written questions.

Grading

When assessing your work, satisfactory achievement is demonstrated, in part, by:

- Simple, minimal, and clear code
- Code utilizes PyTorch methods in a way that simplifies your code
- Code passes relevant tests
- Explanation extends beyond repeating the textbook/material online
- Response are concrete and clearly demonstrate an understanding of the concepts

Multilayer Perceptrons for Classification

You will complete an implementation of a multilayer perceptron trained using gradient descent. You have four tasks:

1. Load data from a csv
2. Implement a model class
3. Train and evaluate your model
4. Complete the plotting function

Data

The problem we are trying to solve with this codelet is to learn the logical operator XOR, which is a staple example. The XOR operator combines two truth values and returns True (1) when only one of the two truth

values is True, otherwise it returns False (0). For example, an input like 1 XOR 0 is True but 1 XOR 1 is False (which is contrary to OR which would return True). The four relevant samples and their output are provided in `xor.csv`. Your task is to load this data in a suitable format to train a PyTorch model on. **Note: Keeping your code minimal is ultimately good. We don't need fancy things to do simple things.**

Evidence of your completion of this task comes from successfully training your model in a later task.

Multilayer Perceptron Model

No basic class structure is provided. Your aim is to build a MLP with one hidden layer with two nodes and that produces a single output given the two inputs. Your hidden layer should use ReLU as its activation function and the output's activation function should be Sigmoid. Before implementing your class, draw the MLP described above and include it in your `codelet4.pdf`. Note, you may have to adapt your model class to help you plot later. You can and should draw on your prior codelets in constructing this class.

Evidence of your completion of this task comes from successfully training your model in a later task.

Training and Evaluation

No train or evaluation function is provided. Your aim is to train your MLP and evaluate its performance on this task. It is helpful to begin with evaluation. You should create a function that calculates the **accuracy** of your model on this task. You should assume that an output greater than 0.5 corresponds to a prediction of True (1), otherwise the prediction is False (0). This function should avoid calculating the gradients, as with `codelet3`.

To evidence completion of this task, and to check your own progress, you should report the accuracy of your model before and after training. You should add to the pdf accompanying your code a screenshot of the output of your code. Mine, for example, looks as follows:

```
Initial Accuracy: 50.0%
Final Accuracy: 100.0%
```

Turning to training, your aim is to train a model to achieve 100% accuracy on this task. As a cost function, you should use Binary Cross-Entropy, which is the same cost function used in Logistic Regression. You can find PyTorch's here. You can and should draw heavily on earlier codelets. The initialization of the network greatly affects its final performance, as does the number of epochs. To assist you, a line of code is included at the top of `codelet4.py` which sets the random seed for PyTorch. You should tweak this until you find a seed that works well. For example, some of my random seeds remain stuck at 50% accuracy.

To evidence completion of this task, and to check your own progress, you should report the initial and final cost value for your model on the data. You should add to the pdf accompanying your code a screenshot of the output of your code. Mine, for example, looks as follows:

```
Initial Cost: 0.7148466110229492
Final Cost: 0.09981650859117508
```

Plotting Representations

Your final task is to complete the `plot` function and to reason about the role of the hidden layer in solving XOR. To help you approach this, `plot` provides the basic scaffolding. The leftmost plot, which plots input features and labels, is already completed for you. On that plot, you'll notice four points, two squares and two stars. The X and Y axis represent different input features. For example, when the input features are both 0, the predicted label is 0 (False), which is represented by a star.

Your aim is to complete the rightmost plot, which gives the relationship between the hidden layer nodes and the labels. Note, you can add whatever parameters you want to the `plot` function.

To evidence completion of this task please provide screenshots of the final plots in the pdf accompanying your code. Additionally, you should provide answers to the following questions:

1. Given the plot on the left, why is XOR impossible to learn using Logistic Regression?
2. Given the plot on the right, how does the hidden layer assist in solving the XOR problem?
3. In general, what kinds of representations do you think a neural network tries to learn before the output layer? What type of model do you think the final weights represent (i.e. the weights mapping from the last hidden layer to the model's prediction)?