Uyeda Lab: An Introduction to Deep Learning

Caleb Charpentier

1/18/23

Table of contents

Preface

I am writing this document to serve as a quick-reference for members of Uyeda Lab who are interested in deep learning.

I am rendering it with Quarto via JupyerLab, but you can follow along with whatever IDE you prefer. I do, however, highly recommend JupyterLab if you have only been exposed to Jupyter Notebooks. The interface is much more similar to traditional IDEs like RStudio, and I find it makes project management much more convenient.

If you are interested in following along with JupyterLab and don't know how to set it up, I recommend installing Anaconda or mamba first. You may then install JupyterLab in the conda environment of your choosing.

Conda is a package management system that allows you to create environments containing Python and R packages, as well as other programs. mamba is a reimplementation of Conda in C++, and its operations have more efficient parallelization. Thus, it tends to be faster.

If you install Conda, you may install Anaconda or miniconda. Installing Anaconda also installs a bunch of extra Python packages that may or may not be useful for you. Therefore, it takes up alot more space on your machine. Unlike miniconda, however, Anaconda comes with Anaconda Navigator preinstalled. This is a graphical user interface that may be convenient if you are unfamiliar with Conda or are uncomfortable with command lines.

I DO NOT recommend installing JupyterLab in the base Conda environment, as base environment installations easily lead to package conflicts. Instead, create a separate environment, activate that environment, and then install it.

If you are an RStudio user and prefer to use RMarkdown or Quarto in RStudio, you can evaluate Python code in cells with the Reticulate package. If you like, you can even evaluate code chunks with an existing conda environment.

1 Introduction

While the history of deep learning can be traced back to 1943, it didn't become terribly useful until 2012.

To understand why this is the case, we must ask three questions:

1. What are neural networks?
2. Why weren't they useful in the past?
3. What makes them useful now?

1.1 What are neural networks?

To put it in oversimplified terms, a neural network is just a recursive generalization of a Generalized Linear Model.

Lets think about simple multiple regression, with coefficients β_0, β_1, and β_2. In neural networks, coefficients are generally referred to as *weights*, but they have the same conceptual meaning as in more traditional linear models.

In our model, lets supppose our weights correspond with three independant variables: x_0, x_1, and x_2. Most likely, our model is being used to predict some real world value.

Because of this, … "universal approximator". … its not so much that they are difficult to interpret to interpret, its that interpretability is inherently not part of the model.

1.2 Why weren't they useful in the past?

Lack of computation resources Lack of datasets Insufficient inductive reasoning

1.3 What makes them useful now?

GPUs (Nvidia/CUDA/TPUs) Modern datasets CNNS –> Transformers

1.4 The issue in Science

1. Interpretability
2. Post-hoc methods vs Interpretable Architecture
3. Inductive Reasoning

2 Summary

In summary, this book has no content whatsoever.

3 Foundational Concepts in Deep Learning

3.0.1 Learning Rates

3.0.1.1 Learning Rate Decay

3.1 Optimizers

3.1.1 Stochastic Gradient Descent

3.1.2 Adam Optimization

3.1.2.1 Momentum

3.2 Batch Learning

3.3 Dropout

4 Neural Network Architectures

4.1 Convolutional Networks

4.2 Recurrent Networks

4.3 LSTMs and GRUs

4.4 Self-Attention and Transformers

4.4.1 Axial Transformers

5 Introduction to PyTorch

It's time to learn PyTorch.

As of 2023, PyTorch is the most common framework for implementing deep learning models in scientific papers. It integrates naturally with other Python packages such as Numpy, Pandas, and Matplotlib, and makes backpropagation both easy to implement and flexible enough to create complex learning models.

As a side note, because neural networks are ultimately graphical models, I find some PyTorch concepts vaguely analogous to model implementation in RevBayes. If you are proficient in RevBayes, approaching the package with this mindset may or may not be helpful for you. Nevertheless, none of it will be very useful if you don't understand the concepts we've discussed so far.

In this chapter, I'm to give an overview of important PyTorch concepts. We will discuss objects, modules, and functions in the packages, and finally how to combine them to train neural networks of your own. Most of the code here is either taken directly from the PyTorch documentation, or comes from this Udemy Course by Fawaz Sammani.

References