Project 1: Neural Networks

Santosh Ganji

March 18, 2024

1 Introduction

Good visualization of back propagation is available in this blog post. Please watch Grant Sanderson's video on Neural Networks to develop good intuition. Christopher Olah's highly influential blog posts might also help improve understanding of neural networks.

Over time, python frameworks evolved and capitalized on new advancements in hardware design, making developers' lives a little easier. Still, there is a lot of room for improvement. For now, we focus our attention on machine learning research and stick with Python.

Training neural networks requires data (a lot of it), and performing a large number of numerical operations. Writing naive code increases time and space complexities. Occasionally, researchers come up with effective linear algebra libraries and numerical techniques. Several frameworks to take advantage of hardware architecture for large-scale deployment of new ideas are under active development. So, we encourage students to persistently explore the landscape and educate themselves with libraries like JAX and Parallel computing libraries like PyOMP.

For developers who rely on Intel processors, Intel One API provides several native optimizations that requires just changing the python interpreter. Nvidia users may find cuNumeric and Numba helpful. Knowledge of any of your favorite optimization libraries may be a highly desirable job skill!

In this tutorial, which is followed by the accompanying Project for this week, we focus on the fundamentals of the training and testing processes.

2 A tutorial on training and testing neural networks

In this tutorial the goal is to understand how to set up and train a neural network to approximate labeled samples. Please download the jupyter notebook from [7].

In the example jupyter notebook, we first show how to create samples that are later used to train and evaluate the neural network approximation.

Consider the function,

$$f(x_1, x_2) = x_1^2 + x_2^2 \tag{1}$$

We call x_1 and x_2 as features. The feature vector is

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}. \tag{2}$$

2.1 Generate training samples and testing samples

Cell 4 in the jupyter notebook shows how to generate 500 training samples $\{(x^{(i)}, y^{(i)}) : 1 \leq i \leq m\}$, where each $x^{(i)} \in R^2$, and $y^{(i)} = f(x_1^{(i)}, x_2^{(i)}) + v^{(i)} \in R^1$. The entries $x_1^{(i)}, x_2^{(i)}$ of each vector $x^{(i)}$ are randomly and independently chosen from $[-1, +1]$ according to a uniform distribution. The labels $y^{(i)}$ are noisy values of $f(x_1^{(i)}, x_2^{(i)})$. The $v^{(i)}$'s are independent and identically distributed with distribution $N(0, 0.01)$, i.e., normally distributed with mean 0 and variance 0.01.

Cell 4 also shows how to generate the test samples. No noise is added to the 50 test samples.

2.2 Define a neural network stack

There are several ways of writing a neural network stack. Cells 7 and 17 in the jupyter notebook provide two different ways of defining a network.

The network defined has 2 input features, 2 neurons at level 1, and 1 neuron at level 2 which produces the output. The network is fully connected. This is shown in Figure 1.

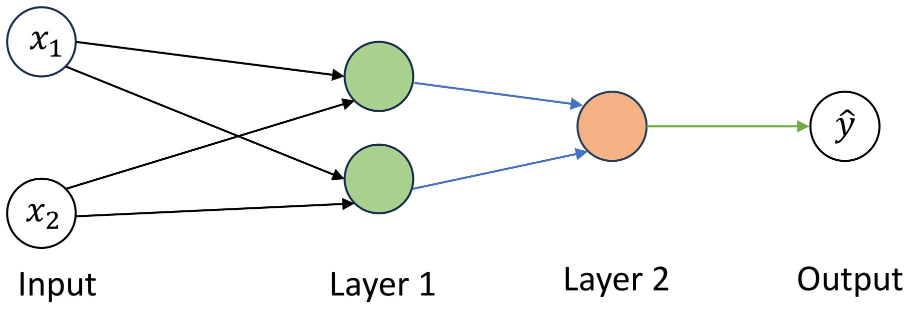

Figure 1: Two-layer network of shape {2,1}, i.e., two neurons at Layer 1, and one at Layer 2. These are fed by 2 inputs, which are completely connected to the neurons at Layer 1. The neuron at Layer 2 generates 1 output. The neurons at Layer 1 are RELU, but the last neuron at the output layer is linear. It has no nonlinearity. It is just a weighted sum of its inputs and bias.

Please note that there are a total of (2x2) weights +(2x1)weights +3biases = 9 parameters.

2.3 Defining a loss function

The loss function we choose is average squared error: $\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)} - \hat{y}(x_{(i)}, \theta)\right)^2$. Here m is the number of training samples. Cell 13 shows how to define the loss function.

2.4 Train neural network

The gradient descent method is called "stochastic gradient method" in the literature, and abbreviated as SGD.

After defining the network stack Cell 25 shows a very simple way to train the network using backprop and gradient descent. The hyperparameters for momentum and learning rate are chosen in Cell 24.

Please note however that this is NOT the code used to actually train the network. That code is provided in Cell 32. The prior cells are only for simple illustratory purposes for your understanding. Please see Cell 32 for the actual code used for training as well as testing.

2.5 Plotting the neural network training error

Please see Cell 36 on how to plot the training losses change as we continue to train.

2.6 Evaluating neural network on test data

Please see Cell 31 to plot the performance of the trained system on the test data.

Cell 36 also shows the performance on the test data as we continue to train.

2.6.1 Magic of over-parametrization

There is an interesting empirical phenomenon called "double descent" that can sometimes be observed: A model with a small number of parameters and a model with an extremely large number of parameters have a small test error, but a model whose number of parameters is about the same as the number of data points used to train the model has a large test error. You can read more about this phenomenon in Preetam et. al work and open AI blog plot.

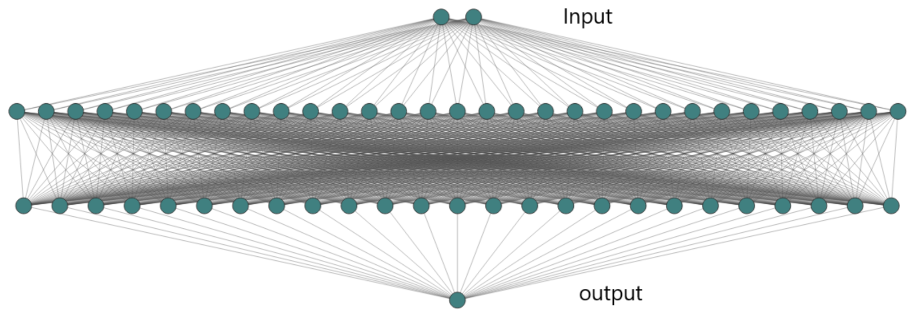

Figure 2: Network with large number of neurons in Layer 1 and 2

We consider neural networks with a large number of neurons in Layer 1, as shown in Fig. 2. Specifically we consider neural networks with 10 or 100 neurons in Layer 1. We repeat the training for each of these large neural networks which have many more parameters. In Cells 37, 43, 50, and 55 we plot the training and test performance of a neural network. Please see the very end of Cell 65 to see this “double descent” phenomenon.

3 Project Assignment

Now we come to the project that you are assigned to do this week. It consists of one problems, carrying 100 points.

What was provided above was only snippets to help you understand. Please write your own code after assimilating the above explanatory material.

Problem:

- Let $f(x_1, x_2, x_3, x_4) := x_1^2 + x_2^2 + 2x_3^3 + 4x_4^3$. It consists of 4 inputs x_1, x_2, x_3, x_4 and one output. Generate training samples by adding independent normally distributed noise of mean 0 and variance 0.01. Create a dataset consisting of 500 training samples with all $x_i \in [-1, 1]$, and 50 noiseless test samples with all $x_i \in [-2, -1] \cup [1, 2]$.
- Train a small neural network of shape (2,1). The four inputs are completely connected to the two RELU neurons at layer 1. These two neurons are connected to a linear output neuron.
- Break the 500 training samples into five minibatches each containing 100 labeled samples. (This is a new detail not present in the above tutorial). Plot the learning performance over 25 episodes, where in each episode all the five minibatches are used once in order.
- Train 5-10 different neural networks that have 10s, 100s, and 1000s of parameters.
- Plot the training and test performances of all networks.
- Summarize all your observations.
- In the summary, mention if you’ve observed the double descent phenomenon.

References

[1] Accelerated linear algebra. URL: https://openxla.org/xla.

[2] Collection of tools to visualize network. URL: https://github.com/ashishpatel26/Tools-to-Design-or-Visualize-Architecture-of-Neural-Network.

[3] Gpu-accelerated supercomputing to the numpy ecosystem. URL: https://developer.nvidia.com/cunumeric.

[4] Intel® distribution for python. URL: https://www.intel.com/content/www/us/en/developer/tools/oneapi/distribution-for-python.html#gs.507phb.

[5] Jax: High-performance array computing. URL: https://jax.readthedocs.io/en/latest/.

[6] Nvdia cuda for python. URL: https://developer.nvidia.com/how-to-cuda-python.

[7] Project 1. URL: https://github.com/shotsan/ECEN_740_Project1.

[8] Pyomp: Parallel multithreading that is fast and pythonic. URL: https://www.openmp.org/wp-content/uploads/OpenMPBoothTalk-PyOMP.pdf.

[9] Pytorch tutorials. URL: https://pytorch.org/tutorials/.

[10] Tensorboard. URL: https://www.tensorflow.org/tensorboard/graphs.

[11] Tensorflow tutorials. URL: https://www.tensorflow.org/guide/.