CS 5600/6600: F24: Intelligent Systems
Assignment 3

Vladimir Kulyukin
Department of Computer Science
Utah State University

September 14, 2024

Learning Objectives

1. Backpropagation
2. MNIST
3. Training and Testing Larger ANNs (aka Bulldozing)

Problem 1 (2 pts)

This assignment will give you a flavor of what it takes to train deeper ANNs on larger datasets. Some AI researchers refer to this process as *bulldozing*. Essentially, one bulldozes model after model until some performance parameter is satisfied.

Many people are crazy about Deep Learning (DL) these days, and for justifiable reasons. ChatGPT, Bing Image Generator, Google Gemini, etc. are on everybody's lips. DL systems have been and are being applied to problems that include, but are not limited to, text/image classification and generation, gaming, speech/audio processing, and music synthesis and analysis, etc. To outside observers, these results are impressive.

But, bulldozing has costs. Let us leave ethical costs (e.g., human replacement, plagiarism) out of the scope of this discussion for now and focus on physically measurable costs – energy and environment. There are few investigations of the energy and environmental footprints of DL systems. Yet, such investigations are urgently needed because of rising energy costs. Let me bring this issue home. Since fall 2022, the city of Logan has levied an electrical surcharge on monthly utility bills "to recover costs Logan City incurred as it purchased electricity on the open market where prices were higher than expected for reasons that were not foreseeable." Such investigations are also fundamental, because of the rising ecological and environmental costs of cloud computing required to support the continuous operation of many DL models that require large GPU farms. Some of the prominent environmental and ecological costs are growing water consumption rates for cooling server farms, rising ocean water temperatures due to submerged data centers, rising ambient temperatures due to the heat generated by server farms, and increasing levels of electromagnetic radiation potentially extremely harmful to many animals. By the way, electromagnetic radiation is another topic frequently brushed aside.

These issues bring us to this week's reading assignment – a 2022 article by Steven Gonzalez Monserrate, which is a condensed version of his M.S. Thesis defended at the MIT Program in History, Anthropology, and Science, Technology, and Society. Mr. Monserrate was an MIT anthropologist who spent a year doing his field research at several U.S. cloud computing farms in 2021–2022. Write a one page analysis of Monserrate's article "The Cloud Is Material: On the Environmental Impacts of Computation and Data Storage." Remember to include in your analysis the following four components: **1)** a brief statement of the problem addressed in the paper; **2)** what you liked in the paper

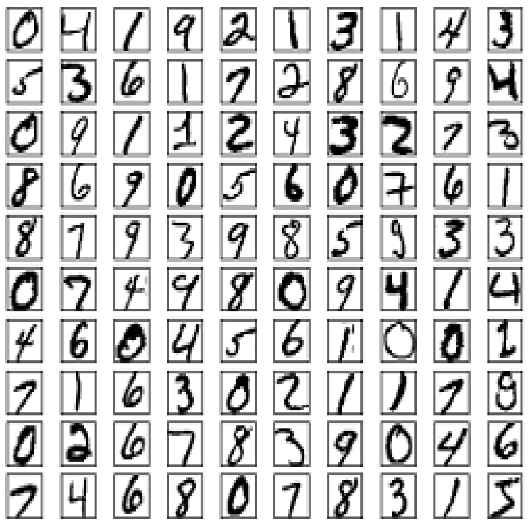
Figure 1: A Sample of MNIST images.

and why; **3)** what you did not like in the paper and why; **4)** any inspirations you found in the paper. Each of these points should be addressed in a separate paragraph. There is no problem if you go over one page, but do not write an opus. Save your analysis in `monserrate.pdf`.

Problem 2 (0 pts)

The purpose of this lab exercise is to prepare you for Problem 3 below. We will play with the MNIST (Modified National Institute of Standards and Technology) dataset. If you have had some exposure with MNIST before, you may quickly scan this section and return to it as necessary as you work on Problem 3.

MNIST is a relatively small written digit image dataset used to test machine learning (ML) algorithms on small and relatively simple images. The dataset contains 60K 28x28 grayscale training images and 10K 28x28 grayscale testing images. In 2017, MNIST was extended to EMNIST which contains 240K training and 40K testing images of handwritten digits. From the perspective of 2024, these datasets are relatively small given the fact that more and more ML applications are trained on terabytes of data. But this is a nice test bed to try different approaches.

Handwritten digit recognition is an excellent prototype problem for learning neural networks. The problem is nontheless challenging and provides an opportunity to learn how neural networks work, because the problem is not as difficult as other problems tackled by NNs. Another useful, pedagogical feature of MNIST is incremental buildup: we can start with very basic NNs and gradually improve them with more advanced techniques.

A sample of MNIST images is in Figure 1. The file `mnist_loader.py` (included in the `HW03` zip) contains the function `load_data_wrapper` that loads the processed training, validation, and testing data for the MNIST dataset. This function unpacks the MNIST images and their targets from `data/mnist.pkl.gz` (also included in the zip). Let us load MNIST and play with it.

```
>>> from mnist_loader import *
>>> train_d, valid_d, test_d = load_data_wrapper()
train_d, valid_d, test_d = load_data_wrapper()
>>> len(train_d)
50000 ### there are 50,000 training images
>>> len(valid_d)
10000 ### there are 10,000 validation images
```

```
>>> len(test_d)
10000 ### there are 10,000 testinging images
```

As the above interaction shows, we have 50,000 training images with targets, 10,000 validation images with targets, and 10,000 test images with targets. We can use the ann class from `ann.py` in the zip to construct ANNs with arbitrarily many layers. Here is how we can construct an 784x30x60x120x10 ANN. Why do we need 784 input nodes? Because 784 = 28x28, i.e., we need 784 input nodes to make each graypixel an input node.

```
>>> from ann import *
>>> net = ann([784, 30, 60, 120, 10])
>>> net
<ann.ann object at 0x7f43d095a560>
```

Let us construct a shallower 784x30x10 ANN and train it with Stochastic Gradient Descent (SGD) on MNIST with the learning rate $\eta = 3.0$, 20 epochs (0 – 19), and a mini-batch of 10.

```
>>> from mnist_loader import load_data_wrapper
>>> train_d, valid_d, test_d = load_data_wrapper()
>>> from ann import ann
>>> net = ann([784, 30, 10])
>>> net.mini_batch_sgd(train_d, 20, 10, 3.0, test_data=test_d)
```

Below is my output on in Python 3.10.12 on Ubuntu 22.04.04 LTS (Jammy Jellyfish).

```
Epoch 0: 9037 / 10000
Epoch 1: 9239 / 10000
Epoch 2: 9291 / 10000
Epoch 3: 9329 / 10000
Epoch 4: 9383 / 10000
Epoch 5: 9394 / 10000
Epoch 6: 9412 / 10000
Epoch 7: 9421 / 10000
Epoch 8: 9425 / 10000
Epoch 9: 9425 / 10000
Epoch 10: 9461 / 10000
Epoch 11: 9454 / 10000
Epoch 12: 9459 / 10000
Epoch 13: 9491 / 10000
Epoch 14: 9457 / 10000
Epoch 15: 9470 / 10000
Epoch 16: 9485 / 10000
Epoch 17: 9487 / 10000
Epoch 18: 9489 / 10000
Epoch 19: 9484 / 10000
```

I got 94.84% accuracy at the end of epoch 19. Not bad for MNIST. Note that in epoch 6, the NN reaches the plateaue of 94%. Your numbers, of course, will be different, because the weights are randomly initialized.

Let us now train a deeper 784x30x60x120x10 ANN.

```
>>> net = ann([784, 30, 60, 120, 10])
>>> net.mini_batch_sgd(train_d, 20, 10, 3.0, test_data=test_d)
```

Here is my output.

```
>>> net = ann([784, 30, 60, 120, 10])
>>> net.mini_batch_sgd(train_d, 20, 10, 3.0, test_data=test_d)
Epoch 0: 1551 / 10000
Epoch 1: 1553 / 10000
Epoch 2: 3424 / 10000
Epoch 3: 5421 / 10000
Epoch 4: 6612 / 10000
Epoch 5: 8208 / 10000
Epoch 6: 8331 / 10000
Epoch 7: 9230 / 10000
Epoch 8: 9317 / 10000
Epoch 9: 9238 / 10000
3Epoch 10: 9345 / 10000
Epoch 11: 9314 / 10000
Epoch 12: 9379 / 10000
Epoch 13: 9405 / 10000
Epoch 14: 9408 / 10000
Epoch 15: 9481 / 10000
Epoch 16: 9424 / 10000
Epoch 17: 9502 / 10000
Epoch 18: 9437 / 10000
Epoch 19: 9510 / 10000
```

This time I got 95.10%, which is is slightly better than 94.84%, which is a gain, but nothing to write home about.

Let us train a shallower network for 100 epochs and decrease the learning rate η to 0.05. My output is below. I deleted some output lines for brevity.

```
>>> net = ann([784, 15, 10])
>>> net.mini_batch_sgd(train_d, 100, 10, 0.05, test_data=test_d)
Epoch 0: 3496 / 10000
Epoch 1: 4758 / 10000
Epoch 2: 5494 / 10000
Epoch 3: 6138 / 10000
Epoch 4: 6568 / 10000
Epoch 5: 6884 / 10000
...
Epoch 95: 9200 / 10000
Epoch 96: 9201 / 10000
Epoch 97: 9204 / 10000
Epoch 98: 9210 / 10000
Epoch 99: 9207 / 10000
```

The net increased its accuracy from 34.96% to 92.07%, which is great, but took ≈25 minutes on my old laptop. Let's keep training the same net. I hope that my laptop doesn't smoke up in the process.

```
>>> net.mini_batch_sgd(train_d, 100, 10, 0.05, test_data=test_d)
Epoch 0: 9210 / 10000
Epoch 1: 9209 / 10000
Epoch 2: 9216 / 10000
Epoch 3: 9212 / 10000
Epoch 4: 9216 / 10000
Epoch 5: 9213 / 10000
...
Epoch 95: 9287 / 10000
Epoch 96: 9290 / 10000
Epoch 97: 9294 / 10000
Epoch 98: 9287 / 10000
Epoch 99: 9290 / 10000
```

From 92.07% up to 92.90%. Nothing to boast about, but still a gain.

Let me increase the hidden layer to 300 neurons and train this network for 100 epochs.

```
>>> net = ann([784, 300, 10])
>>> net.mini_batch_sgd(train_d, 100, 10, 0.05, test_data=test_d)
```

I will go for a walk now to rest my eyes on something green as my old bulldozer chugs its way up (or down!) to a plateau...

OK, I am back. It looks like my bulldozer stopped after ≈40 minutes and bulldozed me the following landscape.

```
Epoch 0: 2640 / 10000
Epoch 1: 3371 / 10000
Epoch 2: 3754 / 10000
Epoch 3: 4052 / 10000
Epoch 4: 4207 / 10000
Epoch 5: 4388 / 10000
...
Epoch 95: 7505 / 10000
Epoch 96: 7509 / 10000
Epoch 97: 7499 / 10000
Epoch 98: 7512 / 10000
Epoch 99: 7517 / 10000
```

Before we continue to bulldoze, I want to tell you that this class has a tradition to have a T-shirt design competition. I ask my students in class and in Canvas to come up with catchy themes/slogans/phrases related to this class that they'd put on a T-shirt. If you have one, please send it to me. One of the best jokes I heard related to this class was last year. Two students approached me after a lecture. One of the asked me a question. I wrote a few formulas on the board and did my best to explain them. After I finished, the other student asked his friend, "Did you backpropagate that, dude?" Some of my personal favorites from previous years are in Figure 2.

Problem 2 (3 pts)

Let us get more systematic about ANN design, training and testing. This problem will give you a flavor of what a NN designer/engineer/data scientist (aka deep bulldozerist) is typically doing with large datasets. We'll confine ourselves to ANNs with 1 and 2 hidden layers to keep it simple (KIS). We will define two global lists - HLS and ETA. HLS holds the values of the hidden layers and ETA the values of the learning rate η that we will experiment with.

I bulldoze the bulldozers
I'M NOT A ROBOT
... I HAVE EYEBALLS
RANDOM FOREST
NATIONAL PARK
RANDOM FOREST NATIONAL PARK
"WHERE EVERY TREE TELLS A STORY"

Figure 2: TOP LEFT: T-shirt by Brianna K. TOP RIGHT: T-shirt by Bryan C. BOTTOM LEFT: T-shirt by Brayden H. (Front pocket); BOTTOM RIGHT: T-shirt by Brayden H. (Back).

```
HLS = [10, 25, 50]
ETA = [0.5, 0.25, 0.125]
```

Write the function `train_1_hidden_layer_anns(hls=HLS, eta=ETA, mbs=10, ne=10)` that takes a list of HLS values (`hls` parameter), a list of η values (`eta` parameter), the mini batch size (`mbs`), and the number of epochs (`ne`), and calls the method `ann.mini_batch_sgd()` to train and test a 784 x h x 10 ANN for each possible h in `hls` at each possible value of η in `eta`. In other words, it will bulldoze $x \cdot y$ ANNs where x is the number of elements in `hls` and y is the number of elements in `eta`. Below is some of my output.

```
*** Training 784x10x10 ANN with eta=0.5
Epoch 0: 7370 / 10000
Epoch 1: 8460 / 10000
Epoch 2: 8716 / 10000
Epoch 3: 8834 / 10000
Epoch 4: 8909 / 10000
Epoch 5: 8932 / 10000
Epoch 6: 8982 / 10000
Epoch 7: 9029 / 10000
Epoch 8: 9015 / 10000
Epoch 9: 9043 / 10000
*** Training 784x10x10 ANN DONE...
...
*** Training 784x50x10 ANN with eta=0.5
Epoch 0: 6853 / 10000
Epoch 1: 8085 / 10000
Epoch 2: 8258 / 10000
Epoch 3: 8341 / 10000
Epoch 4: 8368 / 10000
Epoch 5: 8406 / 10000
Epoch 6: 8443 / 10000
Epoch 7: 8454 / 10000
Epoch 8: 9244 / 10000
Epoch 9: 9293 / 10000
*** Training 784x50x10 ANN DONE...
```

Save your implementation in `cs5600_6600_f24_hw03.py` and use it to fill in the cells of Table 1, where each cell (η, hls) is your accuracy of the ANN with the hls nodes in the hidden layer trained at the given value of η. Keep the mini batch size at 10 and the number of epochs at 10. Save this table with your statistics as a multiline comment in your `cs5600_6600_f24_hw03.py` under the heading Problem 3. What I mean is that you should have something like this.

```
'''
Problem 3: 1-layer NN
------------------------
Eta/HLS | 10 | 25 | 50 |
------------------------
0.5     |    |    |    |
------------------------
0.25    |    |    |    |
------------------------
0.125   |    |    |    |
------------------------
'''
```

Generalize `train_1_hidden_layer_anns()` to the function

```
       train_2_hidden_layer_anns(hls=HLS, eta=ETA, mbs=10, ne=10).
```

This function behaves analogously to `train_1_hidden_layer_anns()` but trains and tests 784 x h_1 x h_2 x 10 ANNs for each possible combination of h_1 and h_2 in hls at each possible value of η in eta. Here is some of my output.

```
*** Training 784x10x10x10 ANN at eta=0.5
Epoch 0: 6000 / 10000
Epoch 1: 7919 / 10000
Epoch 2: 8529 / 10000
Epoch 3: 8729 / 10000
Epoch 4: 8835 / 10000
Epoch 5: 8898 / 10000
Epoch 6: 8934 / 10000
Epoch 7: 8960 / 10000
Epoch 8: 8990 / 10000
Epoch 9: 9004 / 10000
*** Training 784x10x10x10 ANN DONE...
...
*** Training 784x10x25x10 ANN at eta=0.5
Epoch 0: 7577 / 10000
Epoch 1: 8478 / 10000
Epoch 2: 8785 / 10000
Epoch 3: 8945 / 10000
Epoch 4: 9043 / 10000
Epoch 5: 9091 / 10000
Epoch 6: 9108 / 10000
Epoch 7: 9134 / 10000
Epoch 8: 9155 / 10000
Epoch 9: 9149 / 10000
*** Training 784x10x25x10 ANN DONE...
```

Use your implementation of `train_2_hidden_layer_anns()` to fill in the tables below for each possible combination of h_1 and h_2. Save these tables with your statistics in the same multiline comment heading Problem 3 in `cs5600_6600_f24_hw03.py`. In other words, fill in the following tables.

```
'''
Problem 3: two-hidden layer NNs
h1 = 10
------------------------
Eta/h2  | 10 | 25 | 50 |
------------------------
0.5     |    |    |    |
------------------------
0.25    |    |    |    |
------------------------
0.125   |    |    |    |
------------------------

h1 = 25
------------------------
Eta/h2  | 10 | 25 | 50 |
------------------------
0.5     |    |    |    |
------------------------
0.25    |    |    |    |
------------------------
0.125   |    |    |    |
------------------------

h1 = 50
------------------------
Eta/h2  | 10 | 25 | 50 |
------------------------
0.5     |    |    |    |
------------------------
0.25    |    |    |    |
------------------------
0.125   |    |    |    |
------------------------
'''
```

What to Submit

1. `monserrate.pdf` with your summary of Monserrate's paper;
2. `cs5600_6600_f24_hw03.py` with your code and tables;

Happy Hacking, Reading, and Writing!