# CS 5600/6600: F21: Intelligent Systems
# Assignment 3

Vladimir Kulyukin
Department of Computer Science
Utah State University

September 18, 2021

## Learning Objectives

1. Backpropagation
2. MNIST
3. Training and Testing ANNs

## Problem 1 (2 pts)

Forty three years after the seminal paper "A Logical Calculus of the Ideas Immanent in Nervous Activity" by McCulloch and Pitts, three researchers (David Rumelhart, Geoffrey Hinton, and Ronald Williams) published another seminal paper "Learning Representations by Back-propagating Errors," where they formalized some of the ideas originally proposed by McCulloch and Pitts. This paper was one of the first ones to describe a modern version of backpropagation. While many backprop derivatives have been proposed since then, the basic ideas are as applicable today as in the mid 1980's. I included the PDF of a paper scan in the Assignment 03 zip.

Write a one page analysis of this paper. Your analysis should include four components: 1) a brief statement of the problem addressed in the paper; 2) what you liked in the paper 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's no problem if you go over one page, but don't write an opus. Save your analysis in `rumelhart_et_al.pdf`.

## Problem 2 (0 pts)

The purpose of this problem (it's actually a lab exercise!) is to prepare you for Problem 3 below. Let's start playing with the MNIST dataset that we discussed in Lecture 4. The file `mnist_loader.py` (included in the Assignment 03 zip) contains the function f`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's 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` to construct ANNs with arbitrarily many layers. Here's how we can construct an `784x30x60x120x10` ANN.

```
>>> from ann import *
>>> net = ann([784, 30, 60, 120, 10])
>>> net
<ann.ann object at 0x7f57ca22afd0>
```

Let's construct a shallow `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)
>>> net = ann([784, 30, 10])
>>> net.mini_batch_sgd(train_d, 20, 10, 3.0, test_data=test_d)
Epoch 0: 9050 / 10000
Epoch 1: 9166 / 10000
Epoch 2: 9313 / 10000
Epoch 3: 9321 / 10000
Epoch 4: 9356 / 10000
Epoch 5: 9383 / 10000
Epoch 6: 9395 / 10000
Epoch 7: 9404 / 10000
Epoch 8: 9441 / 10000
Epoch 9: 9428 / 10000
Epoch 10: 9440 / 10000
Epoch 11: 9473 / 10000
Epoch 12: 9488 / 10000
Epoch 13: 9464 / 10000
Epoch 14: 9470 / 10000
Epoch 15: 9462 / 10000
Epoch 16: 9485 / 10000
Epoch 17: 9492 / 10000
Epoch 18: 9492 / 10000
Epoch 19: 9493 / 10000
```

I got 94.93% accuracy at the end of epoch 19. Not bad for MNIST. Note that in epochs 17, 18, 19 the accuracy started to plateau. Your numbers, of course, will be different. This training took ≈5 minutes on my Dell laptop (Intel Core i3-7100U CPU @ 2.40GHz × 4 and 7.6 GiB RAM; Python 3.6.7, numpy 1.16.3, Ubuntu 18.04 LTS – Bionic Beaver).

Let's dive deeper and train with a `784x30x60x120x10` ANN.

```
>>> net = ann([784, 30, 60, 120, 10])
```

```
>>> net.mini_batch_sgd(train_d, 20, 10, 3.0, test_data=test_d)
Epoch 0: 985 / 10000
Epoch 1: 1552 / 10000
Epoch 2: 3427 / 10000
Epoch 3: 5421 / 10000
Epoch 4: 6611 / 10000
Epoch 5: 8207 / 10000
Epoch 6: 8330 / 10000
Epoch 7: 9230 / 10000
Epoch 8: 9316 / 10000
Epoch 9: 9238 / 10000
Epoch 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: 9512 / 10000
```

This bulldozing cycle took ≈10 minutes on my laptop, but I got to 95.12% at epoch 19. Nothing to write home about, but, hey, it's a gain, right? Also, note that the accuracy oscillated between 95.02%, 94.37%, and 95.12%, which suggests that it may be worth it to bulldoze some more, because it's not a plateau yet.

Let's modify our shallow network by reducing the hidden layer size to 5 nodes and bulldoze it with the same learning rate, number of epochs and mini batch size.

```
>>> net = ann([784, 5, 10])
>>> net.mini_batch_sgd(train_d, 20, 10, 3.0, test_data=test_d)
Epoch 0: 5879 / 10000
Epoch 1: 6074 / 10000
Epoch 2: 7151 / 10000
Epoch 3: 7428 / 10000
Epoch 4: 7397 / 10000
Epoch 5: 7356 / 10000
Epoch 6: 7442 / 10000
Epoch 7: 7565 / 10000
Epoch 8: 7618 / 10000
Epoch 9: 7264 / 10000
Epoch 10: 7294 / 10000
Epoch 11: 7631 / 10000
Epoch 12: 7688 / 10000
Epoch 13: 7475 / 10000
Epoch 14: 7612 / 10000
Epoch 15: 7495 / 10000
Epoch 16: 7696 / 10000
Epoch 17: 7721 / 10000
Epoch 18: 7800 / 10000
Epoch 19: 7893 / 10000
```

This I-am-like-so-shallow-dude ANN hasn't done as well as the previous two. The accuracy after

20 epochs is 78.93%. But, my bulldozer finished in less than 2 minutes. And, it doesn't look like there's a plateau yet.

Let's increase the hidden layer size to 200 neurons and train.

```
>>> net = ann([784, 200, 10])
>>> net.mini_batch_sgd(train_d, 20, 10, 3.0, test_data=test_d)
```

I'll go for a walk now to rest my eyes on something green and let my old virtual bulldozer chug its way up, up, up to an imaginary plateau...

OK, I'm back. It looks like my bulldozer stopped after ≈30 minutes and produced the following landscape.

```
Epoch 0: 5193 / 10000
Epoch 1: 5957 / 10000
Epoch 2: 6627 / 10000
Epoch 3: 6687 / 10000
Epoch 4: 7609 / 10000
Epoch 5: 7624 / 10000
Epoch 6: 7663 / 10000
Epoch 7: 7702 / 10000
Epoch 8: 7717 / 10000
Epoch 9: 7711 / 10000
Epoch 10: 7948 / 10000
Epoch 11: 8624 / 10000
Epoch 12: 8683 / 10000
Epoch 13: 8680 / 10000
Epoch 14: 8654 / 10000
Epoch 15: 8705 / 10000
Epoch 16: 8704 / 10000
Epoch 17: 8698 / 10000
Epoch 18: 8699 / 10000
Epoch 19: 8714 / 10000
```

We're up from 78.93% to 87.14%! Almost 10%! I'm tempted to jack up the HLS to 500 nodes and see what happens. It'll take about an hour on my laptop though. So, I'll pass and continue punching characters for this LaTeXdocument in my Emacs.

But before I move on, I want to share with you a memory that I just recalled. This class has a tradition to have a T-shirt design competition at the end of the semester. I ask my students in class and on Canvas to come up with catchy themes/slogans/phrases (related to this class) that they'd put on a T-shirt. My all time favorite T-shirt phrase was by Brianna King who designed and sent me the image in Figure 1 in December 2020. There's a funny story behind this phrase that I'll tell you some other time.

## Problem 3 (3 pts)

Let's 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. Let's KIS and confine ourselves to ANNs with 1 and 2 hidden layers. We'll define two global lists - HLS and ETA. HLS holds the values of the hidden layers and ETA the values of the learning rate $\eta$ that we'll experiment with.


Figure 1: Designed by Brianna King (Intelligent Systems: Fall 2020)

```
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)` a list of HLS values, a list of $\eta$ values, 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 $\eta$ in `eta`. In other words, it'll bulldoze $x \cdot y$ ANNs where $x$ is the number of elements in `hls` and $y$ is the number of elements in `eta`.

Here's 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 784x10x10 ANN with eta=0.25
Epoch 0: 6352 / 10000
Epoch 1: 7591 / 10000
Epoch 2: 8077 / 10000
Epoch 3: 8385 / 10000
Epoch 4: 8483 / 10000
Epoch 5: 8593 / 10000
Epoch 6: 8677 / 10000
Epoch 7: 8691 / 10000
Epoch 8: 8751 / 10000
```

```
Epoch 9: 8790 / 10000
*** Training 784x10x10 ANN DONE...
*** Training 784x10x10 ANN with eta=0.125
Epoch 0: 4326 / 10000
Epoch 1: 5425 / 10000
Epoch 2: 6801 / 10000
Epoch 3: 7320 / 10000
Epoch 4: 7604 / 10000
Epoch 5: 7802 / 10000
Epoch 6: 7954 / 10000
Epoch 7: 8148 / 10000
Epoch 8: 8283 / 10000
Epoch 9: 8395 / 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_f21_hw3.py` and use it to fill in the cells of Table 1, where each cell ($\eta$, HLS) is your accuracy of the ANN with the HLS nodes in the hidden layer trained at the given value of $\eta$. Keep the mini batch size at 10 and the number of epochs at 10. Save this table with your stats in `ann_report.pdf` under the heading **Problem 3**.

Table 1: Training ANNs with 1 Hidden Layer on MNIST; mini batch = 10, num epochs=10

| Eta/HLS | 10 | 25 | 50 |
|---|---|---|---|
| $\eta = 0.5$ | | | |
| $\eta = 0.25$ | | | |
| $\eta = 0.125$ | | | |

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 $h_1$ and $h_2$ in `hls` at each possible value of $\eta$ in `eta`. In other words, it'll train $x^2 \cdot y$ ANNs, where $x$ is the number of elements in `hls` and $y$ is the number of elements in `eta`.

Here's 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 784x10x10x10 ANN at eta=0.25
Epoch 0: 4724 / 10000
Epoch 1: 6797 / 10000
Epoch 2: 7900 / 10000
Epoch 3: 8295 / 10000
Epoch 4: 8544 / 10000
Epoch 5: 8650 / 10000
Epoch 6: 8734 / 10000
Epoch 7: 8805 / 10000
Epoch 8: 8835 / 10000
Epoch 9: 8871 / 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 stats in `ann_report.pdf` under the heading **Problem 3**. Briefly (no more than a paragraph) state your observations on the impact of the architecture and $\eta$ on the NN's accuracy.

Table 2: Training ANNs with 2 Hidden Layers on MNIST; $h_1$=10; mini batch = 10, num epochs=10

| Eta/$h_2$ | 10 | 25 | 50 |
|---|---|---|---|
| $\eta = 0.5$ | | | |
| $\eta = 0.25$ | | | |
| $\eta = 0.125$ | | | |

Table 3: Training ANNs with 2 Hidden Layers on MNIST; $h_1$=25; mini batch = 10, num epochs=10

| Eta/$h_2$ | 10 | 25 | 50 |
|---|---|---|---|
| $\eta = 0.5$ | | | |
| $\eta = 0.25$ | | | |
| $\eta = 0.125$ | | | |

Table 4: Training ANNs with 2 Hidden Layers on MNIST; $h_1$=50; mini batch = 10, num epochs=10

| Eta/$h_2$ | 10 | 25 | 50 |
|---|---|---|---|
| $\eta = 0.5$ | | | |
| $\eta = 0.25$ | | | |
| $\eta = 0.125$ | | | |

## What to Submit

1. `rumelhart_et_al.pdf` with your summary of the paper "Learning representations by back-propagating errors";
2. `ann_report.pdf` with your answers to Problem 3 (i.e., your observations and tables).
3. `cs5600_6600_f21_hw03.py` with your code;

Happy Reading, Writing, and, yes, Bulldozing!