ML Challenge 5

Thomas Britton, David Lawrence

June 2020

1 Introduction

Often complex problems have solutions which require a mixture of classification, regression, and prediction. For example, a self-driving vehicle must obey traffic signs (classification) and determine the level of torque to apply to the steering wheel, the amount of acceleration, and the amount of breaking that should be applied (regression). Additionally, decisions may be made on data gathered from a variety of sources, not just a single array of values or a single image. Thus, in order to properly utilize A.I. to solve more complex problems it is important to be able to synthesize data from different sources and different types with the aim of using a combination of, say, classification and regression to accomplish a task.

2 The Challenge

The challenge, is to develop an A.I. calculator that can perform 3 basic integer operations: addition, subtraction, and multiplication. The two integers will be single digit, non-negative, and come from the MNIST data set1. The operator will be given as a string; one from the set '+','-','*'.

To keep things more consistent with the various tutorials that can be found the MNIST the digits will be given in array form. So, as a truncated example if the array for a 9 were:
[0,0,1,1,1,0,0,1,0,1,0,0,1,1,1,0,0,0,0,1,0,0,0,0,1] and a 1 was [0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0] then the problem:
0,0,1,1,1,0,0,1,0,1,0,0,1,1,1,0,0,0,0,1,0,0,0,0,1,-,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0
would have as its answer 8. In actuality each problem line will contain 28x28 (784) integers (representing greyscale), a single character, and a second set of 784 integer values for a total of 1,569 entries per problem.

2.1 Requirements

Entrants will be required to produce a testing script which loads the model and performs the analysis necessary to produce the needed output. This script ***must be done in a Jupyter notebook*** compatible with/run on www.jupyterhub.jlab.org; specifically the AI notebook with slurm tools. The output will be a single column of values (the answers) with each row representing a single problem.

3 Judging

Submissions are **due August 5th at noon** and can be in the form of a **link to github containing the Jupyter notebook to be run** or **directly as a runnable ipynb file**. The judges will then run the testing script on a secret test file via the lab's jupyter hub page. The provided ipynb **MUST** be compatible with the **"ai-notebook (w/ slurm tools)"** notebook image. Entrants will have their sum of squared errors computed over all problems given. Explicitly:

$$\sum_i (Submission_i - Correct_i)^2$$

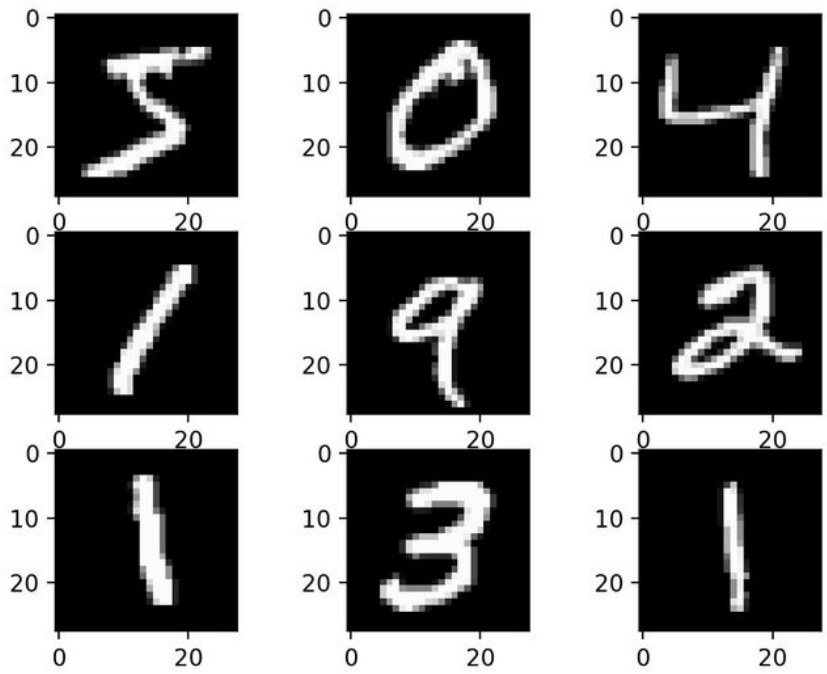

Figure 1: An example of some of the members of the MNIST data set

With the winner having the smallest value. In the case of a tie the winner would be the person who accomplished the task with the fewest number of parameters. Any submission which utilizes built in functions (or operators) to perform the arithmetic will be disqualified. All submissions must write exactly one output and must not use any mathematical operations after the model's output layer which precede the output for a problem. That is to say, a submission must not sum, average, multiply etc a model's output before the answer is determined. It is, however, permitted to do a non-computational interpretation e.g. looking up the label associated with a one-hot output.

4 Appendix

To aid in processing the data we are providing the following function which reads in the data from the zipped files. This is not required and there are many ways to do it.

```
def load_data():
    rootdir=""
    setname="train-images-idx3-ubyte.gz"
    labelname="train-labels-idx1-ubyte.gz"

    images = gzip.open(rootdir+setname, "rb")
    labels = gzip.open(rootdir+labelname, "rb")

    # Read the binary data
    # We have to get big endian unsigned int. So we need ">I"
```

```
# Get metadata for images
images.read(4) # skip the magic_number
number_of_images = images.read(4)
number_of_images = unpack(">I", number_of_images)[0]
rows = images.read(4)
rows = unpack(">I", rows)[0]
cols = images.read(4)
cols = unpack(">I", cols)[0]

# Get metadata for labels
labels.read(4) # skip the magic_number
N = labels.read(4)
N = unpack(">I", N)[0]

if number_of_images != N:
    raise Exception("number of labels did not match the number of images")

# Get the data
x = zeros((N, rows, cols), dtype=float32) # Initialize numpy array
y = zeros((N, 1), dtype=uint8) # Initialize numpy array
for i in range(N):
    if i % 1000 == 0:
        print("i: %i" % i)
    for row in range(rows):
        for col in range(cols):
            tmp_pixel = images.read(1) # Just a single byte
            tmp_pixel = unpack(">B", tmp_pixel)[0]
            x[i][row][col] = tmp_pixel
            tmp_label = labels.read(1)
            y[i] = unpack(">B", tmp_label)[0]

return (x, y)
```