CSC411: Project #2

Due on Sunday, February 18, 2018

Hao Zhang & Tyler Gamvrelis

February 7, 2018

Foreword

In this project, neural networks of various depth were used to recognize handwritten digits and faces.

Part 1 provides a description of the MNIST dataset from which the images of handwritten digits were taken. Part 2 implements the computation of a simple network. Part 3 presents the *sum of the negative log-probabilities* as a cost function, and derives the expression for its gradient with respect to one of its weights. Part 4 details the training and optimization procedures for a digit-recognition neural network, and part 5 improves upon the results by modifying gradient descent to use momentum. Learning curves are presents in parts 4 and 5. Part 6 presents an analysis of network behaviour with respect to two of its weights. Part 7 provides an analysis of the performance of two different backpropagation computation techniques. Part 8 presents a face recognition network architecture for classifying actors, and uses PyTorch for implementation. Part 9 presents visualizations of hidden unit weights relevant to two of the actors. Part 10 uses activations of AlexNet to train a neural network to perform classification of the actors.

System Details for Reproducibility:

- Python 2.7.14
- Libraries:
  - numpy
  - matplotlib
  - pylab
  - time
  - os
  - scipy
  - urllib
  - cPickle
  - PyTorch

Part 1

Dataset description
The MNIST dataset is made of thousands of 28 by 28 pixel images of the handwritten digits: 0 to 9. The images are split into training set and test set images labelled 'train0' to 'train9' and 'test0' to 'test9'. The number of images with each label is as follows:

Label	Number of Images	Label	Number of Images
train0	5923	test0	980
train1	6742	test1	1135
train2	5958	test2	1032
train3	6131	test3	1010
train4	5842	test4	982
train5	5421	test5	892
train6	5918	test6	958
train7	6265	test7	1028
train8	5851	test8	974
train9	5949	test9	1009

Ten images of each number were taken from the training sets and displayed in Figure **??**. The correct labels of most of the pictures can be discerned at a glance by humans However, since the digits are handwritten, some of them may not be completely obvious. For example, Figure **??** is categorized as a 9 but looks like an 8.

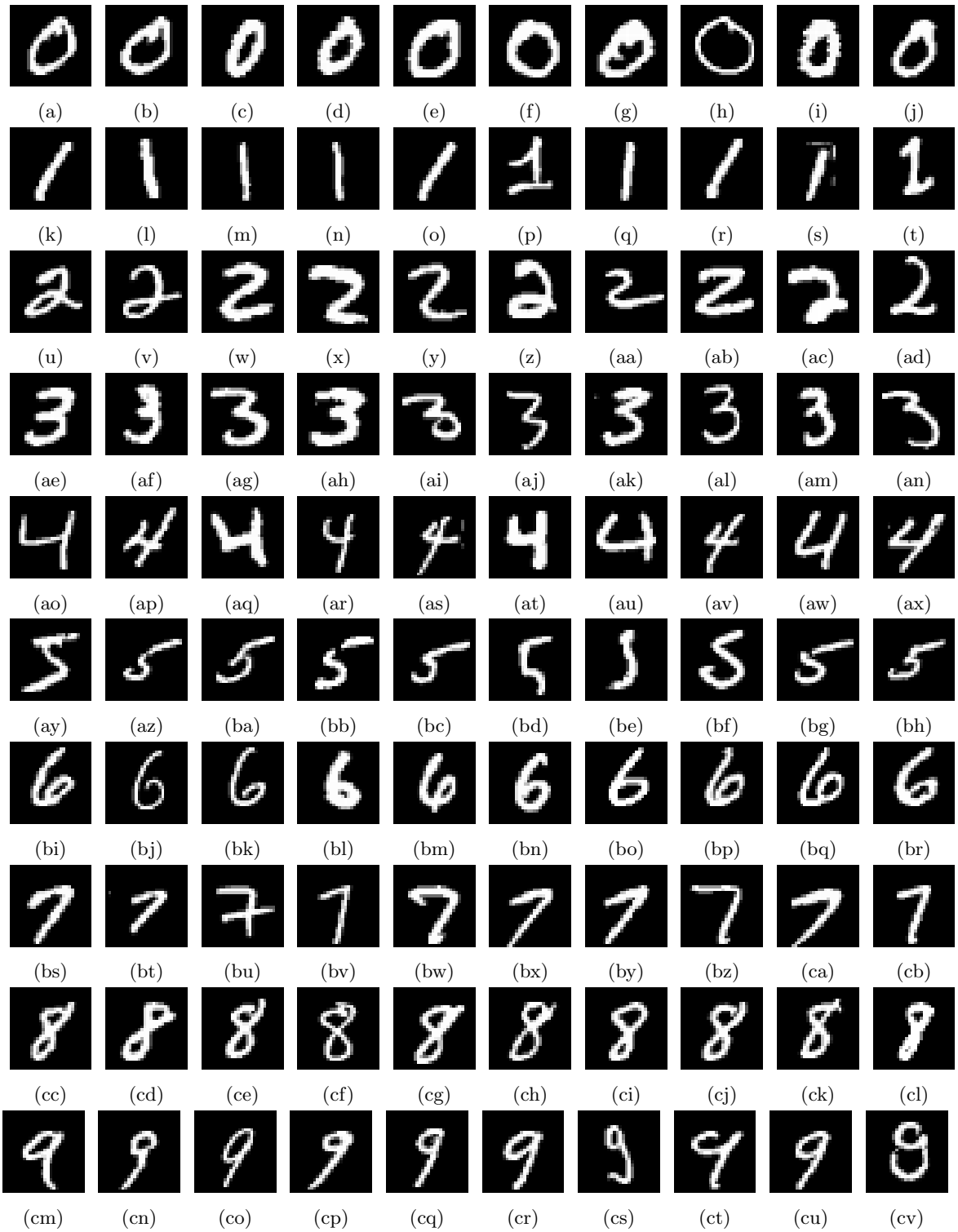

Figure 1: Subset of the MNIST dataset.

Part 2

Computing a simple network

In this part, the simple network depicted in Figure **??** was implemented as a function in Python using NumPy, the code listing for which is presented is Figure **??**.

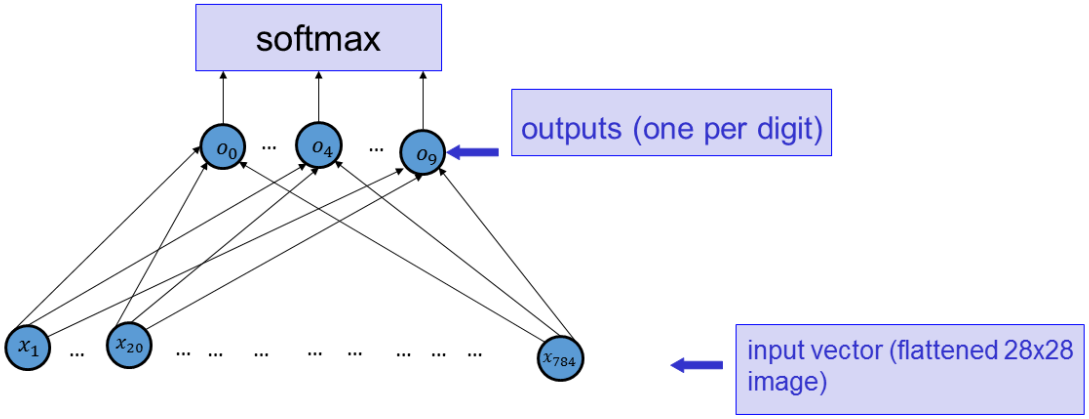

Figure 2: Simple network diagram from project handout.

```
def softmax(y):
'''
Return the output of the softmax function for the matrix of output y. y
is an NxM matrix where N is the number of outputs for a single case, and M
is the number of cases
'''
return exp(y)/tile(sum(exp(y),0), (len(y),1))

def Part2(theta, X):
'''
Part2 returns the vectorized multiplication of the (n x 10) parameter matrix
theta with the data X.

Arguments:
theta -- (n x 10) matrix of parameters (weights and biases)
x -- (n x 1) matrix whose rows correspond to pixels in images
'''

return softmax(np.dot(theta.T, X))
```

Figure 3: Python implementation of network using NumPy.

Part 3

Cost Function of Negative Log Probabilities
The Cost function that will be used for the network described in part 2 is:

$$C = -\sum_{q=1}^{m}(\sum_{l=1}^{k} y_l log(p_l))_q$$

Where k is the number of output nodes, m is the number of training samples, $p_l = \frac{e^{o_l}}{\sum_{c=1}^{k} e^{o_c}}$ and y_l is equal to 1 if the training sample is labeled as l and 0 otherwise.

Partial Derivative of the Cost Function (3a)
Let the matrix $W = \begin{bmatrix} w_{11} & w_{12} & w_{13} & \dots & w_{1k} \\ w_{21} & w_{22} & w_{23} & \dots & w_{2k} \\ \dots & \dots & \dots & \dots & \dots \\ w_{n1} & w_{n2} & w_{n3} & \dots & w_{nk} \end{bmatrix}$ and let W_j be the j^{th} column of matrix W. This results in $o_j = W_j^T x$ where $x = \begin{bmatrix} x_1 \\ x_2 \\ .. \\ x_n \end{bmatrix}$ and n is equal to the number of pixels with an extra 1 to multiply into the bias.

To calculate the partial derivative of C, we will consider only **a single training sample**: $C = -\sum_{l=1}^{k} y_l log(p_l)$
The partial derivative with respect to w_{ij} is:

$$\frac{\partial C(W)}{\partial w_{ij}} = -\sum_{l=1}^{k} \frac{\partial C}{\partial p_l}\frac{\partial p_l}{\partial o_j}\frac{\partial o_j}{\partial w_{ij}}$$

The first term is straight forward: $\frac{\partial C}{\partial p_l} = \frac{y_l}{p_l}$.

For the second term, we have $\frac{\partial p_l}{\partial o_j} = \begin{cases} \frac{-e^{o_l}e^{o_j}}{(\sum_{c=1}^{k} e^{o_c})^2} = -p_l p_j, \text{if } l \neq j \\ \frac{e^{o_j}}{\sum_{c=1}^{k} e^{o_c}} - \frac{e^{2o_j}}{(\sum_{c=1}^{k} e^{o_c})^2} = p_j - p_j^2, \text{if } l = j \end{cases}$
From the equation: $o_j = W_j^T x$, we have $o_j = w_{1j}x_1 + w_{2j}x_2 + \cdots + w_{nj}x_n$ and it is clear to see that: $\frac{\partial o_j}{\partial w_{ij}} = x_i$.

Putting it all together, we have:

$$\frac{\partial C(W)}{\partial w_{ij}} = -(\sum_{l\neq j}(-\frac{y_l}{p_l}p_l p_j) + \frac{y_j}{p_j}(p_j - p_j^2))x_i = (\sum_{l\neq j}(y_l p_j) + y_j(p_j - 1))x_i$$

Considering that we are using 1-hot encoding, only a single y_l can equal 1. Therefore,

$$\frac{\partial C(W)}{\partial w_{ij}} = \begin{cases} (p_j - 1)x_i, & \text{if } y_j = 1 \\ p_j x_i, & \text{otherwise} \end{cases}$$

Which is equivalent to saying: $\frac{\partial C(W)}{\partial w_{ij}} = (p_j - y_j)x_i$. To extend this to all training samples, we just have to sum up each individual contribution, resulting in the equation: $\frac{\partial C(W)}{\partial w_{ij}} = \sum_{q=1}^{m}((p_j - y_j)x_i)_q$.

Vectorized Implementation of Gradient (3b)

To vectorize the gradient, we first define the gradient matrix to be:

$$\frac{\partial C(W)}{\partial W} = \begin{bmatrix} \frac{\partial C}{w_{11}} & \dots & \dots & \dots & \frac{\partial C}{w_{1k}} \\ \frac{\partial C}{w_{21}} & \dots & \dots & \dots & \frac{\partial C}{w_{2k}} \\ \dots & \dots & \dots & \dots & \dots \\ \frac{\partial C}{w_{n1}} & \dots & \dots & \dots & \frac{\partial C}{w_{nk}} \end{bmatrix} = \begin{bmatrix} \sum_{q=1}^{m}((p_1 - y_1)x_1)_q & \dots & \dots & \dots & \sum_{q=1}^{m}((p_k - y_k)x_1)_q \\ \sum_{q=1}^{m}((p_1 - y_1)x_2)_q & \dots & \dots & \dots & \sum_{q=1}^{m}((p_k - y_k)x_2)_q \\ \dots & \dots & \dots & \dots & \dots \\ \sum_{q=1}^{m}((p_1 - y_1)x_n)_q & \dots & \dots & \dots & \sum_{q=1}^{m}((p_k - y_k)x_n)_q \end{bmatrix}$$

This is equivalent to: $X(P-Y)^T$ where:

$$X = \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & x_1^{(3)} & \dots & x_1^{(m)} \\ x_2^{(1)} & x_2^{(2)} & x_2^{(3)} & \dots & x_2^{(m)} \\ \dots & \dots & \dots & \dots & \dots \\ x_n^{(1)} & x_n^{(2)} & x_n^{(3)} & \dots & x_n^{(m)} \end{bmatrix} Y = \begin{bmatrix} y_1^{(1)} & y_1^{(2)} & y_1^{(3)} & \dots & y_1^{(m)} \\ y_2^{(1)} & y_2^{(2)} & y_2^{(3)} & \dots & y_2^{(m)} \\ \dots & \dots & \dots & \dots & \dots \\ y_k^{(1)} & y_k^{(2)} & y_k^{(3)} & \dots & y_k^{(m)} \end{bmatrix} P = \begin{bmatrix} p_1^{(1)} & p_1^{(2)} & p_1^{(3)} & \dots & p_1^{(m)} \\ p_2^{(1)} & p_2^{(2)} & p_2^{(3)} & \dots & p_2^{(m)} \\ \dots & \dots & \dots & \dots & \dots \\ p_k^{(1)} & p_k^{(2)} & p_k^{(3)} & \dots & p_k^{(m)} \end{bmatrix}$$

Where the superscript of each element represents the index of the training samples.

Part 4

Part 5

Part 6

Part 7

Part 8

Part 9

Part 10