CSC411: Project #2

Due on Sunday, February 18, 2018

Hao Zhang & Tyler Gamvrelis

February 5, 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:

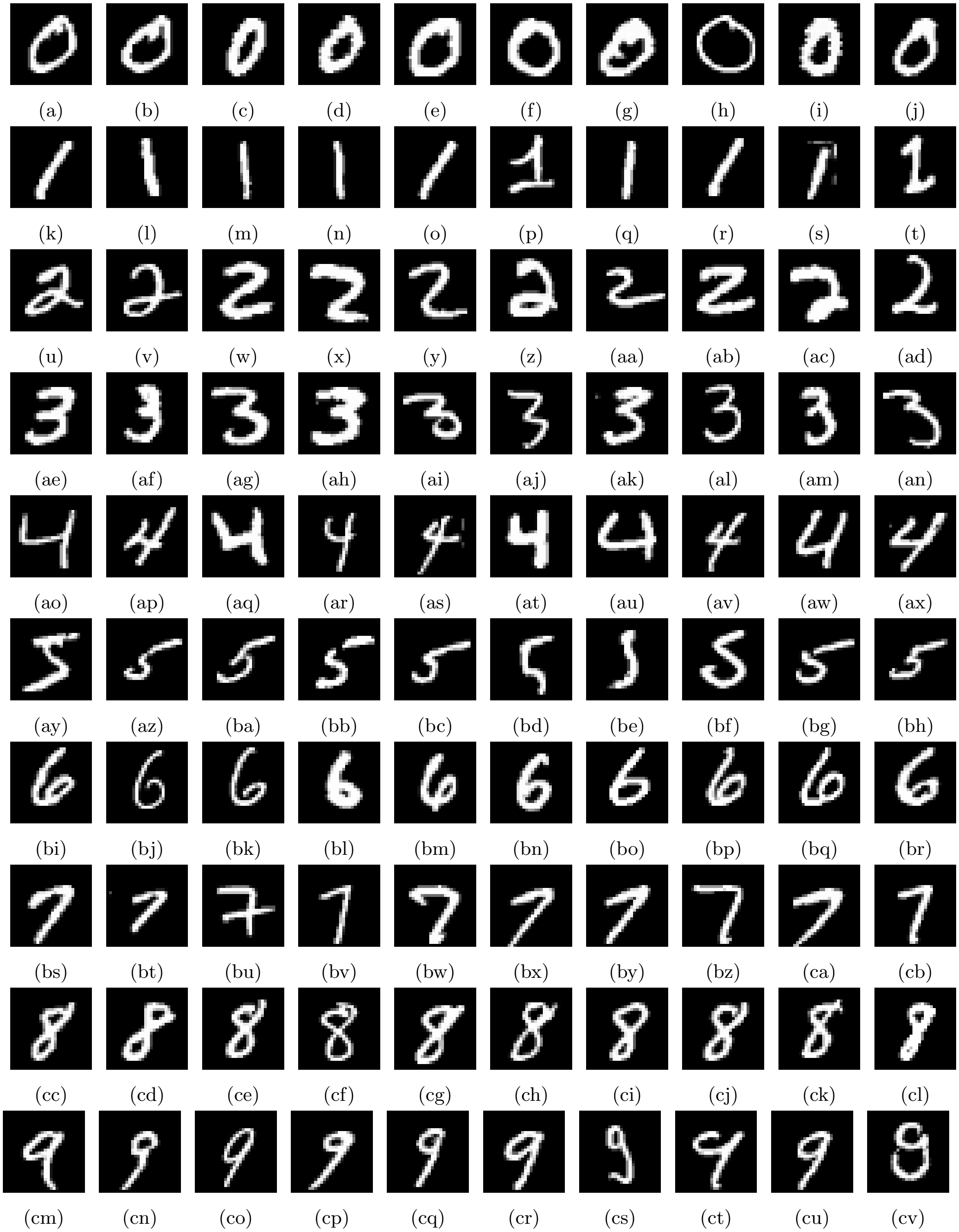
- Python 2.7.14
- Libraries:
  - numpy
  - matplotlib
  - pylab
  - time
  - os
  - scipy
  - urllib
  - cPickle
  - PyTorch

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	test0	1135
train2	5958	test0	1032
train3	6131	test0	1010
train4	5842	test0	982
train5	5421	test0	892
train6	5918	test0	958
train7	6265	test0	1028
train8	5851	test0	974
train9	5949	test0	1009

Ten images of each number were taken from the training sets and displayed in Figure 1. 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 1cv is categorized as a 9 but looks like an 8.



(a) (b) (c) (d) (e) (f) (g) (h) (i) (j)

(k) (l) (m) (n) (o) (p) (q) (r) (s) (t)

(u) (v) (w) (x) (y) (z) (aa) (ab) (ac) (ad)

(ae) (af) (ag) (ah) (ai) (aj) (ak) (al) (am) (an)

(ao) (ap) (aq) (ar) (as) (at) (au) (av) (aw) (ax)

(ay) (az) (ba) (bb) (bc) (bd) (be) (bf) (bg) (bh)

(bi) (bj) (bk) (bl) (bm) (bn) (bo) (bp) (bq) (br)

(bs) (bt) (bu) (bv) (bw) (bx) (by) (bz) (ca) (cb)

(cc) (cd) (ce) (cf) (cg) (ch) (ci) (cj) (ck) (cl)

(cm) (cn) (co) (cp) (cq) (cr) (cs) (ct) (cu) (cv)

Figure 1: Subset of the MNIST dataset.

Part 2

Computing a simple network

In this part, the simple network depicted in Figure 2 was implemented as a function in Python using NumPy, the code listing for which is presented is Figure 3.

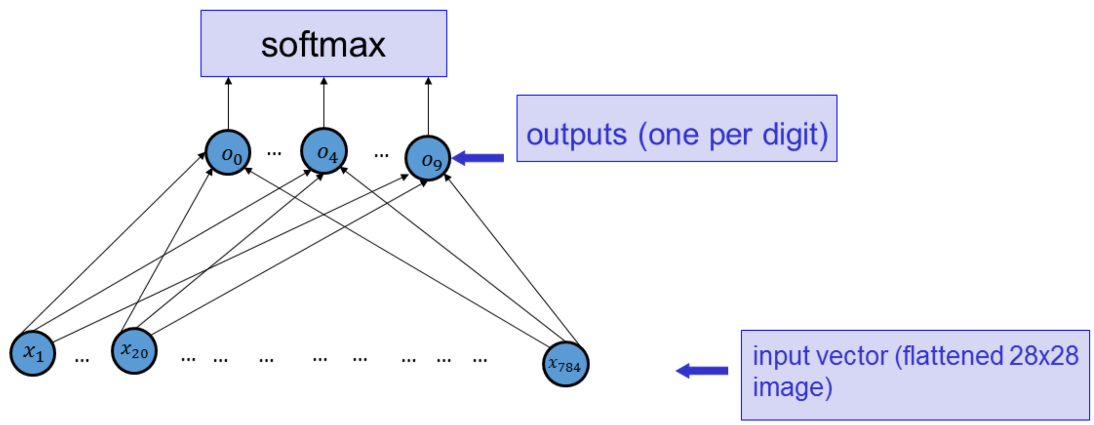


Figure 2: Simple network diagram from project handout.

```python
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

Part 4

Part 5

Part 6

Part 7

Part 8

Part 9

Part 10