Deep Learning

Microsoft

MNIST

- Data Downloading
- Data Manipulation

Objectives

- Understand the MNIST Dataset of handwritten digits
- Understand the process of downloading the data and split into training/testing sets
- Perform data manipulation tasks to evaluate the impact of accuracies

The Problem Domain

In this lab (part A), we will learn how to download and manipulate the well-known MNIST ("Modified National Institute of Standards and Technology") dataset for setting up a feed forward network to recognize handwritten digits. The setting up of the network and classification will be done in part B of the lab.

The official site of the dataset is at http://yann.lecun.com/exdb/mnist/. MNIST ("Modified National Institute of Standards and Technology") is most likely the de facto “hello world” dataset of computer vision. Since its release in 1999, this popular dataset of handwritten images has served as the basis for benchmarking classification algorithms related to vision. As new machine learning techniques emerge, MNIST remains a reliable resource for researchers and learners alike.

Jupyter Notebook: MNIST

In this lab, we will use the Jupyter notebook (Lab 3 Part A) for downloading the MNIST dataset and performing manipulations. Import all the relevant modules to be used for the lab:

```
from __future__ import print_function
import gzip
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
import numpy as np
import os
import shutil
import struct
import sys

try:
    from urllib.request import urlretrieve
except ImportError:
    from urllib import urlretrieve
```

Data Download

MNIST in CSV

For the curious attendees, below is a script to convert the dataset into a csv file. The format is label, pix-11, pix-12, pix-13, … where pix-ij is the pixel in the ith row and jth column. The convert(imgf, labelf, outf, n) function takes input dataset as the first argument, labels data as the second argument and number of examples is the third argument (training set of 60,000 examples, and a test set of 10,000 examples).

The example csv file is shown in Figure 1.

Figure 1: CSV file generated from MNIST Original Data

```
def convert(imgf, labelf, outf, n):
    f = open(imgf, "rb")
    o = open(outf, "w")
    l = open(labelf, "rb")

    f.read(16)
    l.read(8)
    images = []

    for i in range(n):
        image = [ord(l.read(1))]
        for j in range(28*28):
            image.append(ord(f.read(1)))
        images.append(image)

    for image in images:
        o.write(",".join(str(pix) for pix in image)+"\n")
    f.close()
    o.close()
    l.close()

convert("train-images-idx3-ubyte", "train-labels-idx1-ubyte",
        "mnist_train.csv", 60000)
convert("t10k-images-idx3-ubyte", "t10k-labels-idx1-ubyte",
        "mnist_test.csv", 10000)
```

Load MNIST Datasets – Helper Functions

We will download the data into our local machines. The MNIST database of standard handwritten digits consists of a training set of 60,000 images and a test set of 10,000 images with each image being 28 x 28 pixels. This set is easy to use visualize and train on any computer.

loadData(src, cimg): reads image data and formats into a 28 x 28 long array

loadLabels(src, cimg): loadLabels is very similar to loadData, and reads corresponding labels data

try_download(dataSrc, labelsSrc, cimg): try_download is used to download either training or testing data along with the corresponding labels. The two arrays are then combined using numpy's hstack. Hstack takes a sequence of arrays and stack them horizontally to make a single array.

```
def loadData(src, cimg):
    print ('Downloading ' + src)
    gzfname, h = urlretrieve(src, './delete.me')
    print ('Done.')
    try:
        with gzip.open(gzfname) as gz:
            n = struct.unpack('I', gz.read(4))
            # Read magic number.
            if n[0] != 0x3080000:
                raise Exception('Invalid file: unexpected magic number.')
            # Read number of entries.
            n = struct.unpack('>I', gz.read(4))[0]
            if n != cimg:
                raise Exception('Invalid file: expected {0} entries.'.format(cimg))
            crow = struct.unpack('>I', gz.read(4))[0]
            ccol = struct.unpack('>I', gz.read(4))[0]
            if crow != 28 or ccol != 28:
                raise Exception('Invalid file: expected 28 rows/cols per image.')
            # Read data.
            res = np.fromstring(gz.read(cimg * crow * ccol), dtype = np.uint8)
    finally:
        os.remove(gzfname)
    return res.reshape((cimg, crow * ccol))

def loadLabels(src, cimg):
    print ('Downloading ' + src)
    gzfname, h = urlretrieve(src, './delete.me')
    print ('Done.')
    try:
        with gzip.open(gzfname) as gz:
            n = struct.unpack('I', gz.read(4))
            # Read magic number.
            if n[0] != 0x1080000:
                raise Exception('Invalid file: unexpected magic number.')
            # Read number of entries.
            n = struct.unpack('>I', gz.read(4))
            if n[0] != cimg:
                raise Exception('Invalid file: expected {0} rows.'.format(cimg))
            # Read labels.
            res = np.fromstring(gz.read(cimg), dtype = np.uint8)
    finally:
        os.remove(gzfname)
    return res.reshape((cimg, 1))

def try_download(dataSrc, labelsSrc, cimg):
    data = loadData(dataSrc, cimg)
    labels = loadLabels(labelsSrc, cimg)
    return np.hstack((data, labels))
```

Data Download – Train & Test

The MNIST dataset is provided as train (60000 images) and test (10000 images) sets. Using try_download for each of the train and test datasets/labels, you can download the datasets.

```
# URLs for the train image and labels data

url_train_image = 'http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz'
url_train_labels = 'http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz'
num_train_samples = 60000

print("Downloading train data")
train = try_download(url_train_image, url_train_labels, num_train_samples)

url_test_image = 'http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz'
url_test_labels = 'http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz'
num_test_samples = 10000

print("Downloading test data")
test = try_download(url_test_image, url_test_labels, num_test_samples)
```

Visualization

Visualization can be performed using matplotlib, a plotting library for the Python programming language and its numerical mathematics extension NumPy. The below code can be used to plot a random image by reshaping the 1-D array.

```
# Plot a random image
sample_number = 5001
plt.imshow(train[sample_number,:-1].reshape(28,28), cmap="gray_r")
plt.axis('off')
print("Image Label: ", train[sample_number,-1])
```

CNTK Text Format

We can now save the images in a local directory. The data can be flattened to a vector of 28x28 image pixels and the labels are encoded using 1-hot encoding (label of 3 with 10 digits becomes 0010000000. The CNTK Text Format can be described as follows:

CNTK Text Format (CTF)

Each line in the input file contains one sample for one or more inputs. Since (explicitly or implicitly) every line is also attached to a sequence, it defines one or more **<sequence, input, sample>** relations. Each input line must be formatted as follows:

[Sequence_Id](Sample or Comment)+
where

Sample=|Input_Name (Value)*
Comment=|# some content

- Each line starts with a sequence id (a number and can be omitted) and contains one or more samples. If sequence id is omitted, line number will be used as the sequence id.
- A sample is effectively a key-value pair consisting of an input name and the corresponding value vector.
- Each sample begins with a pipe symbol (|) followed by the input name (no spaces), followed by a whitespace delimiter and then a list of values.
- Each value is either a number or an index-prefixed number for sparse inputs.
- Both tabs and spaces can be used interchangeably as delimiters.
- A comment starts with a pipe immediately followed by a hash symbol: |#, then followed by the actually content (body) of the comment. The body can contain any characters, however a pipe symbol inside the body needs to be escaped by appending the hash symbol to it (see the example below). The body of a comment continues until the end of line or the next un-escaped pipe, whichever comes first.

The below helper function savetxt(filename, ndarray) can be used for saving the data files into the CTF format.

```
def savetxt(filename, ndarray):
    dir = os.path.dirname(filename)
    if not os.path.exists(dir):
        os.makedirs(dir)

    if not os.path.isfile(filename):
        print("Saving", filename )
        with open(filename, 'w') as f:
            labels = list(map(' '.join, np.eye(10, dtype=np.uint).astype(str)))
            for row in ndarray:
```

```
            row_str = row.astype(str)
            label_str = labels[row[-1]]
            feature_str = ' '.join(row_str[:-1])
            f.write('|labels {} |features {}\n'.format(label_str, feature_str))
    else:
        print("File already exists", filename)

savetxt(r'data/MNIST/Train-28x28_cntk_text.txt', train)
savetxt(r'data/MNIST/Test-28x28_cntk_text.txt', test)
```

Data Manipulation

In this section, we'll look at ways to manipulate the data to improve the model. Some of the techniques involves shuffling the training data, distorting the images to improve generalization:

1. Shuffle Training Data

Shuffling the training data ensures that you do not obtain entire minibatches of highly correlated examples. If you have a large amount of training data and you use batches, the order of the training data might have a dramatic impact on the results. The below two lines would shuffle the training data using numpy's permutation function.

```
permute_indices = np.random.permutation(train.shape[0])
train = train[permute_indices]
```

Plot the same image used earlier with the permuted data and notice a new random digit image displayed.

```
# Plot a random image

sample_number = 5001
plt.imshow(train[sample_number,:-1].reshape(28,28), cmap="gray_r")
plt.axis('off')
print("Image Label: ", train[sample_number,-1])
```

2. Image Distortions - Translation

For building robust image classification, the training data should include images that are distorted (rotation/translation) and/or contain additive noise. Numpy provides various

techniques to rotate 2D arrays. An easy way of doing this is using numpy.rot90(*m*, *k=1*, *axes=(0*, *1)*). By passing k (second parameter), we can rotate an array *m* by *k* * 90 degrees.

The below lines of code rotate 5% of the training data randomly by 90 degrees and another 5% by 270 degrees. You can do the same with testing data to evaluate the performance of classification on rotated images.

```
train_elements = train.shape[0]
permute_indices = np.random.permutation(train.shape[0])
percentage_rotate = 0.10
permute_indices[0:int(percentage_rotate * train_elements)]

for permute_index in permute_indices[0:int(percentage_rotate * train_elements)]:
    train[permute_index,:-1] = np.rot90(train[permute_index,:-
1].reshape(28,28)).reshape(28*28)

percentage_rotate = 0.05
for permute_index in permute_indices[0:int(percentage_rotate * train_elements)]:
    train[permute_index,:-1] = np.rot90(train[permute_index,:-
1].reshape(28,28),3).reshape(28*28)
```

3. Image Distortions - Smoothing

Image filtering methods, including the popular Gaussian filtering, are used to reduce background noise and improve signal-to-noise ratio of the image with better contrast. However, Gaussian filtering also generates image distortion, considering the size of filter as shown below. The two lines of code below can be used to produce image distortions as shown in Figure 2.

```
from scipy.ndimage import gaussian_filter
plt.imshow(gaussian_filter(train[sample_number,:-1].reshape(28,28),2), cmap="gray_r")
```

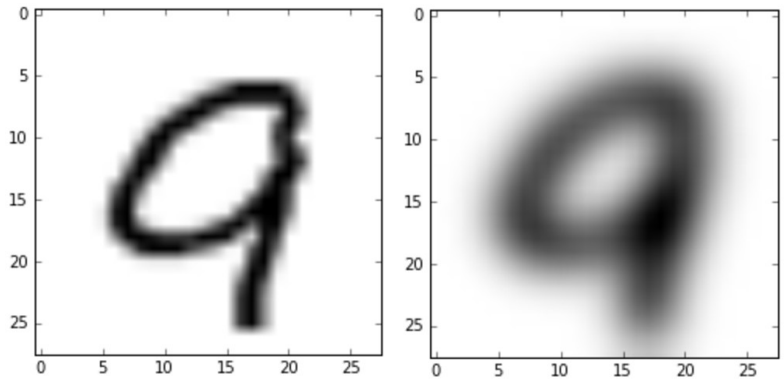

Figure 2: a) Original Image representing 9. b) Smoothed image.