Project 2: Revenge of MNIST

Mid Progress Check – Sept. 28, 2017

Final Due Date – Oct. 5, 2017

Note on collaboration:

Collaboration's great! You may look at others' code, talk to them about the project, etc. as much as you like – but - you must write your own code and report, and be prepared to explain what it does.

Introduction:

We've seen 2 network architectures in class for classifying MNIST:

- the beginning example in Tensorflow, with 92% accuracy, NETWORK A,
- and the more complex network w/ 2 hidden layers, with 98% accuracy, NETWORK B.

The code for each of these is in the Project 2 directory on Canvas, in files P2_A_Canvas.py and P2_B_Canvas.py. These are using written in Python using TensorFlow, which you should have installed on your machine in order to complete Project 1.

NOTE: I had difficulty accessing the weights in the earlier files I uploaded, so I reworked the code to make that easier.

This project will attempt to give us more insight into what the nets are doing, and how robust they are to changes that would be encountered in real-world computer vision applications.

Part A) Weight Visualization

Both networks take in MNIST data which has been reshaped from 28x28 images to 784(x1) vectors. To get started, create a figure (or figures) displaying 1 image of each digit, 0-9.

Then, after training NETWORK A, so that we have 784 weights for each node (ignoring the bias), reshape the weights into images, and show each of these for the 10 output nodes (so we should get out 10 images). These should resemble averaged versions of the MNIST digit images.

NETWORK B has 256 nodes in the first hidden layer. Perform the same process of reshaping and visualizing the weights for each of the first 10 nodes (so we should get out 10 more images). These images will not look the same as those from the NETWORK A weights.

Part B) Investigating Rotation

A problem that's frequently encountered in computer vision applications is the rotation due to unknown camera angles and orientation of objects in images. If, for example, the camera is held at 45 degrees, or we're trying to recognize a can of soda laying on its side, we need algorithms that are rotationally invariant.

In order to test the robustness of the networks to rotation, we'll need to generate a new training and validation set by changing the input images. Specifically, generate a new set of images by rotating each image in the current set of MNIST images by a random angle ranging from 0 to 360 degrees. The image size will stay the same.

NB: scipy already has a rotation function for images, so you don't need to write your own.

After producing the new training set, use it to train both NETWORK A and NETWORK B, and report the final accuracies. Also generate confusion matrices for both.

Part C) Investigating Scaling

Another problem that's frequently encountered in computer vision applications is size variation due to differences in distance from the camera to a given object. (If the soda can's further away, it takes up fewer pixels in the image.) Ideally, computer vison algorithms are also scale invariant.

In order to test the robustness of the networks to scale, we'll need to generate a new training and validation set by scaling the input images. Specifically, generate a new set of images by scaling each image in the current set of MNIST images by a random factor ranging from [0.5 to 1]. The image size will stay the same.

NB: scipy already has a zoom function as well.

After producing the new training set, use it to train both NETWORK A and NETWORK B, and report the final accuracies. Also generate confusion matrices for both.

Produce a Final Report, including your code, that discusses your implementation and observations.

NB: If you're using Spyder as an IDE for Python/TensorFlow, you may want to change the settings to Spyder to plot in independent windows. You can do so using:

Tools > Preferences > Ipython Console > Graphics > Graphics Backend > Backend: "automatic"

(you may need to restart Spyder after making the change)