

The Short document

All About My Neural Network



December 2, 2013

Franklin Leung

COSC6313 Fall 2013

Table of Contents

[Motivations 2](#_Toc373738828)

[Challenges 2](#_Toc373738829)

[Strategies 2](#_Toc373738830)

[Problems 3](#_Toc373738831)

[Concluding Thoughts 4](#_Toc373738832)

[Links to Third-party Libraries 4](#_Toc373738833)

# Motivations

Since the essential goal was to complete option 2 (feed a computer generated letter to be recognized by a neural network), I decided to pick one of the earlier neural networks we covered in class that dealt specifically with letter recognition – the hetero-associative neural network.

One of the reasons for doing so was because it was relatively simple, especially compared to some of the latter, more complex networks, and as I’ve never programmed a neural network before, I thought it would be better for me to have a simpler neural network (and inversely, less powerful) to serve as a proof-of-concept that I could later improve to the more complex networks.

# Challenges

The program is coded in Python, a language I have never used in the past. Python has been incredibly popular in recent years due to its relative simplicity (declaring variable types is not necessary, the compiler will figure it out for you) and relaxed syntax rules (there are simpler ways of doing lots of things when compared to C/Java).

I felt like I had sufficient basic knowledge of the “classic” programming languages of C/Java and other C-like derivatives (e.g. C#) to quickly pick up in Python, and I was right – defining functions and handling variables only required minor documentation lookups for new syntactical rules. The way for loops work feels much more straightforward than in other languages, and Python had support of some powerful third-party libraries that made my life significantly easier.

A notable disadvantage to Python seems to be the weak bundled IDE (called IDLE) that I use for Python editing. While it works fine for lexical and syntax checking, semantic checking is weak and there aren’t any options for debugging or even line numbers. There also does not seem to be any document handler for managing multiple Python files in a single project.

# Strategies

One of the reasons that I stuck with Python was the availability of third-party libraries. Specifically, I used Python Imaging Library and NumPy, used for image processing/manipulation and scientific computing respectively.

Using Python Imaging Library allowed me a very easy way to view pixel values of any image and it had built-in functions to resize images as necessary, and gave me options for converting images from color/RGB to monochrome/black and white in case the source image was not already in the correct format.

NumPy was essential in constructing the neural network because of the ease of matrix multiplication – I could use my matrix objects in exactly the same syntax as typical math (e.g. *print a\*b*, which would print to console the product of those two matrices). The transpose for matrices was easy to access (e.g. *x.T*, with *x* being the matrix) and printing matrices was automatically properly formatted for easy viewing. NumPy itself is an incredibly useful tool for all sorts of scientific computing, and I only needed to use it for a little bit of matrix manipulation.

Given the popularity of Python, user-generated questions and answers relating to various problems I had were easy to find (especially on stackoverflow.com) and this allowed me to quickly move from one problem to the next. It made me appreciate all that Python can do, while showing me that there are so many better ways to do what I am already doing in a better, more efficient manner with certain styles/ideals that are associated with Python (e.g. index counters that are meant for counting indices in a loop are considered “non-Pythonic”).

# Problems

As with any new project, there were (and still are) a number of problems that I have not resolved in a satisfactory manner, either because I ran out of time or I have not been able to come up with a solution to the problem.

Python works extremely well as something like a script, where the goal isn’t necessarily to have a complex and fancy program and more as something that can be run from the command line (not that it can’t be one, though). The GUI is an entirely different beast from the usual Python, and it is one that I have not been able to firmly grasp. I have seen beautiful interfaces both as a standalone application and as a web application built solely from Python – I don’t doubt its abilities, only mine.

I have also struggled to come up with a good way to implement the neural network when the data is not all set/hard-coded beforehand. For example, if I want to store and recall 10 different letters of random choosing, I only know to hardcode those beforehand. I’d really like a way to be able to dynamically create an output pair from the input pair that will still check to be different enough from other existing pairs in order to minimize crosstalk.

Speaking of which, I noticed that trying to recall the first seven letters of the alphabet (A through G) resulted in many incorrect answers, which I chalked up to crosstalk. This seems reasonable because my output pair consisted only of a 3x5 matrix, and with so few elements, B would look like D, C would look like D, E would look like F, and G would look like C. When I reduced it to only a few different-looking letters (I used A, T, and X), it was much easier to get the correct responses. I implemented a function to randomly introduce noise to the input pair and was happy to see the neural network work as well as it does (50% noise was still manageable).

I attempted to come up with my own normalization algorithm to essentially shrink images while keeping the general shape of the letter, but I found the problem to be too difficult for me to complete in time (“Should I compare only left/right neighbors or also in rows above/below the pixel I’m looking at? At what threshold should it be set it if a pixel is black versus if it’s white?). I elected to use the resize feature built into PIL (Python Imaging Library) which gave decent results, but certainly not as good as a handcrafted, resized image.

My code is still messy from having all my code stuffed into a single document, and I really should have made a separate class for the neural network itself with its own store and recall object methods. I’ve done my best to comment various sections to help other readers understand what I did do.

# Concluding Thoughts

Overall, I feel pretty good about the success of the hetero-associative neural network. I know that it’s one of the weaker pattern association neural networks that we covered in class, but it still works well enough with small enough sets of data to show how useful neural networks can be.

Of course I’d like to have a well-functioning GUI to complement the visual nature of character recognition, but at least seeing it work with some basic *print* commands shows it working as expected.

It also seems like the differences of implementing options 1 and 2 are very minor, in that the image is stored and recalled in exactly the same way - the only modification is how the file is generated in the first place, either by manual input from the user or from a character typed by some computer program.

If I were to continue improving this on my free time, I think I would try to figure out how to store all 52 characters (upper and lower cases) into a neural network with good recall. After that, I would be interested to see how voice recognition would be implemented and to see if I could re-use all or most of the code and/or concepts used for character recognition.

# Links to Third-party Libraries

Python Imaging Library  
<http://www.pythonware.com/products/pil/>

NumPy  
<http://www.numpy.org/>