Tingyu Thomas Lin, Juan M. Navarro
CS 263B – Natural Language Processing, Prof. Michael Dyer
Project Proposal
Tuesday, April 28th, 2009

A Perceptron-Based Approach for Image Processing

I Purpose and Overview

Our project will attempt to associate an image to a concrete noun. For example, when we feed the word "tree" into our system, we want the system to generate an image of a tree. Furthermore, we will attempt to train the system to recognize relative positions (e.g. "to the right of", "to the left of", "above", etc.), so the system should generate the appropriate image if we feed as input "tree to the right of house".

Ultimately, the goal is to have any arbitrary "<NOUN> <RELATIVE POSITION> <NOUN>" output the correct image. Of course, the image for the noun has to already been learned by the first system, but this should work even if that specific combination hasn't explicitly trained in the second system.

We will use multilayer perceptrons, as they are more powerful than the simple perceptron. It isn't entirely clear if our application is or isn't linearly separable, so we'll err on the side of the more capable network. One multilayer perceptron with three layers (input, output, and a hidden layer) will be used to build the association between the concrete noun and its image. Then, this multilayer perceptron will be used as a sub-component, in conjunction with another multilayer perceptron, to a second system to handle the relative positions between two concrete nouns. Input to both systems will be encoded as ASCII text vectors, and the output will be grayscale intensity values (a real value between 0 and 1) for each pixel of the image.

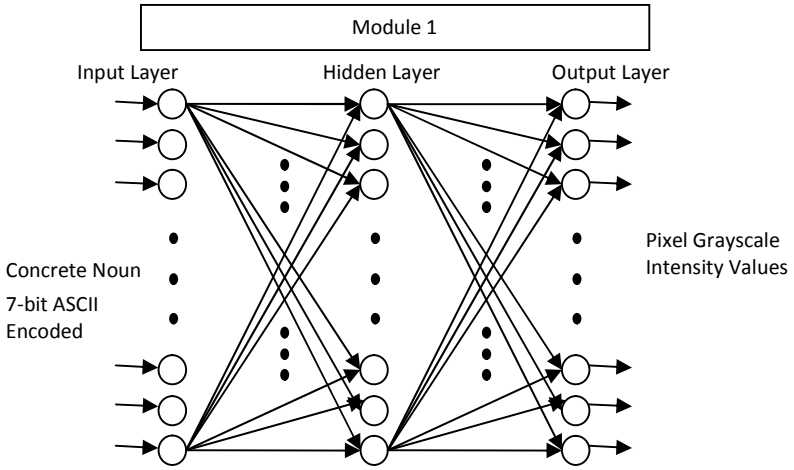

Figure 1

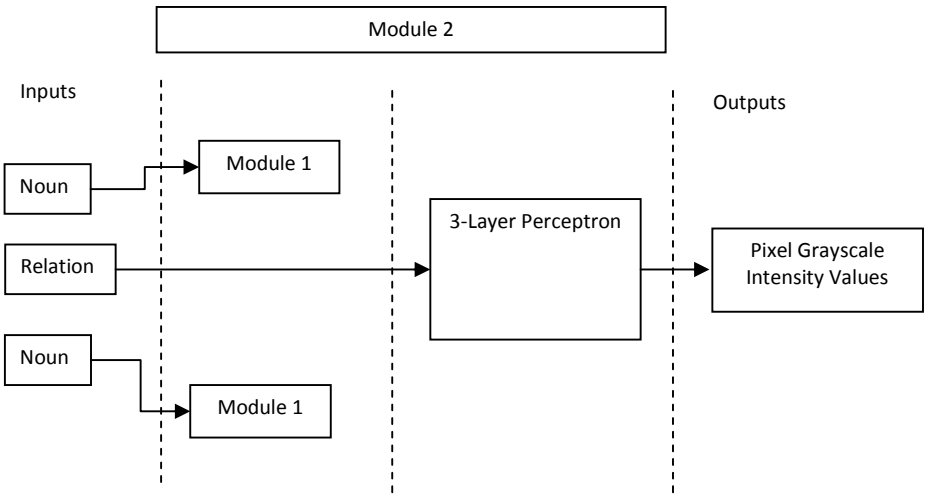

Figure 2

II Description of Neural Network Architecture

We use a two-module approach. The first module is responsible for learning noun-image associations. It is a multilayer perceptron with three layers: an input, output, and hidden layer (see Figure 1). It will take in 7-bit ASCII text vectors as input:

1. tree
2. house
3. car, etc.

There will be 7 neuron inputs for each letter, each neuron for a bit, and the number of letters in a word will be limited so that the number of input neurons is finite. The output neurons will have real values of 0 to 1, indicating grayscale intensity, and each neuron will correspond to a particular pixel in an image. The image size will be fixed to some MxN pixel size that will be determined later.

The second module will be similar to the first module, except there will be enough input neurons to accept three 7-bit ASCII text vectors and that the first module, with weights locked, will be used as components (see Figure 2). The three text vectors form a sentence of form "<NOUN> <RELATION> <NOUN>" where the noun is one of those trained in module 1 and the relation is in the set:

{right-of, left-of, above, below, front-of, behind}

The output of the second module, like the first module, will be a set of output neurons with values for grayscale intensity on the range of 0 to 1, and each neuron corresponds with a pixel of the output image.

III Multilayer Percepetron – Output and Learning

The structure of the 3-layer perceptron will be fairly standard. For module 1, if X_i as the ith input neuron's output (that is, the value that's fed to that particular neuron) then the expression for H_j, the jth hidden layer neuron's output, is:

$$H_j = \sum_i \sigma(w_{i,j} X_i)$$

where $w_{i,j}$ is the weight on the edge from input neuron X_i to hidden layer neuron H_j, and $\sigma(t)$ is the sigmoid function:

$$\sigma(t) = \frac{1}{1+e^{-t}}$$

Similarly, the expression for O_k, the value of the kth neuron, is:

$$O_k = \sum_j \sigma(\bar{w}_{j,k} H_j)$$

where if $\bar{w}_{j,k}$ is the weight on the edge from hidden layer neuron H_j to output neuron O_k. As for updating the weights, if Y_k is the correct output that O_k should ideally be, then the adjustments to weights $w_{i,j}$ and $\bar{w}_{j,k}$ are:

$$\Delta\bar{w}_{j,k} = \alpha * \delta O_k * H_j$$

$$\Delta w_{i,j} = \alpha * \delta H_j * X_i$$

where:

$$\delta O_k = O_k(1 - O_k)(Y_k - O_k)$$

$$\delta H_j = H_j(1 - H_j)\sum_k \Delta\bar{w}_{j,k}\delta O_k$$

We run this weight update step across all training data until the global error ε is below a threshold, where:

$$\varepsilon = \frac{1}{2}\sum_k {\delta O_k}^2$$

IV Sources of Training Data

All of our training data will be generated by us. The concrete nouns, encoded in ASCII, will be chosen by us from a list of words, and we will create the appropriate images corresponding images. We will create "ideal" output images for each module, so that we may compare the output results of each module and measure noise quantitatively.

V Implementation

Currently, we are considering using MATLAB to implement this system. If we find MATLAB to be inadequate for our purposes, we may implement this system using C++.

As for performance, we have little insight to the performance of this approach, as we have no experience with making neural networks. However, we can expect this to be slow, as with a three layer neural net, there is n*(m+p) weights to update, where m is the number of inputs, n the number of hidden neurons, and p the number of outputs.

VI Stages of Project

Our project consists of two main stages or modules:

- **Module 1:** This systems involves using a perceptron to generate the image of an object given an ASCII text input string.
- **Module 2:** This module will build on the functionality provided by Module 1. This module will take in two ASCII text input vectors, specifying the objects to be displayed in the output image. A third ASCII text input vector will specify the relative position of the objects in the output image.