pdfpagewidth 8.5in pdfpageheight 11in

Intelligent Agents in a Probabilitsic Game : Proximity

Matthias Denu and **Donald Hamnett**
Northeastern University
College of Computer and Information Science
440 Huntington Ave #202, Boston, MA 02115

Abstract

The aim of this project was to implement intelligent agents to defeat a greedy agent in the game of Proximity.

Introduction

Background

Artificial Neural Networks have their origins in 1943, with the McCollough-Pitts Threshold Logic Unit (TLU), a mathematical interpretation how formal logic is carried out in the human brain.The TLU representation of a neuron, is as a unit that takes in a set of inputs and calculates a weighted sum. This weighted sum would then be processed as:

$$f(\mathbf{x}) = \begin{cases} 0, & \big(\sum_i w_i x_i\big) + b < \text{threshhold} \\ 1, & \text{Otherwise} \end{cases}$$

Figure 1: TLU Activation

Any discrete binary task could be represented by these units, but their usefulness was simply as a model of the brain and offered no deductive improvements upon formal logic.

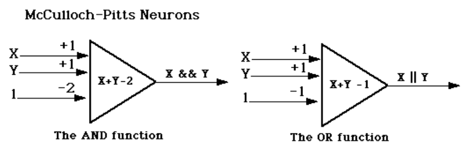

Figure 2: The McCollough-Pitts Neuron

In 1958, the precursor of the modern-day neural network was born with Rosenblatt's development of the "perceptron." Expanding on the MuCollough-Pitts representation of a neuron, his perceptron was a unit that similarly took in a set of inputs and calculated a weighted sum, activating with the sign function.

$$sgn(\mathbf{x}) = \begin{cases} -1, & \big(\sum_i w_i x_i\big) + b < 0 \\ +1, & \text{Otherwise} \end{cases}$$

Figure 3: The Sign Function

The key insight of Rosenblatt was that opposed to weights being fixed, they were adjustable parameters with positive and negative weights, and could hence be learned.

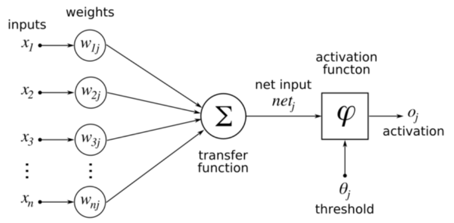

Figure 4: The Rosenblatt Perceptron

The full history of perceptrons is outside the scope of this report, but after a period of falling out of fashion, the perceptron has made a massive resurgence in recent years due to the development of the ANN, where several perceptrons are combined in parallel and/or in layers to compute complex predictive models that can be applied to non-linearly separable decision boundaries. While the ANN was first developed in the 1960s, notably ADALINE and MADELINE, it was not until the recent increases in both computational power and available data that their widespread use was adopted. Since then, these networks have had success in a vast collection of problems, ranging from natural language processing with recurrent neural networks, to computer vision with convolutional neural networks (CNNs).

A convolutional neural network is a popular tool used mainly in image classification and computer vision, but has been applied in recent years to a growing number of use cases. The basis of the CNN, and what makes it well suited to computer vision, is the convolutional layers. A convolution in this sense is a kernel with trainable weights which has the same depth of the image being convolved (i.e. and RGB image would have a depth of 3). However, the other dimensions of this kernel, or "filter," are usually smaller than on the other axes, which allows it to stride across windows of the image. The output of a single stride is a scalar, the sum of the element-wise product of the kernel with the input image. Each of these scalars is placed in a new grid, where spacial relativity of the original image and the kernel outputs are preserved. By using several filters and layers of convolutions (and a lot of training data), individual kernels can begin to recognize features such as edges and foreground, which when combined can lead to meaningful pattern recognition of the images.

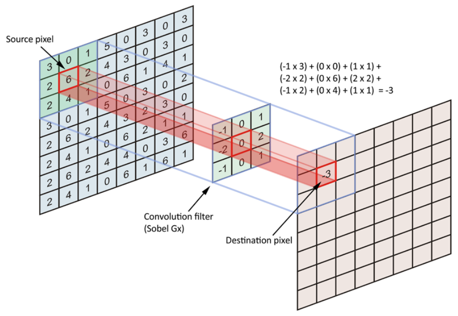
Source pixel
(-1 x 3) + (0 x 0) + (1 x 1) +
(-2 x 2) + (0 x 6) + (2 x 2) +
(-1 x 2) + (0 x 4) + (1 x 1) = -3
Convolution filter
(Sobel Gx)
Destination pixel

Figure 5: A convolution (Pjreddie)

Project Description

Using the Keras deep learning library in Python with its functional API made it incredibly easy to build an ANN. Below is the pseudocode used to build a fully-connected feed-forward network.

```
1: procedure GETNETWORK(NumberOfHiddenLayers)
2:     InputLayer ← $Input$(NumberOfFeatures)
3:     x ← InputLayer
4:     for $i = 1$ to NumberOfHiddenLayers do
5:         x ← $HiddenLayer$(NumberOfFeatures)(x)
6:     OutputLayer ← $Layer(1)$(x)
7:     ANN← Model(InputLayer, OutputLayer)
8:     Compile(ANN)
9: return ANN
```

In all networks, I used the rectified linear unit (ReLU) for the hidden layer activations, and the logistic sigmoid with a threshold of 0.5 for the output activation.

$$sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

$$ReLU(x) = \max(0, x) \tag{2}$$

Figure 6: The sigmoid and ReLU functions

Below are the relevant parameters of the networks:

Parameters

- Optimizer : Adam (Kingma and Ba 2014)
  - Learning Rate : 0.001
  - β_1 : 0.9
  - β_2 : 0.999
- Normalization : Batch Normalization on the Input Layer
- Dropout : Rate of 0.5 on all Hidden Layers
- Batch Size : 32
- Epochs : Range $[10, 1000]$

References

Kingma, D. P., and Ba, J. 2014. Adam: A method for stochastic optimization. *CoRR* abs/1412.6980.

Pjreddie. pjreddie/cnn-primer.