Neural Networks with Python and TensorFlow

Manchester Metropolitan University
Student: Alexander J. Johnson   Supervisor: Dr Stephen Lynch

Brief History

The design of artificial neural networks were initially inspired by biological neurons. The first mathematical models of neurons was created by McCulloch and Pitts, 1943, using a simple activation rule: if at least one excitatory connection is active and all inhibitory connections are inactive, the cell will be active.

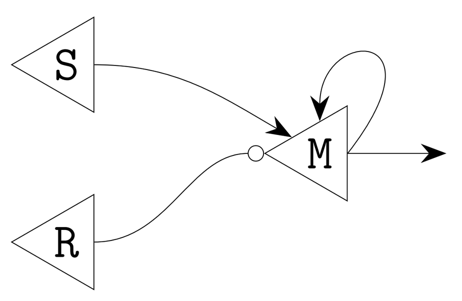

Figure 1: Example SR Flip-Flop using McCulloch's neurons.

Later, a concept called the perceptron was built by Rosenblatt, 1958. The perceptron consisted of a number of photovoltaic cells, which connected to a number of association cells, which then connected to a number of response cells.
Each photovoltaic cell was connected to every association cell, and each association cell was connected to every response cell, this kind of connectivity is referred to a being densely connected. The output of each association cell was a boolean value based on the weighted sum of the input values, where the weights are automatically adjusted by the perceptron.

Modern Neural Networks

The most common architecture for artificial neurons was outlined by McClelland, Rumelhart, and Group, 1986, where the activation is given by

$$y_i = \phi\left(b_i + \sum_j w_{i,j} x_j\right),$$

where x_j is the j^{th} input, $w_{i,j}$ is the connection weight from j to i, b_i is the input bias, and ϕ is some activation function.
Learning is performed using a technique called backpropagation, which applies the chain rule to differentiate an value function with respect to each weight.

Curve Fitting

An implementation of a neural network was written in python to predict the values of houses within Boston, based on three attributes: number of rooms, highway accessibility, and percentage of lower status population. The network consisted of three input neurons and one output neuron, using tanh as the activation function.

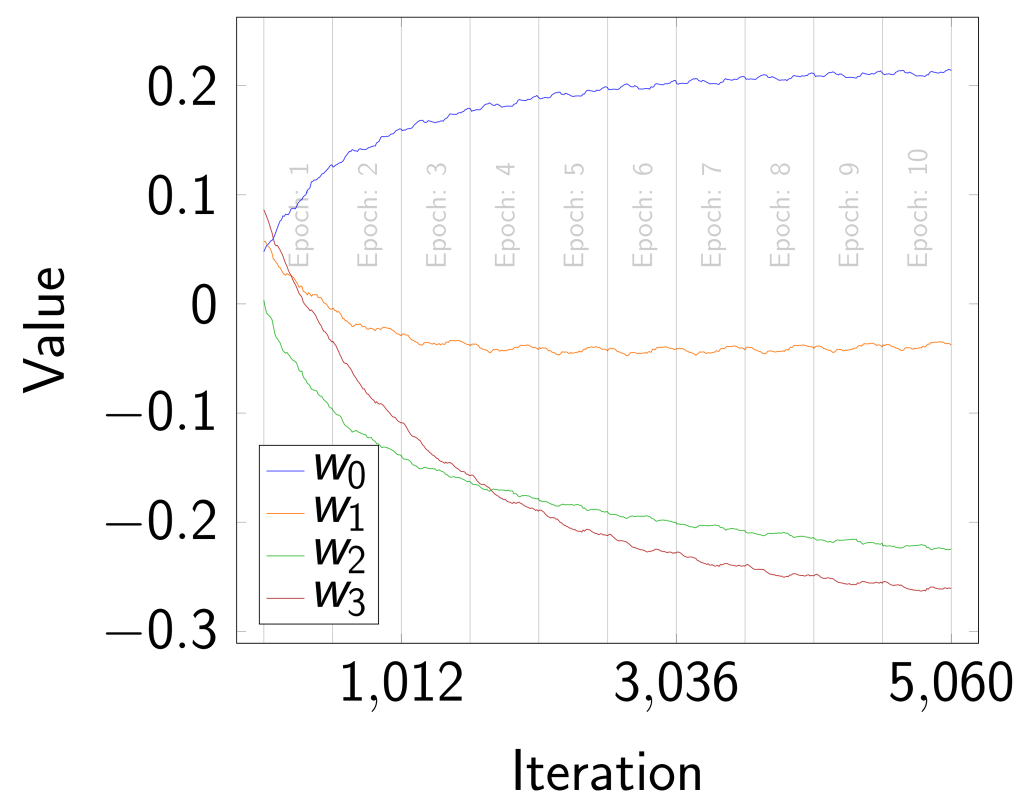

Figure 2: Network weights against iteration number.

Image Recognition

Creating neural networks in python can be repetitive without the use of a library. TensorFlow provides an easy-to-use framework for creating neural networks in python which includes a variety of options for layers, activation functions, and optimisers.
For image recognition, it is advantageous to use convolutional layers, in which each neuron is connected to a small region of the previous layer, and all the neurons share the same weight scheme. A convolutional neural network was in python using TensorFlow to perform character recognition on the MNIST dataset, which contains a large number of handwritten numbers. The network also contained "dropout" layers, which prevent the network from over fitting.

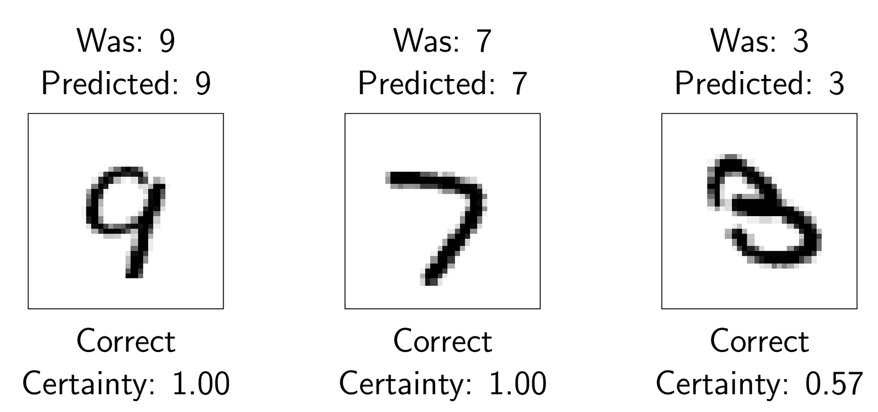

Figure 3: Network predictions for MNIST dataset.

Deep Q-Learning

Neural networks can also be used to interact with systems. The basis for the method is an algorithm called Q-Learning (Watkins, 1989), which uses the Bellman equation,

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a)),$$

where $Q(s,a)$ is the estimated future reward of taking action a from state s, s' is the state after the action, r is the reward of taking the action, γ is the discount rate, and α is the learning rate. These Q-values are stored in a table.
Once trained, the action with the highest Q-value from a given state is the optimal choice.
The Q-table can be replaced with a neural network, which is trained to match its output with the Bellman equation.

Policy Gradient

For many problems, a deterministic policy is not optimal, and a stochastic policy is required. Sutton et al., 2000 introduced an algorithm for learning probabilities, which defined the policy gradient as

$$\frac{\partial \rho}{\partial \theta} = \sum_s d(s) \sum_a \frac{\partial \pi(s,a)}{\partial \theta} Q(s,a),$$

where ρ is the expect reward, π is the policy, θ is the parameter vector of π, $d(s)$ is the state distribution, and $Q(s,a)$ is the state-action pair value.
Policy gradients and Q-Learning can be combined to form Actor-Critic methods, which are the current state-of-the-art for reinforcement learning.

Bibliography

McClelland, James L, David E Rumelhart, PDP Research Group, et al. (1986). "Parallel distributed processing". In: *Explorations in the Microstructure of Cognition* 2, pp. 216–271.
McCulloch, Warren S and Walter Pitts (1943). "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4, pp. 115–133. DOI: 10.1007/BF02478259.
Rosenblatt, Frank (1958). "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6, p. 386. DOI: 10.1037/h0042519.
Sutton, Richard S et al. (2000). "Policy gradient methods for reinforcement learning with function approximation". In: *Advances in neural information processing systems*, pp. 1057–1063.
Watkins, Christopher John Cornish Hellaby (1989). "Learning from delayed rewards". In:

Email: alexanderjjohnson@outlook.com