Faculty of Science and Engineering

Alexander J. Johnson

Mathematics

Neural Networks with Python and TensorFlow

March 13, 2020

School of Computing, Mathematics and Digital Technology

Abstract

Artificial Intelligence (AI) still remains as one of the greatest challenges in scientific research to this date, but much progress in the field has been made using artificial neural networks. The design of artificial neural networks is loosely inspired by that of biological brains, and serves as an expansion of an earlier concept called the perceptron (Rosenblatt, 1958). By using multiple layers of these artificial neurons, we can form a highly connected system that is referred to as a neural network, these networks can then be trained on a large data set to predict the output with high accuracy.

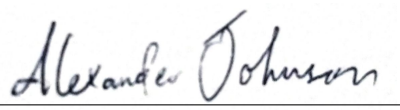
The range applications for neural networks is wide: they can be used to classify data, predict future states of chaotic systems, apply stylisations to images, and control physical/physically-based system in real-time.

TODO: *Abstract*.

Declaration

With the exception of any statement to the contrary, all the material presented in this report is the result of my own efforts. In addition, no parts of this report are copied from other sources. I understand that any evidence of plagiarism and/or the use of unacknowledged third party materials will be dealt with as a serious matter.

Signed



Contents

Chapter 1

Introduction to Neural Networks

TODO: *Chapter: Introduction*

Biological Neurons

Biological neurons are electrically excitable cells that are found in almost all animals. These neurons can transmit and receive electrical signals to one another via synaptic connections, which maybe either excitatory or inhibitory. Any given neuron will be either active or inactive depending on whether or not its input exceeds a threshold.

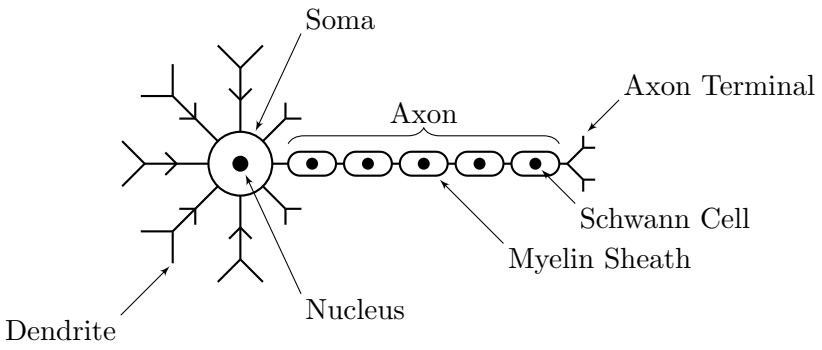


Figure 1.1: Diagram of a biological neuron.

Signals are received by the neuron via connections to dendrites and soma. If the threshold is met, electrical signals are sent along the axon to the terminal, where it is connected to other neurons, or to a controllable cell such as a neuromuscular junction.

TODO: *Section: Biological Neurons*

Artificial Intelligence

The idea of artificial beings capable of human intelligence can be traced back to mythical stories from ancient Greece. One such story was that of a mythical automaton called Talos, who circled an island's shores to protect it from pirates and other invaders.

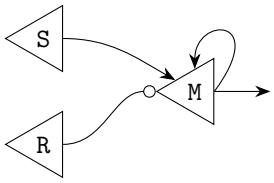
In the 19$^{\text{th}}$ century, other notions of artificial intelligence were explored by fiction in stories such as Mary Shelley's "Frankenstein", and Karel Čapek's "R.U.R.". Some of the fictional writings of the 20$^{\text{th}}$ century further continued exploring the concept in novels such as Isaac Asimov's "I, Robot".

Academic research into artificial intelligence began around the 1940's, primarily due to findings in neurological research at the time. The first explorations of artificial neural networks was done by McCulloch and Pitts, 1943, who investigated how simple logic functions might be performed by idealised networks. The neurons within these networks operated using some basic logic rules, applied to a discrete time system which, can be summarised using the expression

$$N(t) = (E_1(t-1) \vee E_2(t-1) \vee \dots) \wedge \neg(I_1(t-1) \vee I_2(t-1) \vee \dots),$$

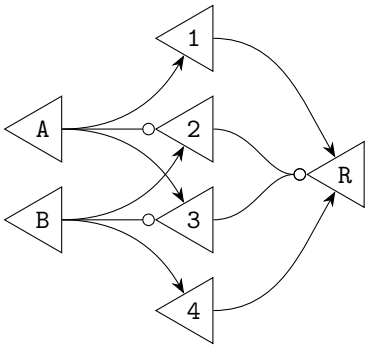
where $N(t)$ is the state of a neuron at time t, and $E_i(t-1)$ and $I_i(t-1)$ are the states of the excitatory and inhibitory connections from the previous time step respectively. The result is such that the neuron will only be active if at least one excitatory connection is active and all inhibitory connections are inactive. The versatility of this definition is demonstrated in the following examples.

SR Flip-flop



$$N_M(t+1) = (N_S(t) \vee N_M(t)) \wedge \neg N_R(t)$$

AND Gate



$$N_R(t+2) = N_A(t) \vee N_B(t)$$

Figure 1.2: Two common logic circuits using McCullochs neurons where the arrows and circles indicate excitatory and inhibitory connections respectively.

While this model provided insight into the mechanisms by which neurons operate, the structure

was static, and incapable of learning.

McCulloch's work was later cited by psychologist Hebb, 1949, who proposed that the structure of biological neural networks was dynamic, and that frequently repeated stimuli caused gradual development. At the scale of neuron, it was theorised that if one neuron successfully excited another, the connection between them would strengthen, hence increasing the likelihood that the former would be able to excite the latter again in the future. His theory was supported by research conducted by himself and others that showed that intelligence-test performance in mature patients was often unaffected by brain operations that would have prevented development in younger patients, which suggested that learnt stimuli are processed differently to unknown stimuli. This hypothesis became known as Hebbian learning.

Computer simulations applying this theory to a small network were done by Farley and Clark, 1954. The actions of the network were compared to that of a servo system which must counteract any displacements so as to maintain a steady position. The network was trained using a set of input patterns, which were subject to non-linear transformations. Similar to the Hebbian theory, when the network produces the correct responses, the active connections are strengthened. Although the results were of little neurophysiological significance, the results were of great use for demonstrating computational simulations, which were considerably slower at the time.

TODO: *Section: Artificial Intelligence*

Perceptrons

The idea of the perceptron was originally conceived by Rosenblatt, 1958, to represent a simplified model of intelligent systems free from particularities of biological organisms, whilst maintaining some of their fundamental properties.

The perceptron was built as a dedicated machine that consisted of a number of photovoltaic cells, analogous to a retina, that feed into an"association area". This association area contains a number of cells that each calculate a weighted sum of the receptor values and output a signal if it exceeds a threshold. Expressed mathematically, the output of a given association cell is

given by

$$A_i = \begin{cases} 1, & \sum_j w_{i,j} x_j > \theta \\ 0, & \text{otherwise} \end{cases},$$

where x_j is the value from the j^{th} photovoltaic cell, $w_{i,j}$ is the weight of the connection between association cell i and photovoltaic cell j, and θ is the threshold. These value weights were implemented using variable resistance wires that the perceptron could adjust automatically. The outputs from the association area are then connected to response cells, which operate in a similar fashion to the association cells. The activation of these response cells are the outputs of the perceptron, and indicated the classification of the input.

Similar to Farley and Clark, 1954, the method by which the perceptron adjusted it weights was also based on which cells were active, and whether the correct output was produced; except that the perceptron was also able to "penalise" weights when an incorrect result was outputted.

This machine was initially trained to reliably identify three different shapes: a square, a circle, and a triangle; and did so with a better than chance probability. When attempting to use the perceptron for more complicated tasks, such as character recognition, it failed to produce better than chance results.

After a decade of unsuccessful real world application attempts, a book titled "*Perceptrons: An introduction to computational geometry*" by Minsky and Papert, 1969, was released. The book provided a rigorous mathematical analysis of the model, the results showed that single layered, simple linear perceptron networks could not calculate XOR predicates. A 2017 reissue of the book contained a foreword by Léon Bottou, who wrote "Their rigorous work and brilliant technique does not make the perceptron look very good..."

Following the book's release, perceptron research effectively halted for 15 years until the first successful uses of multilayer networks by McClelland, Rumelhart, and Group, 1986, which also served as a departure from the neuron outputs being boolean values. The multilayered structure of this new model allowed it to calculate the XOR predicates that the single layer perceptrons could not.

The output of the units within these networks were defined by

$$\mathbf{a}(t+1) = \mathbf{F}(\mathbf{a}(t), \mathbf{net}_1(t), \mathbf{net}_2(t), ...),$$

where $\mathbf{net}_i$ is the i^{th} propagation rule applied to the inputs, $\mathbf{F}$ is the activation function, and $\mathbf{a}(t)$ is the activation of the units at time step t. The model usually uses a simplified version which can be summarised as

$$a_i = F\left(\sum_j w_{i,j} o_j\right),$$

where o_j is the output of unit j. Hebbian learning could be performed the network by using iterative methods, the most simple of which was given by

$$\Delta w_{i,j} = \eta\, a_i o_j,$$

where η is the learning rate, which is a constant.

TODO: *Subsection: Perceptrons*

Backpropagation

In order for a neural network to learn, it must undergo some form of optimisation process. For the perceptron, this process was one of positive and negative reinforcement.

In the field of control theory, an optimisation method known as gradient descent was developed by Kelley, 1960, in which a given function of the system is either maximised or minimised. This is achieved by taking partial derivatives of the function with respect to each parameter, which gives an approximation of how the function value will change as the parameter changes. By evaluating the partial derivatives, multiplying them by a constant, and adding them to their respective parameters, the parameter values can be updated. Using these new parameter values, one can expect to improve the function value. This can be written as

$$w_i' = w_i + \eta \frac{\partial f}{\partial w_i}(\mathbf{x}),$$

where $f(\mathbf{x})$ is the function to be optimised, w_i is a parameter of f, w_i' is the updated parameter, and η is ascent/descent parameter. Positive η values will maximise the function value, where as negative values will minimise it. The magnitude of η determines the rate at which the method will attempt change the parameters: if the value is too large, the method will overshoot the optimal values; if the value is too small, the method will be too slow to converge.

When this method was applied to neural networks, researchers sometimes encountered an

issue now known as the vanishing gradient problem. A computer program will typically calculate the gradient via repeat applications of chain rule; if there are many small terms, the gradient will tend to zero, and the learning rate of the network will be minimal.

One of the methods that overcame this problem was developed by Schmidhuber, 1992, where each layer of the network was pre-trained to predict the next input from previous inputs. Once each layer had been pre-trained, the network was then fine tuned using backpropagation. The method also provided a way of calculating which inputs were least expected, so that more training time could be devoted to learning them.

Since then, computational power has significantly increased, and the slow convergence caused by the vanishing gradient problem is less significant. Further more, backpropagation and a simple variant the model outlined by McClelland, Rumelhart, and Group, 1986, have become the standard for neural networks. Namely

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

where x_i is the output of neuron i, $w_{i,j}$ is the weight of connection from j to j, b_i is the input bias of i, and ϕ is the activation function.

TODO: *Subsection: Backpropagation*

Types of Artificial Neurons

The preceding discussion has been focused on densely connected neural networks, where each neuron in a layer is connected to every neuron in the previous, but it is important to note, that many other neural network architectures are often used together, and maybe more suitable under certain contexts.

TODO: *Section: Types of Neurons*

Convolutional Neural Networks

Often used in image processing, a Convolutional Neural Network (CNN) layer contains neurons that sample different regions of the previous layer using the same weight scheme.

A Convolutional Neural Network (CNN) layer contains sets of neurons that sample regions

of the previous layer, with all neurons within a set using the same weight scheme.

The 1D case may be expressed mathematically as

$$y_i = \phi \left(b + \sum_{j=0}^{S} w_j x_{i+j} \right),$$

where S is the kernel size.

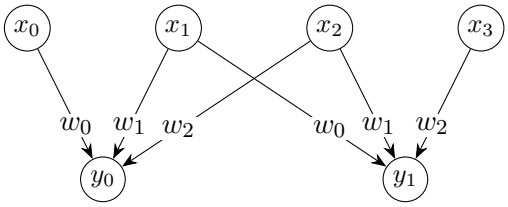


Figure 1.3: Example of a 1D CNN.

CNNs have the advantage of encoding spacial information implicitly, which makes them well suited for image processing.

RNN

LSTM

Chapter 2

Neural Networks in Python

Python is a general-purpose programming language designed by Guido van Rossum, with an emphasis on readability and reusability (Rossum, 1996). It comes with an extensive standard library and is one of the most popular programming languages.

There are multiple options for interacting with Python, these include:

- typing commands into an interpreter,
- writing files and running them with an interpreter,
- using an online service such as Google Colab.

TODO: *Chapter: Neural Networks in Python*

Single Perceptron Boston Housing Data

TODO: *Section: Single Perceptron Boston Housing Data*

Multi Layer Perceptron Boston Housing Data

TODO: *Section: Multi Layer Perceptron Boston Housing Data*

XOR Gate

TODO: *Section: XOR Gate*

Chapter 3

Introduction To TensorFlow

TensorFlow TODO: *Chapter: Introduction To TensorFlow*

Linear Regression

TODO: *Section: Linear Regression*

XOR

TODO: *Section: XOR*

Boston Housing with Keras

TODO: *Section: Boston Housing with Keras*

Chapter 4

Deep Learning

TODO: *Chapter: Deep Learning*

Recurrent Neural Networks

TODO: *Section: Recurrent Neural Networks*

Convolutional Neural Networks

TODO: *Section: Convolutional Neural Networks*

Image Processing

TODO: *Subsection: Image Processing*

Supervised Learning

TODO: *Section: Supervised Learning*

Unsupervised Learning

TODO: *Section: Unsupervised Learning*

Autoencoding

TODO: *Section: Autoencoding*

Reinforcement Learning

TODO: *Section: Reinforcement Learning*

Chapter 5

Null

TODO: *Decide parent chapter for CNN, RNN, and LSTM sections*

Bibliography

Farley, B. and W. Clark (1954). "Simulation of self-organizing systems by digital computer". In: *Transactions of the IRE Professional Group on Information Theory* 4.4, pp. 76–84. ISSN: 2168-2704. DOI: 10.1109/TIT.1954.1057468.

Hebb, Donald O (1949). *The organization of behavior*. John Wiley & Sons, Inc.

Kelley, Henry J (1960). "Gradient theory of optimal flight paths". In: *Ars Journal* 30.10, pp. 947–954.

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.

Minsky, Marvin and Seymour A Papert (1969). *Perceptrons: An introduction to computational geometry*. MIT press.

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.

Rossum, Guido van (1996). *Foreword for "Programming Python" (1st ed.)* URL: https://www.python.org/doc/essays/foreword/ (visited on 02/07/2020).

Schmidhuber, Jürgen (1992). "Learning Complex, Extended Sequences Using the Principle of History Compression". In: *Neural Computation* 4.2, pp. 234–242. ISSN: 0899-7667. DOI: 10.1162/neco.1992.4.2.234.