Bachelorproject 2019

Rasmus og Roald

May 6, 2019

1 Introduction

1.1 Motivation

Neurale netværk er rigtigt seje [**?**]

1.2 Proteins

1.2.1 What is

1.2.2 Secondary Structures

1.2.3 Solvent accessible surface area

1.3 Neural networks

The common perception is that humans and animals process information, i.e. transform perceptional stimuli and physiological conditions into behaviour, by using their brains. This is imprecise however, as the brain as such is only one functioning part of what is called the *nervous system*, which is in turn responsible for the internal workings of human and animal behaviour.

This nervous system is an abstaction over a number of *neurons* interconnected by synapses. Neurons in turn are so-called electrically exitable cells. In a gross over-simplification, this can be translated into the case that each neuron can have different internal states, depending on the internal states of the neurons it is connected to, thus forming a *neural network*.

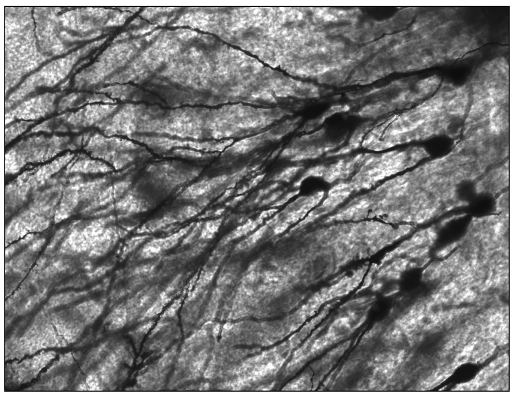

Figure 1: Neurons in the dentate gyrus of an epilepsy patient.

While obviously interesting within the fields of biology or psychology, this structure has shown to be enourmously interesting in the field of computation, as one can in fact model this very thing and use it to make predictions based on prior observations, for example in regards to the aforementioned structures of proteins folding.

Analogously, or rather, digitally, we can use this model to construct an *artificial neural network*. These set themselves apart from biological neural networks in a couple of ways: most urgently in that while neurons in biological neural networks are connected to each other via synapses, so that each neuron has an either inhibitory or activating effect on whether or not a connected neuron 'fires', in artificial neural networks neurons are connected by *edges*, each with an associated *weight*, allowing the artificial networks to leave the discrete domain and enter the continuous.

1.3.1 Basics

An artificial neural network is a specific instantiation of a *multi-layer perceptron*. A multi-layer perceptron in turn is a system of nodes arranged into layers and each receiving their value from the value of the nodes in the layer before them (adjusted by some weight) in combination with a bias (a scalar) and an activation function of some sort.
In this way the first layer will be called the *input layer*, the last layer the *output layer* and all layers in between *hidden layers*.
In the case of neural networks, the nodes in the system are referred to as neurons. Thus, if one was to describe it more precisely, the value z_i of neuron layer i with d number of neurons in it, given the matrix of sets of weights w (the biases being the very first row) and the activation function $h()$ can be expressed as follows:

$$z_i = h\left(\sum_{j=0}^{d} w_{ij} \cdot z_{i-1} + w_{i0}\right)$$

Such a model is useful for calculating continuous values as well as for classification purposes. A common example of the latter is the MNIST data set of handwritten digits in 28x28 pixel greyscale images. In this case the light intensity of the 784 (28^2) pixels could fittingly be the input layer, while a layer consisting of 10 nodes (corresponding to digits 0 through 9) could be the output layer, such that the system could be used to predict which digit is written in a given image. A common implementation of a neural network to perform this task is to have a single hidden layer of 800 nodes, however the most successful implementations in regards to the MNIST data set have been made with *convolutional neural networks*, but we shall return to these later.

Figure 2: Handwritten digits in the MNIST data set.

1.3.2 Training

Simply having defined the system of the neural network as above (denoted f_{NN}) naturally does not give us a reliable method for classifying, as the accuracy of the prediction inherently depends on the correctness of the weights and biases of the system (denoted Θ), which must be initialized with random values.
Enter a rather nifty idea named *backpropagation*. Having a set of predicted values ($\hat{y}$) for a data set (x) along with the observed correct values (y), one can establish a *loss* by applying a loss function (that we shall elaborate on later), but which can be as straightforward as a root mean square error. We shall denote this loss function $l(\hat{y}, y)$, so that the total loss of the system when applied to data set x is $l(f_{NN}(\Theta, x), \hat{y})$.
Now backpropagation refers to the practice of essentially taking the derivative of the loss function in regards to the matrix of weights and biases, thereby calculating a *gradient* $\nabla l(f_{NN}(\Theta, x), \hat{y})$ of these values with respect to the accuracy of the system, and then adjusting the weights and biases by the gradient multiplied by a learning rate scalar.
In laymans terms, by doing this one gets a matrix of 'change your weights by this and that much in this and that direction to increase prediction accuracy'.

1.3.3 Convolutional neural networks

Before talking about convolutional neural networks, we have to specify what a convolution is. Again the MNIST data set serves as an obvious example.
One can understand each of the images in the data set as a matrix of 26×26 values in the range of 0 - 255. The formal definition of a convolution is *TODO*, however the more practical explanation is that it is the resulting matrix of the product of a matrix and a filter. For example if one has the matrix a and the filter f:

$$a = \begin{bmatrix} 2 & 3 & 4 \\ 1 & 2 & 3 \\ 0 & 1 & 2 \end{bmatrix} \qquad f = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

the result of convolving a over f would be:

1.3.4 Multitask learning

2 Methods

3 Results

4 Conclusion

5 Appendix