AI4

Liquid State Mahine
Implementation and Testing

Daniel Tank Hviid
Dahvi14@student.sdu.dk

November 23, 2018

Abstract

Abstract here

Contents

0.1 Introduction

0.2 Motivation

In this section we look at why Artificial Neural Networks differ from biological neural networks, and why this is worth emulating and investigating. Then we look at some of the ways they can be modified to better emulate the biology of the brain

Standard neural networks are based on mathematical matrix models. These models share little with the functions of the brain, as they are a much oversimplified version of the biological network that constitutes brainmatter. As biological neural networks have so far unparallelled computational powers and uses, this report is motivated by the idea that these biological networks are worth emulating in search of more powerful and general neural models.

There are several differences between biological neural networks, such as a human brain, and artifical neural networks, such as a feed forward network. These include:

- Temporal states
- Non-differential signals
- Activation functions
- Structure

Looking only at the first three items on this list, there is a artifical neural network model that does take these into considerationg and as such better immitates the workings of the brain. This is the spiking neural network model, which uses the concept of spike trains instead of linear transformations to process the input information. These are said to have higher expressional power than standard neural networks, but in return require more computational power to run. The last item is structure. A biological brain is not made to be uni-directional, such as a feed-forward neural network, but instead is much better expressed as a reservoir computing framework. When one combines these two extensions of the feed-forward network, one arrives at what is called a Liquid State Machine.

0.3 Theory

In this section we look at the teory behind Spiking neural networks, reservoir computing, and Liquid State Machines, assuming that the reader is familiar with Artificial neural networks.

0.3.1 Spiking Neural Network

The main difference between a Spiking Neural Network and a non-Spiking Neural Network is the way it processes the weighted input sum in order

to find the output. For a non-Spiking NN, this is the activation function, while a Spiking NN uses a model that stores information from activation to activation. This main difference is what provides the Spiking NN with temporal information that permeates through the network.

Several such models exist, but only the Leaky Integrate-and-fire model [**?**] will be explained and used in this project.

The Leaky Integrate-and-fire model attempts to emulate the membrane potential and voltage spikes of a biological neuron. The biological neuron uses two different ion channels in order to build up potential and then dump it as a spike, which is modelled using the following equation, from [**?**]:

$$\tau_m \frac{dv}{dt} = -v(t) + RI(t) \tag{1}$$

Where $[\tau_m]$ is the time constant, $[v(t)]$ is the membrane potential at a given time t, R is the resistance of the membrane, and $I(t)$ is the integral of the input current. There are three different parts to this. The first is the change in membrane potential; This change in potential is the main part of the equation, as it is what we wish to find at each time step. The second part is the "leaky" part of the Leaky Integrate-and-fire, where $[v(t)]$ is the membrane potential at a given time t. At each timestep a part of the stored potential is lost, which is proportional in size with the stored potential. The last part is the integral of the weighted inputs, where R is the resistance of the membrane, and $I(t)$ is the integral of the input current, as with a non-Spiking NN. The time constant $[\tau_m]$ then scales all of this to fit with the time between each activation.

0.4 Implementation

In this section we look into how the Liquid State Machine was implemented in a code environment, and which choices were made along the way. The code implementations of this project used C++ as a language, and generally used Object Oriented design philosophy, aiming to be modular and easy to use.

The core of the implementation is the hidden layer of the Liquid State Machine, from now on refered to as the Pool. The Pool have input and output neurons, that are interfaced with by the input and output layers. The pool is implemented a 3 dimensional tensor of pointers to individual neurons. These neurons each contain a set of parameters, the important ones listed below:

- Input
- Internal potential
- Output
- Synapses

- Weights

As well as a list of constants pertaining to the specific neuron model used, see **??**.

The secondary part of the implementation is the input and output layers. In this implementation, these are implemented as standard feed forward networks. This method was chosen in order to facilitate the back propogation capabilities of a feed forward network in the output layer, which was then reused as the input layer for simplicity. As the two models are not compatible, the data needs to be converted from one to another at the transitions. The input transition is simple, it just adds to the input parameter of the Pool inputs, but the output transition is a bit more tricky. Because the implementation allows for two different kinds of activations, one at each time step and one that sums over a single time unit, two different transitions needs to be implemented. On the timestep activation the internal potential of the Pool outputs are simply given to the output layer as inputs, unless the neuron had just fired, in which case the output is 1. The timeunit activation instead takes the mean of the activations for the duration of the activation. Because of this, both activation methods then provide a number below 1, which allows for consistency when training the outputlayer.

0.4.1 Algorithm

Now that the elements are accounted for, the algorithm that is run each time the LSM is activated can be explained.

At each step the first thing that needs to be done is running the input layer. This yields a set of values that are then provided to the Pool input layer. The next step is activating all of the neurons whose internal potential exceeded the threshold potential last timestep. The third step is to check the internal potential of each neuron, to see which are going to fire next timestep. The last step is to gather the output from the Pool outputs.

0.5 Testing

0.6 Results

0.7 Discussion

0.8 Conclusion