The classes SubsetHopfieldNN and HopfieldNeuralNet

Robin Glattauer

April 10, 2012

Abstract

An explanation of how the classes SubsetHopfieldNN and Hopfield-NeuralNet work.

1 What the classes are for

First the classes where designed to help with track reconstruction, but as the principle behind it is much more general, they got transformed to templates, being able to handle tracks as well as entirely different things.

The overall goal of these classes is to find subsets of things that can be incompatible, but within the subset are entirely compatible and maximize some sort of quality. An example: You plan a concert where a lot of different artists shall perform together. You have 20 artist, that you are interested in asking. BUT: not all of them get along together very well. So artist 1 might be incompatible with artist 2 and 7 for some arbitrary reasons (e.g. they ate his cookies). And artist 2 is incompatible with artist 1, 3 and 13 and so on. In order not to completely mess up the concert you can only invite artists who are entirely compatible with each other. AND: not all artists are equal: some are really famous and some are pretty mediocre. So you want to find a set of artists with the highest possible quality which is completely compatible.

For tracking this is used, when you end up with reconstructed tracks, that share hits, i.e. are incompatible. So you ask yourself, which ones to keep and which to throw away. The quality in this case can be something like the χ^2 propability or the length or something else (or a mixture of course).

This is done by this class. The algorithm used for it is a Hopfield Neural Network (HNN). To sum it up: you have objects and some of them are compatible, some of them are not. They have different qualities and for whatever reason you want a subset of them, where all are compatible with each other.

2 How to use them

The HopfieldNeuralNet class does the core calculations of the Hopfield Neural Net. It is used by SubsetHopfieldNN, so you don't need to use it directly. The SubsetHopfieldNN is a template, so it works with any kind of objects, but in order to search for the best subset, you must provide two things: a measure of quality and compatibility. (Cause how else is the class going to know which ones are compatible, and which ones are the best).

This information is provided by functor classes (if you've never heard of them before, now's the time to google, they can be really handy).

Here is how the use of SubsetHopfieldNN could look like, if we work on our artist example:

```
class AristQI{

   public:

   // returns the quality of an artist
   // (a value between 0 and 1, 0 being bad and 1 being fantastic)
   inline double operator()( Artist artist ){

      return artist.numberOfFans()/nPeopleOnEarth;    // a number between 0 and 1

   }

};

class ArtistCompatibility{

public:

   // returns whether two artists are compatible
   inline bool operator()( Artist artistA, Artist artistB ){

      if( artistA.hates( artistB ) ) return false;
      else return true;

   }

};


//Somewhere within the program:

ArtistCompatibility comp;
ArtistQI qi;

SubsetHopfieldNN< Artist > subset;
subset.add( vecOfArtists );
subset.calculateBestSet( comp, qi );

std::vector< Artist > artistsToInvite = subset.getAccepted();
std::vector< Artist > artistsToStayAtHome = subset.getRejected();
```

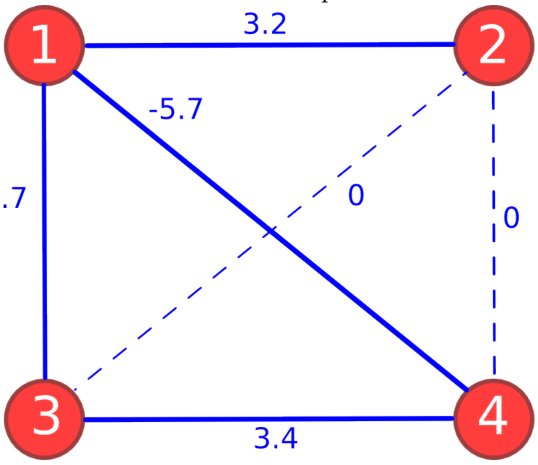
3 What is a Hopfield Neural Network

It is as the name suggests a neural network invented by J.J. Hopfield[2]. Neural means all elements are called neurons, because they can be (like neurons in biological terms) connected to each other. The neurons are binary, so they are either in state 0 or 1. Each neuron updates its state from time to time. The new value it gets (again 0 or 1) depends on the other neurons around it. Every neuron may be connected to it and have an activating or damping effect. If a neuron is updated the activation is calculated by:

$$a_i = \sum_{j \neq i} w_{ij} x_j$$

The x_j are the states of the neurons and the matrix w_{ij} gives the strength of the connection. The new state of the neuron x_i is then either 0 if a is below a certain activation threshold or 1 otherwise.

Let's have a look at an example:



Here we have 4 neurons (in red) connected to each other (connections are blue). The strengths of the connections are written besides them. (The dashed ones are 0, which just means there is no connection). The matrix w_{ij} tells us the strength of the connections:

$$w_{ij} = \begin{pmatrix} 0 & 3.2 & 0.7 & -5.7 \\ 3.2 & 0 & 0 & 0 \\ 0.7 & 0 & 0 & 3.4 \\ -5.7 & 0 & 3.4 & 0 \end{pmatrix}$$

In this case the connections are symmetric and therefore is w_{ij}. Let's assume all neurons have state 1 at the moment and update neuron number 1:

$$a_1 = \sum_{j \neq 1} w_{1j} x_j = 0 \times 1 + 3.2 \times 1 + 0.7 \times 1 - 5.7 \times 1 = -1.8$$

Now the question is: what is the activation threshold. If we just choose 0 as activation threshold, below 0 means inactive and above 0 means active, so the neuron number 1 is now inactive (its state is now 0).

Though the original idea of Hopfield was to describe a way information can be handled by neural cells and how memory storation can work within that context, the Hopfield Neural Net can be used as a tool for many different areas. Like the search for subsets.

4 How the classes use the HNN

As shown in [3] the Hopfield Neural Network can be used to find maximal compatible sets and with a modification [1] also to find nearly optimal subsets, with optimal meaning as large as possible and with a quality as high as possible.

It works the following way: the matrix w_{ij} is set up like this:

$$w_{ij} = \begin{cases} -1 & \text{if i and j are incompatible,} \\ (1-\omega)/N & \text{if i and j are compatible.} \end{cases}$$

N is the number of neurons in the network and ω is a tunable parameter, that controls how important the quality of a neuron is: the higher, the more influence comes from the quality.

The updating of the neurons is done asynchronous (one at a time in a random order) to avoid cycling between states.

There's an additional vector holding the quality of the neurons:

$$w_{i0} = \omega \cdot q_i$$

,where q_i are the qualities of the neurons.

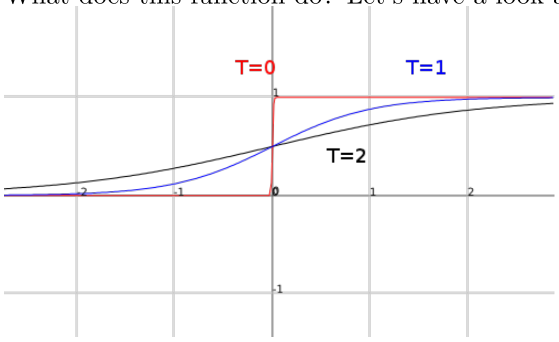
At the beginning the states of all neurons are a small random number. Then the iterations begin. In every iteration the state x_i of a neuron is calculated like this:

$$a_i = w_{i0} + \sum_{j \neq i} w_{ij} x_j$$

So you see, how additional to the influence from the other neurons the quality of one specific Neuron helps it getting activated. Instead of checking whether the value a is below or above a threshold, a so called activation function is used.

$$x_i = \frac{1}{2}(1 + \tanh(a_i/T))$$

What does this function do? Let's have a look at it:



We see, that all values are mapped to something between 0 and 1. So instead of having states with either 0 or 1, we now allow any value between. Also we see the dependency on the parameter T. The lower T gets, the sharper the assignment to either 0 or 1. With $T = 0$, we are back where we were before: we either assign full activation (1) or complete deactivation (0).

The idea behind T is, that it resembles a temperature. So instead of neurons going into full activation or deactivation, while its "hot" the temperature hinders them from doing this and lets them only partly that way. This way, if we start with a certain temperature and cool down gradually, we can make sure, that the neurons get enough time, that the system can gradually evolve. The benefit is, that this avoids local minima, i.e. situations where the neural network reaches a stable state, that is not optimal.

So we start at a finite temperature T (like 2.1) and update all neurons in a random order. After the first round we lower the temperature a bit by:

$$T_{n+1} = \frac{1}{2}(T_n + T_\infty)$$

To put it that way: at first we let them play with each other nice and friendly and as the atmosphere gets cooler, the winners and losers become clearer. We stop the procedure, once the changes of the states are sufficiently small (like below 0.01 for example). At this time all states above a certain threshold (like 0.75) form the best subset.

References

[1] Rudolf Frühwirth. Selection of optimal subsets of tracks with a feed-back neural network. *Computer Physics Communications*, 78:23–38, 1993.

[2] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proc. Natl. Acad. Sci. USA*, 79:2554–2558, 1982.

[3] Yash Shrivastava. Guaranteed convergence in a class of hopfield networks. *IEEE Transactions on Neural Networks*, 3(6):951–961, 1992.