

Project 2 : Machine Learning
Simulation of Short Term Memory and Long
Term Memory using Hopfield Networks

Achint O. Thomas
Aditya Joag
Sarvesh A. Telang
Vinuchandar Krishnaswamy

December 20, 2005

Contents

| | | |
|----------|------------------------------------|-----------|
| 1 | Introduction | 4 |
| 2 | Motivation | 4 |
| 3 | Model Description | 4 |
| 3.1 | Data Description | 4 |
| 3.2 | Model Description : | 5 |
| 4 | Algorithm | 6 |
| 5 | Experiments | 8 |
| 5.1 | Choice of Model | 8 |
| 5.2 | Degeneration of Memory | 12 |
| 6 | Comparison with Human Brain | 14 |
| 7 | Division of Work | 16 |
| 8 | Future Work | 16 |
| 9 | References | 16 |

Abstract

This project aims at simulating the human memory system a.k.a the short term memory and the long term memory and the interaction between them. This project aims at simulating the above system using hopfield networks, which is an associative memory system. i.e. its weights help in reproducing the input on the output, thus making it a memory. The project also aims at considering different models in the hopfield network and compare how they are similar to or differ from the human memory system. The system aims at modeling the human brain in the aspects given below

1. Remembering data (by training the hopfield networks)
2. Forgetting older data first (by weight capping)
3. Forgetting data over time (adding noise over time)

The model also describes an algorithm that will simulate the interaction between the short term and the long term memory.

The introduction and motivation for the project is given in the sections one and two. Section 3 describes the model that is used. Section 4 describes the experiments and analysis carried out to select the model as well as to compare the human system to the model. Improvements that can be done on the model presented is handled in Section 5.

1 Introduction

The human memory has a short term memory (STM) that stores about 7 items at a time. Memories in the STM decay over time and are forgotten. Only the ones that are reinforced are pushed into the Long Term Memory (LTM). The ones that are not reinforced decay over time and will get forgotten. The LTM is a more permanent memory as compared to the STM. Theoretically, it has infinite memory and does not forget the data as easily as the STM does. The forgetting function is a function over time. i.e. if at time t , the memory remembers some data, at time $t+1$, its memory will deteriorate. Both the memories work in the same way, but the LTM is more immune to this deterioration.

2 Motivation

Though Machine Learning has a lot of applications in classification and identification, its main purpose lies in making a system that will resemble the human brain. This project is a small step in that direction. Lot of work has been done in the field to model a human memory, especially using associative memory models. This project aims at doing the same.

3 Model Description

3.1 Data Description

The data set selected is a simple one. It is a set of $7*7$ image representation of alphabets and numbers. A random set out of it is taken to train the STM and the LTM. A pixel is represented by a 1 if that pixel needs to be drawn to form the letter. It is -1 if the pixel should be turned off to form the alphabet. This pixel representation is turned to a data point by treating each pixel value as a dimension.

Total dimensions in the data = 49.

There are no preprocessing steps done on the data.

Data is not visible to the LTM while training. It is presented to the STM one at a time. The STM tries to remember it. Next time, it remembers the data seen and then redirects its output (the remembered memory) to the LTM. The LTM is trained on this output. This is done to model the human system perfectly. In the human system, LTMs always only learn from the short term memory and does not see the data directly.



Figure 1: Datapoint for STM



Figure 2: Datapoint for LTM

3.2 Model Description

The model consists of two hopfield networks, one for STM and the other for the LTM. The STM is a hopfield network of 49 nodes. i.e. the data is of size 7×7 . The LTM is a hopfield network of 196 nodes. The data is scaled up while being passed from the STM to LTM in such a way that one pixel in the STM data corresponds to a square of 4 pixels in the LTM data. Dimensions are increased to accommodate the data in the number of nodes available in LTM. LTM has more nodes to increase its memory power. The STM here can remember about 6 different data points. The LTM can remember about 25 different sets of data.

The Hopfields for both STM and LTM are a hebbian-rule based hopfields. They are not gradient-descent hopfields. The analysis of both the models and the reasoning behind the choice is given in the experimentation section.

Hebbian Rule Hebbian rule states that "On seeing a data, if two nodes are activated simultaneously, then the weights between them should be increased. The same goes for two nodes that are both deactivated on seeing a data point. Also, if one node gets activated and the other doesn't, then the weights between them decrease".

4 Algorithm

The training algorithm used for the project is as follows:-

- Any data that comes in is first passed through the testing cycle to check if either of the memories remembers it.
- There can be following cases
 1. Neither STM nor LTM remembers the given data. In this case, we train the STM with the given data but not the LTM. This is analogous to a human brain or mind sensing (seeing/hearing/smelling) and learning something for the first time ever.
 2. The STM remembers it but the LTM does not. In this case we take the output that we got from the STM during the testing phase and pass that as a training input to the LTM. This is analogous to something being pushed into the long term memory because it was learnt repeatedly.
 3. The LTM remembers it but the STM does not. In this case we pass the data again to the short term memory as the training data. This is analogous to the human mind not recollecting immediately something that it senses. But it still can identify it because it is in the long term memory and as it has been learnt some time back.
 4. Both STM and LTM remember it. In this case we will not train either STM or LTM. This is analogous to a human brain already knowing something very well and not bothering to learn it again.
- Whenever training data is passed to either STM or LTM its weights are updated accordingly and these will be used in the next testing cycle.

The reasoning behind doing testing cycle before every training cycle is as follows: It was seen that when the network already remembers some things and we pass one of those things repeatedly as the training data then the weights are adjusted such that the network gets biased towards that particular learning. For example, if the STM already recognizes 2 and 3 and we repeatedly pass the image 2 as training, its weights would get biased towards the 2 and it will tend to forget the 3. This is one of the aspects in which the Hopfield networks are different as compared to the human memory. The human memory does not forget one thing if some other thing is learnt again and again.

The training algorithm used for the project is as follows:- While testing the Hopfield networks for STM and LTM we take a similar approach to what a human brain would do.

It would pass the pattern seen to both its short term as well as long term memory. If the short term memory remembers it, it would not bother consulting the long term memory.

On the other hand if the short term memory does not remember it, it would check whether the long term memory remembers it.

If the long term memory remembers it, the result will be shown to be coming from only the LTM.

If the LTM does not remember it either then the human memory would say that it does not recognize the pattern.

The Hopfield networks used here to simulate the memories show similar outcome. If the pattern given as input is actually learnt by the Hopfield network previously then it would increment the value of a variable (mem in this case).

The algorithm is as follows

- Load the weights stored during the training. Define a variable for storing the result (mem) and initialize it to zero.
- Pass the data point (7 X 7 matrix image converted to 1 X 49)
- Calculate the output for the input using $\text{Output} = \text{weight} * \text{input}$ (as this is a case of Hebbian learning, we do not have a separate output function. Instead $\text{Output} = \text{Activation}$ i.e. $y(a) = a$)
- The output is compared with the target to find the error.
- If the error is zero mem is incremented.

Thus if mem is non-zero we can say that the particular memory has learnt the particular input and remembers it properly.

For the long term memory, the testing algorithm is similar except for the following differences:

1. In STM we gave the input as a 7 X 7 image converted to a 1 X 49 matrix. Instead here we will give the input as a 14 X 14 image converted to a 1 X 196 matrix. This is because we are considering the long term memory to be stronger and hence has more neurons. Therefore to match the matrix dimensions while calculating the outputs we are expanding the data to four times its original size. (In the above given algorithm we are considering only one data point as the input. But we can give more

than one inputs and simply by incrementing mem in a loop we can see how many of them can be remembered).

2. In case of long term memory, we would get better results if the image has already been learnt. This is because we are giving an expanded image as the input. Also the LTM has more neurons. So it can handle the distortions in the data in a better way.

Considerations:-

- In the testing algorithm above, we are passing only non-distorted data as the input. So the value of the variable mem will increment if and only if there is a perfect match found in the memory. i.e. the output calculated using the stored weights and the inputs exactly matches the input.

In case we are passing distorted data during testing then a case may arise where the Hopfield network may find output equivalent to some other stable state and falsely or erroneously remember the given input. To counter this we can pass the targets along with the input and check if that target also finds a match in the memory. Only in the case that the target finds a match in the memory should we say that the memory remembers that particular input. i.e.increment mem.

5 Experiments

5.1 Choice of Model

Hopfield network is the obvious choice because it is an associative memory. There are two versions of Hopfield, gradient-descent and Hopfield-based. Both were considered. The models considered and their advantages and disadvantages are listed below.

- Batch gradient-based Hopfield network The data for training arrives one at a time; therefore batch training is not possible. A batch training algorithm can be converted into an online algorithm by training the model on all the data that has previously arrived plus the current data. i.e. when the 3rd data arrives, we train the data on the 1st to 3rd data items. This is not a very good model because it doesn't simulate how the brain functions. The brain can take one data at a time and remember it without any relation to the previous data.

This can also be achieved by reducing the learning rate to a very small number like 0.0000001 and the number of iterations also to be very

small, like 1000. This doesn't help either. The network only remembers the latest data point and forgets the previous points. This is expected because in the second round of training, we do not consider the first data point and so lose its optimal weight. We do start from the weights found by learning the first data point but still the move away from the weights is too big for the network to remember the first data point.

We can still decrease the learning rate or the number of iterations, but it would not learn the second data point then. If we do this, our algorithm reduces to the variation 1 of the online gradient-based network described below.

- Online gradient-based Hopfield network - Variation 1 In this model, A batch algorithm is run when the first data point comes in. The weights now model the first data point perfectly. For consecutive points, the weights take one step towards the direction of that data point. So, when we pass a data point frequently enough, it will remember it. But, if there is another data point that makes the weights move in an opposite direction, then there is a conflict. The network doesn't learn anything at all.

Example: Data points (3,2,5). 1st Step: Weights obtained for data-point 3. 2nd Step: Single step for 2. (Remembers both 3 and 2) 3rd Step: Single step for 3. (Remembers only 3)

If Instead of 5, we pass a totally unrelated data, say 9, then it remembers 3,2 and 9 provided we train on 9 a few times. Again, anything similar to 3, 2 or 9 and which will push the weights in the opposite direction will make it forget the other data.

Another disadvantage of the model is that, the weights are always near 3, and so 3 is not forgotten easily. But, it being the oldest data point, it has to be forgotten first.

To avoid keeping on training the datapoint n , we can pass it through the system for k number of iterations where k is the iteration in which it just starts remembering the n th point as well. This model is just a small variation of the above said model and yields the same results.

In the above said models, it is seen that data points that are very similar to each other are remembered better than the ones that are dissimilar. This is expected because, in most of the dimensions, the weights for the previous data point model the current data point as well. This is seen in the case of 3 and 2, where just one iteration of 2 is enough to remember both 3 and 2. The corollary is also seen when we learn a 9

after 3. It takes about 25 iterations with a learning rate of 0.0001 to model both 3 and 9. And this also proves futile if we start learning a 5 after this. Then, 9 is forgotten. Only 3 and 5 are remembered.

- Online gradient-based Hopfield network - Variation 2 This model tries to overcome the disadvantage of the first data being remembered all the time. So, the first data point is also run through only once. This doesn't help because after passing in a set of points, the weight is in such a position that it doesn't remember anything. This also happens if we run each of the data points for a very less number of iterations, say 25.

In each of the above scenarios, the weights never reach any local maxima to classify the data properly. So, it is at a "confused" state which does not remember anything. But, when we pass similar data like 3 and 2, then it remembers both of them just for one iteration of the algorithm. But, this is just a special case. For normal unrelated data, this model performs the worst.

- Online hebbian-based Hopfield network The weight matrix is calculated from the input data such that it follows hebbian rule.

$$W_{k*k} = W_{k*k} + r \left(\sum_{i=1}^k I_i^t * I_i \right)$$

Here, when the inputs to two nodes are 1 (they are activated together), then the weights between them increase. This happens when two nodes get deactivated together.

The model is trained online. It succeeds in remembering upto 6 non-similar data points. It remembers more if the data points are similar to each other. For retrieving the outputs, the input is passed through the network and the outputs are calculated using the following equation

$$Y = I * W$$

The outputs are then compared with the inputs. This is an iterative associative memory. So, the outputs are passed on to the network recursively until it reaches one of the stable states. If this stable state is the input under consideration, then the network has remembered the input. The number of iterations required to move to a stable state differs depending on how much the data is garbled. In our experiments,

| $A(O(1/\sqrt{49}))$ | No.of Data Points | No.of old data forgotten | No.of Data not trained |
|---------------------|-------------------|--------------------------|------------------------|
| 0.38 | 3 | 0 | 0 |
| | 4 | 0 | 1 |
| | 5 | 0 | 0 |
| 0.38/200 | 3 | 0 | 0 |
| | 4 | 1 (Not oldest) | 0 |
| | 5 | 1 (Not oldest) | 0 |
| 0.38/300 | 3 | 0 | 0 |
| | 4 | 1 oldest | 0 |
| | 5 | 2 oldest | 0 |
| 0.38/400 | 3 | 1 (Not oldest) | 0 |
| | 4 | 2 oldest | 0 |
| | 5 | 2 oldest | 0 |

Table 1: Weight Capping Analysis for STM

as we are passing only non-distorted data, it has to go to a stable state within a few iterations. So, the model has been hard coded to iterate 10 times before it can decide if the data is remembered or not.

To model forgetting the oldest data first, weight capping was tried out. In weight capping, the weights are controlled not to go above or below a particular weight.

$$-A \leq w_{ij} \leq A \text{ with } A \leq O(1/\sqrt{N})$$

Here, given a random alphabets and training the network, the capacity of the network goes down to 4. Though, most of the times the older data is forgotten first, sometimes the new data doesn't get learnt. When the weights have to be updated for a new data point but it is capped, it doesn't learn the new data. This also manages to forget the older data.

The capping point was decided based on a trial and error experimentation of how well the old data gets forgotten. Few of the results are tabulated below. For the short term memory,

From the experiments, we can infer that $A=0.38/300$ is a good cap, because of two reasons.

1. It doesn't restrict the learning of new data.
2. The forgetting is gradual

Similarly, for the Long Term Memory the table is given

| $A(O(1/\sqrt{49}))$ | <i>No.of DataPoints</i> | <i>No.of olddata forgotten</i> | <i>No.of Data not trained</i> |
|---------------------|-------------------------|--------------------------------|-------------------------------|
| 0.0714 | 5 | 0 | 0 |
| | 8 | 0 | 1 |
| | 10 | 1(Not oldest) | 0 |
| 0.0714/100 | 5 | 0 | 0 |
| | 8 | 1 (Not oldest) | 0 |
| | 10 | 1 (Not oldest) | 0 |
| 0.0714/350 | 5 | 0 | 0 |
| | 8 | 1 oldest | 0 |
| | 10 | 2 oldest | 0 |
| 0.0714/500 | 5 | 0 | 0 |
| | 8 | 2 oldest | 0 |
| | 10 | 3 oldest | 0 |

Table 2: Weight Capping Analysis for LTM

5.2 Degeneration of Memory

Both STM and LTM degenerate over time. The STM forgets memories faster than the LTM.

This is simulated by the addition of a random gaussian noise to the weights in both the STM and LTM. Addition of noise is made continuous through a timer in matlab. A random noise is added every minute. This effect on the weights succeeds in degenerating the memory over time. The noise added to the STM is larger than the noise in LTM

When STM is considered As we have trained the weights on alphabets 3,T,U,H,O,X,E, using the hebbian rule. The weights are able to remember 3 terms in all when we test it. The weights in the short term memory are then corrupted by random Gaussian noise. The Gaussian noise value is adjusted in such a way that after a few number of noise entries the network is able to reproduce values less than 3 which finally will result in a null memory state.

The graph shows that while the weights get corrupted by noise the associative memory of the network decreases from 3 to 2 alphabets. This recall rate is inversely proportional to time since as time passes by for an STM noise factor in small amounts increase or decrease the value of weights. If the gaussian noise is high the loss of memory rate is high and vice-versa.

The figure below shows the a short term memory loss when the noise is kept high. As the values of the weights keep changing due to noise, the random nature of noise might cause a proportional increase. This will lead the short term memory to remember forgotten values once in a while. This happens for a very short duration though.

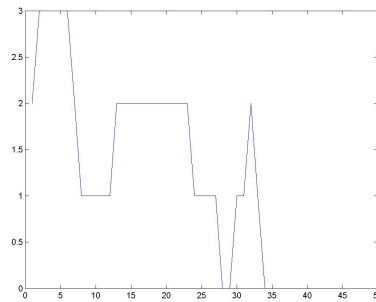


Figure 3: Degeneration of STM

Learning weights while adding noise

As the short term memory gradually gets corrupted by random noise the new data fed into it has to be learnt. Hence this cause a new change of weight vectors to be moving a direction close to the new data vector. Thus the new weights are correctly able to identify the new value of data. These weights are again repeatedly corrupted with noise which leads to the network forget the new data point. This is shown in figure 3 where the network is able to remember 3 values in the beginning and later gradually due to the addition of noise in it , it eventually remembers none and here is where the network Is trained again so that it remembers the previous values. This process keeps going on. It might happen that the rate of memory loss decreases is less as the network is kept training.

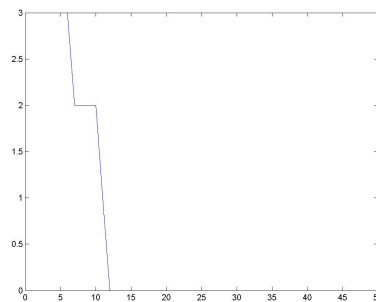


Figure 4: STM Memory Loss

Learning without noise When no Gaussian noise is added to the network the behavior is quite different from what is was when the noise is added. As we keep on increasing the inputs to the network the weight values adjust in a way so as to remember most of the input values. But as the number of

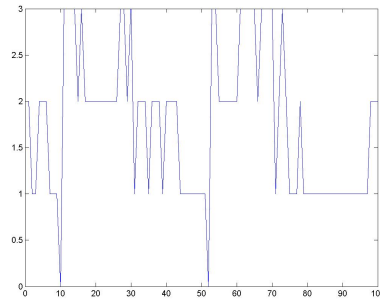


Figure 5: Learning while adding noise in STM

input values in increased the threshold value of memory is reached and later the new input values fed into the network are not learnt by the network.

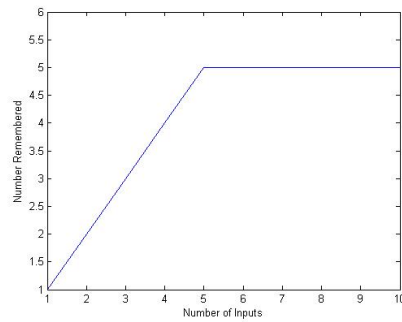


Figure 6: STM without noise

6 Comparison with Human Brain

Comparison with actual human memory:

* Similarities

1. The training process for the human memory is similar to what we have tried to implement here. i.e. The input weights change as and when new data is learnt and those weights are used while analyzing new things seen.
2. The learning rate of STM and LTM differs in a similar way in the actual brain and in what we have implemented here. i.e. The LTM will not learn or remember something until the STM has learnt it repeatedly.

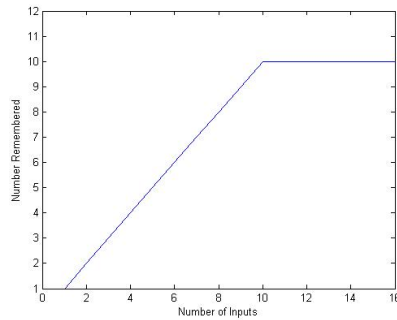


Figure 7: LTM without noise

3. The unlearning or forgetting is also similar to the human brain in the sense that LTM remembers anything for a longer time than the STM. This has been achieved by adding noise to STM and LTM at different rates.
4. In human brain if the capacity is reached the oldest things learnt are forgotten from the short term memory. The model designed here behaves in a similar way.
5. In this model when a data is passed through for training, it can also reinforce parts of another data point. i.e. when similar data like 3 and 5 are trained one after another, the 5 will reinforce the top most row of 3. So, it might also happen that one data becomes strongly learnt because of this. So, forgetting it will be more difficult. That is one of the possible reasons why an older data doesn't get forgotten very easily. It is similar to the working of the human brain where when we see data, we also reinforce the related data.

* Differences

1. This being a very basic model has much less capacity as compared to a human brain.
2. As mentioned in the algorithm for training, human memory does not forget one learning when some other learning is done repeatedly, but the model designed here is susceptible to such unlearning.
3. The human memory identifies a lot many different distorted variations of the pattern it knows which is not the case with this model. It can identify patterns distorted only up to a certain extent.

7 Division of Work

- Vinu: Analysis of the hopfield models and deciding on one Analysis of the component for "forgetting old data while remembering new ones"
- Achint: Short-term and Long-term memory interaction and Data analysis
- Sarvesh: Design and Analysis of component for forgetting data over time using gaussian noise. Analyse the capacity of STM and LTM with and without noise
- Aditya: Design and analysis of the algorithm for training and testing. Analyse the similarity and difference to the human brain

8 Future Work

As of the present work we have done, the memories can only remember exact matches of data that it is presented. This means that if distorted data is passed to the memories, the Hopfield network may find output equivalent to some other stable state and falsely or erroneously remember the given input. We could think of ways to enable the network to remember distorted versions of the same datapoint, in further work. This could be done by presenting distorted versions of the same datapoint to the network during the training phase for the datapoint. However, this could result in a case where the network remembers the original and distorted datapoints as separate entities, which is undesirable. The network must generalize over the datapoints to account for small errors in a datapoint as due to noise and remember them as a single datapoint.

9 References

1. <http://www.goertzel.org/papers/ANNPaper.html>
2. <http://www.gamedev.net/reference/articles/article771.asp>
3. Paul Thagard, Mind-Introduction to Cognitive Science II Edition, Cloth April 2005