MŰEGYETEM 1782

Budapest University of Technology and Economics

Faculty of Electrical Engineering and Informatics

Department of Telecommunications and Media Informatics

Csongor Pilinszki-Nagy

Optimizing Hierarchical Temporal Memory for Sequence Learning

Supervisor:
Bálint Gyires-Tóth, PhD

October 29, 2018

Kivonat

A Hierarchikus Temporális Memória (HTM) egy olyan gépi tanulási módszer, amely az agykéreg működésének újszerű elméletét használja fel. Ez az elmélet többek között új megközelítést ad arra, hogy mit jelent az intelligencia és hogyan működik az emberi agy ezen része. Az elmélet szerint a agykéreg egy olyan homogén rendszer, amely különböző absztrakciós rétegek hierarchiájába rendeződik, melyek ugyanazt az időalapú minta-tanulást végzik. Ennek a hierarchiának csak egy részét kell megfejteni ahhoz, hogy megértsék és lemásolják az emberi szintű általános intelligenciát. A HTM egy új típusú neurális hálózatnak tekinthető; alkalmas idősorokból való tanulásra anélkül, hogy a hálózat bementéhez biztosítani kellene az összes korábbi bemenetet.

Habár a téma még nem hozott olyan látványos eredményeket, mint amit a mély neurális hálózatoknál láthattunk az elmúlt években, mindamellett rendkívül erős gépi tanulási módszernek bizonyulhat. A legígéretesebb eredmények a Numenta csoport publikációiból és könyvéből származtak; a Biological and Machine Intelligence, amelyben beszámolnak kutatási eredményeikről és egy lehetséges megvalósítást is nyújtanak. Ez a NuPIC nevű program Python-ban, amely bemutatja az ilyen típusú hálózatok tanulási képességeit. Ennek az elméletnek azonban jelenleg nincs publikált optimalizált megoldása.

A HTM hálózatot ritka mátrix reprezentációt igényel, vagyis annak elemeinek csak egy kis hányada nem nulla értékű. Ez a fajta ritka mátrix szükségessé teszi az új típusú ritka mátrix operátorokat, amelyek nincsenek megfelelően implementálva az elterjedtebb gépi tanuló keretrendszerekben. Továbbá a jelenlegi megvalósítás nem alkalmaz nagymértékben párhuzamos hardvert, mint a grafikus processzorok (GPU), amelyek jelentős módon hozzájárultak a gépi és mély tanulás által elért eredményekhez.

Jelen munkámban ezekre a problémákra szeretnék megoldást nyújtani, implementálni egy optimalizált alap HTM hálózatot, a GPU-kon végrehajtott ritka mátrix műveletek használatával, és összehasonlítani őket a szekvenciális tanulással a jelenlegi LSTM-ek (Long Short-Term Memory) segítségével. A

HTM hálózatok belső mechanizmusának vizsgálata és a teljesítményének kiértékelése szintén része a munkámnak.

Abstract

Hierarchical Temporal Memory (HTM) is a machine learning method that utilizes the theory of how the neocortex functions. This theory presents a novel approach as to what it means to be intelligent and what are the core principles of this part of the human brain. It suggests that the neocortex is a homogeneous system arranged into a hierarchy of different abstraction layers, which do the same time-based pattern learning. Only one section of this hierarchy needs to be reverse engineered to understand and reimplement a human-like general intelligence. The HTM can be considered as a novel type of neural networks; it is suited to learn from sequential input, without providing the whole history of inputs to the network.

There has not been as much research on this topic as on deep learning, however, it could also be a powerful machine learning method. The most promising results came from Numenta group and their book; the Biological and Machine Intelligence, which described their progress and possible implementations extensively. They also published a simple implementation called NuPic in Python, which demonstrates the learning capabilities of this type of network. However, there is no published optimized implementation of this theory, which I would like to contribute to during my research.

The main challenges are: a much more complex network structure, which requires more work to express as parallel operations. The network is represented in sparse matrices, meaning it only has a small fraction of its values are different from zero. This type of sparseness requires a new type of sparse operators, which are not well implemented in mainstream machine learning libraries. Finally, no implementation uses massively parallel hardware, like Graphical Processing Units, which are behind the current developments of machine learning.

In the current work, I would like to address these problems, implement an optimized version of baseline HTM networks, using sparse operations executed on GPUs and compare them on sequence learning with the current state of the art LSTMs. Investigation of the inner mechanism of HTM networks and

evaluation of their performance are also part of my work.

Contents

1 Introduction

Machine learning gained extraordinary popularity in the last decade, thanks to both research progress and the advancements of computational power. These changes led to achievements that weren't imaginable before. However, these solutions require an enormous amount of data and computational power to train, especially when training on multimedia, like photos or videos. In the case of sequence learning the length of the learned sequences presents additional challenges. A shorter learning horizon can miss essential correlations in data, while longer might be inefficient. There have been attempts to counter this by sharing resources through cloud services to spread the costs related and also transfer learning which enables only partial training of these vast models. Unfortunately, these solutions are still computationally expensive and not so effective compared to what the human brain is capable of.

When people talk about neural networks, it is understood that the structure of those resembles the human brain's intelligence or nerve system. However, Numenta argues that this is not the case, current neural networks have little or nothing to do with how the human brain or nervous system works. They say that there are fundamental differences between current neural networks and the human brain.[Haw+16] Some of the problems current neural networks suffer from are overfitting, vanishing or exploding gradients, false positives or just the general amount of computation needed to train with many epochs. According to Numenta first we need to understand how the human brain works, then we could implement such a system that works on those principles that make us truly intelligent.

The Hierarchical Temporal Memory (HTM) network has some apparent, currently mostly theoretical advantages over the regular feedforward neural networks. It is a simpler structure that uses binary activations, binary connections and a simpler method for training, i.e., updating those connections during learning. This simplicity comes at a cost, this network has more neurons than a feedforward network, since the neuron activity is always low. However,

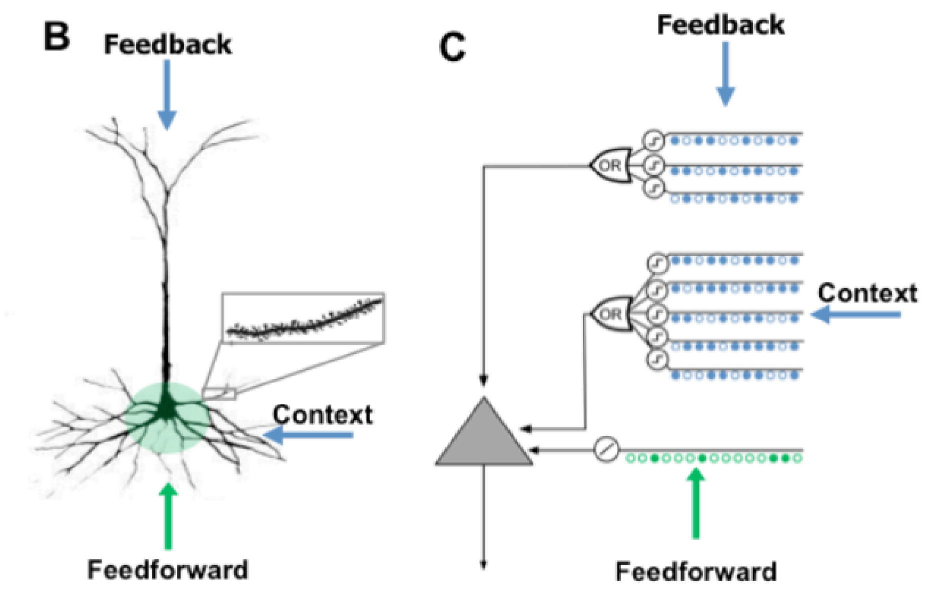

Figure 1: Real and artificial neural networks, Source: [Num]

this sparseness can be used to make this network more efficient.

Numerous changes need to be implemented for such a network. Some of those resemble well-known operations from other networks like max pooling in convolutional neural networks.[L+95] In Figure 1 it can be seen how an HTM cell models a real neuron. It receives inputs from three sources: the Feedforward input, the Context representation and the Feedback information from cell higher on the hierarchy. First, the encoding of data needs to change. Instead of the apparent one-hot encoding, there is a more efficient encoding that retains the semantic information and also remains efficient, and it is called Sparse Distributed Representation (SDR). Every data can be represented as an SDR of a combination of those. In Figure 2 there are two different encodings. The one-hot encoding uses N bits to represent N classes, and for every class one bit is active. In the SDR case, there are buckets which can be classes or scalar ranges. For each of these buckets, a few adjacent bits are active. This encoding has favorable properties over the usual one-hot encoding, which produce too much overhead or densely packed representations of data which are prone to false positives in the inference phase. More on the SDR representation in the HTM - SDRs subsection.

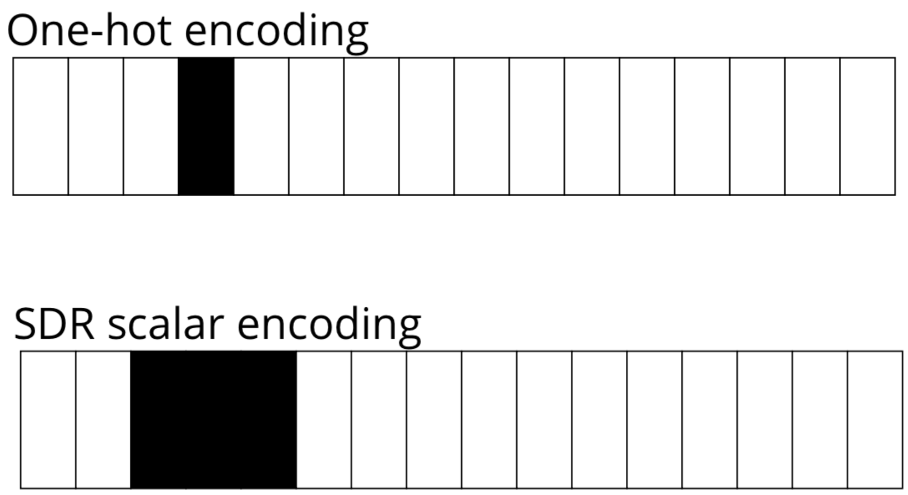

Figure 2: Encoding differences

Then comes the Hierarchical Temporal Memory which is the system perfected by the research of Numenta, and also currently under massive development. This is still far from a final product, but they managed to create a demonstration system which can handle temporal data exceptionally. The HTM network only accepts SDR inputs, so an encoder is always needed. It consists of two parts a Spatial Pooler and a Temporal Memory part. The first part is responsible for the normalization of data while reserving its spatial and semantical meaning. The second part learns the temporal patterns and is capable of coding the context of the current input in a way that makes different sequences distinguishable.

After the spatial pooler coded what the network sees and Temporal Memory the context, an SDR classifier needs to decide what the next element in the sequence is going to be.

All the layers described here use Hebbian learning which means separable learning for every layer. This method gets rid of the gradient vanishing and exploding problems, while also enables easy debugging for every layer.[Haw+16][KGV99]

There is an implementation of HTM network by Numenta called NuPIC, but it is not an optimized code, more of a proof of concept prototype.[Haw+16] The goal of my work is to apply massively parallel training possible in HTM networks. Thus, it will be more scalable to more complex tasks. Furthermore, I investigate the sequence learning capabilities of an HTM network against an LSTM baseline network. The training data is a multidimensional superposed sinusoidal signal with added noise.

My goal is to implement this method because of its favourable learning properties of one-shot learning and robustness against noise. The current implementation can not yet handle vast amounts of data, but it is a model that should scale well, depending on the implementation. I contacted the Numenta team about their optimization roadmap, and their response was positive towards my plans. At this point they are still developing the models of other functionalities of the neocortex, like the grid cell, so the performance optimization is not yet a priority for them.

2 Deep learning background

2.1 Neural networks

Deep neural networks are the cutting edge technology in machine learning today. These networks are capable of almost any optimization task, which can be translated to image recognition, asset price prediction or video generation. As the computational power started to increase thanks to the development of graphics accelerator cards more complex neural networks started to appear to tackle even more complex tasks with better accuracy. However, the problem with these solutions is that as the model size increases it requires more and more hyperparameter tuning and that demands either professional human knowledge or even more computational power to optimize. Since the impact of these variables is not known the best method is still trial and error which is a massive loss in efficiency.[Tho+13]

2.2 Recurrent neural networks

Recurrent neural networks were created to learn from sequences and predict some future value, so there is a time-based correlation between the data points in addition to the spatial features.[SSN12] Since time only goes forward, it is not advised to treat the time dimension like any other spatial dimension. A typical RNN has an input value x_t and an output value h_t. The network deals with the time-based correlation by having a recurrent connection with itself, where the last state of the neural network is the input of the network in the current timestep.

This method presented in Figure 3 convert the learning to a feedforward like neural network training, which is trained with the so-called backpropagation through time (BPTT) algorithm.[Wer90] This approach presents a new challenge which is the fixed length sequence learning problem. How long should this fixed length be and what is the maximum it can be. Since the error cannot backpropagate through many layers due to vanishing or exploding gradients,

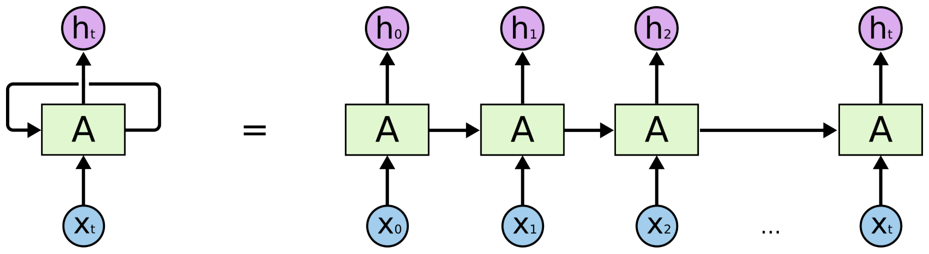

Figure 3: Recurrent Neural Network unrolled, Source: [Ola]

this is a limited model and can only learn short sequences well.

2.3 Long Short-Term Memory

When learning sequences the first problem that comes to mind is the length of the sequences that the neural network is working with. There are multiple levels at which we are interested in the context of the input. For example, when learning sentences, there are character based sequences, sequences of words and also connections that go through sentences as well. Building a character based learning means that the network is not able to learn the connection between sentences. If we were to learn at a bigger scale, by words, there might be a problem where new worlds that don't fit into the encoder confuse the network into predicting something unexpected given the context. Recently the use of Recurrent Neural Networks dominates the field of the sequence learning methods because they deal with the time-based structure of the data more efficiently than a fully connected neural network. However, the problem with sequence learning with RNNs is that training those is hard. The only way it is done is rolling out a few steps of the network as a feedforward one. This way the training is similar to those, but through the many layers, the vanishing gradient problem can occur. To fight this a new way was developed to hold onto some critical information during the use of a network, by inserting an LSTM cell into the unrolled RNN network.[HS97] LSTM stands for Long Short-Term Memory. Sequence learning usually is done by Long Short-Term Memory or one-dimensional Convolutional

Neural Networks.[L+95] Both of these solutions use the same basic concept, which rolls out the temporal part of the data into a multi-layered network. This approach has an obvious limitation; it is a fixed length sequence learning method, and also this length is restricted by how much layers can the error backpropagate during learning.

3 HTM background

3.1 HTM overview

Human brain model

HTM is a unique approach to artificial intelligence that starts from the neuroscience of the neocortex. The older parts of the brain that are below the neocortex are involved in the essential functions of life, like sleeping and eating. The neocortex is involved in all that is considered intelligent behavior. The structure of the neocortex is identical across the whole human brain; this tells us that the brain processes the different pieces of information coming from different senses in the same way. The neocortex has a hierarchical structure where lower parts process the stimuli, and higher parts learn more general features. The neocortex consists of neurons, segments, and synapses. In Figure 4 there are illustrations about the neuron columns in the neocortex. There are vertical connections that are the feedforward and feedback information and there are horizontal connections that are the context inputs. In this case the Feedback inputs won't be used since only one layer of HTM network will be used. The neurons can connect to other nearby neurons through segments and synapses. There are different types of segments, those that connect to the neurons found lower in the hierarchy are called proximal. The ones that come from distant neurons are the distal ones. There are also distal segments that connect to the upper neurons.

The feedforward neural network somewhat resembles these principles, it is homogeneous, meaning every cell does the same computations and it has a hierarchy, every cell in higher layers learn something more abstract from the layers beneath.

Basic HTM network

Figure 5 shows an overview of the HTM network for scalar value sequence learning. All the parts of HTM have different roles in understanding and predicting

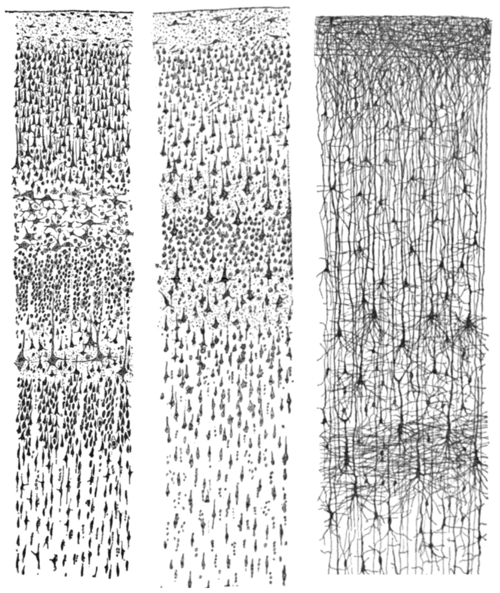
Figure 4: Neuron column, Source: [Caj]

the input data.

- The SDR scalar encoder, which is capable of representing multidimensional scalar data as a Sparse Distributed Representation. There are encoders for just about any type of data, but those are beyond the scope of this paper. SDR representation is a form of coding data into bit arrays so that it retains the semantic similarity between similar input values.

- The spatial pooler decides which columns should be activated given the SDR representation of the input. The spatial pooler acts as a normalization layer for the SDR input, which makes sure the number of columns and number of active columns stays fixed. This is crucial for the HTM network to work.

- The Temporal Memory receives input from the spatial pooler and does the sequence learning, which is expressed in a set of active cells. Both the active columns and active cells are sparse representations of data just as the SDRs. These active cells not only represent the input data but provide a distinct representation about the context that came before the input.

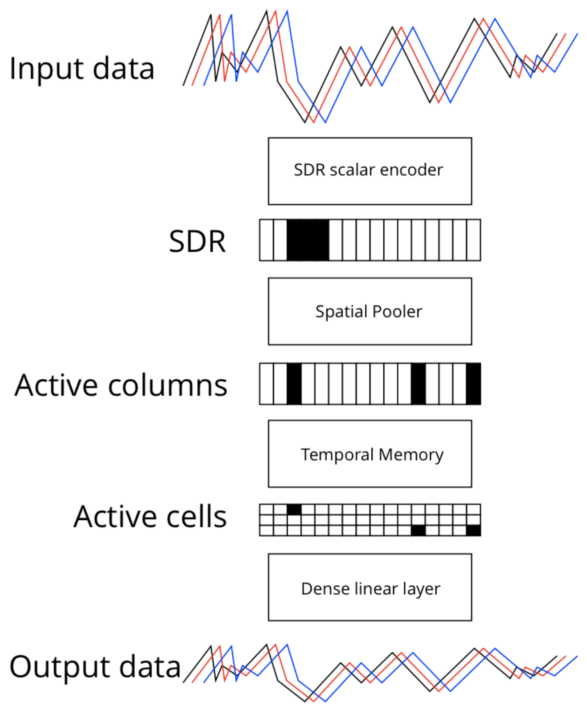

Figure 5: HTM overview

Using this multi-tiered input the SDR classifier has to decode the active cell values to a scalar value that should come next in the sequence. This output can also be multidimensional like the input values. The HTM network can not only predict the future values of sequences but detect anomalies in sequences.

HTM core principle

HTM starts with the core assumption that everything the neocortex does is based on the memory and recall of sequences. These sequences are patterns of the SDR input, which are translated into the sequences of cell activations in the network. This is an online training method, which doesn't need multiple epochs of training. Most of the necessary synapse connections are created during the first pass, so it is a one-shot learning capability. The HTM network can recognize and predict sequence with such robustness, that it does not suffer from the usual problems hindering the training of conventional neural networks. HTM builds a predictive model of the world so every time it receives input, it is attempting to predict what is going to happen next.

3.2 HTM training

Training in the HTM network differs from other neural networks. In the network all neurons, segments and synapses have binary activations. The networks connections are decimal values between 0 and 1. Since this network is binary, the typical loss back propagation method will not work in this case. The learning suited for such a network is Hebbian-learning.[KGV99] It is a simpler but unsupervised learning method, where the learning only occurs between neighboring layers. Other methods help the learning mechanism, but these are only present in the spatial pooler and Temporal Memory.

Segments and synapses

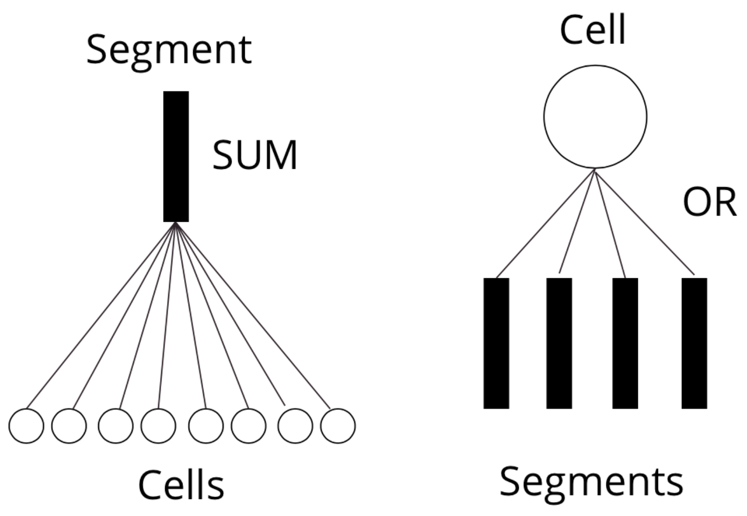

Figure 6: HTM segments and synapses

In the HTM network, cells are not connected directly through synapses with each other. Figure 6 shows how the cells, synapses and segments are connected. A cell has one or multiple segments connected to it. If one of these segments is activated, then the cell is activated too. This is like an OR gate between the segments. The segment has synapses that connect to other cells. If one cell is active, the synapses connected to it will be activated too. If the segment has enough active synapses, then it becomes active. This is a summation operation with a gate for a given threshold. The synapses are only potential synapses meaning there could be a connection if the strength of that connection is enough. At initialization, these synapses will have strength around the threshold, and with time these can change. If a potential synapse strength is above the threshold, it is a connected synapse and with enough other synapses can activate the segment.

Hebbian learning

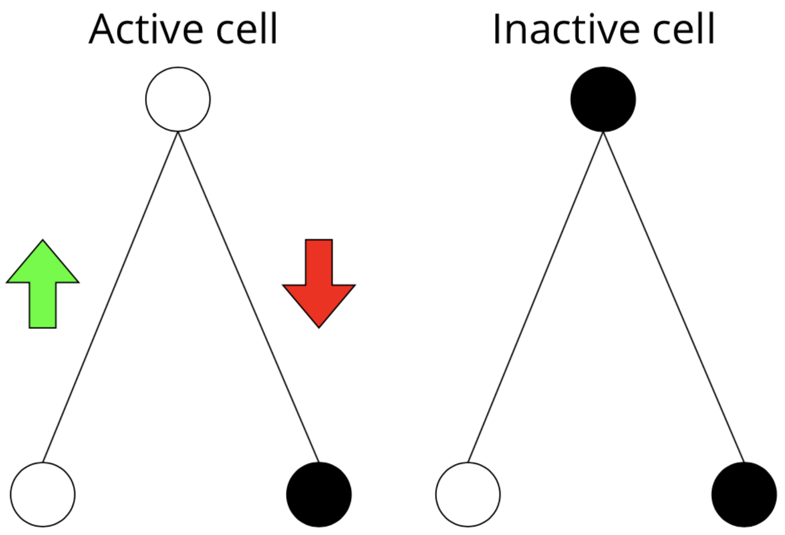

Figure 7: Hebbian learning, white cells are active and black cells are inactive

Hebbian learning[KGV99] rules shown in Figure 7:

- Only those synapses that are connected to an active cell through a segment learn
- If one synapse is connected to an active cell, then it contributed right to the activation of that segment. Therefore its strength should be incremented
- If one synapse is connected to an inactive cell, then it did not contribute right to the activation of that segment. Therefore its strength should be decreased

The learning has one more added complexity at learning in the Temporal Memory but more on that in the Temporal Memory learning section.

3.3 Sparse Distributed Representation (SDR)

Bit arrays

The capacity of a dense bit array in 2 to the power of the number of bits. This gives the bit array a large capacity but little resistance to noise. The first operator for two-bit arrays is the OR operation, which is used for creating a union of multiple bit arrays. The second operator for two-bit arrays is the AND operator which can be used for getting the matching bits between two arrays. The population of a bit array is the number of ones in it. The population is also the Hamming weight of the bit array. In Figure 8 a dense and a sparse matrix is shown. dense representation is a bit array where the ones and zeros are for example in a 50-50% ratio. On the other hand, a spare representation has only a 2% ratio; this enables favorable properties for use in the HTM network. In this sparse case, the capacity is much smaller. In a good representation, every bit in this bit array can have a specific meaning which represents a given object. A sparse bit array can be stored efficiently by only storing the indices of the ones.

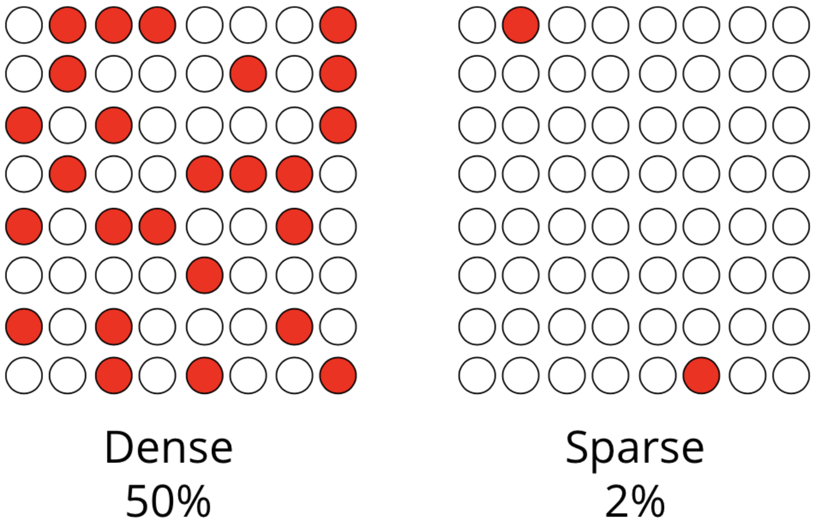

Figure 8: A dense and sparse binary matrix, red is active white is inactive

SDR capacity and comparison

HTM networks use Sparse Distributed Representations (SDR). This is a representation for any data in the form of a sparse bit array, where only a small fraction of bit are ones. As discussed above the capacity of a sparse bit array is much smaller than a dense one, but still, huge capacity is achievable with not a modarate sized SDR representation. Let's take the example of an SDR. N is the length of the array, while W is the number of active bits in the array. The sparsity is the proportion of these two variables. Numenta says that in the brain only 2% of the cells are active, so let's see the capacity of a bit array of this sparsity. In the example, the capacity of a 256 bit SDR with a population of 5 is 8,8 * 10e9. Storing this array would only require five integer values, which hold the indices of these active bits. In a bigger example, let's calculate the capacity of a 2048 bit SDR, which is the recommended minimum size by Numenta for an SDR. Ideally, an SDR should be much bigger, for example, 65k bits, but for the sake of implementation and visualization purposes, we will only work with the

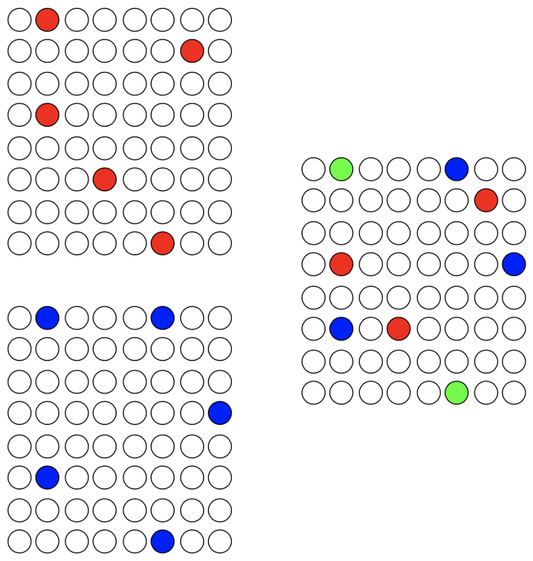

Figure 9: SDR overlap score, the green cells are the shared active bits of the two matrices

recommended minimum. This case the population should be 40. The capacity comes out to 2,37*10e84, which is more than enough to code any classification or regression task. In order to enable classification and regression there needs to be a way to decide whether or not two SDRs are matching, so if the HTM network encountered such an SDR before. This is what SDR comparison is for. In Figure 9 there is an example of a two SDR overlap. In the example, there are two random SDRs, and the third is the overlap between them, which is a simple AND operation between the two inputs. The population of the overlap is the overlapping score for these two compared SDRs. With two dense arrays, the expected overlap score is high. In the rare case, the overlap is much more unlikely, so the overlap scores are typically lower. To decide whether or not overlap is a match or not a threshold is needed called θ. If the population of the overlap array is higher than the threshold, it is a match. Otherwise, these are two different SDRs. The accidental overlaps in SDRs are rare so the matching of two SDRs can be done with high precision. The rate of an SDR matching as a false positive is meager. The sparseness of the SDRs also makes for good noise tolerance properties. Presenting large amounts of noise to the second SDR has little effect on the overlap score of the comparison. In summary, an SDR representation is a reliable way of comparing representation for matching.

SDR overlap sets and subsampling

SDR overlap sets are an array of SDRs that fit a specific overlap score criteria to an SDR. Let's take an example, where a 256 bit SDR with five population. The following formula calculates the number of overlapping SDRs that have B overlap score: Wx over b x (n-w_x) over (w-b). With SDRs, there is another way of compressing data apart from only saving the current bit indices, which is subsampling. For comparing two SDRs, it is not needed to use every active bit. Take a subsample from the active bits, calculate the overlap score and the chance of false positives will not increase much while saving storage and computational capacities.

SDR sets and unions

The use of SDR sets is that these can reliably represent a large number of SDRs. Let's say that we have seen 100 SDRs and we want to decide about the new ones if any of the previous one is matching. The naive approach would be to take every SDR and compare it to the new one. Instead, a union can be built from these 100 arrays. Since these are sparse, the chance of colliding bits is low. Therefore the sparsity of the union will be significantly higher. Despite the union SDRs almost saturating this is still a reliable way of matching the SDR with the previous 100. If the new one matches with the union, it matches with one of the previously seen with high certainty. If it does not match with the union, it probably is not in the collection. This method only works for a limited sized collection, but effectively can reduce the computations required by many folds.

3.4 Encoding

Scalar encoding

The language of the HTM network is an SDR. There needs to be an encoder for it so that it can be applied to real-world problems. The first and most crucial encoder for the HTM system is the scalar encoder, and I will be focusing on this in the rest of this paper. An encoder for the HTM must meet the following criteria.

The principles of SDR encoding:

- Semantically similar data should result in SDRs with overlapping bits. The higher the overlap, the more the similarity.
- The same input should always produce the same output, so it needs to be deterministic.
- The output should have the same dimensions for all inputs.
- The output should have similar sparsity for all inputs and should handle noise and subsampling.

An example of such an encoder if shown in Figure 10. That is a scalar encoder for values between 0 and 100. There are 8 bits and 3 of them should be active. All values below 0 or over 100 are capped to the range of the scalar encoder.

3.5 Spatial pooling

Input and output

The spatial pooler is the first layer of the HTM network which comes after the encoder. It takes the SDR input from the encoder and outputs a set of active columns. These columns represent the recognition of the input. The columns also have a normalizing effect. There are two tasks for the spatial pooler, maintain a fixed sparsity and maintain overlap properties of the output of the encoder. These properties can be looked at like the normalization in other

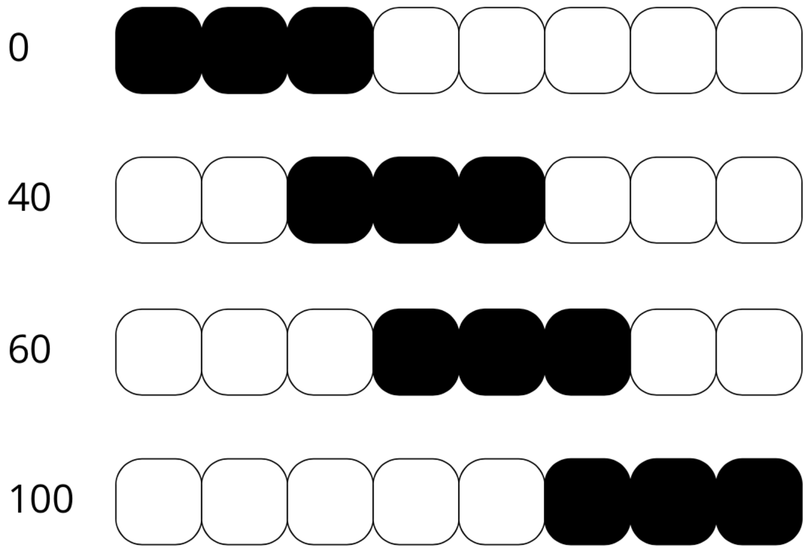

Figure 10: Scalar encoder

neural networks which helps the learning process by constraining the neurons behavior.

Columns

The column in a spatial pooler means a set of cell that shares the same segment connecting to the encoders SDR input. Presented with the input of these cells that are in a column would like to be activated simultaneously, this means that the column is activated.

Connections

The spatial pooler has connections between the SDR input cells and the spatial pooler columns. Every synapse is a potential synapse which can be connected or not depending on its strength. At initialization, there are only some cells connected to one column with a potential synapse. This already has some effect to fight overfitting the input data. If the potential synapse strength is over the threshold, it is connected.

At initialization, the synapses are initialized around the threshold strength so at the start half of the potential synapses are connected as well. The values of synapses strength are ranging from 0 to 1, and the threshold can be chosen anywhere between these two. The network's performance is invariant to this variable according to Numenta.

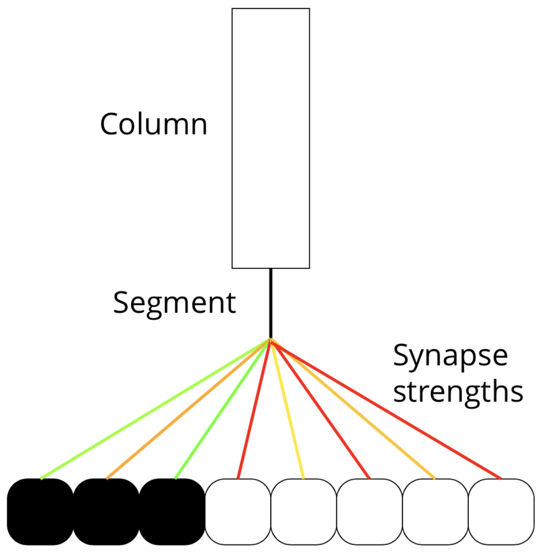

Figure 11: Spatial Pooler connections (green stronger connections)

Inhibition

For a given SDR input the column activations are calculated. The activations are the number of potential synapses that are connected to an active input cell. Usually a fixed threshold would decide whether or not a column activation score makes the column active or not, but in this case, the sparsity needs to be fixed. The solution is inhibition between columns, that only allow the top n most active columns to be active. These are the columns that can best represent the input SDR. It can be seen that this method not only maintains sparsity but ensures that the semantic information is the same. By changing only a small fraction of the active bits the activation scores of the columns would only change a little, thus not resulting in entirely different winners of the inhibition. A completely different SDR would result in different activation scores and different winners, meaning there's no semantic similarity between the two inputs.

Boosting

There is one problem with using inhibition for column activations; there can be columns that never get to be active therefore never have the chance to learn and increase their synapse values. This means that a lot of columns is not representing any information from the SDR inputs, leaving more information for fewer columns to represent. This results in worse performance than the more distributed column activations. What we are aiming for here is homeostasis between the column activations over time, so every column has some role and has a chance to improve synapse permanences. The solution to this problem is boosting, shown in Figure 12. The figure shows the cell activations, which is a moving average over the active cells. To counter the dominant cells the boosting factors are calculated. The boosting effect if shown on the same figure below with or without boosting. Without boosting some of the cells have high activations, while after boosting is applied the activation is more spread across the cells. The real effects are also shown on Figure 13 and 14 Boosting happens before inhibition. Every column activation is multiplied by, and then the inhibition occurs based on these adjusted scores. The more active columns have a lower than one boosting score while the less active columns have a higher than one boosting score. The boosting factors are the inverse of the active duty cycle of the columns, capped by some minimum and maximum value, for example, 0.5 and 2.

Learning

The randomly initialized spatial pooler already satisfies the two criteria, but a learning spatial pooler can do an even better representation of the input SDRs. As it can be seen on Figure 15 the weights are randomly initialized, and on

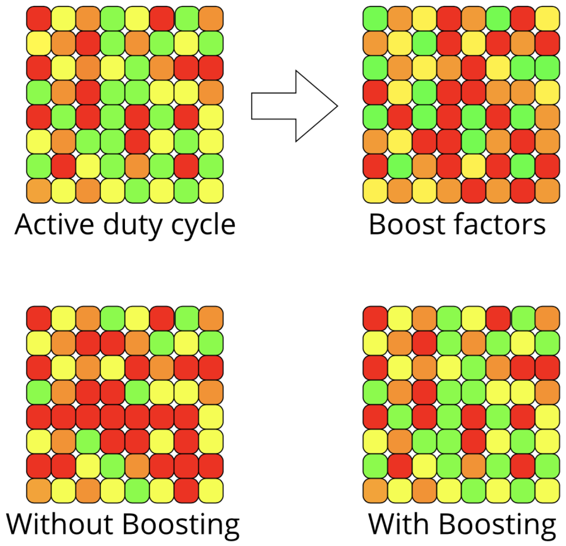

Figure 12: Boosting for equal cell activations (green high value, red low value)

the Figure 16 there are the learned weights. A lot of the not useful synapses are decrease and the important ones where the inputs are are strengthened. It uses Hebbian learning between the input cells and the spatial pooling columns. The Hebbian learning rule is simple; it is only applied to columns that are active. The synapses coming from these columns that are connected to active cells are reinforced, meaning their permanence values are increased. The synapses that are connected to inactive cells are punished, their permanence values are decreased. After learning the columns should have better activation scores because the connected synapses better aligned with those active cells that the column represents.

3.6 Temporal Memory

Input and output

The Temporal Memory receives the active columns as input and outputs the active cells which represent the context of the input in those active cells. At

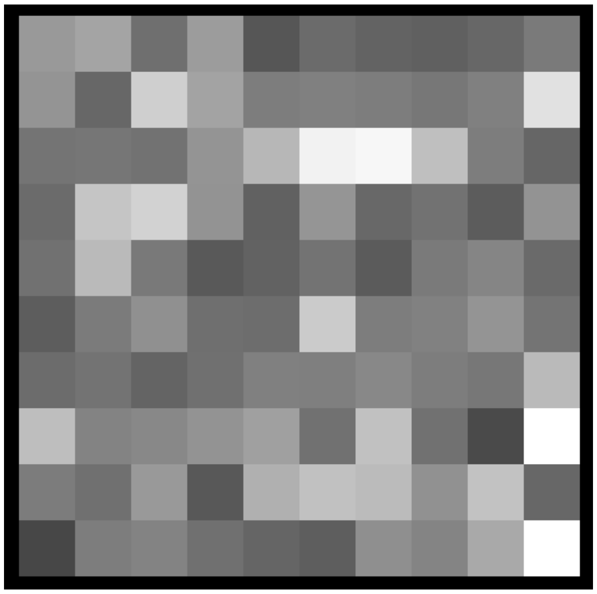

Figure 13: Boosting factors, some of the cells are boosted

any given timestep the active columns tell what the network sees and the active cells tell in what context the network sees it.

Cells

The cells in the Temporal memory can be either inactive, active, predictive or winner cells. A cell is activated if it is in ana active column and was in a predictive state in the previous timestep. A cell is a predictive cell one of its distal segments is activated because it recognized a pattern in the previous activation of the cells. Winner cells are chosen when there are no predictive cells in an active column. The winner cells can grow new segments and synapses to previous winner cells, therefore recognizing the patterns in the future.

The spatial pooling columns have one segment with potential synapses connected to the input SDR. In order to have temporal learning capabilities the HTM needs another mechanism called Temporal Memory which handles the sequential column activations. The Temporal Memory should expect what is going to happen in the next timestep, which columns will be activated. This is

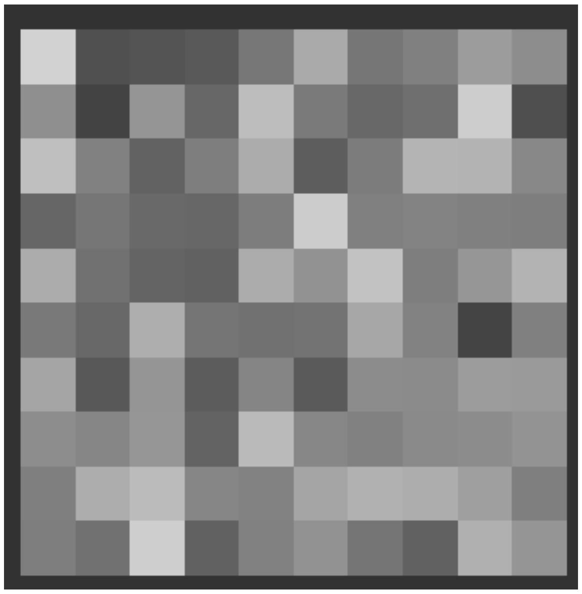
Figure 14: Activations with boosting, the cell activations are homogeneous

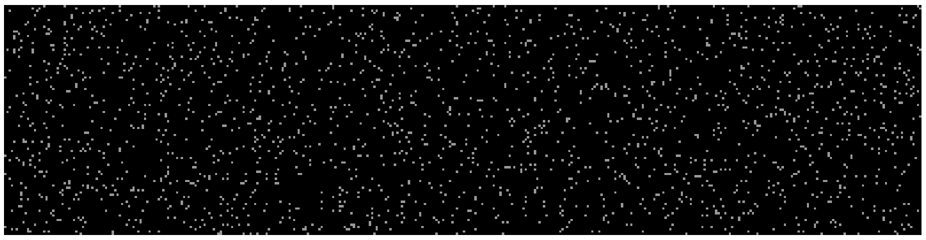
Figure 15: Spatial Pooler synapses before learning

implemented by multiple cells in every column, which not only have an active and inactive state but a predictive tool. This can be understood as preparing a cell for activation. Going further into the column structure all the cells share the same segment input from the encoder. This means that for a given input cell in one column want to be active at the same time. However, there's another inhibition inside the columns which prevent all cells from becoming active depending on the cells state.

There are three scenarios for a columns cells activation:

Figure 16: Spatial Pooler synapses after learning

- No cells were predicted inside the column
- Some cells were predicted inside the column
- Some cells were predicted inside a column that should not be activated in the first place

In the first case, there is no inhibition, and all the cells become active, meaning that the column encountered a sequence which it had never seen before. In the second case, the sequence is already known since there was an accurate prediction inside the column, only the predicted cell becomes active. Lastly, when there were cells predicted in the column, but the column itself is not active no cells will activate in this column, and the synapses that led to the false prediction will be punished.

Connections

The connections in the Temporal Memory between cells are created during learning, not initialized like in the spatial spooler. When there is an unknown pattern in the previous cell activations, then new winning cells need to be chosen, and new segments formed to those winner cells before.

Bursting

Bursting occurs when a column is activated, and there were no cells in a predictive state so the race for activation is a tie and all the cells become active

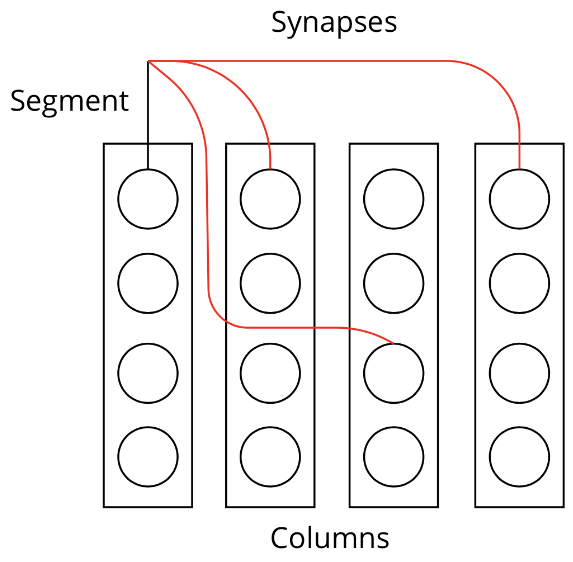

Figure 17: Distal connections in Temporal memory

shown in Figure 18. This is important because the activation of all cells is the union of all the context representation that could have come before. Since we do not know this pattern yet, we do not have a context representation for it. Therefore a good choice would be to express all the possible contexts that could come before, as all the cells activated.

Winner cells

To later recognize this pattern a winner cell is needed to choose to represent the new pattern the network encountered. The winner cells are chosen based on two factors, matching segments and least used cells.

- If there is a cell in the column that has a matching segment, it was almost activated. Therefore it should be the representation of this new context.
- If there is no cell in the column with a matching segment, the cell with the least segments should be the winner.

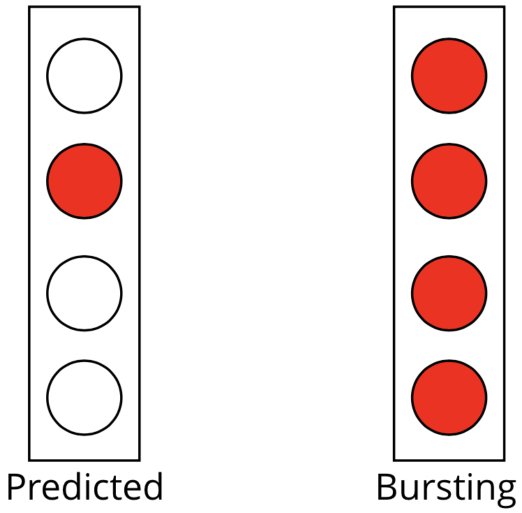

Figure 18: Predicted and bursting column

Training

The training happens similarly to spatial pooler training. The difference is that one cell has many segments connected to it, and the synapses of these segment do not connect to the previous layer's output but other cells in the temporary memory. The learning also creates new segments and synapses to ensure that the unknown patterns get recognized the next time the network encounters them.

The synapse reinforcement would be the same as the spatial pooler if the cell were correctly predicted in the column. The segment that led to the prediction of the cell has updated the synapses that were connected to active cells will increase in permanence, and the ones that were not will be decreased. Also if there were not enough active synapses, the network grows new ones to previous active cells to ensure at least the desired amount of active synapses.

If the cell is a bursting cell, then the learning is different in this case. Since there were no segments that were activated by synapses, there needs to be a new segment that will recognize this unknown pattern. First, there is the question of which cell should be active next time the network encounters the same pattern.

This is called a winner cell, and only these will take part in the learning process. Correctly predicted cells are automatically winner cells as well, so those always learn. The winner cell can be chosen in two ways. There could be equal segments attached to the cell, which means they have a slightly less activation to become activated but still pass the matching threshold. In this case, the segment almost matches the input so there only needs to be a slight change to the synapses in this matching segment, some new synapses to the previous winner cells.

In the case where there are no equal segments connected to the cell then there needs to be a new segment that will recognize this unknown pattern. The winner cells should be the one that has the least segments so that no cell would be responsible for detecting too many patterns. This is an equalizer of responsibilities across cells. The new segment will connect to some of the winner cells in the previous timestep.

4 Proposed work

First I had to get familiar with the concept of the HTM network. For this the two most helpful sources were the Numenta Biological and Machine Intelligence online book[Haw+16] and the HTM school video series created also by Numenta.[Num18] After reading and watching these material I wrote a text prediction program using their NuPIC package, which could predict the coming characters with high accuracy given that it was only one epoch training. However I thought that a different implementation could achieve better results in terms of performance. So I strated to lay out my plans about rebuilding this structure using matrix and tensor operations using the math package for Python called NumPy. First and SDR encoder for scalar input was created, then the Spatial Pooler with training. Then I started to work on the Temporal Memory. After a couple of months the basic network was implemented and it could read a sequence of data and output the active cell states. One holdup in the development was the growing of segments in the Temporal memory, which proved to be a much more complex operation than anticipated. During the debugging phase new functions were added to the network, like restoring a trained model and visualizations about different network metrics. When the network was finally capable of learning one stream of sequence data, then it was time to make it a multidimensional data stream network. For this only the first and layst layer of the network needed change, since the rest of the network uses SDR representations. With this the network could learn from a bigger stream of data, using the same number of internal cells, since the capacity of the SDR representations are so high. My plans about accelerating the network by using sparse tensors and graphical processors were hindered by the Temporal Memory problems and also the lacking implementation of sparse tensor packages for Python. Finally an LSTM baseline network were created to measure the results. These networks were created using the Tensorflow Keras package.

The training consists of test with multidimension scalar input sequences. These are created from a sum of different frequency of sine functions with added noise. The HTM network needs to predict the scalar value in the next time step receiving the current scalar value and also using its own SDR context representation. The LSTM network uses the last few inputs and has to predict the next scalar values also for every dimension.

The results show that both the networks are capable of learning the undisturbed sequences. However the HTM network seems to handle the noisy inputs a little better. Here are the results of the experiment:

5 LSTM baseline

5.1 Network structure

The LSTM baseline network had 4 configurations for learning multidimensional sinusoid sequences. For this one and two layered and 64 and 256 size LSTM cells were used. The input dimension of the sequences contained 100 different frequiencies of these sinusoids. After the LSTM cells a simple linear feedforward layer predicted the next element for every sequence dimension. The LSTM used the previous 20 samples form the input sequence to determine the next element. Below are the training times and losses of the training of the LSTM network configurations.

LSTM configurations

Config	Number of layers	Size	Training time	Training Loss
LSTM 1-64	1	64	50.1s	0.5493
LSTM 1-256	1	256	50.1s	0.1356
LSTM 2-64	2	64	100.15s	0.7071
LSTM 2-256	2	256	100.15s	0.1356

Figure 19 20 show three of the 1000 dimensions of sequences, the the network should've learned. The pictures show that the network couldn't fully learn all the noisy transitions since the blue (training) and green (predicted) lines don't overlap fully.

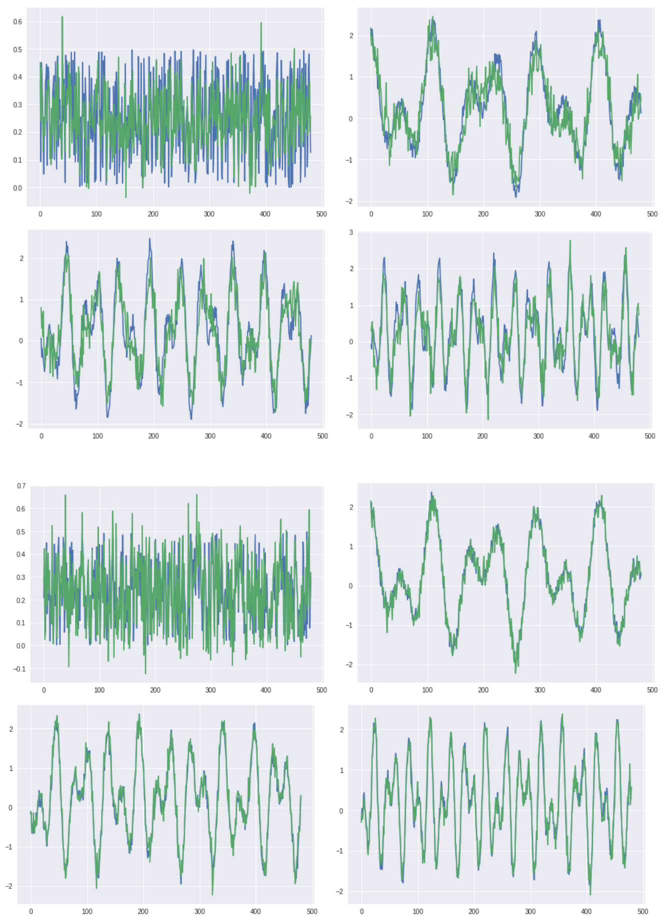

Figure 19: LSTM 1 Layer (64 (above) and 256 (below) size)

6 Methodology

Following the results of Numenta, first, an SDR scalar encoder was implemented, then a spatial pooler followed. The Temporal Memory first version used dense matrices then the sparse implementation was executed as well. Since the Temporal Memory was the computationally heavy part, the sparse matrix representation had a significant increase in performance so the network could be much. Furthermore, I would like to use a sparse representation for all layers in the future. A standard perceptron classifies the output of the network with softmax output. The implementation of the output processing by Numenta is different;

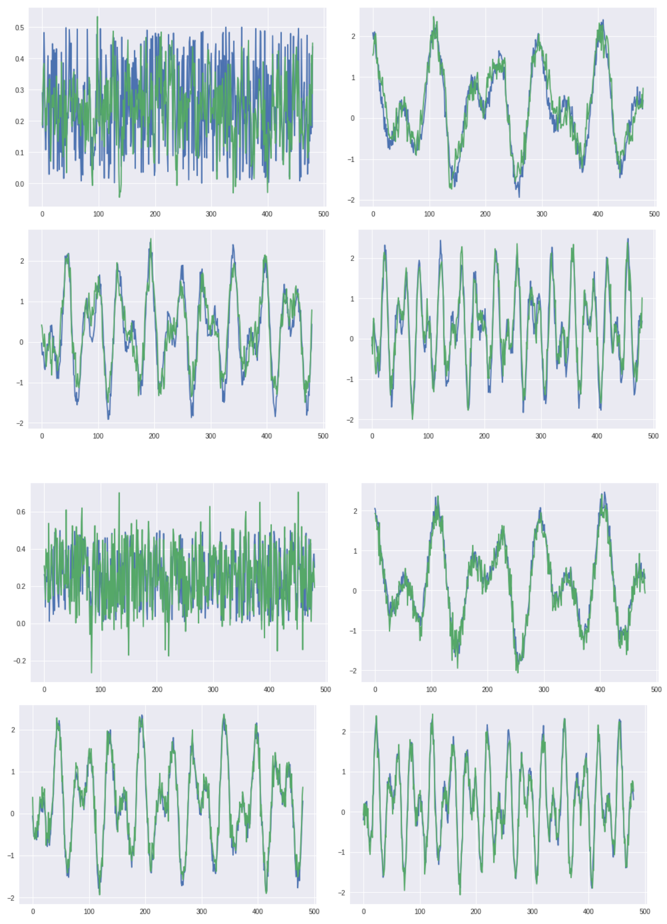
Figure 20: LSTM 2 Layer predictions (64 (above) and 256 (below) size)

they use an SDR classifier at the end. The documentation on this part is vague so the functionality of this part is not well understood but will take part in future developments with a sparse matrix representation as well.

6.1 Scalar encoder

The scalar encoder makes the following steps:

- Cap the value to the value range.

- Get the number of buckets from the input size and the number of active cells.
- Find the fitting bucket to the input value.
- Encode the bucket into an SDR with the first active cell at the bucket index.
- Return the SDR.

6.2 Spatial pooler

The Spatial Pooler makes the following steps:

- Initialize SP with potential synapses.
- Calculate segment activations from input SDR. (matrix multiplication between input vector and SP synapse matrix)
- Multiple activations by boosting factors.
- Choose the top N activations.
- Activate the columns that are connected to those segments.
- Update column activity and boost factors.
- Update synapse strengths with Hebbian learning.

6.3 Temporal memory

The Temporal Memory makes the following steps:

- Initialize the TM synapses with no synapses.
- Calculate segments activations from previous active cells. (tensor multiplication between the previous active cells and the TM synapses
- Determine matching and active segments depending on which threshold the activations on the segments pass.

- The cells that have an active segment become predictive state.
- The cells that were previously predictive and are in an active column should be active.
- These cells are automatically winner cells.
- All the cells in the columns that are active but had no active segments are bursting cells and become active.
- If the column has a matching segment, then the segment with the maximum activation wins.
- If the column has no matching segments the cell with the least segments becomes active.
- The cell connected to the winner segment is a winner cell.
- The segments that are winner segments update the synapses connected to them and grow new segments connected to the previous winner cells.

6.4 Linear layer

The linear layer is a one layer feedforward neural network with linear activations from the TensorFlow and Keras packages.[Mar+15][Cho+15]

7 Experiments

The HTM network should be able to learn the same sequence as the LSTM network, so the same data is passed on. The network consist of a Scalar Encoder with 100 bits for each encoded value. Next a Spatial pooler with 100 columns. Then a Temporal memory with 10 cells in each columns. Last the same feedforward neural network with linear activations for fair comparison.

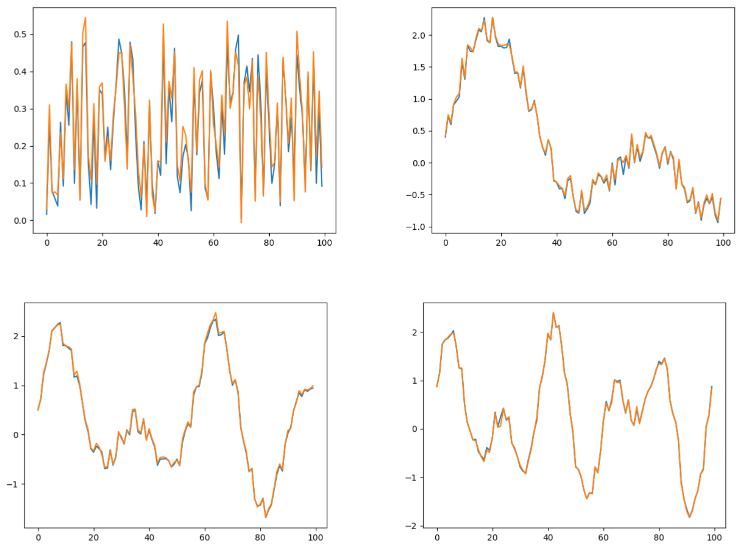

Figure 21: HTM layer output, 100 columns

The network as shown in Figure 21 could learn the patterns seemingly better. The overlap between the blue (training) and orange (prediction) lines is high, which means accurate prediction. The training time for this is 160s, with the three Temporal Memory epochs consuming most of this. It is worth noting that the average bursting rate, which can be translated to the network's confidence in the predictions significantly drops after the first epoch. At first all the columns bursts all the time since there are no learned connections to predict cells. This was an average 100 bursting cells in the first epoch. Then at the second pass the averege drops to 2 bursting columns and stays there in the third epoch as well. This means that the columns rarely burst, mostly only at the first input. This means that most of the necessary connections are formed during the first epoch, and then not much is needed to change in the TM snypases. This means that the HTM is capable of the desired one shot learning. The loss of the learning drops to low values which are in the range of 0.001 for mean squared error.

8 Future work

In this paper, the optimization of the HTM network consisted of rethinking the algorithm regarding using matrix operations and using those to accelerate the HTM network.

Since the Temporal Memory needed the most computational power, this is the first part that is needed to be implemented using sparse tensors. However, when increasing the network size, the other parts of the HTM network can be also computationally so expensive that it is worth implementing those sparsely as well.

In the future, I would like to extend the sparse implementation to the whole network so that the network can scale up optimally.

Furthermore, I would like to add new functionality to the network as well. First, an SDR classifier needs to be made, but the lacking documentation makes it hard to rework the algorithm into sparse matrix operations. Then to use this method in as many domains as possible a lot of new SDR encoders are needed, like date, sound, image, video and GPS encoders.

Since Numenta is continually evolving its HTM theory, it is most likely they will introduce new functionalities into NuPIC as well, so these need to be also reimplemented into the optimized HTM network. One clear path they are going now is the way grid cells code the spatial movements of the person and how these cells can also have something to do with the way people make a mindmap of the objects and concepts they know.

9 Summary

In conclusion, I presented the HTM networks theory and the beneficial properties of it. Proposed a way to execute it using matrix and tensor operations and implemented such a network using Numpy and Tensorflow. Compared the results to a baseline LSTM networks performance on noisy multidimensional periodic sequences, which showed good results in terms of accuracy and similar training times. For fair comparisons I tried to match the hyperparamters of the two networks. The results showed the true learning capabilities of the HTM network.

However the scalability of the network could not been achieved in the scope of this paper, since the current sparse packages lacked either sparse tensor operations or tensor mutability. This leads to the path of reimplementing a sparse tensor package which enables both fast operations between tensors and also mutability for synapse updates. Without the sparse implementations the graphical processor implementation is not achievable neither, since the overhead of copying sparse matrices in dense format between the processor and the graphical processor outweigh the benefits of parallel execution. Graphical processor acceleration works for big batches of data, but for this case batch learning is not yet worked out. That is another hindering factor of such an implementation.

List of Figures

References

[Wer90] Paul J Werbos. “Backpropagation through time: what it does and how to do it”. In: *Proceedings of the IEEE* 78.10 (1990), pp. 1550–1560.

[L+95] Yann LeCun, Yoshua Bengio, et al. “Convolutional networks for images, speech, and time series”. In: *The handbook of brain theory and neural networks* 3361.10 (1995), p. 1995.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[KGV99] Richard Kempter, Wulfram Gerstner, and J Leo Van Hemmen. “Hebbian learning and spiking neurons”. In: *Physical Review E* 59.4 (1999), p. 4498.

[SSN12] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. “LSTM neural networks for language modeling”. In: *Thirteenth annual conference of the international speech communication association.* 2012.

[Tho+13] Chris Thornton et al. “Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms”. In: *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM. 2013, pp. 847–855.

[Cho+15] François Chollet et al. *Keras.* `https://keras.io`. 2015.

[Mar+15] Martın Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems.* Software available from tensorflow.org. 2015. URL: `https://www.tensorflow.org/`.

[Haw+16] J. Hawkins et al. “Biological and Machine Intelligence (BAMI)”. Initial online release 0.4. 2016. URL: `http://numenta.com/biological-and-machine-intelligence/`.

[Num18] Numenta. *HTM school*. 2018. URL: https://www.youtube.com/playlist?list=PL3yXMgtrZmDqhsFQzwUC9V8MeeVOQ7eZ9 (visited on 10/26/2018).

[Caj] Santiago Ramón y Cajal. *Neuron column*. https://upload.wikimedia.org/wikipedia/commons/5/5b/Cajal_cortex_drawings.png. [Online; accessed 2018.10.28.]

[Num] Numenta. *Real and artificial neural networks*. https://numenta.com/neuroscience-research/research-publications/papers/images/why-neurons-have-thousands.png. [Online; accessed 2018.10.28.]

[Ola] Christopher Olah. *Recurrent Neural Network unrolled*. http://colah.github.io/posts/2015-08-Understanding-LSTMs/img/RNN-unrolled.png. [Online; accessed 2018.10.28.]

Abbreviations

- RNN: Recurrent Neural Networks
- LSTM: Long Short-Term Memory
- HTM: Hierarchical Temporal Memory
- SDR: Sparse Distributed Representation
- SP: Spatial Pooler
- TM: Temporal Memory

Acknowledgements

The research presented in this paper has been supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.2-16-2017-00013).