Hierarchical Temporal Memory for Human Action Recognition

Andrew Srbic

School of ICT

Griffith University

A thesis submitted for the partial fulfilment of

Bachelor of IT Advanced with Honours

December 2012

Abstract

Hierarchical temporal memory (HTM) is a novel pattern recognition algorithm which is receiving increasing attention from the machine learning community. It is inspired by the current understanding of the structure and behaviour of the neocortex. This work focuses on the HTM cortical learning algorithms (CLA) proposed in 2010. To date, these algoritms have received little attention in the literature. In particular, there has been no specific evaluation of the CLA temporal pooler. This thesis seeks to address this lack by evaluating a combined spatial and temporal pooler on a difficult action recognition problem. The results indicate that the algorithm is not capable of handling such high dimensional video data. Nonetheless, some interesting behaviours are reported.

Contents

List of Figures

List of Tables

LIST OF TABLES

1 Introduction

Biologically inspired algorithms have been the focus of many recent machine learning publications. They combine the robust and powerful recognition capabilities of biological systems with the speed and scalability inherent to computer systems. Hierarchical temporal memory (HTM) is a biologically inspired algorithm developed by Numenta (1). The latest version of this algorithm, particularly the temporal pooler component, has seen almost no attention in the literature. Thus, a opportunity exist for a novel study into the performance of this algorithm.

The pattern recognition problem examined by Wang et. al. (2) provides a benchmark on which feature extraction algorithms can be tested. In that work the authors evaluated a variety of feature extraction techniques and reported their accuracy when used in a classification pipeline. The datasets used were videos of humans performing actions. This thesis aims to evaluate the performance of a HTM system as a feature extractor on a complex problem such as this. This required a HTM system to be implemented and evaluated in an exploratory study prior to applying it to the classification problem.

The motivation of the thesis is that, at the time of writing there has be no existing benchmark evaluating temporal pooler. Given this, the algorithm was evaluated in a exploratory fashion to determine its fitness on the action recognition problem.

This contribution of this study is to indicated that the temporal pooler implemented for this thesis would be unable to function adequately in a classification pipeline like that of Wang et. al. (2). Interesting behaviours observed in this exploratory study are presented in this thesis.

In the next chapter a brief review of the relevant research areas will be presented. This will include a brief section on neuroscience to to give a background for several of the algorithms discussed in subsequent sections. After this will be a sectionintroducing the key machine learning paradigms. Given the broad scope of machine learning, this will be followed by only a selection of machine learning algorithms deemed relevant to this work. There will then be a review of recent literature relevant to this thesis.

The HTM chapter will extend the review of HTM papers given in the background chapter, giving a thorough explanation of the data structures and algorithms of the HTM spatial and temporal poolers.

The research methodology chapter will set out the plan which this research project follows. This includes a clear specification of the research problem and question and a section on research design. In the outcomes chapter the research question will be answered and the knowledge gained from the execution of the research will be discussed. Finally the concluding chapter will analyse the outcomes, summarise the work in this thesis and discuss future work in this research area.

2

Background

2.1 Outline

This thesis brings together work from several disciplines in machine learning as well as biology. This chapter will provide an outline of relevant fields of study in order to construct the conceptual framework into which HTM applied to human action recognition fits.

2.2 Machine Learning Background

Machine learning techniques can be applied successfully to almost any form of information. Its goal is to exploit the underlying statistical properties of data to produce a more useful form of information which is in some way representative of the original data. Many machine learning techniques draw inspiration from neuroscience.

In neuroscience, neurons are complex cells which are capable of information transfer and processing (3). Information is transmitted to them through dendrites connected to other neurons, sensory organs and the nervous system. Given sufficient electrochemical input through such dendrites a neuron will fire. Firing results in a charge being sent down the axon attached to it which in turn connects to many other neurons. These basic principles form the basis of many machine learning algorithms.

Mitchell (4) gives the following definition of machine learning: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at task T, as measured by P, improves with experience E." There is a large, diverse range of machine learning algorithms. This section will give an account of key machine learning paradigms and algorithms. The threads of work which lead to the various algorithms will also be discussed. This review will allow the distinguishing features of HTM to be illustrated.

Machine learning algorithms are typically run in two distinct phases; training and testing. It allows algorithms to be evaluated on novel inputs in a uniform fashion. In the training phase an algorithm is given a set of inputs which it uses to alter internal states such that it will produce a better output given that same input in future. This is done until the algorithm is fully trained, known as reaching a state of convergence. The testing phase typically commences after this. The algorithm should be used to generate outputs in the same fashion as in the training phase but internal states must not be modified. This preserves the state of the algorithm on completion of training which allows any number of novel test inputs to be evaluated in precisely the same way. The effectiveness with which the algorithm can do this from a converged state forms a useful performance metric which can be conveniently compared against different configurations of the algorithm, other algorithms and even other datasets.

The converged state refers to the global minimum of the algorithm's search space being reached. An example of a simple search space is shown in Figure 2.1. A search space is the set of all possible outputs for all possible inputs. Consider the function $f(x) = x^2$. To make $f(x)$ as close as possible to 0, x should be as close as possible to 0. In the types of problems which machine learning algorithms typically deal with the goal is the same - find the configuration of the algorithm's states which results in either a maximum or minimum for as many inputs as possible. Typical machine learning problems only differ from this example in that they involve hundreds or thousands of input dimensions and potentially as many output dimensions. Convergence can often be recognised when an algorithm ceases to substantially modify internal states during training as no better solution can be found.

Machine learning algorithms typically use one of two learning methods; supervised or unsupervised. The key difference between these is the information they get about training data. Supervised learners have full access to the classification of a given input - unsupervised learners do not. Supervision seeks to improve classification rates by allowing algorithms to learn the relationship between an input and its class. Given several inputs labelled "dog", and several labelled "cat", a supervised learner should

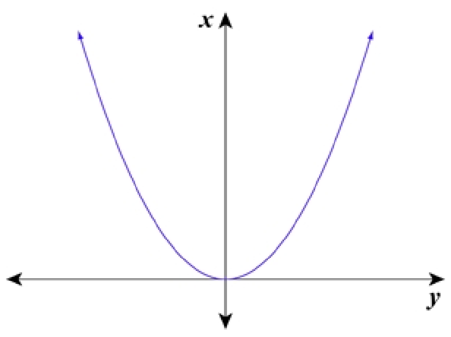


Figure 2.1: Simple Search Space - The minimum of this search space occurs at $x = 0$.

ideally be able to learn the attributes which are indicative of an input being a dog or cat respectively. It is only useful if the labels of the input are available.

When data labels are not available an unsupervised learning approach is appropriate. Its goal is to use the structure of data to generate similar outputs for similar inputs. This allows inputs to be roughly taxonomised into one of many structurally different outputs. The structure of the input as well as the corresponding output is difficult to conceptualise. It can be thought of as an algorithm's abstract idea of what makes up an input. It seems that this type of learning cannot achieve anything useful due to its inability to give a classification to input data. All it is doing is changing the form of the data. It becomes useful when its output is given to a supervised learning algorithm for classification. This can reduce the complexity of the input data before it is classified.

Some additional machine learning background is required before the machine learning algorithms are discussed; convolution and stacking. Convolution is a process used by feature extraction techniques and stacking hails from the deep learning sub-field of machine learning. These two concepts have seen widespread adoption in pattern recognition literature.

Convolution is a mechanism whereby large inputs can be handled in a memory and time efficient manner. The complexity of many machine learning algorithms typically scales non-linearly when the size of the input increases. Small increases in input size can lead to much larger increases in computational and requirements. Convolution solves this by taking smaller patches of data from the input. For example, given an 9 x 9 pixel image, instead of requiring the algorithm in use to cater to 81 inputs a patch size of 3 x 3 can instead be used requiring only 9 inputs. Clearly, this would require at least 9 patches to be sampled from the original image to include every pixel. Nonetheless, this process typically leads to reduced convergence time due to the reduced dimensionality of the search space. It is common for the convoluted patches to overlap each other to improve the probability that small, important features in the data are not cut off by the border of a patch. Convolving input data can greatly reduce computation complexity. Information lost in this process can be reconstituted by forming hierarchies.

Stacking refers to pattern recognition algorithms whose outputs are given as input to another pattern recognition algorithm. The goal is that the higher layer algorithm which receive input from another algorithm will learn better features. Algorithms arranged in this fashion are said to form a hierarchical structure. It is possible to form a hierarchy out of uniform or different algorithms. The higher level algorithms typically form a more abstract representation of the input data. This level of abstraction leads to the representation being invariant to noise and small differences in the input data. Given the same input several times with different noise introduced, the higher level algorithm's representation will vary less than the lower level one. This hierarchical structure can be extended such that the higher level algorithm receives inputs from several lower level algorithms. Additionally, any number of levels can be used to produce more and more invariant and abstract representations. Most problems and algorithms have a limit where additional layers do not further improve the representation of the original input. Many of these machine learning concepts draw inspiration from neuroscience.

2.3 Machine Learning Algorithms

The story of statistical modelling algorithms begins in 1957 with the perceptron (5). Figure 2.2 shows its basic structure which is inspired by the biological structure and behaviour of neurons. It consists of inputs which are connected to a neuron through weighted connections. The weighted connections are given a weight value, typically between 1 and -1. This is used to model the importance of the connected input. The sum of all weights multiplied by their connected input determines the input to the activation function of an output neuron. The result of the activation function is considered the output of a neuron. It uses a simple learning algorithm based on the difference between the actual outputs of the perceptron and the desired outputs. For each output neuron, all connected weights are either increased or decreased to make the actual output more like the desired output in future. Perceptrons are robust to noise but cannot be successfully applied to many machine learning problems. This led to the creation of multilayer perceptrons, known as feed forward neural networks.

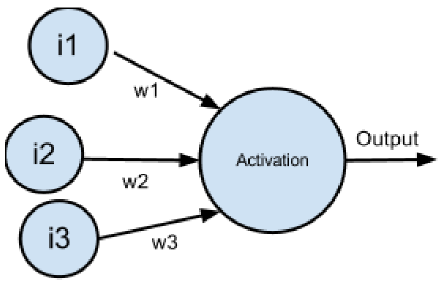


Figure 2.2: Perceptron - A simple one layer perceptron. The inputs are labelled i1 through i3, the weights are labelled w1 through w3.

Multilayer feed forward neural networks build (6, 7, 8, 9) on single layer perceptrons by adding an additional layer of "hidden" neurons between input and output neurons. An example is shown in Figure 2.3. This additional layer means that the output of input neurons is now the input to the hidden neurons and similarly the output of the hidden layer is the input to the output neurons. The way in which the activations in each neuron are calculated remains the same; the weighted inputs are summed and put through an activation function. One popular activation function is the sigmoid activation function (10) The learning method for the weights connecting to output neurons involves a simple derivative. However, weights between other layers can no longer be adjusted so easily. The learning method for the output weights required the desired output of the output neurons to be known. This is unknown for hidden neurons. Instead the back propagation algorithm (6) is used to update these weights. This is the most popular of the neural network algorithms. This type of neural network is applicable to a wide range of problems and is robust to noise.

There are many other variants of artificial neural networks (ANN) with different structures but they all utilise neurons with weighted connections. Perceptrons are considered a form of ANN along with their successors; Helmholtz machines and Boltzmann machines. Added to this are and Kohonen self organising maps. This list is not comprehensive. These are, however, the key neural networks which are relevant to this paper. As such a high level understanding of each of them is necessary.

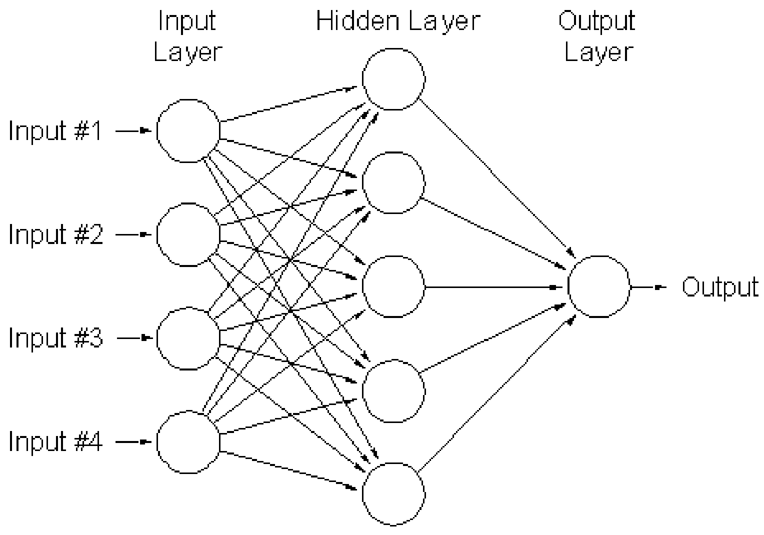


Figure 2.3: Neural Network - A three layer neural network.

Helmholtz machines (11) are a form of recurrent neural network (RNN). The key property distinguishing RNNs from feed forward neural networks is that the connections in an RNN form a directed cycle. This means that neurons, can influence themselves. An example of this can be see in Figure 2.4. This mechanism allows global information to be considered in calculations local to a single neuron, learning of sequences and several other applications. In a Helmholtz machine the recurrent connection occurs between bottom up neurons feeding into top down neurons and vice versa. Helmholtz machines also introduce an energy term.

The energy in a Helmholtz machine is a measure of how far the system is from the best possible state. To get the lowest possible energy in the system the surprise must be reduced. Surprise is a complex measure of how unexpected an input is. The wake-sleep learning algorithms is a derivative of the function of this energy and attempts to minimise it. This mechanism is also used in restricted Boltzmann machines (RBMs).

RBMs (12) combine an energy function with random variations on inputs among other mechanisms. Introducing the randomisation makes the network more robust. Real world inputs are rarely uniform and always contain slight variations. Training in this way makes that variation something that the network expects. It is common for RBMs to be stacked in a hierarchical manner.

Kohonen self organising maps (13) (SOM) solve a different problem to a typical

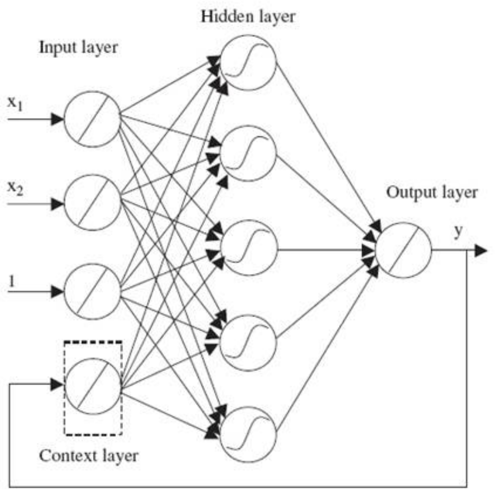


Figure 2.4: Recurrent Neural Network - A simple recurrent neural network. The output of the network forms part of the network's input. In this example it is done to give each neuron context information regarding what is occurring throughout the entire network.

neural network. They perform clustering on input data and learns in an unsupervised fashion. An important aspect of clustering is reduced data complexity. This will be discussed further in a review of recent action recognition literature. Figure 2.5 shows the structure of a SOM. Clustering is the process of turning an input into one of a predetermined number of outputs. These outputs are ideally as similar to the input as possible so the change from input to its cluster point causes as little error as possible. This is integral to the algorithm's success. If there is a substantial difference between the input and the cluster point then it is failing to produce accurate representations of it.

SOMs use a simple learning process which requires all the instances in a given dataset to be input to the algorithm multiple times. Initially, every output neuron is given set of randomly initialised weights. The number of weights is given by the size of the input. For example, if the input to the algorithm was a colour composed of a red, green and a blue intensity then there would be three weights. Afterwards, the iterative learning process involves selecting the output neuron which is most similar to the current input and recording it as the best matching unit (BMU). The measure of similarity is the Euclidean distance between each input and the weight connecting

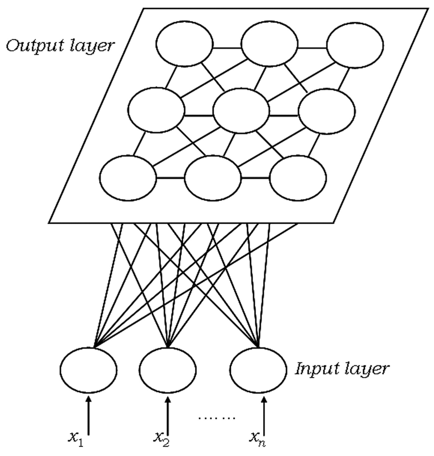


Figure 2.5: SOM - A self organising map. Each output neuron is connected to every input neuron.

that input to a given output neuron. The BMU and the neurons in a large neighbourhood area around it have their weights modified to be more similar to the input. The strength of this modification is determined by a Gaussian distribution centred around the BMU so neurons closer to it will be modified more than those on the edge of the neighbourhood area. The result of the process being repeated multiple times for each instance in the dataset is a map in which similar cluster points will appear next to each other. There will typically be more cluster points bearing similarity to inputs if they are more common than others. After training, selecting the best cluster point for a given input means simply selecting the BMU.

Common to all neural networks is the presence of weighted connections. These connections are used to increase or decrease the importance of a particular input to a particular neuron. The net effect of this is neurons which are sensitive to certain patterns and less sensitive to others. Several neurons combined together allows more complex patterns to be assembled and given as input to other neurons or output from the network. The key aspect to remember is that the input coming from a neuron is typically multiplied by the weight of the connection. This is not true for SOMs as they have a substantially different architecture to most neural networks. SOMs solve the clustering problem, as does another machine learning algorithm known as "k-means".

K-means clustering(14) seeks to bin n instances into k clusters. Selecting a cluster point for particular instances simply involves selecting the cluster point which is most similar to the input. The algorithm uses an iterative training process to determine what these cluster points should be. Initially, k random, cluster points, known as means in this algorithm, are generated. Each training instance is then associated with its closest mean. After this the iterative training process begins. Each iteration involves relocating the mean to the centre of all instances associated with that mean. This leads to some instances now being closer to a different mean and thus they have a new cluster point. This is repeated until convergence when the means are no longer being adjusted. K-means is simple to implement and efficient in most cases.

Support vector machines (15) leverage impressive learning mechanisms to linearly separate data. Linear seperation solve the classification problem. If a line can be drawn which separates all elements of one class from all elements of the other class then they can be easily classified. In addition to this, SVMs can also use non-linear separation methods to add dimensions to input data to improve the way in which it can be separated. SVMs make excellent classifiers but can only distinguish between two classes. In order to classify more classes, several SVM can be trained. Each one determines if an input either what it is looking for, or something else.

Principal component analysis (PCA) (16) is a mathematical process which transforms input data. If the data was plotted on a graph, the nature of the transformation would be to translate and rotate the axes to the centre of the data. This results in the data being distributed around the axes. This process is also known as whitening and is essential for independent component analysis (ICA).

ICA(17) is an advanced signal processing technique which seeks to extract source information out of one or more combined signals. It is impractical to perform ICA without having first performed PCA on input data as the algorithm typically fails. The algorithms uses a learned matrix to transform a combined signal, which may be an image or sound, into its source. The source is the algorithm's pure representation of what makes up the input and can be used as a feature. The training process of the algorithm is essentially a search problem for the matrix which best satisfies some constraint and produces good sources as a result. ISA(18) is an extension of ICA.

Another important machine learning algorithm is the Bayesian belief network (19). These networks leverage Bayesian probability theory to infer the probability of an event occurring. An extension of Bayesian belief networks is the dynamic Bayesian network which can infer probabilities on temporal data.

Hidden Markov models (20) are similar to dynamic Bayesian networks. They are capable of learning sequences. The key to the algorithm is the presence of hidden states. For every observed state, or effect, there is assumed to be a corresponding hidden state, or cause. The algorithm learns relationships between cause and effect as well as the temporal relationships between causes. The algorithm deals with a finite set of causes.

2.4 Hierarchical Temporal Memory

Hierarchical Temporal Memory (HTM) is a biologically inspired learning algorithm. The current algorithm is the latest of several generations, detailed in (1). There are many publications utilising the older versions of HTM but these can, for the most part, be considered irrelevant as the current HTM algorithm differs substantially from them.

The spatial pooler produces sparse codes, or features, from a given input. The process through which these features are produced is novel in machine learning literature. Its closest relation is a neural network such as a restricted Boltzmann machine (11) in that they both contain abstractions of neurons which connect to input data. One key difference is that connections can be in an on or off state rather than always on with some weighted input value. This, in combination with ihnibition separates the spatial pooler from any other machine learning algorithm.

The temporal pooler can be used to predict sequences of sparse codes and potentially clean up temporal noise. This allows HTM to be applied effectively to temporal datasets, such as videos. A useful property of the temporal pooler is that it is capable of making predictions about future inputs given some inputs in a temporal sequence. There is no existing algorithm in the literature which bears similarity the the temporal pooler so it is an entirely novel approach. The algorithms key distinguishing property is that new connections can be established as it learns in a a similar fashion to neurons in the brain. In contrast, a hidden Markov model (20) can only learn transitions between a finite set of states.

Mountcastle (21) expanded on the basic understanding of the neocortex. In his research he dissected neocortical tissue and observed columnular structures. The neurons appeared to be arranged into rough columns. In addition to being structured this way they also fired together in a roughly uniform manner, as a column. This forms an integral part of the HTM cortical learning algorithms as it too utilises a columnular structure.

Previous versions of HTM proposed by Hawkins and George (22) have utilised an integrally hierarchical structures. The most recent version can function as a single region. The older version used a hierarchical Bayesian network for learning and inference. The hierarchical structure of the inputs started with many very small low level sensors examining the input data. The output of these is passed up to progressively larger upper levels until a recognition node is reached. Numenta released software package of the old algorithm, known as NuPIC. It was not capable of learning effectively on high dimensional data. In a recent lecture, Jeff Hawkins stated that the new HTM was not designed to learn and predict high dimensional input data such as videos.

It should be noted that Hierarchical Temporal Memory including HTM Cortical Learning Algorithms is a whitepaper produced by a company with a commercial interest in the success of HTM. It does not provide any performance metrics on the algorithm, only pseudocode and biological justifications. There is a clear possibility that any positive statements regarding the performance of the algorithms will be considerably biased so care must be taken to recognise this and form an objective evaluation.

In Thornton and Srbic (23) the sparse encoding capabilities of the spatial pooler were evaluated. The results indicated that the algorithm is capable of producing what can be classified as good features. The lifetime and population kurtosis of the spatial pooler trained on natural images were compared against those produced by other sparse encoding methods. The result was favourable for the spatial pooler indicating that it excels at producing unique sparse codes.

Other recent works in HTM have focused almost entirely on the old HTM algorithms(24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36) and are therefore irrelevant to this work. Price (37) does provide an interesting study into the parallelisation of the new temporal pooler but does present any classification performance metrics.

2.5 Human Action Recognition

Wang et. al. (2) propose a common testing pipeline for feature extraction techniques and benchmark the previous state of the art of these techniques. This classification pipeline is comprised of feature extraction followed by clustering with K-means then bag-of-features Support Vector Machine (SVM) classification as shown in Figure 2.6. The datasets contain videos of human actions with varying classification difficulties. These datasets include, in descending order of difficulty, Hollywood2 (38), UCF sport action (39), YouTube (40) and KTH (41). Their results indicated that there was no clear winner among the feature extraction techniques on all datasets - those which performed well on some did not on others. The best result on KTH they produced was 92.1%.

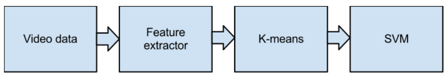


Figure 2.6: Wang Classification Pipeline - The pipeline as used in Wang et. al.

Le et. al. (42) utilises the classification pipeline and datasets from Wang et. al. (2) with a hierarchical Independent Subspace Analysis (ISA) feature extractor to achieve competitive results. This paper builds on previous work from Wang et. al. by replacing the benchmarked feature extraction methods with hierarchical ISA as shown in Figure 2.6. They show that their stacked ISA feature extractor performs consistently well on all datasets. At best, using norm thresholding and dense sampling, they achieved a classification accuracy of 93.9%.

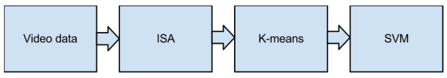


Figure 2.7: Le Classification Pipeline - The pipeline as used in Le et. al.

There are several recent publications which achieved impressive results on the KTH dataset. Zhang et. al. (43) used context constrained linear coding to achieve 95.06%. Their pipeline was the same as that of Wang et. al. and Le et. al. so any results produced in this thesis will be comparable to theirs. Sadanand and Corso (44) presented an impressive classification accuracy of 98.2% but the pipeline they used differed substantially with their action bank approach. This makes their result interesting but irrelevant. Both Wang et. al. (45) and Shabani et. al. (46) achieved approximately 93.9% accuracy. These results are inferior to that of Le et. al. and can thus be discarded. Castrodad and Sapiro (47) used sparse modelling to attain a classification accuracy of 97.9%. This is the current state of the art recognition rate against which HTM will be compared.

3

Hierarchical Temporal Memory

Hierarchical temporal memory is a biologically inspired learning algorithm developed by Numenta (1) and is composed of a hierarchically structured collection of spatial and temporal poolers. Its goal is to extract the underlying structure from input data, form a useful representation of it and predict future inputs. The spatial pooler forms a sparse distributed representation of input data which the temporal pooler then makes prediction on. This chapter will draw from Numenta's HTM cortical learning algorithms whitepaper (1) to give an overview of these components.

3.1 Structure

A HTM system is made up of several levels of data structures. At the top level there is a hierarchy. A hierarchy is is one or more regions linked together in a hierarchical structure such that the output of a lower level region forms the input to a higher level region. Each region can function independently and is composed of a number of columns. These columns are arranged in a rectangular shape over the input to the region. There are weighted connections, or synapses, leading from the input to the columns. Additionally, each column also contains one or more cells. The cells in turn have one or more dendrite segments. Dendrite segments are use to combine synapse connections from a set of nearby cells. The purpose of these synapses differs substantially from those which connect columns to the input. The spatial pooler utilises columns and synapses connected to the input while the temporal pooler uses the cells inside each column and their dendrite segments, including the synapses the segments contain. Because each algorithm operates on separate data structures they can be trained in parallel or in sequence. Figure 3.1 shows Numenta's visualisation of a HTM region.

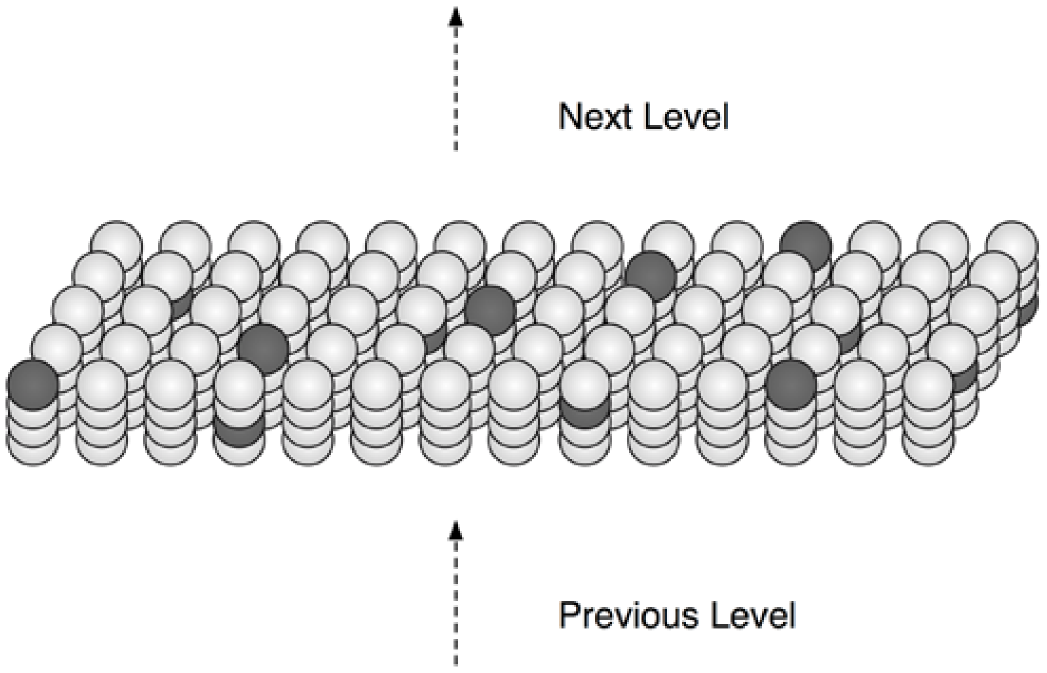


Figure 3.1: A Single HTM Region - This image depicts a HTM region with four cells in each column. Some cells are active in this example. Source: (1)

3.2 Spatial Pooler

3.2.1 Overview

The spatial pooler creates a new representation of input through a process of activation and inhibition. This part of the algorithms takes inspiration directly from neuroscience. In the spatial pooler, columns become active when they are have sufficient connections to inputs which are exhibiting activity. After activation, inhibition is the key to the sparse distributed representation. It causes only a small portion of the original active columns to be active.

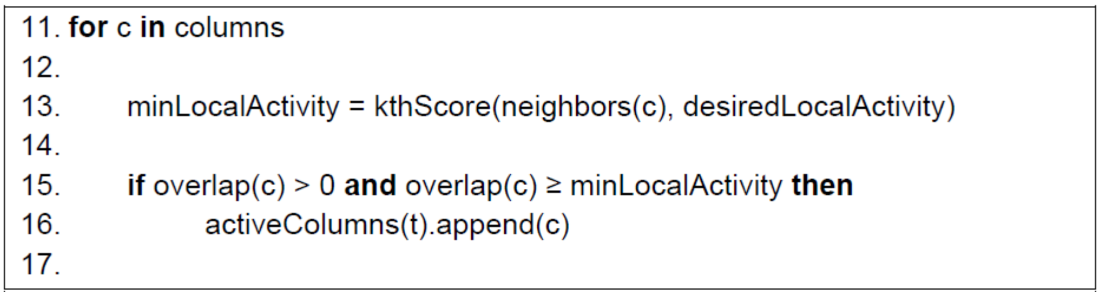
3.2.2 Algorithm

Activation of a given column is determined by the strength of the inputs connected with active synapses. An active synapse is one that has a permanence value higher than minOverlap, an arbitrarily set threshold (see line 7, Figure 3.2). Numenta's spatial pooling algorithm was intended to only deal with binary inputs with only two possible states. Thornton et. al. (23) extend this functionality to being capable of reading grey-scale inputs with 255 possible values with augmented spatial pooling. In this modified version of the algorithm a column is considered active when the intensity of all of the inputs with active synapses is above the mean of the entire input. An important aspect to note is the way in which the synapses are randomly initialised. Each column is more likely to make connections with inputs directly below them. This causes columns to be more sensitive to nearby inputs and makes them behave like local receptive fields. Figures 3.2, 3.3 and 3.4 shows pseudocode for the first, second and third phase of the spatial pooler. See Numenta's whitepaper (1) for a detailed explanation of each method in the pseudocode.

```
1.  for c in columns
2.
3.          overlap(c) = 0
4.          for s in connectedSynapses(c)
5.              overlap(c) = overlap(c) + input(t, s.sourceInput)
6.
7.          if overlap(c) < minOverlap then
8.              overlap(c) = 0
9.          else
10.         overlap(c) = overlap(c) * boost(c)
```

Figure 3.2: Calculate Overlap - Pseudocode for phase one of the spatial pooler. Source: (1)

The inhibition mechanism (Figure 3.3) suppresses the activation of cells near what are deemed to be the most active cells. It can cause some columns to become useless. If they form similar connections to a nearby column with more active synapses on initialisation then they might never enter an active state. To counter this a boosting mechanism is introduced (Figure 3.4, lines 28 - 36). Boosting monitors a column's past activity and if it does not become active often enough it is given an artificially higher activation level. There is another form of boosting which temporarily increases the permanence values of synapses connected to columns which do not become active prior to inhibition often enough. Both of these mechanisms ideally lead to the column surviving inhibition in future iterations and contributing to the sparse distributed representation of the input. Should a column survive the inhibition process, all of its synapses' permanence values are incremented if they are connected to active inputs or decremented if they are not (Figure 3.4, lines 18 - 26). These mechanisms occur over three phases; calculate overlap, inhibition and learning.



```
11. for c in columns
12.
13.       minLocalActivity = kthScore(neighbors(c), desiredLocalActivity)
14.
15.       if overlap(c) > 0 and overlap(c) ≥ minLocalActivity then
16.             activeColumns(t).append(c)
17.
```

Figure 3.3: Inhibition - Pseudocode for phase two of the spatial pooler. Source: (1)

3.3 Temporal Pooler

3.3.1 Overview

The temporal pooler builds upon the column activations of the spatial pooler. It uses these activation patterns to form sequences with the cells inside each column. This columnular structure takes inspiration from neuroscience where neurons were observed in structural columns (21). Cells in a columns will only become active if the column itself is in an active state as determined by the spatial pooler. Its goal is to form predictions regarding whether or not a cell will be active in the next iteration. The advantage multiple cells in each columns brings is a representation of context. Each column would typically be part of many different sequences so the cells allow a particular route to be taken through the column. This prevents a sequence from suffering from the interference of other sequences using the same column as they are independent of each other. Figure 3.5 depicts the active and predictive state of a HTM region. This

```
for c in activeColumns(t)

    for s in potentialSynapses(c)
        if active(s) then
            s.permanence += permanenceInc
            s.permanence = min(1.0, s.permanence)
        else
            s.permanence -= permanenceDec
            s.permanence = max(0.0, s.permanence)

for c in columns:

    minDutyCycle(c) = 0.01 * maxDutyCycle(neighbors(c))
    activeDutyCycle(c) = updateActiveDutyCycle(c)
    boost(c) = boostFunction(activeDutyCycle(c), minDutyCycle(c))

    overlapDutyCycle(c) = updateOverlapDutyCycle(c)
    if overlapDutyCycle(c) < minDutyCycle(c) then
        increasePermanences(c, 0.1*connectedPerm)

inhibitionRadius = averageReceptiveFieldSize()

```

Figure 3.4: Learning - Pseudocode for phase three of the spatial pooler. Source: (1)

is the result of running the temporal pooler on column activations from the spatial pooler.

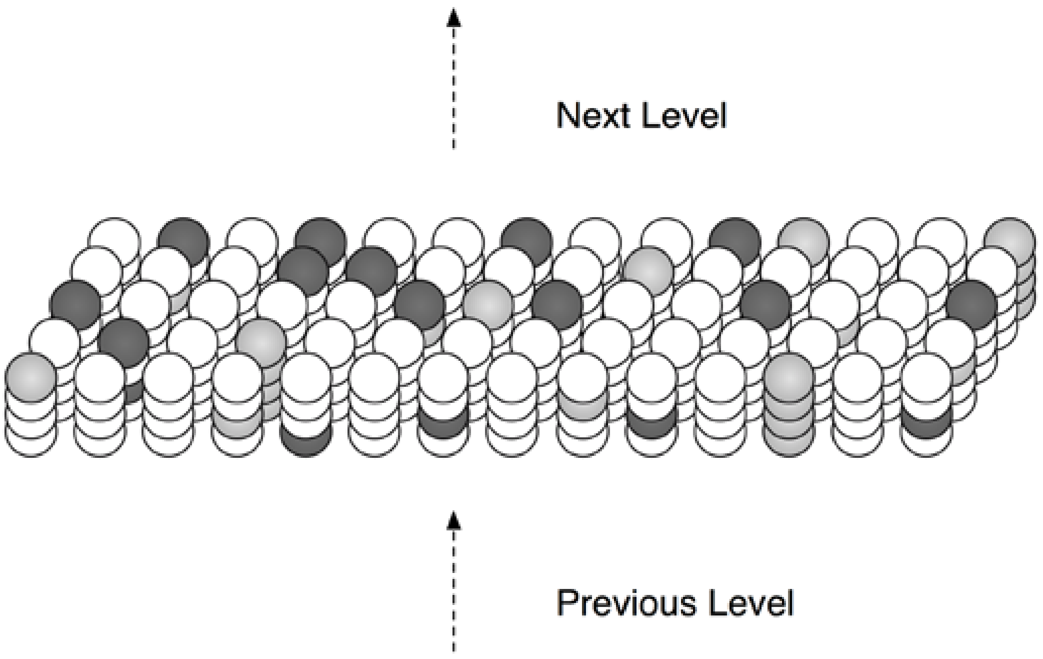


Figure 3.5: Temporal Pooler Predictions in a HTM Region - This image depicts the active (light grey) and predictive (dark grey) cells in a HTM region after temporal pooling of spatial pooler column activations Source: (1)

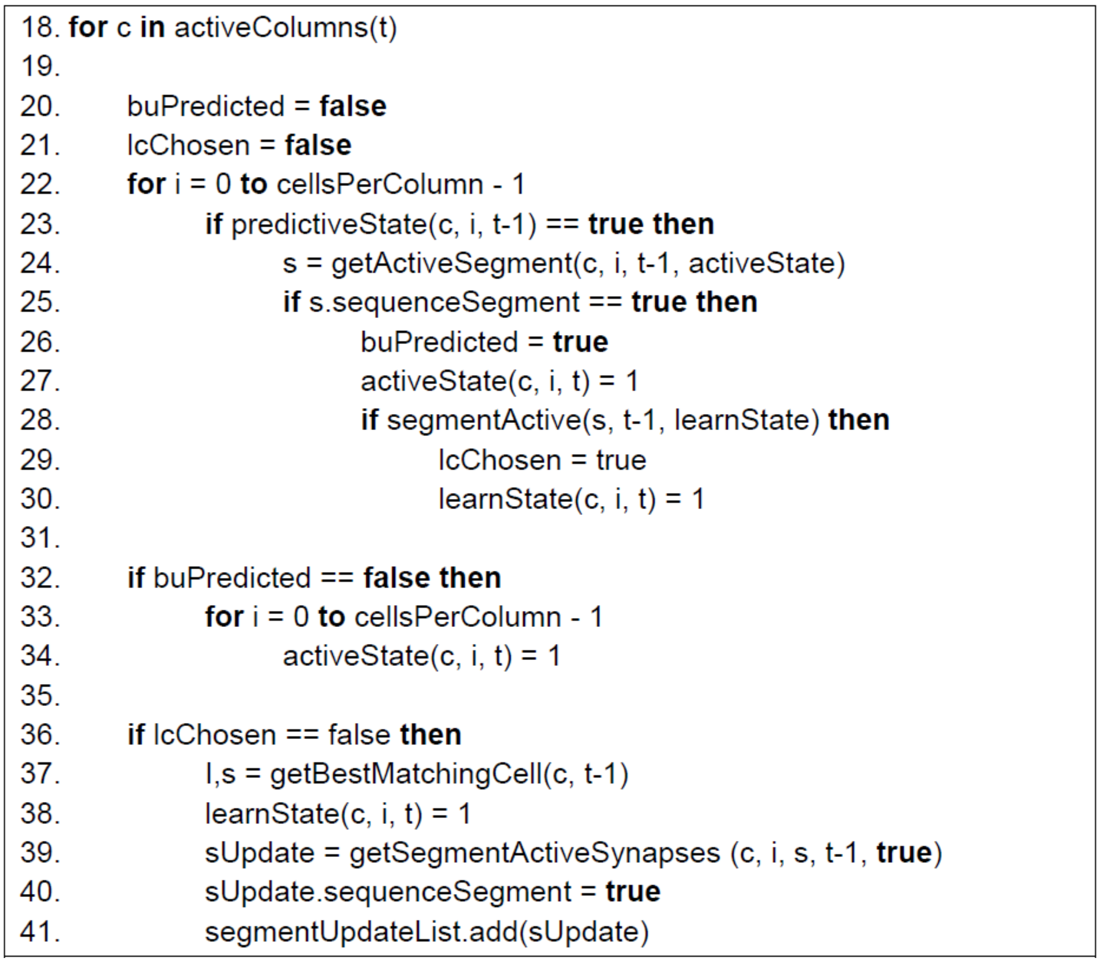
3.3.2 Algorithm

Initially, the temporal pooler contains no synapses. As new sequences are encountered they are established between cells and dendrite segments. Each cell can be in an active, learn and predictive state. These states are not mutually exclusive. An active state means that a cell is considered active in the current iteration. The learn state indicates that the given cell should be adapted at the end of the iteration. Predictive states are indicative that a cell will be active at some stage in the future. Additionally, the dendrite segments in each cell can be in an active or learn state. These states are used in the calculation of a cell's state. Dendrite segments also have an important sequence segment flag. This indicates that a dendrite segment is predicting that the cell it resides in will be active in the next iteration should the segment also be in an active state. The temporal pooler has three phases; calculate active states, calculate predictive states and learning.

The first phase, shown in Figure 3.6, determines if a cell should be in an active or learn state in the current iteration. If a column is active (Figure 3.6, line 18) then there are several possibilities for the activations of the cells it contains. If no cells in this column are in a predictive state from a sequence segment then every cell is put in an active state (Figure 3.6, lines 32 - 34). If such a cell does exist, then it is given an active state (Figure 3.6, lines 23 - 27). If the dendrite segment of the predictive cell is also in a learn state then that cell is also put in a learn state (Figure 3.6, lines 28 - 30). Otherwise, a learning cell is selected out of the active cells in the column based on which one has the dendrite segment with the most connections to active cells (Figure 3.6, lines 36 - 41). Such a cell should be predicting activity in this iteration so it is flagged for learning and a segment update is queued up for it.

Segment updates are data structures which contain information on how a dendrite segment should be modified. There are several possibilities for the update. It will create a new dendrite segments if there are none present in the cell. In this case, the segment can optionally be flagged as a sequence segment in the update. Otherwise, an index will be provided for a specific segment to update. It will also include a list of the active segments in that segment from that iteration. The list is required because multiple segment updates may be queued up for a single segment so they will be able to modify only the synapses which were present in the iteration of the activity. When new synapses are to be added they will be connected with cels in the learn state in the given iteration. These learn state cells must also be within a certain distance of the segment being updated.

In the second phase, depicted in Figure 3.7, the predictive state of each cell is calculated. A cell is placed in a predictive state if one of its dendrite segments is active (Figure 3.7, line 42 - 45). Such a segment becomes active by having sufficient active synapses with permanence values above a minimun threshold. An active synapse is one connected to an active cell in the current iteration and the threshold is an arbitrarily set paramater for the entire HTM system. Should this be the case, the segment is marked to be reinforced in the learning phase with a segment update (Figure 3.7, lines 47 - 48). Another segment update is also added which seeks to cause this predictive state cell to become predictive earlier (Figure 3.7, lines 50 - 53). It does so by adding synapses



```
for c in activeColumns(t)

    buPredicted = false
    lcChosen = false
    for i = 0 to cellsPerColumn - 1
        if predictiveState(c, i, t-1) == true then
            s = getActiveSegment(c, i, t-1, activeState)
            if s.sequenceSegment == true then
                buPredicted = true
                activeState(c, i, t) = 1
                if segmentActive(s, t-1, learnState) then
                    lcChosen = true
                    learnState(c, i, t) = 1

    if buPredicted == false then
        for i = 0 to cellsPerColumn - 1
            activeState(c, i, t) = 1

    if lcChosen == false then
        l,s = getBestMatchingCell(c, t-1)
        learnState(c, i, t) = 1
        sUpdate = getSegmentActiveSynapses (c, i, s, t-1, true)
        sUpdate.sequenceSegment = true
        segmentUpdateList.add(sUpdate)
```

Figure 3.6: Calculate Active States - Pseudocode for phase one of the temporal pooler. Source: (1)

to the most active segment in the previous iteration. This causes the segment to make predictions about predictive states instead of active states, making the predictions occur further and further back in time.

```
42. for c, i in cells
43.     for s in segments(c, i)
44.         if segmentActive(s, t, activeState) then
45.             predictiveState(c, i, t) = 1
46.
47.             activeUpdate = getSegmentActiveSynapses (c, i, s, t, false)
48.             segmentUpdateList.add(activeUpdate)
49.
50.             predSegment = getBestMatchingSegment(c, i, t-1)
51.             predUpdate = getSegmentActiveSynapses(
52.                             c, i, predSegment, t-1, true)
53.             segmentUpdateList.add(predUpdate)
```

Figure 3.7: Calculate Predictive States - Pseudocode for phase two of the temporal pooler. Source: (1)

The final phase, learning, implements the segment updates queued up in the other two phases. The pseudocode for this is shown in Figure 3.8. If a cell is in a learn state then the segment updates are carried out with positive reinforcement (Figure 3.8, lines 55 - 57). Otherwise, if it was previously in a predictive state but is not anymore then the update will carry out the segment update with negative reinforcement (Figure 3.8, lines 58 - 60). The latter of these possibilities will negatively reinforces a failed prediction.

```
54. for c, i in cells
55.     if learnState(s, i, t) == 1 then
56.         adaptSegments (segmentUpdateList(c, i), true)
57.         segmentUpdateList(c, i).delete()
58.     else if predictiveState(c, i, t) == 0 and predictiveState(c, i, t-1)==1 then
59.         adaptSegments (segmentUpdateList(c,i), false)
60.         segmentUpdateList(c, i).delete()
61.
```

Figure 3.8: Segment Updates - Pseudocode for phase three of the temporal pooler. Source: (1)

4

Research Methodology

4.1 Research Question

4.1.1 Research Problem Statement and Significance

The research problem is as follows:

The viability of the temporal pooler as a feature extraction technique remains unknown.

It is a significant problem for several reasons. First and foremost, feature extraction tests the algorithms ability to determine what is important in input data. A result would be a clear indication of the algorithms capacity to identify the underlying structure of a set of inputs. Another important reason is that, as previously mentioned, Numenta is in the process of commercialising this algorithm so a benchmark would be of considerable importance to potential consumers. The final reason which also makes any novel feature extraction technique significant is that the algorithm could be capable of outperforming the current state of the art.

4.1.2 Research Question Statement

The research question is as follows:

Is the HTM temporal pooler a viable feature extraction technique when given high dimensional movie data as input?

The theoretical framework for this work is based on the classification pipeline from Le et. al. (42) and the HTM cortical learning algorithms paper (1). Given that the pipeline forms an effective benchmarking platform for feature extraction techniques it is logical use this to evaluate HTM. This will involve replacing the existing ISA feature extractor in the pipeline (as seen in Figure 2.7) with a HTM feature extractor, as shown in Figure 4.1, then evaluating it. One can assume that the temporal pooler is capable of producing at the very least interesting results given the conclusions drawn in Thornton et. al. (23).

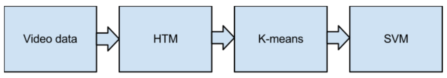


Figure 4.1: HTM classification pipeline - The classification pipeline after replacing ISA with HTM.

4.1.3 Research Aims

The primary aim is to determine whether or not HTM can handle complex data. Should an initial evaluation prove that it might be able to, its performance as a feature extractor in the pipeline from Le et. al (42) will be investigated. Additionally, any modifications or tweaks to the algorithm which improve performance will be recorded and analysed to determine the cause of the behaviour. This has the aim of improving the algorithm by gaining a better understanding of it.

4.1.4 Contribution

The contributions this research will bring to the field are unbiased performance metrics for the new HTM algorithms and potentially a new state of the art feature extraction technique. The former of these two is important because any performance metrics which Numenta releases will be potentially biased by commercial interest. This study will provide alternative performance information which is valid and reliable. The latter would cause considerable interest in HTM from the feature extraction community should it be the case. If not, useful information will still be gained for use by anyone performing further work on feature extraction with HTM.

4.2 Research Design

This research design contains a discussion of the underlying assumptions which can be determined from this and the nature of the problem. These assumptions then will be used in the formulation of a research strategy and approach. The methodology used in the identified research area will then be discussed. Finally, rigour, validity, reliability and ethics will be discussed in relation to the methodology and a timeline will outline completed and future work on the research project.

4.2.1 Assumptions

Several philosophical assumptions were made through an examination of the research area and research question.

The ontological assumption for this work is one of an objective reality. An ontological assumption is concerned with the state of reality. An assumption of objectivity is the only possibility. Pattern recognition is concerned with representing the structure of external inputs. Such external inputs can only be processed under the assumption that they are real and testable.

An epistemological assumption relates to our perceptions of the world. It is customary in pattern recognition to take a positivist view. Such a view postulates that we perceive the world in its entirety. Everything that we know is either sensed directly from the world or formulated by applying some transformation to sensed information. This is the only possibility given that a pattern recognition algorithm seeks to treat sensory data as input and apply some transformation to it.

A quantitative methodological assumption applies to this research. Methodological assumptions relate to the input data. A quantitative assumption is used in pattern recognition as the input data is typically treated independently of the nature of its origin. As far as the algorithm is concerned it is nothing more than a mass of quantifiable numbers upon which operations can be performed.

4.2.2 Research Strategy

The established research strategy for machine learning projects involves a literature review, gathering data, implementation of some algorithm, testing of the algorithm and then written documentation of the results and the experimental set up. The literature review previously discussed analysed the relevant disciplines of machine learning and paved the way for a research question to be formulated. Gathering data will be discussed in the data section. The implementation and testing of the algorithm will use constructive paradigms and follow the research approach of a typical pattern recognition problem discussed in the research approach section. For this research project the written documentation will be a dissertation thesis.

This research strategy takes all assumptions made into consideration. The objective ontological assumption is taken into consideration in that input data is assumed to represent a real pattern which exists in nature. The epistemological assumption of an positivism is based on the premise that all the information needed to perform complex tasks can be extracted from the environment. If this is not the case and there are aspects which cannot be calculated then pattern recognition will fail to effectively process these aspects. The methodological assumption will be discussed in the data section below.

4.2.3 Data

Data collection for this research project is a simple matter of selecting one from many datasets in the nominated field of human action recognition. As discussed, quantitative methodological approach will be taken to data collection. This dictates that the input data must be quantifiable and it should be possible to treat the data independently of its origin. Any human action recognition dataset will satisfy this constraint as they are just blocks of data from which patterns should be extracted as far as the algorithm operating on them is concerned. Given this wide selection the ideal dataset for a novel study such as this is the simplest one. The KTH human action dataset (41) is among the simplest and most used in the field so it is the best candidate. Ethical concerns are not an issue with this dataset as it was produced with paid actors for the sole purpose of being given as input to machine learning algorithms in another work.

4.2.4 Approach

Applying a constructivist approach to the research design of a typical pattern recognition research methodology results in a straightforward research approach. The first step is to identify a specific pattern recognition problem which, in this case, is human action recognition. A dataset based on this problem must then be procured. As previously mentioned, the KTH human actions dataset is the best candidate for this. Given this input data, an algorithm should be produced which can cater to the structure of the input data. This is not an attempt to bias the algorithm but a measure to ensure that it will be able to correctly process that specific type of data so the algorithm will not need to be restructured later on.

Implementation of the algorithm requires four components; the spatial pooler, temporal pooler, spatial and temporal pooler interface and integration into the existing classification pipeline from Le et. al. (42). This pipeline is the same as in Wang et. al. (2) and is publicly available. The spatial pooler had been implemented and tested in Thornton et. al. (23) so all that was left to implement was the temporal pooler and integration with the pipeline. Implementing the temporal pooler was a substantial task given the complexity of the algorithm. It required a clear understanding of the temporal pooler pseudocode presented in (1) followed by a careful implementation. The integration process involved investigating the structure required of the output of the feature extraction technique. This was followed by the process of evaluating the algorithm on the dataset.

An exploratory study into the HTM's feature extraction capabilities must be conducted prior to attempting to integrate the temporal pooler into the classification pipeline. This will involve the implementation of a mechanism to illustrate internal workings of the HTM. It it proves that the temporal pooler is incapable of handling samples from the KTH dataset then it follows that it should not be integrated into the pipeline in order to perform a more complex task.

Feature extraction algorithms are evaluated in much the same fashion as typical pattern recognition algorithm. The first step is to train the algorithm. With a feature extractor this will involve unsupervised training for the HTM and supervised training on the output of the HTM and the action classes for the classification algorithm (SVM). Once the algorithms are trained they can then be tested. Learning is turned off for this process and the classification accuracy of the SVM will be the key performance metric. Once this result is obtained it can either be accepted or rejected as a publishable result. Should the result be rejected then the researcher must make modifications to the algorithms then restart the process from the training phase. In this research approach only the HTM feature extractor will be modified and tweaked to improve performance to preserve the rigour, validity and reliability of the research.

4.2.5 Rigour, Validity and Reliability

Rigour, validity and reliability will all be maintained in this research by closely following existing pattern recognition and feature extraction paradigms and methodologies. Rigorous, uniform treatment of input data and test results is achieved by using the existing classification pipeline with precisely the same experimental set up. This pipeline treats all data and feature extractors in an unbiased manner. The pipeline in combination with the well known input dataset attests to the validity of the research. Use of an old, much used input dataset increases this work's comparability to existing literature. The pipeline further improves this by making it valid to compare any experimental results with results using the pipeline in the literature. It also simplifies the process of repeating this study. Reliability is not an issue as the dataset will never change and the algorithm will never change assuming the random seed does not change. The experimental results should be precisely repeatable.

5

Outcomes

An implemented, functioning temporal pooler and the behaviours it exhibited form the key outcomes of this research project. Considerable prepatory work was undertaken to integrate the temporal pooler into the classification pipeline from Le et. al.(42). This chapter will outline the outcomes of this process and the concurrent effort to build a working temporal pooler.

5.1 Pipeline Preparation

As previously discussed in the literature review, Le et. al. (42) used a classification pipeline with ISA as a feature extractor to produce state of the art results. This work aims to use HTM as a feature extractor in place of ISA. To achieve this, a rewrite of some sections of the pipeline's Matlab code was required. There were two possible routes this rewrite could have taken; a MEX file to interface the C code for the HTM with the Matlab pipeline or writing feature data from the HTM to disk to be later read by the pipeline. The MEX option would have the benefit of consolidating the pipeline into one procedure started through the Matlab pipeline. The latter is a much simpler approach and it was deemed the best option for two reasons. First, there would be a substantial time cost associated with learning the structure of MEX files and correctly building one. Second, saving everything to disk has the added benefit of being able to re-run the pipeline with the same features without having to re-train the HTM. This would save a considerable amount of time given the substantial time required to train

HTM systems. Additionally, saving features to disk allows the output to be manually inspected.

The pipeline was prepared to take input from the temporal pooler After careful research into the pipeline's various components. The raw patch data was previously being passed to the to ISA to extract features which were then passed to k-means for clustering. The modified process involves passing raw patch data to the spatial pooler and producing a feature for each patch. The features are then passed to the temporal pooler which then produces an output file of features to be read in by the modified pipeline. At this stage the focus was shifted to ensuring the temporal pooler was functioning as expected.

The patch data was extracted from the KTH dataset in the same fashion as in Le et. al. (42). A sample of two million 16×16 patches were randomly extracted from the videos in the KTH dataset. The samples were taken in tranches with a temporal length of ten reproducing the experimental setup of Le et. al. (42). For the temporal pooler, the tranches were made longer in order to allow it to learn more substantial temporal sequences. ISA treats the temporal dimension the same as spatial ones so this gave no advantages. The temporal pooler, however, is specifically intended to model temporal sequences so a temporal length of 100 was used instead. Thus, twenty thousand $16 \times 16 \times 100$ tranches were extracted.

5.2 Temporal Pooler Implementation

5.2.1 Graphical Interface

The complex processes occurring in the temporal pooler are difficult to grasp, especially so when the internal states can only be examined through verbose output. This led to the creation of the graphical user interface shown in Figure 5.1. It allows customised patterns to be submitted to the temporal pooler via mouse input and displays the state of every cell and their containing columns. Each column has a square outline in which one or more cell squares reside. Dendrite segments are not represented by this interface but cells with active sequence dendrite segments are differentiated from those activated by normal dendrite segments. This interface will be used to explain findings in other sections. The meaning of each colour is as follows:

Single States

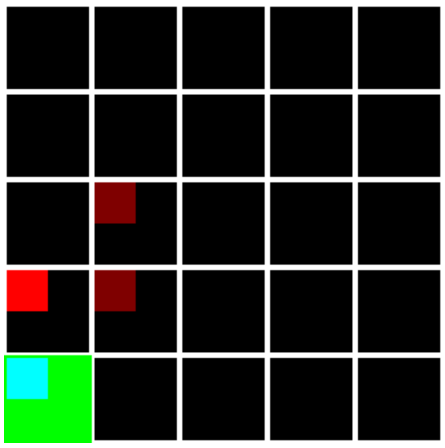


Figure 5.1: Temporal Pooler GUI - An example of the temporal pooler GUI with 25 columns.

Bright red: The cell is predicted to be active on in the next iteration.

Dark red: The cell is predicted to be active in some future iteration.

Blue: The cell has been selected as a learning cell for its containing column in this iteration.

Green: The cell or column is active in this iteration. A green border around a column is indicative of an active column while a green square is indicative of an active cell.

Compound States

Cyan: A cell which is in both a learn and active state.

Yellow: A cell which is in a bright red predictive state and an active state.

Light green: A cell which is in a dark red predictive state and an active state.

Violet: A cell which is in a learning and a dark red predictive state.

White: A cell which is in a predictive, active and learn state simultaneously.

These relationships can also be seen in Table 5.1.

The temporal pooler implementation progressed faster subsequent to the implementation of the GUI. The GUI provided an invaluable testing mechanism for the complex

Colour	Active	Predictive	Sequence	Learn
Bright red	No	No	Yes	No
Dark red	No	Yes	No	No
Blue	No	No	No	Yes
Green	Yes	No	No	No
Cyan	Yes	No	No	Yes
Yellow	Yes	No	Yes	No
Light green	Yes	Yes	No	No
Violet	No	Yes	No	Yes
White	Yes	Yes	Yes	Yes

Table 5.1: Temporal Pooler GUI Colour Key - Each colour is indicative of the cell being in some combination of an active, predictive, imminently predictive or learn state.

state transitions. The next section will detail the implementation of the algorithm itself.

5.2.2 Temporal Pooler

The temporal pooler was implemented as specified by the pseudocode in Hawkins et. al. (1) with two deviations. One is the place in which cell and segment states are calculated (see getActiveSegment method calls in Figures 3.6 and 3.7). Some of the calculations performed in the first phase in Numenta's algorithm are now performed in the second phase. This change does not affect the behaviour of the algorithm. It was done to improve the algorithm's efficiency and reduce computation time. The second change is that there is no learning radius for new synapse connections. In Numenta's version of the algorithm, when segment updates are implemented, new synapses are only connected to cells in a learning state within a certain radius. In the implementation completed for this thesis connections can be made with any learn state cell. Numenta (1) provides little information regarding the correct implementation of this mechanism and so a further study would have to be conducted to find an appropriate setting or the specify an algorithm which can calculate it.

5.2.3 Spatial to Temporal Pooler Interface

Interfacing the spatial pooler with the temporal pooler was a relatively simple task. The features output from an augmented spatial pooler was written to disk. This data was then read in as the input to the temporal pooler through the previously mentioned GUI. This allowed visualisation of the algorithm to be maintianed while operating on more complex data.

5.3 Temporal Pooler Behaviour

This section describes the key outcomes of this research project. The Temporal pooler exhibited several interesting behaviours including variable length sequence prediction and contextual sequence prediction. additionally, some limitations were encountered which halted progress towards evaluating it in the classification pipeline.

5.3.1 Sequence Prediction

The temporal pooler showed a clear capability to learn simple sequences. The GUI provided a method to evaluate this by inputting simple patterns and observing the resulting states of the temporal pooler. Figure 5.2 shows the temporal pooler learning a four pixel long vertical line. Note that the segment updates and all states are cleared before drawing the line so no recurrent connection is made from the top of the line back to the column at the bottom. The first time the line is drawn the algorithm makes no predictions as no connections have been made. Please refer to the GUI colour key from the graphical interface section to determine the states of each cell and column in each iteration. Because there are no existing connections all of the cells in the active column are activated and the first of those is selected as a learning cell. What is unseen is the segment update which connects this learn state cell with one or more learn state cells from the previous iteration.

The segment update is created in the active state calculation phase and causes sequence segments to join cells together. The sequence segments are indicative of a cell being active in the next iteration. To create this link the algorithm works backwards from a learning cell and connects it with cells which were in a learn state in the previous iteration. The connection is formed with the sequence segment flag set to true, causing the cells to instead be in a bright red predictive state the next time the line is drawn.

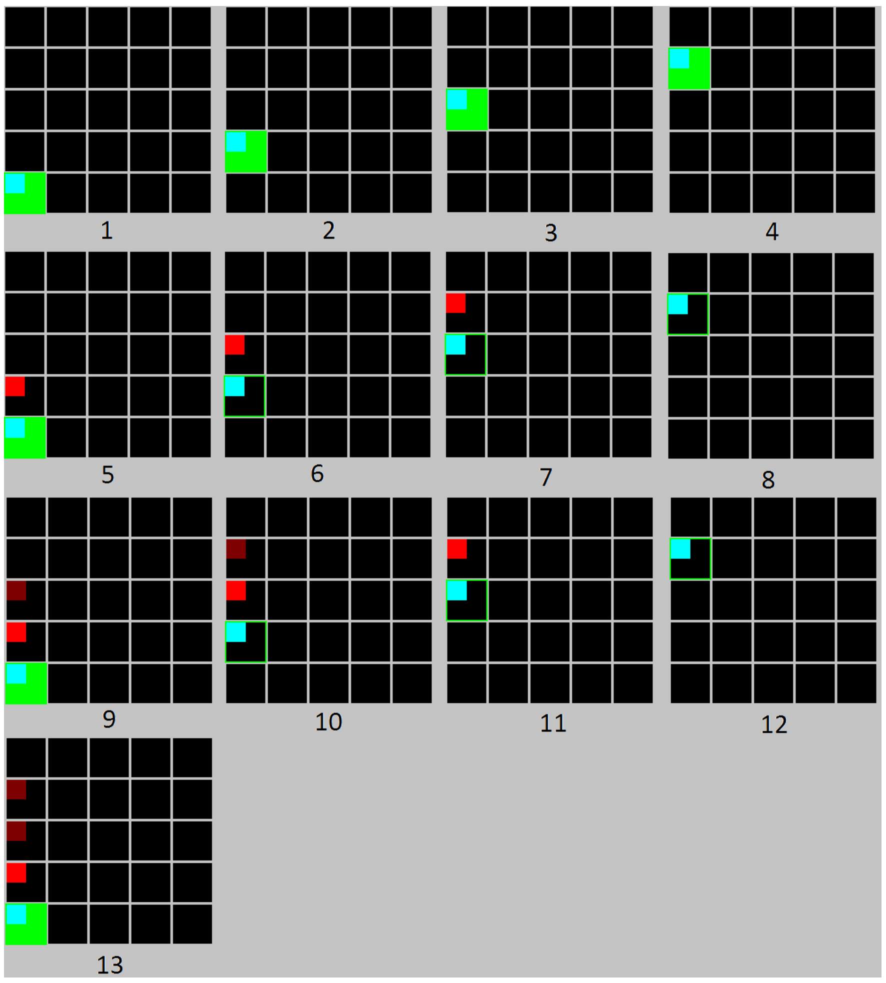


Figure 5.2: Simple Sequence - Training iterations on a simple sequence.

The third time the line is drawn another form of prediction can be seen. The dark red cells are cells predicted to become active in some future iteration - not specifically the next iteration. This is perhaps the most complex mechanism to grasp in the temporal pooler. The behaviour is caused by one of the segment updates queued up in the calculate predictive state phase. This type of update is queued when a cell enters a predictive state. The algorithm will then attempt to establish connections from this predictive cell to a learn state cell in the previous iteration. The more times the cell becomes predictive in the same sequence the further back its connections will extend. This mechanism works in reverse to make connections further and further back in time.

5.3.2 Contextual Sequence Prediction

The purpose of having multiple cells in each column comes to light when a more complex input is given to the temporal pooler. Figure 5.3 depicts the states of the temporal pooler when learning a 'plus' pattern. This pattern is important because the horizontal and vertical lines it is comprised of both contain the centre column. The key to this behaviour can be seen in a juxtaposition of the second and fourth frames. When the centre column of the vertical line is reached the first cell in the column is automatically deemed the learning cell, as no other candidates exist. In the centre of the horizontal line drawn afterwards all cells are activated to indicate that this column being active was not predicted once again. The key difference is that now the second cell in the column is chosen as the learning cell. This is because the first cell has existing connections so is no longer the most desirable learning cell. At this stage, a cell in the column below the centre is still predicting to be active in the next iteration. This is the correct prediction for the vertical line, not the horizontal one.

A few more iterations allow the algorithm to correctly predict this pattern. The three frames starting at the 15th and 18th frames show the sequence being correctly predicted for the vertical and horizontal line components respectively. Note that the vertical line sequences includes the first cell in the the centre column while the horizontal line includes the second cell. This follows from them establishing connections with cells in the columns which start each sequence.

These behaviours indicate that the algorithm was implemented correctly. The same behaviours are described in the cortical learning algorithms whitepaper. Given this,

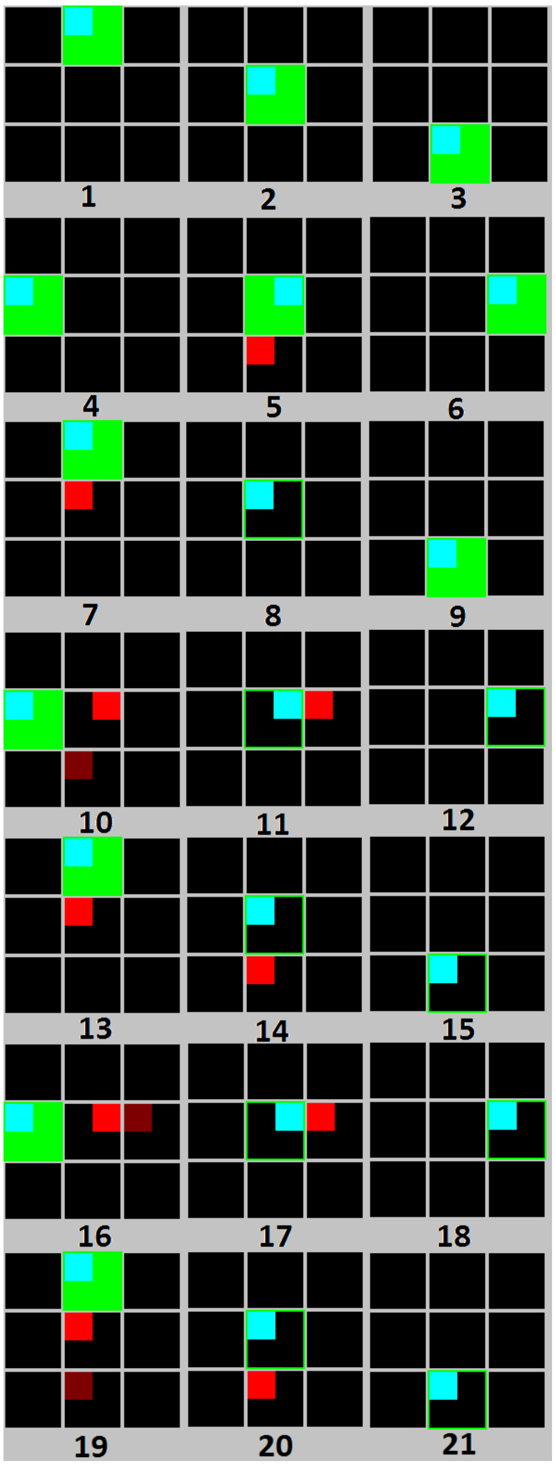


Figure 5.3: Complex Sequence - Training iterations on two overlapping sequences.

work commenced on the next phase of the project to train the temporal pooler on features extracted from the KTH dataset.

5.3.3 Limitations

When attempting to train the temporal pooler on the features extracted by the augmented spatial pooler two debilitating limitations were encountered.

5.3.3.1 Full Dendrite Segments

The first limitation relates to how the temporal pooler handles large amounts of input. The settings provided by Numenta indicate that there should be 128 segments in each cell and 40 synapses in each segment. This limit is hastily reached when attempting to train the temporal pooler on anything more than a single sequence of 100 spatial pooler codes. The result of this is that no new sequences are learned. This typically occurs after the temporal pooler sees approximately 40 sequences. Not all segments are full at this stage but those that are will be unable to connect with other cells. Given that this happens after only 40 of the 2000 sequences in the dataset, attempting to continue learning is impractical. Successfully learning the requires each sequence to be seen several times.

5.3.3.2 Predict Everything

As the number of sequences viewed increased, so too did the number of predictions being made by each cell. Juxtaposing Figure 5.4 with Figure 5.5 illustrates this. After training on 40 sequences the codes the temporal pooler produce contain approximately 200 columns with cells predicting activity in the next iteration. If almost everything is being predicted then the accuracy of the prediction is very poor. Spatial pooler codes typically contain 7 to 10 active columns so an ideal prediction should contain a similar amount of columns. With activation levels over 20 times higher than expected, inserting the temporal pooler into the classification pipeline was no longer practical.

Several actions were taken to attempt to solve this issue. Increasing the amount by which synapses are decremented when their predictions do not become true had a negligible impact. This should reduce the number of cells in a predictive state because a higher portion of their connections will fall below the connected permanence threshold.

The solution actually increased the rate at which dendrite segments became full because active cells were not being predicted and were thus being reinforced with new synapses. Another approach taken examined the result of using several connected permanence, initial permanence, activation threshold, minimum threshold and new synapse count values. This did not lead to any significant results.

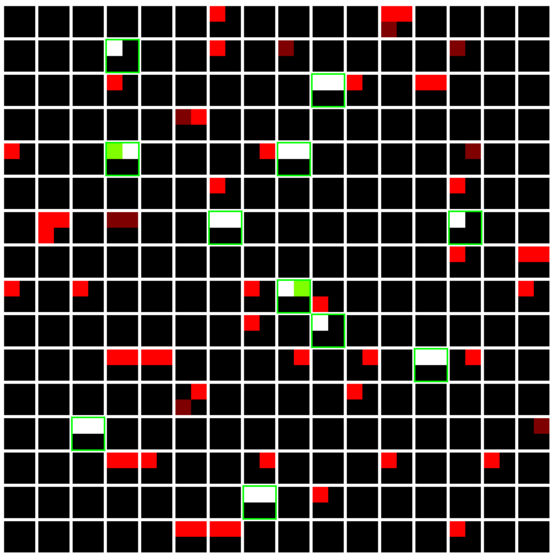


Figure 5.4: Moderate Predictions - The temporal pooler after training on 1 sequence of 100 features.

5.4 Interpretation of Results

What can be interpreted from the results at this stage is that the temporal pooler is not fit to be used in a classification pipeline. Measures implemented to address this did not yield any significant results. Further modifying Numenta's temporal pooler is beyond the scope of this thesis. The goal was only to implement Numenta's algorithm and evaluate its performance on complex video data.

Implementing the learning radius (see Section 5.2.2) could possible have alleviate this issue, but in all likelihood it would not solve it. In its current state, the algorithm encountered full cells after only 40 sequences. There are twenty thousand sequences in the dataset so the rate at which synapses are created would have to be reduced by as much as five hundred fold for one run through of the dataset. The temporal pooler

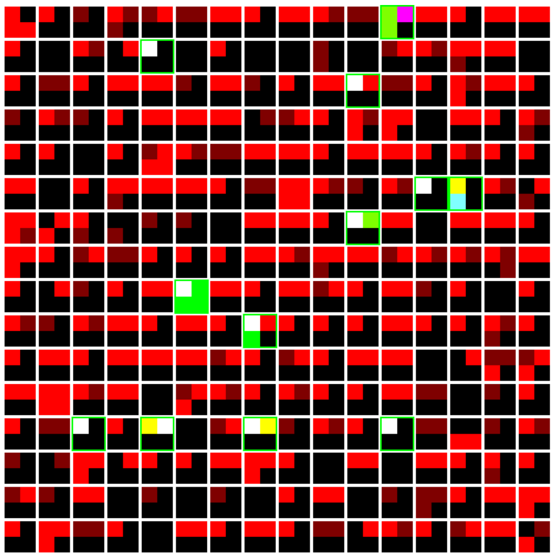


Figure 5.5: Excessive Predictions - The temporal pooler after training on 40 sequences of 100 features.

requires multiple runs through input data to allow predictions to be made further ahead in time, making it even less likely that the learning radius could solve this issue.

A good setting would also have to be found for the learning radius parameter. If it is too large, then the excessive predictions problem would not be alleviated. If it was too small then temporal connections would not be made to nearby cells which precede a given learn state cell. An investigation would have to be conducted to find an appropriate setting for this parameter.

The sequence prediction capabilities demonstrated are indicative of a sound prediction algorithm. The temporal pooler is able to form increasingly far sighted predictions the more data it receives on a particular sequence. Making accurate predictions more than one iteration ahead is an impressive feat. Additionally, it can take a cell's context information into account when making decisions on which sequence will be seen in future. These mechanisms make the temporal pooler more powerful than typical sequence prediction algorithms such as hidden Markov models which only operate within a fixed time frame. As such, it would be worthwhile to attempt to improve the algorithm in future research projects.

Currently, the limitations of the temporal pooler prevent it from being useful in a complex pattern recognition problem such as the KTH dataset (41). Predictions that

70% of all columns will be active in the next iteration represents a failure to form and predict sparse distributed representations. The algorithm is intended to result in small amounts of activation in the region less than 5%. Predictions should involve far fewer cells being active in the very next iteration. Had the majority of these been predicted only at some stage in the future then it would have had the possibility of making reasonable predictions. The problem of segments becoming full after a relatively small number of iterations also needs to be solved.

6

Conclusion

6.1 Summary

This thesis presented an evaluation of the temporal pooler as a feature extractor. The results indicate that the temporal pooler is not capable of effectively learning complex features present in the classification pipeline in Le et. al.(42). Doing so causes the algorithm to be overwhelmed and predict too much activity in addition to quickly reaching its synapse limit. This thesis was successful in evaluating the temporal pooler for the task. It can be concluded that it is not possible for it to produce impressive results for the human action recognition problem, unless modifications are made to the temporal pooler algorithm.

6.2 Future Work

There are several courses of action which are outside the scope of this research project. These include dynamically allocating memory for new dendrite segments, altering the way in which segment are negatively reinforced, applying HTM's multi-cell context approach to other algorithms, sequence segmentation and implementing the learning radius mechanism.

Dynamic memory allocation for dendrite segments would greatly increase the number of possible sequences which could be learned. The upper limit would of course be the memory of the system. This many not prove beneficial due to the excessive predictions that occur with the current dendrite segment limit. It will likely just confound this problem so it must be implemented in conjunction with measures to reduce the excessive predictions. One possible measure is to evaluate if there are other options to how synapses are negatively reinforced. A method which removes synapses with an insignificant activation rate will allow more important sequences to be learns in place of them.

The multi-cell contextual sequence prediction approach in HTM systems could be applied to other machine learning algorithms. Recurrent neural networks are the best fit for such a study. They could be enhanced with context-specific cell activations and predictions. This mechanism is beneficial because many sequences contain overlap where cells are part of several different sequences. The context allows one of several paths to be taken through a given cell, improving the accuracy of the sequence.

Another possible route of research which resides outside the scope of this project is the evaluation of the temporal pooler as a sequence segmentator. This would require performance improving measures to be implemented first. Sequence segmentation could allow for variable length sequences to be taken from the spatial pooler based on when the temporal pooler believes one sequence ends and another begins. The variable length sequences would still be the size of a normal features and could be used in the classification pipeline. Given that other methods such as Le et. al.(42) treat the temporal dimension as just more spatial data, such sequence segmented features may achieve superior performance.

Implementing the learning radius is a logical goal for future work. Doing so would allow the impact it has on the algorithm's two key limitations to be evaluated. Should this lead to a substantial improvement, the temporal pooler could then be evaluated in the classification pipeline for a more thorough test.

References

[1] Jeff hawkins, Subutai Ahmad, and Donna Dubinsky. **Hierarchical temporal memory including HTM cortical learning algorithms**. whitepaper, Numenta Inc., September 2011. 1, 12, 15, 16, 17, 18, 19, 20, 22, 23, 25, 29, 34

[2] H. Wang, M.M. Ullah, A. Klaser, I. Laptev, C. Schmid, et al. **Evaluation of local spatio-temporal features for action recognition**. 2009. 1, 13, 14, 29

[3] M. Glickstein. **History of Neuroscience**. *eLS*, 2008. 3

[4] T. Mitchell. *Machine Learning (Mcgraw-Hill International Edit)*. McGraw-Hill Education (ISE Editions), 1st edition, October 1997. 3

[5] F. Rosenblatt. *The Perceptron, a Perceiving and Recognizing Automaton Project Para*. Cornell Aeronautical Laboratory, 1957. 6

[6] A.E. Bryson, Y.C. Ho, and G.M. Siouris. **Applied optimal control: optimization, estimation, and control**. *Systems, Man and Cybernetics, IEEE Transactions on*, **9**(6):366–367, 1979. 7

[7] PJ Werbos. **Beyond regression: new tools for prediction and analysis in the behavioral sciences. 1974**. *Harvard University*, 1974. 7

[8] Y. Lecun. **Une Procedure dśapprentissage pour reseau a seuil assymetrique cog'nitiva 85: A la Frontiere de lqIntelligence Artificielle des Sciences de la Connais' sance des Neurosciences**. *Paris, France*, 1985. 7

[9] D.E. Rumelhart, G.E. Hintont, and R.J. Williams. **Learning representations by back-propagating errors**. *Nature*, **323**(6088):533–536, 1986. 7

[10] J. Han and C. Moraga. **The influence of the sigmoid function parameters on the speed of backpropagation learning**. *From Natural to Artificial Neural Computation*, pages 195–201, 1995. 7

[11] P. Dayan, G.E. Hinton, R.M. Neal, and R.S. Zemel. **The helmholtz machine**. *Neural computation*, **7**(5):889–904, 1995. 8, 12

[12] G.E. Hinton and R.R. Salakhutdinov. **Reducing the dimensionality of data with neural networks**. *Science*, **313**(5786):504–507, 2006. 8

[13] T. Kohonen. **Self-organized formation of topologically correct feature maps**. *Biological cybernetics*, **43**(1):59–69, 1982. 8

[14] J. MacQueen et al. **Some methods for classification and analysis of multivariate observations**. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, **1**, page 14. California, USA, 1967. 10

[15] C. Cortes and V. Vapnik. **Support-vector networks**. *Machine learning*, **20**(3):273–297, 1995. 11

[16] K. Pearson. **LIII. On lines and planes of closest fit to systems of points in space**. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, **2**(11):559–572, 1901. 11

[17] P. Comon. **Independent component analysis, a new concept?** *Signal processing*, **36**(3):287–314, 1994. 11

[18] A. Hyvärinen, J. Hurri, and P.O. Hoyer. *Natural Image Statistics: A Probabilistic Approach to Early Computational Vision.*, **39**. Springer, 2009. 11

[19] J. Pearl. *Bayesian Networks: a model of self-activated: memory for evidential reasoning.* Computer Science Department, University of California, 1985. 11

[20] L.E. Baum and T. Petrie. **Statistical inference for probabilistic functions of finite state Markov chains**. *The Annals of Mathematical Statistics*, **37**(6):1554–1563, 1966. 12

[21] V.B. Mountcastle et al. **Modality and topographic properties of single neurons of cats somatic sensory cortex**. *J. Neurophysiol*, **20**(4):408–434, 1957. 12, 18

[22] J. Hawkins and D. George. **Hierarchical temporal memory: Concepts, theory and terminology**. *Whitepaper, Numenta Inc*, 2006. 13

[23] J. Thornton and A. Srbic. **Spatial pooling for greyscale images**. *International Journal of Machine Learning and Cybernetics*, pages 1–10, 2012. 13, 17, 26, 29

[24] A.J. Perea, J.E. Meroño, R. Crespo, and M.J. Aguilera. **Automatic detection of urban areas using the Hierarchical Temporal Memory of Numenta®**. *Scientific Research and Essays*, **7**(16):1662–1673, 2012. 13

[25] AJ Perea, JE Meroño, and MJ Aguilera. **Hierarchical temporal memory for mapping vineyards using digital aerial photographs**. *African Journal of Agricultural Research*, **7**(3):456–466, 2012. 13

[26] I. Kostavelis and A. Gasteratos. **On the optimization of Hierarchical Temporal Memory**. *Pattern Recognition Letters*, 2011. 13

[27] R.J. Rodriguez and J.A. Cannady. **Towards a hierarchical temporal memory based self-managed dynamic trust replication mechanism in cognitive mobile ad-hoc networks**. In *Proceedings of the 10th WSEAS international conference on Artificial intelligence, knowledge engineering and data bases*, pages 320–328. World Scientific and Engineering Academy and Society (WSEAS), 2011. 13

REFERENCES

[28] D.R. Fernández, F.B.R. Ortiz, and P.V. Martinez. **Analysis and Extension of Hierarchical Temporal Memory for Multivariable Time Series**. 2011. 13

[29] S. Sinkevicius, R. Simutis, and V. Raudonis. **Monitoring of Humans Traffic using Hierarchical Temporal Memory Algorithms**. *Electronics and Electrical Engineering*, **115**(9):91–96, 2011. 13

[30] D. Rozado, F. Rodriguez, and P. Varona. **Gaze gesture recognition with hierarchical temporal memory networks**. *Advances in Computational Intelligence*, pages 1–8, 2011. 13

[31] L. Rodriguez-Cobo, P.B. Garcia-Allende, A. Cobo, J.M. López-Higuera, and O.M. Conde. **Raw Material Classification by means of Hyperspectral Imaging and Hierarchical Temporal Memories**. 2011. 13

[32] T. Kapuscinski. **Vision-Based Recognition of Fingerspelled Acronyms Using Hierarchical Temporal Memory**. In *Artificial Intelligence and Soft Computing*, pages 527–534. Springer, 2012. 13

[33] A. Motiwala, C. Fox, N. Lepora, and T. Prescott. **Sensing with artificial tactile sensors: an investigation of spatio-temporal inference**. *Towards Autonomous Robotic Systems*, pages 253–264, 2011. 13

[34] D. Rawlinson and G. Kowadlo. **Generating Adaptive Behaviour within a Memory-Prediction Framework**. *PloS one*, **7**(1):e29264, 2012. 13

[35] D. Lorenzi and J. Vaidya. **Identifying a critical threat to privacy through automatic image classification**. In *Proceedings of the first ACM conference on Data and application security and privacy*, pages 157–168. ACM, 2011. 13

[36] R. Dove. **Patterns of Self-Organizing Agile Security for Resilient Network Situational Awareness and Sensemaking**. In *Information Technology: New Generations (ITNG), 2011 Eighth International Conference on*, pages 902–908. IEEE, 2011. 13

[37] R.W. Price. *Hierarchical Temporal Memory Cortical Learning Algorithm for Pattern Recognition on Multi-core Architectures*. PhD thesis, PORTLAND STATE UNIVERSITY, 2011. 13

[38] M. Marszalek, I. Laptev, and C. Schmid. **Actions in context**. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 2929–2936. IEEE, 2009. 14

[39] M. Rodriguez, J. Ahmed, and M. Shah. **Action mach: Maximum average correlation height filter for action classification**. In *Proc. of IEEE Conf. on CVPR*, 2008. 14

[40] J. Liu, J. Luo, and M. Shah. **Recognizing realistic actions from videos in the wild**. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 1996–2003. IEEE, 2009. 14

[41] C. Schuldt, I. Laptev, and B. Caputo. **Recognizing human actions: A local SVM approach**. In *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, **3**, pages 32–36. IEEE, 2004. 14, 28, 41

[42] Q.V. Le, W.Y. Zou, S.Y. Yeung, and A.Y. Ng. **Learning hierarchical invariant spatio-temporal features for action recognition with independent subspace analysis**. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 3361–3368. IEEE, 2011. 14, 25, 26, 29, 31, 32, 43, 44

[43] Z. Zhang, C. Wang, B. Xiao, W. Zhou, and S. Liu. **Action Recognition Using Context-Constrained Linear Coding**. *Signal Processing Letters, IEEE*, **19**(7):439–442, 2012. 14

[44] S. Sadanand and J. Corso. **Action Bank: A High-Level Representation of Activity in Video**. CVPR, 2012. 14

[45] H. Wang, C. Yuan, W. Hu, and C. Sun. **Supervised class-specific dictionary learning for sparse modeling in action recognition**. *Pattern Recognition*, 2012. 14

[46] A.H. Shabani, D.A. Clausi, and J.S. Zelek. **Improved Spatio-temporal Salient Feature Detection for Action Recognition**. In *Jesse Hoey, Stephen McKenna and Emanuele Trucco, Proceedings of the British Machine Vision Conference, pages*, pages 100–1. 14

[47] A. Castrodad and G. Sapiro. **Sparse modeling of human actions from motion imagery**. *International Journal of Computer Vision*, pages 1–15, 2012. 14

Declaration

This work has not previously been submitted for a degree or diploma in any university. To the best of my knowledge and belief, the thesis contains no material previously published or written by another person except where due reference is made in the thesis itself.

Signed: