# Utilizing the HTM algorithms for weather forecasting and anomaly detection

Alexandre Vivmond

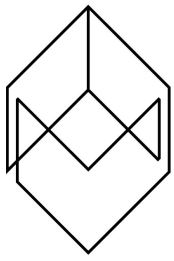
Master's thesis in Software Engineering at

Department of Informatics
University of Bergen

Department of Computing, Mathematics and Physics
Bergen University College

Simula@UiB

November 2016

HØGSKOLEN
I BERGEN

# Abstract

Various studies have utilized different artificial neural networks (ANN) for weather forecasting. This thesis examines how well the official implementation of a novel online ANN called the Hierarchical Temporal Memory (HTM) can forecast the weather and detect anomalies in the weather data. Created by Numenta[1], the HTM emulates the brain's neocortical structures and processes to mimic its capabilities of memory retention. By using sparse distributed representations instead of binary representations as its foundation for information storage and representation, it is able to learn complex patterns in noisy data sets that can be used to make predictions and detect anomalies in streamed data. Numenta has officially implemented the theory of HTM in an open-source Python platform called NuPIC. Although there are slight differences between the theory of HTM and its implementation, the most important factor about NuPIC is the addition of several purely engineered algorithms. Two of the most notable additions, are an algorithm that enables NuPIC to make the final decisions in cases when more than one possible prediction is possible, and an algorithm that makes it possible to simultaneously input multiple metrics to NuPIC.

The weather data that was to be predicted consisted of several weather factors, wind direction, wind speed, atmospheric pressure, precipitation, temperature, and relative humidity measurements spanning over a period of 12 years. Originally, the goal was to input the data sets simultaneously. However, because the functionality responsible for enabling this feature was malfunctioning at the time of the thesis work, every weather data set had to be input separately. The results showed that NuPIC was able to make decent forecasts, but was for the most part outperformed by a simple technique that made predictions by calculating the average of the last few days. The main reasons for this was due to the weather's lack of similarity between past and current conditions, and NuPIC's inability to generalize its knowledge in order to factor weather trends in its predictions. Although there is also a minor issue with the current engineered prediction algorithm, the results indicate that prediction is not NuPIC's strongest suit. NuPIC was completely unable to detect any noteworthy anomalies in the weather data, which again is most likely due to the weather data's chaotic nature.

[1] www.numenta.com

Despite the negative results, there were also some positive ones. An unrelated experiment that detected anomalies in the oil price, revealed that NuPIC was able to detect anomalies that were linked to major real world economic and/or geopolitical events. This indicates that the quality of NuPIC's results are highly dependant on the properties of the data set that it is given. Data sets that conform to NuPIC's strengths can lead to both decent predictions and anomaly detections, while those that do not produce poor results.

# Acknowledgments

I would like to express my sincere gratitude to my adviser Professor Kjell Jørgen Hole for his continuous support. His enthusiasm, patience and motivation have helped me make it through to the very end. I am incredibly humble and glad to have had him as my adviser and mentor, and wholeheartedly recommend him to students searching for an adviser.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Motivation

The field of machine learning is experiencing a steady rise in popularity amongst students [1] wishing to learn about the subject and amongst investors [2] and IT companies wanting to reap the benefits of the technology's potential. In truth, this should come as no surprise since we wish to achieve increasingly more complex tasks with more collected data than we know what to do with. Most people associate machine learning with artificial neural networks, in particular feed-forward neural networks (FNNs) with backpropagation as the best known training algorithm. Although FNNs were originally based on our understanding of neurology as being an electrical network of neurons, most modern-day FNNs such as convolutional neural networks are only inspired by biological processes and are first and foremost based on mathematical and statistical models. Hierarchical Temporal Memory (HTM) is a relatively new online machine learning model that was developed by Jeff Hawkins [3]. Hawkins' long-term goal is to emulate the structural and algorithmic properties of the neocortex as understood by modern neuroscience. This implies that HTM is an artificial neural network with a much more solid biological foundation than most others. The model is still in development with many challenges still remaining to be solved before it can emulate the neocortex.

As it stands today, HTM can accomplish impressive tasks such as learn complex temporal patterns in streamed data, use those learned patterns to predict future patterns, and carry out anomaly detection on the data, all in real time. For some applications, the ability to predict values based on previously learned patterns in noisy data environments is invaluable. Of equal importance, is the ability to raise 'red flags' when something anomalous or abnormal is happening in that noisy data. After a lot of contemplation and much dismay over the weather here in Bergen, I decided to examine how well HTM could forecast the weather, when put to the task.

## 1.2 Objectives

The main objective of this master's thesis is to examine how accurately the weather can be forecast using the the official implementation of HTM, called the Cortical Learning Algorithm (CLA) located in the software package NuPIC which was developed by Numenta. The second main objective is to examine how well NuPIC can detect anomalies in the weather data. The CLA in NuPIC is quite a large and complex program, which requires some expertise both in the theoretical understanding of HTM and the practical understanding of NuPIC with all its capabilities and quirks. Finally, thorough testing and reviewing will be needed to achieve satisfiable and trustworthy results, no matter whether they are positive, negative, or somewhere in between.

## 1.3 Thesis overview

This thesis is structured the classical way most dissertations follow. It starts with a thorough literature review of all the relevant background subjects such as the theory behind HTM and its core algorithms in Chapter 2. In Chapter 3, a code overview of NuPIC provides insight into the technology and features that have been implemented to extend the HTM's capabilities. Once the basis of this thesis is established, Chapter 4 discusses the methodologies that were used to accomplish the thesis project. The actual research is conducted in Chapter 5, where NuPIC is run on a multitude of data sets for various experiments with the results displayed. Chapter 6 analyses the results and Chapter 7 provides the final conclusions about HTM and NuPIC.

# Chapter 2

# Literature review

## 2.1 Classic artificial neural networks

In the field of machine learning, the term 'classic' artificial neural networks (ANNs) is sometimes used to refer to ANNs which are based on the principles of feedforward neural networks (FNNs) with some form of backpropagation training. ANNs were inspired [4] by our historical understanding of the communication processes between biological neurons. Neurons in FNNs [4] are simplistic entities which are interconnected in a structurally uniform network of neuron layers with weights on the edges between neurons. Input values were propagated forward by taking the input on each incoming branch to a neuron and multiply it with the branch weight and then sum all values. The sum was then processed through a threshold function that generated an output value. The output was used as input to the next neurons and so on until the final layer of neurons generate their output, which needs to be appropriately interpreted and evaluated for correctness. Depending on the degree of correctness, the edge weights between all neurons were adjusted by performing backpropagation training which is a rather tedious task. The long training time is a particular concern for large networks consisting of multiple layers and many thousands of edge weights. Training networks is exacerbated furthermore by the fact that the process of forward and backpropagation needs to be performed many thousands of times, which makes for a time consuming and computationally taxing task. Despite these disadvantages, great progress is being made in the field of machine learning with these techniques.

One thing to note however is that there are undeniable differences between the network structures and processes of classic ANNs, and our biological neurons and neurological processes. This is in no way a bad thing, but it does mean that classic artificial neural networks are much more artificial networks than neural networks. The theory of HTM is much more strongly founded on modern principles of neural biology, which we will soon look at.

## 2.2 Modern science on the mammalian brain

The mammalian brain can be roughly divided into three [5] main parts: the largest part is called the cerebrum, the smaller back part is called the cerebellum and the small undermost part is called the brainstem. The cerebrum itself consists of a right and left cerebral hemisphere, each of which can be divided into the top outermost part called the neocortex and the rest called the allocortex.

The neocortex is involved in higher cognitive functions [6] such as reasoning, conscious thoughts, language and motor commands. All these functions mean that it has highly desirable capabilities worth exploring. The neocortex consists of a wide variety [7] of different neurons that fulfill different tasks. Although the neurons in our brains are highly interconnected with each other, a very small percentage ($\approx$1%) of neurons are firing at any given moment in time [8]. This property is due to inhibitory neurons, which as their name suggests, inhibit neighboring neurons from firing at the same time as they do. Of particular interest are pyramidal neurons which slightly resemble pyramids and can be found in layered structures dubbed cortical layers or cellular layers. All mammals have six such layers [6] in their neocortex, named with roman numerals from I to VI, with the layers II and III often combined together and referred as layer II/III or layer 2/3 as shown in Figure 2.2. Pyramidal neurons are the main cells found in layer III and V and play a central role in the theory of HTM. If one were to zoom in on a layer, a pattern would emerge as neurons appear to be arranged in cortical columns spanning the entire layer [9]. Because of a class of inhibitory cells, all the cells within a column are forced to receive the same information and thus represent the same value, but in different contexts. It should be noted that the theory of HTM largely ignores layer I, focusing mostly on layer II/III and often only references the last three layers.

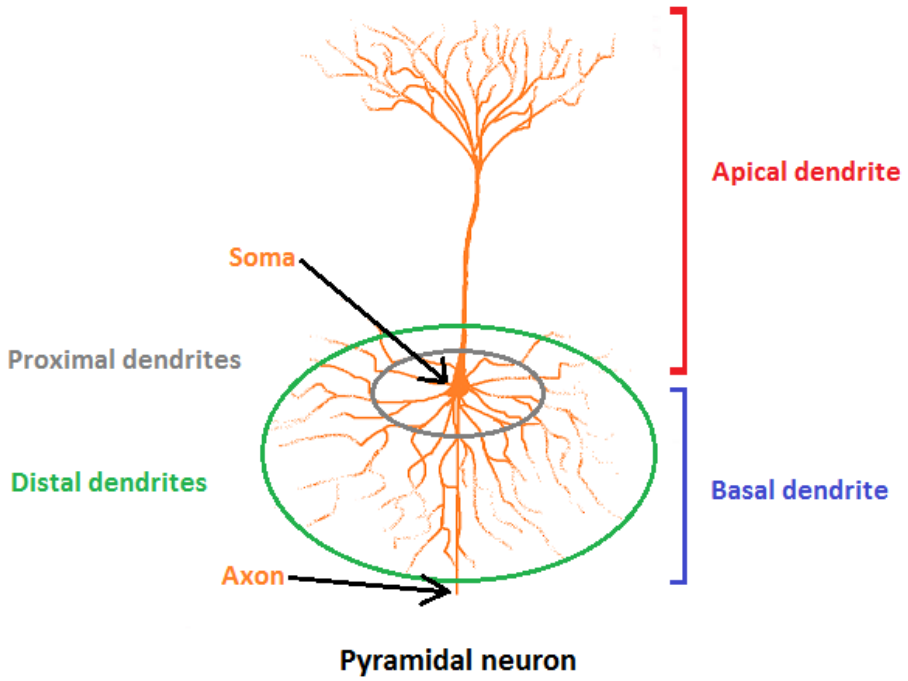



**Figure 2.1:** A figure of a pyramidal neuron with the names of the various parts. The gray circle encompasses the cell's soma and proximal dendrites, while the green circle encompasses the distal dendrites, excluding the gray circle. Notice that the pyramidal neuron's axon is depicted by the straight line stretching from the soma to the green circle.

While most neurons are depicted as a cell body with a few dendrites and an axon, pyramidal neurons (see Figure 2.1) have thousands of dendrites, and can be divided into four main parts, the cell body (soma), basal dendrites, apical dendrite, and axon. The basal dendrites emanate directly from the soma while the apical dendrite is a single long and thicker dendrite that branches out profusely further away from the soma. Although the axon is rarely displayed in pictures and diagrams of pyramidal neurons, it is very long and branches out extensively. Basal dendrites and the apical dendrite can be further divided into two groups, proximal dendrites and distal dendrites. The dendrite branches closer to the soma are called proximal dendrites, while the dendrite branches farther away from the soma are called distal dendrites. Pyramidal neurons can receive inputs from synapses on their proximal and distal dendrites [10]. Inputs received from proximal dendrites can lead to an action potential (AP), while input from distal dendrites can generally only lead to the cell being depolarized. A cell in a polarized state will require a significant amount of input for it to lead to an AP, while a cell in a depolarized state will require much less input. An AP is a process that can be simply described as a signal being fired down a cell's axon and passed on to other connected cells.

## 2.3 The HTM model

### 2.3.1 Overview

Jeff Hawkins was the founder of Palm Computing and one of the founders of Handspring, which developed the popular and successful PDAs (Personal Digital Assistant) Palm and Treo in the 1990s [11]. In the early 2000s, he decided to delve into his deepest field of interest, which was brain research at the Redwood Center for Theoretical Neuroscience in Berkley, California. In 2004, he released the book *On Intelligence* in which he discusses the brain's intelligence and the key principles that enable it. He explains (amongst other things) that the neocortex is a complex memory system with hierarchical regions that works by constantly trying to predict the future based on stored past experiences. He calls this theory the 'memory-prediction framework'. In 2005, Hawkins, Donna Dubinsky, and Dileep George founded the private company Numenta, which aims to discover the operating principles of the neocortex and build intelligent systems based on those principles.

During the past 11 years, Numenta has worked through three semi-official generations of learning algorithms [12]. Their first generation of algorithms called Zeta 1 was mainly focused on vision tasks and had little basis in neuroscience, relying heavily on mathematical principles alone. Their second generation of algorithms called the Cortical Learning Algorithms (CLA) was much more focused on neuroscience. These algorithms were based on the company's newly developed HTM theory, which was based on and expanded upon the memory-prediction framework. One notable expansion, was the introduction of sparse distributed representations which play a foundational role in HTM and permeate most functions and processes within. HTM consists of a collection of algorithms working in unison to bring about real-time learning of temporal and spatial patterns in streamed data sets. These algorithms provide HTM with advanced anomaly detection and prediction capabilities. The CLA was successfully used in several different applications that benefited from anomaly and prediction capabilities, such as server metric anomalies, rogue behavior detection, and natural language processing [13]. Although the HTM theory discussed how the hierarchical aspect was involved in the overall processes, it did not provide any detailed explanations on the subject, since there is currently very little empirical knowledge available. This means that the CLA does not implement hierarchical processes. In 2013, the company decided to open-source its CLA implementation on a Python platform called the Numenta Platform for Intelligent Computing (NuPIC)[1], which will be used in this thesis. Finally in 2014, after making advances in their theoretical work and continually improving NuPIC, they entered the current third generation of algorithms, which has no official name. We will simply refer to the third generation of algorithms as NuPIC.

[1]Repository url: https://github.com/numenta/nupic

### 2.3.2 Structure and elements of HTM

The structure of HTM [10] is not identical but quite similar to the neocortex' biological structure. Figure 2.2 shows the elements of an HTM model and simultaneously contextualizes it by showing which elements are connected with each other and what they represent. The four elements are regions, layers, columns (often called mini-columns) and cells or neurons.

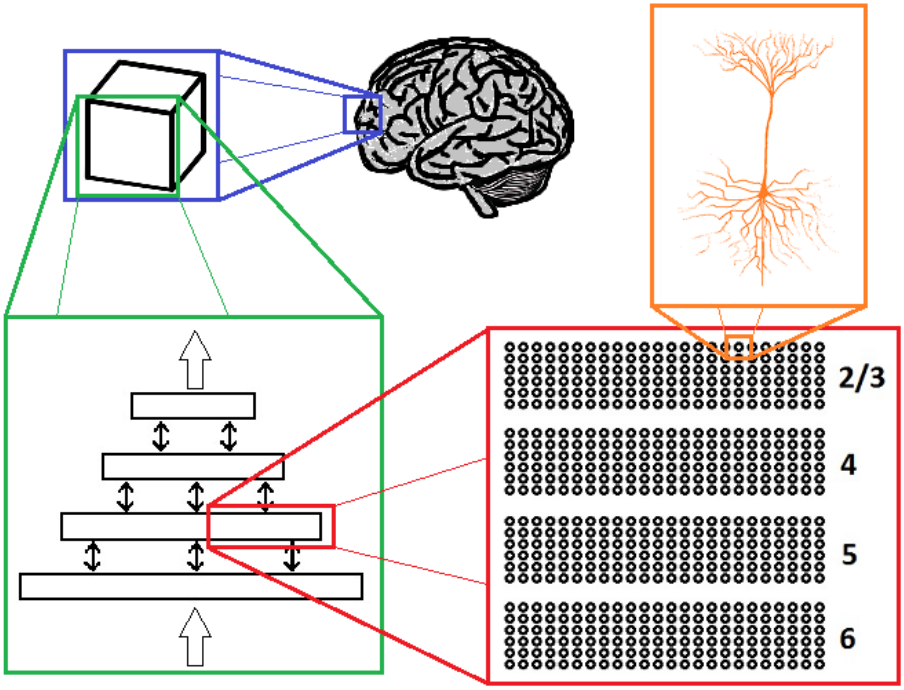



**Figure 2.2:** A figure showing the relationship between the brain and the elements of HTM. At the top middle, a human brain is shown with a small cube of cortical tissue cut out in the blue rectangle. The green rectangle displays the hierarchy of regions which get their input from below and pass their processed output at the top. The red rectangle shows how every regions is composed of the same set of cortical layers, each of which are arranged in columns of cells. The orange rectangle depicts such a cell from a column.

Figure 2.2 shows a (human) mammalian brain, with a $2mm^2$ patch of neocortical tissue cut out representing a hierarchy of multiple regions. All regions in a hierarchy are fundamentally and structurally similar [14]. Input from sensors gets processed by the bottom region which outputs its results up to the region above it, and so on until the top region outputs its results. Each region learns sequences, and the output from a region is more abstract then the input it was given. Although HTM explains that regions are supposed to communicate with each other in such a hierarchical manner, the actual details and technicalities behind this communication and their outputs to each other are still being researched. Each region consists of four layers, layer 2/3, 4, 5, and 6. A layer consists of columns which are made up of neurons. A typical layer has about 2048 columns, with 32 cells in each column. Cells in HTM are modelled more

realistically than cells in most other ANNs, and imitate the properties of pyramidal cells that were explained earlier, albeit a little simplified.

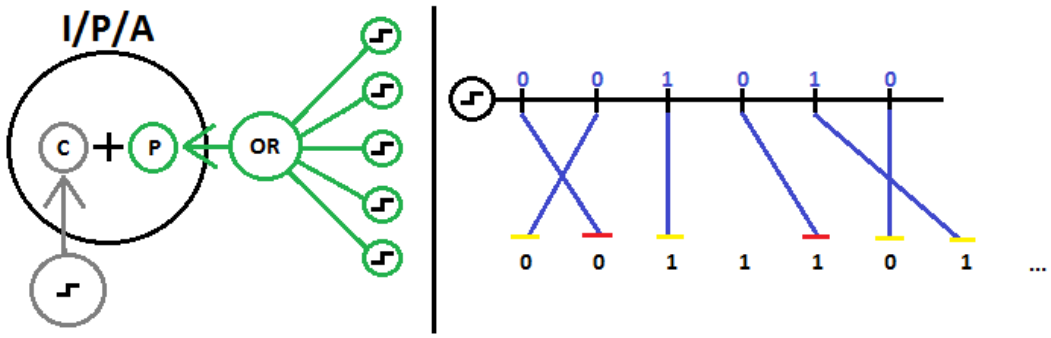



**Figure 2.3:** On the left side of the black line, an HTM cell is drawn, with the three possible states being inactive (I), predictive (P) and active (A). An HTM cell receives input from both its proximal dendrite in gray and from its distal dendrites in green. The binary OR of all distal dendrite values determines whether a cell is set to the predictive state, while the proximal dendrite determines if a cell is chosen (C). If a cell is both chosen and in a predictive state from before, it is set to the active state. A cell can also be set to the active state without being in a predictive state from before. On the right side is a depiction of a segment. The segment is drawn in black, with blue synapses connecting it to some input bit array. A synapse with a yellow bar means that it is over some threshold while a red bar means that is below. A segment can only read input from synapses above that threshold, in this example resulting in 0 0 1 0 1 0. Since a segment can never perceive the entire input space, some input bits are not connected by a synapse and never will. Synapses with red bars can eventually turn yellow and vice versa.

As can be seen in Figure 2.3, an HTM cell has one proximal dendrite and multiple distal dendrites, both of which receive input in the form of a binary vector. A proximal dendrite has a set of potential synapses, each of which can perceive a fraction of the input space from feedforward input. Likewise, a cell has multiple distal dendrites, each of which has a set of potential synapses that can receive information about the state of other cells in the same layer. In both cases, the synapses on both dendrites have a permanence value which must be above some threshold for a synapse to be active, i.e. be able to receive information. Feedforward input comes from sensors while input coming from distal dendrites is called feedback input. No single cell perceives the entire input space from a sensor, nor perceive the state of all other cells in a layer. Furthermore, all cells in a common column share the same synapses on their proximal dendrite that receives feedforward input, meaning that they all receive the exact same information from sensors. Finally, a cell can be in three states: inactive, predictive, and active. A cell is inactive when it does not get any input at all, predictive when it gets feedback input, and active when it gets feedforward input.

### 2.3.3 Data representation

The theory of HTM uses sparse distributed representations (SDRs) as its foundation for representing data and learning sequences of patterns [15]. SDRs are long binary vector representations with extremely few ON or 1 bits, which constitute compressed semantic representations of input data. SDRs emulate the brain's property of having a small percent of firing neurons, through a process that will soon be revealed. HTM theory uses 2048 bit long vectors with 40 ON bits. Although SDRs are represented as binary vectors, it will be explained why no single bit is critical to a representation, unlike in the binary number system. Unfortunately, the science of SDRs is too complex and mathematically demanding to include in this thesis. Therefore, a summary of the remarkable properties is given, assuming that the SDRs consist of 2048 bits ($n$) of which only 40 are ON bits ($w$). The representation space is astronomically high ($2.37\text{x}10^{84}$), while the likelihood of two random SDRs being identical is astronomically low ($4.2\text{x}10^{-85}$). Two SDRs can be easily compared for similarity, by performing a bitwise AND operation on the two, and counting the amount of ON bits in the result to see if it is above some threshold ($\theta$) to be considered a match. Impressively, even with a $\theta$ that is half of $w$, there is still an extremely low likelihood ($\approx 2.5\text{x}10^{-26}$) of false positives when testing for matching SDRs. Similarly, the chance of a false positive when comparing a subsampled SDR with $w$ = 20 to a random SDR with $w$ = 40 and $\theta$ = 10 is also unlikely ($\approx 1\text{x}10^{-12}$). Finally and most strikingly is the small probability ($\approx 4.48\text{x}10^{-12}$) of false positives when matching one SDR to another SDR that is the union of 20 SDRs with $\theta$ = 18. In other words, let $x$ be an SDR that is the result of calculating the bitwise OR of 20 random SDRs. If $x$ is matched to some SDR $y$, then it is practically guaranteed that $y$ is contained in $x$. The process for input data to be sparsely distributed is accomplished in the first two steps of the three core HTM processes explained below.

#### Encoding

The first core process and step in creating SDRs is encoding, which converts raw input data into distributed binary representations [15, 16]. The actual output representation from an encoder is an array of zeros and ones, which can be viewed as a vector. Binary representations must have a fixed total length and number of subsequent 1-bits to accommodate the range of raw input values. Since representations are discrete, values are sorted into 'buckets', which represent all values within their range with the same binary representation. The number of buckets must be chosen appropriately depending on the range of possible input values. The actual number of 1-bits should be at least 20–25 for robust subsampling and good noise tolerance. The total number of bits required and the index of the bucket that a value is to be placed into, can be calculated with the following formulae. Here, $n$ is the total number of bits in a vector, $b$ is the number of

buckets, $w$ is the number of ON bits, $i$ is the bucket index and $v$ is the input value:

$$n = b + w - 1 \tag{2.1}$$

$$i = \left\lfloor \frac{b \cdot (v - minValue)}{maxValue - minValue} \right\rfloor \tag{2.2}$$

| Value | → | bucket index | → | binary representation |
|---|---|---|---|---|
| $\leq 0$ | → | $b_0$ | → | 11111000000000 |
| 0.99 | → | $b_0$ | → | 11111000000000 |
| 1 | → | $b_1$ | → | 01111100000000 |
| 5 | → | $b_5$ | → | 00000111110000 |
| 9.99 | → | $b_9$ | → | 00000000011111 |
| $\geq 10$ | → | $b_9$ | → | 00000000011111 |

**Table 2.1:** Table showing an example of the relation between raw values, bucket indexes and binary representations. This particular example demonstrates how the raw input numerals 0 to 10 are encoded into binary vectors, using 10 buckets and five subsequent 1-bits

In the example from Table 2.1, each value falls into some bucket and thus gets represented accordingly. Performing a bitwise AND operation on two representations tells the semantic relationship between them. Here, the bitwise AND of the representations of 0 and 1 would result in 01111000000000 which has an overlap of four 1-bits, meaning that the semantic similarity between those values is very strong, yet not identical. The semantic similarity between the values 1 and 5 is only one meaning that it is very weak, and between 1 and 10 it is non-existent.

### Spatial pooling

Once raw inputs are encoded into distributed representations, they are passed on to the next step called spatial pooling (SP), while the algorithm itself is called the spatial pooler [15]. SP is the second core HTM process and final step in creating SDRs. This process uses the HTM structure previously described in Section 2.3.2 about cortical layers and cells. It was explained that all cells in a column have a proximal dendrite with synapses that get the same feedforward input from sensors, albeit once encoded as is now known. To simplify and optimize on this fact, the SP forgoes working with each individual cell, and works only at the column level assigning each column the same input space that all of its cells share. The spatial pooler has two objectives, turn distributed representations into sparsely distributed representations, and learn to better recognize recurring inputs.

SP creates SDRs by mapping the 2048 columns to the input vector, and then choosing 40 of them to be marked as active. Each column in the spatial pooler is connected to a random subset

of the input space, i.e. each column's "synapses" are connected to random elements in the input vector received from the encoder. The default number of connections is 80% of the total input array length, and the process is partly illustrated in Figure 2.3. The synapse connections are static, meaning that a column will never create nor loose connections throughout the entire HTM algorithm cycle. However, each connection has a permanence value, which must be above some threshold for that connection to be considered active. A column has an overlap score, which is calculated by counting the amount of active connections that are mapped to ON bits in the input vector. The columns are then compared to their neighbouring columns based on their overlap score, with the ones with the highest relative score becoming the aforementioned active columns. To ensure that the 40 active columns represent that particular input in the future, for each of those columns, the connections that were mapped to ON bits in the input vector have their permanence values slightly increased. Inversely, all the other connections that were mapped to OFF bits in the input vector have their permanence values slightly decreased. This process ensures that each column learns to recognize very few inputs, but effectively. The idea is that two similar input vectors will be largely recognized by the same columns, albeit not exactly the same, and thus create similar SDR representations. On the contrary, very differing input vectors will create highly contrasting SDR representations. The output of the spatial pooler is a 2048 long vector/array with 40 ON bits.

**Temporal memory**

The temporal memory (TM) algorithm, which was formerly called the temporal pooler is the third and final core HTM process [15]. It has two purposes, the first is to turn the SDR input received from the spatial pooler into a representation that captures the temporal context of the current input. The second purpose is to create a prediction of the future input based on the sequences that followed before.

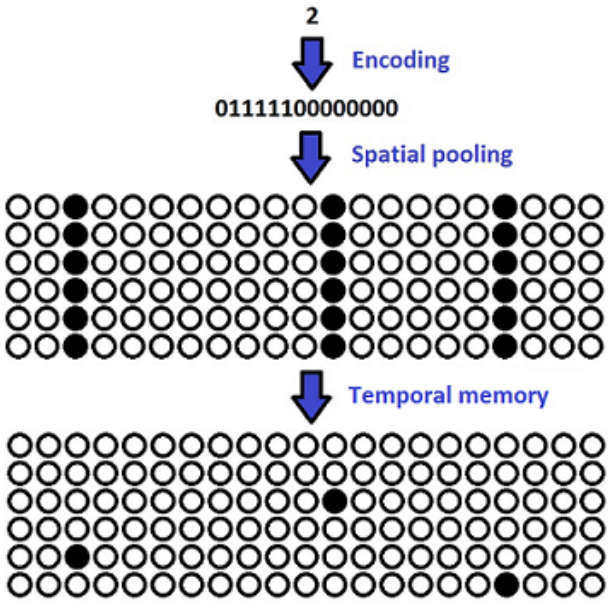



**Figure 2.4:** This Figure shows the three core steps of HTM and how they affect the internal state of the cells in a layer. The encoder converts raw values into binary distributed representations. The spatial pooler converts the binary encoding into a sparsely distributed representation by choosing a subset of columns. The temporal memory turns the SDR into a contextual SDR by choosing cells amongst each chosen column. Note that this illustration is general and therefore does not include cells in the predictive state and the important role that they play.

When the TM algorithm receives input from the spatial pooler, it must decide which cells inside each of the active columns should become active, i.e. changed to the active state. In each active column, if a cell is found to be in the predictive state, it is switched to the active state. If no predictive cells are found in an active column, then all cells inside that column are turned active, in a process called 'bursting'. The idea is that while an SDR of active columns can represent a value, the distinct combination of active cells in them represents that same value but in different contexts. Once done, the algorithm goes through every cell's distal dendrites, and counts how many of the synapses are both active and connected to active cells. If at least one dendrite's count is higher than some threshold, then that distal dendrite is marked active, and the corresponding cell is put in the predictive state. If in the next time step, a predictive cell happens to be in an active column i.e. it was correctly predicted, then the cell's active distal dendrite increases the permanence values of all its synapses that were connected to active cells. On the other hand, a predictive cell that does not become active, will reduce the permanence values of its synapses on the distal dendrite that was marked active. In the case of a bursting column, no cell in that column was in a predictive state from before. Therefore, the algorithm tries to find if there are any cells in that column that have dendrites with enough synapses connected to previously active cells that are above some minimal threshold (that is obviously below the threshold that marks a dendrite active). If such cells are found, the one with the dendrite that

has the most connected synapses is selected, and their permanence values are increased. If no such cell is found, the cell with the currently least amount of distal dendrites receives a new distal dendrite that has synapses connected to a subset of the previously active cells with above minimal threshold permanence values. The TM algorithm outputs the set of cells that have been marked active in a pretty long vector, since there are 2048 columns with 32 cells per column, such a vector array contains 65536 bits. An alternative way to store this information is to only keep the indexes of the active cells which drastically reduces the length of the array down to a minimum of 40 elements.

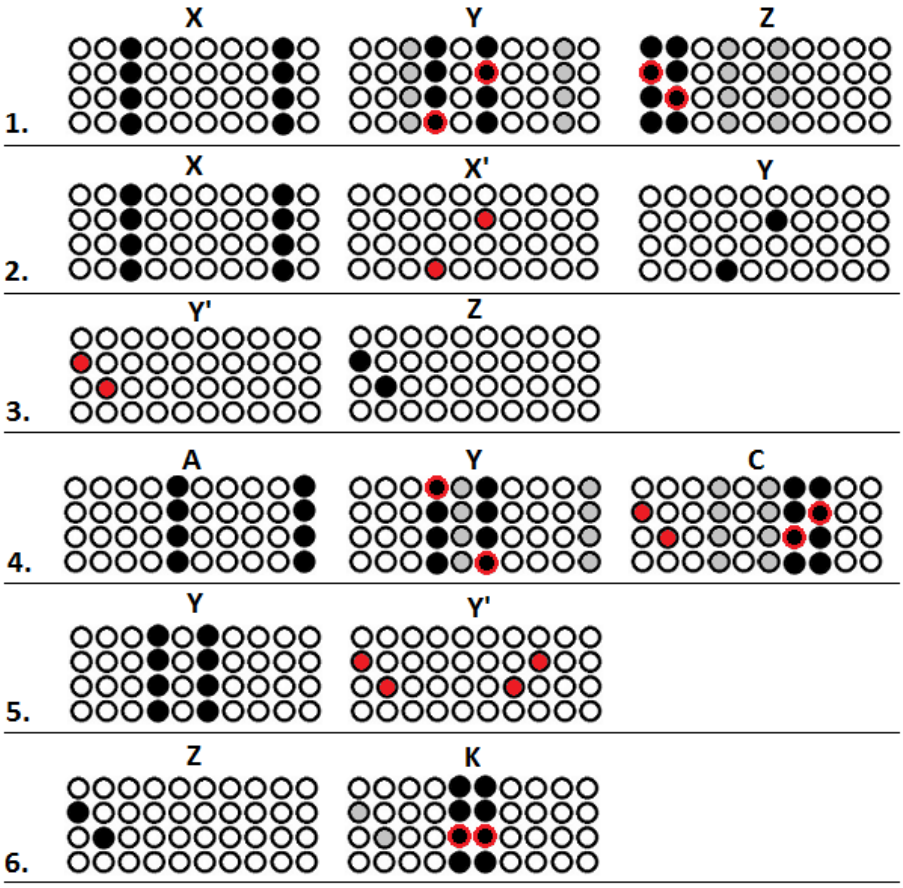



**Figure 2.5:** Here are several examples demonstrating how the TM learns sequences, using a cortical layer with only 10 columns and 4 cells. To keep it simple, the SDRs are composed of only 2 active columns and the leftmost column is the zeroth column. Circles with a white center represent inactive cells, while circles with black centers represent active cells. Circles with a gray center represent cells that were active in the previous time step. Red circles with a black center represent active cells that have been chosen to represent the current contextual SDR, while black circles with red centers represent predictive cells. Each row has been numerated to make it easier to follow the description.

Looking at the example Figure 2.5, the first row shows how the TM reacts when receiving the unpredicted values X, Y, and Z from the SP which are represented with the respective column

arrays {2,8}, {3,5} and {0,1}. Since the values had never been seen before, the TM 'bursts' the cells in those columns, marking them all active. In addition to bursting, the Y SDR selects a cell from each burst column (red circle with black center) and creates a connection between all the previously active cells (in gray) and those selected active cells. These connections are the basis for the previous cells' predictions to the current chosen active cells. The same process is repeated for the Z SDR. In the second row, when the X SDR appears once again, the TM checks the active cells' connections and finds two cells which it marks as predictive (depicted as black circles with red centers in the X' SDR). The next input from the SP is {3,5} which represents the value Y, and since there already are predictive cells in those columns, the TM marks them active. Before the next input is received the TM once again checks the current active cells' connections and marks them predictive, as shown in the Y' SDR. Finally when the last input {0,1} is received, it only marks the predictive cells in those columns active.

In the fourth row, at some later time into the data set, the new value A is received which leads to bursting, followed by the value Y which had previously been seen before. Since this Y value occurred in a new context, the chosen active cells are different from the ones in the first row. When the TM checks the Y SDR's active cells' connections, it still finds the two predictive cells representing Z that were learned in the first row, shown in the C SDR. However, the prediction is unsuccessful as it receives the new value C. In the fifth row is a scenario where the value Y is unexpectedly received in a new context. Because of the active cells' previous connections, the TM finds and marks both the cells that represent the values Z and C as possible predictions. In the previous examples, all the previously active cells (from burst columns) had to create connections to the chosen active cells. In most cases however, a lot fewer cells will have to create connections to chosen active cells as depicted in the last row, which correctly predicted the value Z but then receives the new value K. Only the two cells from Z create connections to the chosen active cells in K.

While it would be natural to think that the cells that have been marked as predictive by the TM would be somehow used as the actual predictions of the HTM, they are in fact not. Those cells are only used by the TM for its own internal predictions. Their ultimate purpose is to be marked as active if they correctly predict the next input. For in doing so, they create a representation of the current input that is unique to the current context. The actual output predictions come from an algorithm called the CLA classifier, which uses the active cells generated by the TM, and makes the final decision when multiple predictions are likely. The CLA classifier will be explained in the next chapter.

### 2.3.4 Detecting Anomalies in patterns

The process of anomaly detection is enabled by performing mathematical and statistical calculations based on the results from the TM's predicted and active columns [17]. There are two anomaly detection metrics in HTM, raw anomaly scores and anomaly likelihoods, of which the latter is calculated based on the results of the former. Once the TM algorithm is done creating a contextual SDR, the anomaly score can be easily computed by doing a calculation based on the amount of active columns that burst. If the input $x_t$ is converted into the SDR vector $SDR(x_t)$ which represents active columns, and $SDR(p_{t-1})$ is the SDR vector of predicted columns from the last time step (i.e. columns in which there are cells in the predictive state), then the equation [18] is:

$$\textit{Anomaly Score} = \frac{|SDR(x_t) - (SDR(p_{t-1}) \cap SDR(x_t))|}{|SDR(x_t)|} \tag{2.3}$$

The resulting raw anomaly score will be between 0 and 1, where 0 indicates that the input was correctly predicted, 1 indicates that the input was completely unpredicted, and in between if the input was somewhat predicted depending on the adjacency to either 0 or 1. In the case where multiple different values are simultaneously predicted, the amount of predicted columns will outnumber the amount of active columns in the current SDR. According to the equation, this means that if any one of the predicted values turn out to be correct then the score will be 0.

The advantage of calculating raw anomaly scores is that it gives direct feedback as to how expected or unexpected a certain input value is at any time. However, in certain environments there is a constant stream of noise and deviating values, leading to a perpetual barrage of high anomaly scores which would be quite troublesome. The second anomaly factor called anomaly likelihood was made to handle this exact problem by focusing on the change in raw anomaly scores over time as indication of actual anomalies. The statistical calculations behind anomaly likelihoods are beyond the scope of this chapter, but can be read in Numenta's paper [17].

One of the major points about anomaly detection algorithms, is that they may detect anomalies in data streams prior to major events [19]. Numenta has demonstrated that the HTM successfully detected a temporal anomaly in the temperature data stream of a windmill, prior to the windmill catching fire [20]. This aptitude of HTM is one of its most compelling qualities.

## 2.4 Forecasting the weather

The weather is an extremely vast, complex, and chaotic system that is affected by the interaction of a multitude of factors to create the diverse weather phenomena observed on our planet. Despite all this complexity and chaos, thanks to the combined scientific advances of numerous fields, we are today able to predict the weather for the next four days with decent accuracy [21], albeit less for some places with especially intricate weather such as Bergen, Norway. Since

HTM is to be used for weather prediction, it is necessary to acquire a basic understanding of the weather and science behind forecasting.

### 2.4.1 Overview

Solar radiation warms our ellipsoidal planet unequally, in such a way that the equatorial regions gain more heat than the polar regions [22]. Grossly simplified, the oceans and the atmosphere redistribute excess heat from the equator to the poles which fortunately ensures that neither of the two become completely inhospitable to life. When the planet's surface is heated up, it transfers its heat to the air above it. The hot air rises up and is replaced by descending cooler air in a process called convection. However, because of the earth's size and rotation, convection doesn't directly bring hot air from the equator to the poles and vice versa. There exists three major convective cells between the equator and the poles, each of which circulates the air in a loop, but ultimately transfers heat from the equator to the poles.

Convection cells are one of the main factors that create pressure systems, which are relative highs and lows in the sea level pressure distribution, where air is forced down in high pressure systems and up in low pressure systems. These systems have a very influential role on the generation of rain, wind (in particular the wind's direction and strength), and temperature. High pressure systems are associated with clear skies, dry, and cooler air, while low pressure systems are associated with overcast skies, precipitation, moist and warmer air, although these associations do not always hold true. Air masses of cold and warm air have a similarly influential role on precipitation, temperature, and wind. When a cold front undercuts a mass of warm air, it usually results in localized heavy precipitation, abrupt increase in wind, and drop in temperatures. On the contrary, a warm front rides over the cooler air until it is slowly displaced, which brings about lighter but more prolonged precipitation and over a wider area with slowly increasing temperatures. Cloud formation is caused when water vapour reaches its dew-point temperature by rising into the air where it is cold enough for condensation to occur. Air can be lifted in several ways, the first is simply through convection, the second is with the help of fronts, and the last is through orographic lift. The first two have already been covered, while orographic rainfall is caused when air is forced to rise because of mountains or volcanoes.

### 2.4.2 How forecasts are made

Throughout time, several forecasting methods have been developed. These range from simply looking at the weather with the naked eye to highly complex mathematical models of the atmosphere and oceans that can only be run in a timely manner with supercomputers. The last method described is called numerical weather prediction (NWP) and is the modern way to forecast the weather. The first step to forecasting is almost always to determine the weather's

current state by measuring different factors and constantly monitoring them. Unsurprisingly, the monitored factors are air pressure, temperature, humidity, wind strength and direction. Additionally, both satellite and radar images are extensively used to track the movements of large clouds. Most of these factors are measured using different instruments, at multiple altitudes, and spanning over as wide an area as possible. Once all these measurements are collected, they are analyzed, processed, and modelled using NWP methods to create detailed hourly forecasts. Despite our advances in science and technology, there still remains many challenges before forecasts can be made both further in time and with greater confidence. One of the greatest amongst them, is the fact that the atmosphere is a chaotic system. This implies that even extremely small errors in the initial conditions will amplify and double every five days when fed into numerical models, effectively capping reliable forecasts to no more than 4 to 5 days [21].

While it is obvious that HTM cannot be used to create numerical models in its current state, there exists a forecasting method that could be decently suited for HTM's capabilities called the analogue method/technique. The analogue technique aims to forecast the weather by remembering previous weather events similar to the present ones and predicting the same weather that was recorded. Although there is rarely a perfect analogue of the current weather conditions, HTM could succeed at discerning patterns that no other methods have found.

# Chapter 3

# Code overview

## 3.1 NuPIC - HTM in practice

The intent of this chapter is to give the reader an overview of how the open source project NuPIC implements HTM and discuss some important aspects about the implementation. One matter that needs to be expressed, is that while HTM/NuPIC conceptually works with infinite streams of data, it also works just as well with finite data sets. The data from a file is simply inputted to NuPIC sequentially as if it was a true data stream.

### 3.1.1 The encoder

The encoder that was illustrated in the literature review is actually only one amongst many in NuPIC. Other encoders have been created that are more efficient or that encode other types of input such a category encoder, date encoder, and spatial coordinate encoder. Although the encoders may work in slightly different ways and for different inputs, the same principles that have already been explained apply to them as well. The illustrative example in Section 2.3.3 was a scalar encoder with a vector length $n$ of only 14 and number of ones $w$ of 5 while in practice those values should be no smaller than 100 and 21, respectively. One such encoder is called the Random Distributed Scalar Encoder (RDSE) which uses an $n$ of 400 and $w$ of 21 by default, indicating how long a real distributed representation should be. Using large enough encoder values is critical for the overall HTM process to produce good results. The reason is that if an encoder uses too small parameter values compared to the input space, it may result in encoded values that are poorly distinguishable from each other. Such a scenario would lead to the spatial pooler creating poor SDRs, which ultimately result in the TM struggling to learn sequences when all its input values are seemingly the same.

### 3.1.2 The SP and TM algorithms

The implementation of the spatial pooler follows the theory very close, although the code does have to deal with some minor technical details that were left out in the theory. The first such technicality is inhibition, which is the reason for the brain's small number of simultaneously firing neurons. Inhibitory neurons suppress the other neurons in close vicinity from firing. In the review, this process was accomplished by choosing the 40 columns that had the highest relative overlap score compared to their neighbour columns. This process is called local inhibition since the columns are chosen from the columns' local neighbourhoods. Unfortunately, this choosing procedure is very computationally expensive, creating a serious bottleneck in the system. To alleviate this problem, a selection process called global inhibition has been made, which simply chooses the 40 columns that have the highest overlap score amongst all columns in the layer. While this procedure does not guarantee that all the chosen columns are a certain distance from each other, it actually produces decent SDRs and boosts performance up to 60 times [23]. In fact, global inhibition is so good that it is the default selection process in the spatial pooler implementation.

The second technicality that is implementation specific concerns both the SP and TM algorithms. The theory of SP and TM is based on the manipulation and processing of the properties of HTM cells by the SP and TM algorithms. This means that both algorithms work with the same set of global HTM cells, while in reality the algorithms work with their own set of cell-like objects. It was stated earlier that the SP does not work with cells at the individual level, but rather with columns of cells that all share the same proximal dendrite. The SP implementation simply works with arrays that represent columns that only have proximal dendrites. Likewise, the TM algorithm implementation works with arrays that only have distal dendrites. The author of this thesis speculates that since HTM cells do not act on their own, there was never any incentive or benefit from using global cell objects shared by SP and TP algorithms.

Lastly, here is a look at *some* of the parameter values located in a file called `model_params` that the SP and TM use as their 'cell' attributes. While the principles of HTM are solid, each unique data set that is fed to an HTM system will require subtle tweaking of some of those parameters to make sure that the HTM learns as effectively as possible.

```
spParams:
    columnCount: 2048
    # 1 = global inhibition, 0 = local inhibition
    globalInhibition: 1
    # Number of columns to select as active //
    numActiveColumnsPerInhArea: 40
    # Size of the random subset relative to the input space
```

```
    potentialPct: 0.8
    # Use the C++ or Python implementation of the SP
    spatialImp: cpp
    # Percent amount by which an active synapse is incremented
    synPermActiveInc: 0.003
    # Permanence threshold above which a synapse is considered active
    synPermConnected: 0.2
    # Percent amount by which an inactive synapse is decremented
    synPermInactiveDec: 0.0005
tpParams:
    # Number of synapses that must be both active and connected to
    # active cells for distal dendrite to be marked active
    activationThreshold: 20
    cellsPerColumn: 32
    columnCount: 2048
    # How much the permanence value of
    # cells is to be decreased by
    globalDecay: 0.0
    initialPerm: 0.24
    maxSegmentsPerCell: 128
    maxSynapsesPerSegment: 32
    minThreshold: 13
    newSynapseCount: 31
    permanenceDec: 0.008
    permanenceInc: 0.04
    temporalImp: cpp
```

## 3.2 The CLA classifier

The CLA classifier (CLAc) is the last necessary algorithm for generating actual predictions [24]. It was not included in the literature review because it is not viewed as part of the core HTM theory. The reason for this is that the CLAc is a purely engineered algorithm with no basis in biology. The purpose of the CLAc is to generate predictions, as many steps into the future as desired. Generally speaking, it accomplishes this by connecting past values that were represented in a contextual way by the TM to present ones and then make predictions based on the likelihood distribution of those past values. The CLAc can make predictions $k$ steps into the future for $k$ = 1,2,3,4,... In addition, it can make multiple predictions simultaneously, for example three predictions: 1, 3, and 5 steps into the future. The only issue with the CLAc is that NuPIC can only be used for either anomaly detection or for getting multiple simultaneous predictions. If anomaly detection is chosen, the CLAc will only be able to generate one pre-

diction at each time step instead of multiple predictions. Previously, SDRs have been explained to represent 40 active columns chosen amongst 2048. Although the output from the TM can also be viewed as an SDR, it is more of a contextual SDR, which consists of a vector of the indexes of the active cells. I will therefor refer to these contextual SDRs as CSDRs. Figure 3.1 demonstrates an example CLAc working with a cortical layer consisting of only 10 columns with 4 cells per column.

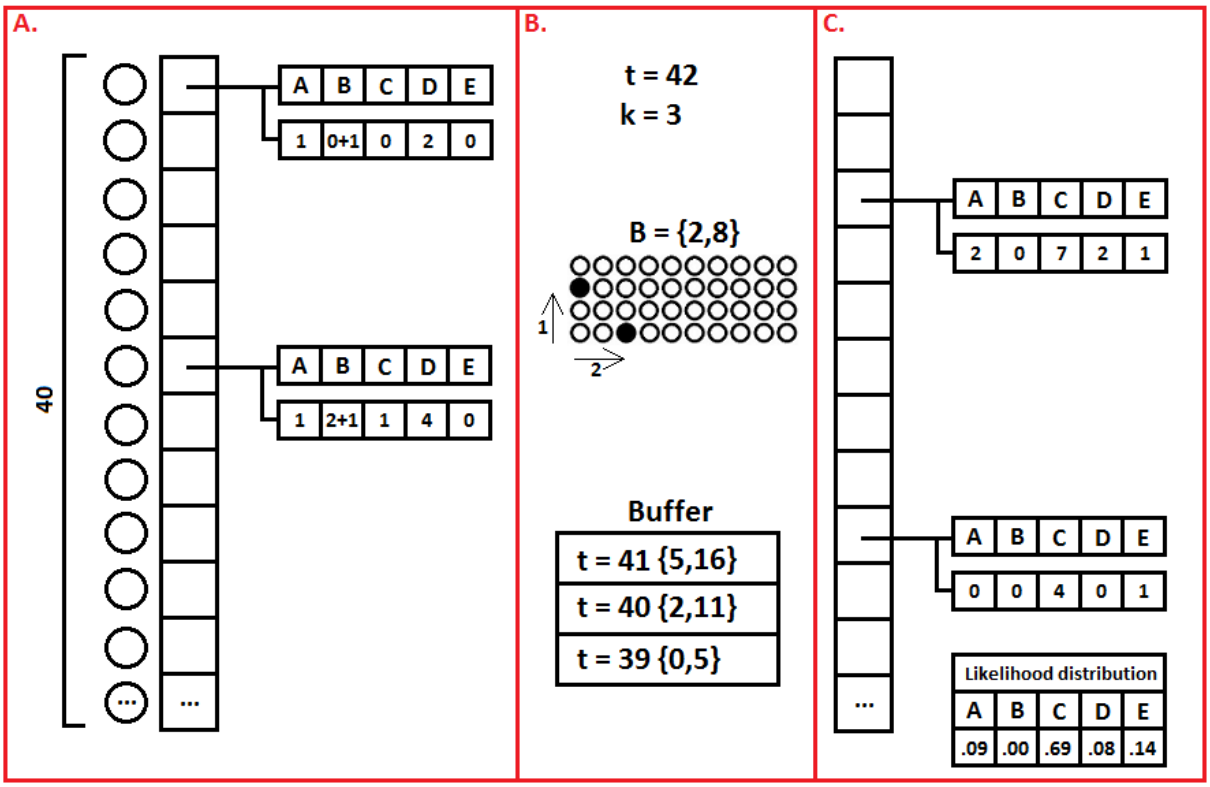



**Figure 3.1:** This figure illustrates the components and processes of the CLAc using an example. In section A, a cortical layer consists of 40 cells, which are displayed on the left. Since the CLAc does not actually operate with cell objects, it represents them with an array of length equal to the amount of cells. Each element in the CLAc's array has two arrays of its own that are the same length. The first array of an element represents the value range in each bucket, which is admittedly irrelevant when working with letters but needed when dealing with real numbers, since the bucket range determines whether a number should fall into one bucket or another. The second array holds a counter for the amount of values that have been placed into each bucket in the first array. In section B, the current time step is 42, and the CLAc must make predictions 3 time steps into the future. The current CSDR received from the TM is the value B represented by the vector $\{2,8\}$ (starting at zero from the bottom left and counting up and then to the right). To bind the past to the present, inside the CLAc's buffer is the stored CSDR from time step 39, which is the vector $\{0,5\}$. Back at section A, looking at the CLAc's elements at the indexes corresponding to the old vector values 0 and 5, the counters for the value B are increased by one. Finally in section C, in the CLAc's elements at the indexes corresponding to the current CSDR vector $\{2,8\}$, the total likelihood distribution is calculated based on the average of all the normalized counter values.

The actual CLAc uses an array that represents all 65536 cells that are in a standard size cortical layer (2048x32). It has a buffer of the last $k$ CSDRs, and receives input from both the encoder and TM algorithms. From the encoder it receives the raw value that it was supposed to encode (at the current time step), the index of the bucket into which it placed the value, and the record number (i.e. number of inputs received so far). From the TM it receives a vector consisting of the indexes of the cells that have been marked active, in other words the current contextual representation of the raw value. Each HTM cell that is represented as an element in the CLAc's array, holds an array of buckets that is similar to the encoder's buckets. Each bucket in that array covers a similar range to the encoder's buckets, which determines whether a number falls within one bucket or another.

Before the CLAc is able to make any predictions, it must first bind the present to the past. It does this by retrieving the $k$th CSDR from its buffer, which was given to it $t$-$k$ time steps ago by the TM algorithm. The vector values of that old CSDR correspond to the indexes of the CLAc's array. Inside each of the CLAc's elements at the corresponding index, lies an array of buckets (explained earlier). Depending on the raw value that the CLAc received from the encoder, it will place that value in the array of buckets, in the accommodating bucket. For each element in the array of buckets, a counter keeps track of the number of values placed inside each of them. The process of placing a raw value inside an array of buckets, is repeated for each of the elements in the CLAc that corresponded to the $k$th CSDR's vector values. In simpler terms, the CSDR from $k$ time steps ago, is taught that the current value appears $k$ time steps after it. Since this process is repeated for each new input, it should be no surprise that some indexes in the CLAc array will have to store more raw values than others in their array of buckets.

With the past stored away, the CLAc can finally generate predictions. Looking at the currently received CSDR from the TM, the focus will once again be on the indexes of the CLAc's array that correspond to the CSDR's values. Each of those elements have their own array of buckets filled with a certain amount of values, which creates a likelihood distribution when normalized. Those probabilities are averaged together to create a grand total likelihood distribution for the current CSDR. In that total, each raw value has a certain likelihood of occurring $k$ time steps after the current one. The value with the highest likelihood is chosen by the CLAc to be the prediction for $k$ time steps into the future. For each additional simultaneous prediction, the CLAc will have to hold an additional array of buckets for each index in its array of size 65536 and repeat the two processes. This means that although the processing of it all is rather moderate, the memory usage can become excessive if too many simultaneous predictions are desired.

## 3.3 Code architecture

The architecture of NuPIC is quite elaborate, as it implements the HTM algorithms to be run in a variety of ways with speed in mind, and works as a flexible platform for Numenta's experimental research. The NuPIC code base consists of 2 main repositories called nupic[1] and nupic.core[2], with the former containing a collection of Python classes and scripts, while the latter contains a collection of C++ classes. These two repositories work in tandem to provide users with three ways to use NuPIC.

**1st way - Low level algorithm routines**
All the HTM concepts that have been explained such as encoding, SP, TM, and anomaly calculation are coded into their own separate classes as low level routines in Python. Running HTM using these routines directly requires that the user has good understanding of HTM theory and extensive knowledge about the NuPIC architecture. Users will need to know how to properly run the different algorithms with proper parameter values, connect them together, and how to handle inputs and outputs. One thing to note is that many of the low level routines that are in the nupic repository have identical C++ implementations in the nupic.core repository, which can be used instead of the Python ones for shorter running times. Although this low level way of usage is the most demanding, it provides the most flexibility and control.

**2nd way - The Network API**
Inside the nupic.core repository is the Network API, which is a flexible API that enables users to run HTM, or specific HTM algorithms and other algorithms in any sequence hierarchy [25]. The Network API consists of a network of regions that can be arranged in almost any topology, which is a highly convenient feature to (Numenta) researchers for research purposes and testing. A region is a container that implements a specific HTM algorithm, although it can implement any algorithm. It provides three things, input(s) to and output(s) from the algorithm, a compute() method that runs one iteration of the algorithm, and get()/set() methods for the algorithm's parameters. A region that implements an encoder algorithm is called a 'sensor region'. Regions are added to a network, which connects them according to the user's chosen topology. Once a network has been created, it can be run, stopped, saved, or loaded at the user's convenience. A 'normal' bare minimum HTM system is simply run by creating a network consisting of a sensor region that implements some encoder, a region that implements the spatial pooler algorithm, and a region that implements a temporal memory algorithm. Such a network would be able to read input from a CSV file and only output contextual SDRs. Adding regions that implement an anomaly score algorithm and CLAc algorithm would give the network the ability to output

[1]Repository url: https://github.com/numenta/nupic
[2]Repository url: https://github.com/numenta/nupic.core

anomalies and predictions as well. The API supports multiple language bindings, with Python being already bound, meaning that although a user runs a network from Python code, the actual implementation is run in C++. Using the Network API is meant to reduce many of the burdens and technicalities of the first method, while still preserving some flexibility and control.

**3rd way - The Online Prediction Framework**

The Online Prediction Framework or OPF for short, is a framework made to work with online learning algorithms, including HTM, to provide predictions from streamed data sources [26]. The OPF is collection of major algorithms working together. Together, the algorithms can run a 'plain' HTM network in a simple and convenient way. The framework is a client of the Network API, that uses a network consisting of a sensor region, SP region, and TM region at the very least. It has several utility classes with interfaces that make it easy to feed raw input from bulk sources (such as CSV files) into the sensor region and neatly format outputs from the system. Most importantly, it has a swarming algorithm that attempts to find the best values for the parameters in the `model_params` file that was displayed in Section 3.1.2. This is an invaluable tool that makes using HTM simpler, although not carefree, as will be revealed later on. In addition to running the three core HTM algorithms, the OPF can either detect anomalies or make multiple simultaneous predictions. In the OPF's configuration file, the former must be marked as doing temporal anomaly, while the latter must be marked as 'temporal multi step'. The OPF can also provide further services although they are not used in this thesis.

## 3.4 Multiple simultaneous inputs

So far, the process of HTM has only been about how HTM works by demonstrating how a single input, or metric, gets processed from start to finish. HTM can in fact be fed multiple metrics simultaneously [27]. It does this by encoding each metric value from each metric separately, and then concatenating all the encodings into one final input. This encoding is then treated as if it were a simple encoding generated from a single metric and passed along to the SP and TM algorithms as normal. Since the multiple inputs are treated as one input, one of the metrics needs to be predesignated as the 'predicted metric' (PM) for which the HTM will attempt to predict its values. This means that the input from the other metrics are treated as extra information which might or might not be helpful in predicting the PM.

One problem to consider is that if there are too many non overlapping encoded values, then the concatenated result would be a vector with much too many ON bits. The difference between the resulting encodings would be so small that the SP would rarely be able to distinguish one input from the next, which would create highly similar SDRs. The TM would view this as a repeating pattern and fail to learn anything useful. According to Numenta, no more than 5 input metrics should be used. Unsurprisingly, not all additional metrics may help predicting the PM.

The additional data should act as a precursor to some change and/or event in the PM that is not easily found in the PM data itself. While it is hard to determine which metrics to include, the swarm takes care of this in the OPF, by looking for helpful correlations over all the additional metrics. If some metric is found to help, then it will be expressed as a percentage of the increase in correct predictions. On the contrary, some correlations may even confuse the HTM and lead to a decrease in correct predictions, which would also be noted.

# Chapter 4

# Methodology

## 4.1 About the data

To conduct the research for this thesis, a number of weather factors were needed. Based on the literature review's climate section, those factors were temperature, air pressure, wind speed, wind direction, and humidity. While weather services need to acquire all these factors over a wide area to create trustworthy models and forecasts, HTM can only deal with those five factors from a single place at a time. Although NuPIC could technically be simultaneously given multiple factors from multiple places, such a thing would be unwise for already discussed reasons. Likewise, numerical weather prediction (NWP) forecasting relies on radar imagery to observe the movement of clouds and cold/warm fronts, something which again cannot be used in NuPIC.

As any ANN algorithm, HTM requires an abundant amount of data in order to properly learn patterns. The consensus is that HTM needs at least 3000 inputs to get a good overview of the data set. The reason for this number is that in such a large data set, diverse patterns would occur enough times for the HTM to discern them, learn them, and be able to make decent predictions based on them. This suggests that giving HTM the weather factors from January only once would not be enough, or even be useful in forecasting the weather for July. So either HTM would need factors from several equal months to properly learn anything about a particular month, or it would need factors from several entire years to learn the variances between months and even seasons. This conclusion is in line with the principles of the analogue method of forecasting.

The next question is whether to use hourly factors or daily factors. In reality those options are not exhaustive, although circumstances dictate there is but one. The only way to find many months and years of weather data is in historical weather archives. Archives do not record

hourly factors because of the shear amount of data that would be stored each year, so they usually only record daily metrics which are the average of the max and min values of the day. There are archives with measurements from several decades ago, although the downside is that they do not store forecasts. Some weather websites provide APIs enabling users to retrieve current weather conditions and forecasts. The forecasts usually range from hourly for the current and next day to daily forecasts up to 25 days. From the previous discussion, several years worth of data from such websites would need to be collected just to get forecasts, something which is impossible to achieve in the span of a master's thesis. Had it been possible though, NuPIC's forecast accuracy could have been compared to NWP forecasts.

Having settled for finding a historical weather archive, I searched extensively and found only a handful of websites/services that provide free weather archives that are downloadable en masse. Since a minimum of 3000 records were needed, it meant that the data set would have to consist of at least eight years of daily measurements. One such website that passed this criteria was eklima.met.no which is the climate database of the Norwegian Meteorological Institute (NMI). The NMI collects weather data from automatic weather stations (AWS) all over Norway including some airports, which provided enough places to choose from. However, data selection was still quite difficult since many AWS do not record all the weather factors required for this research. Additionally, not all stations that do record all the required factors today, have done so in the past eight years. And on top of that, almost all stations have missing measurements, some of which span the length of entire months. The final requirement was that the altitude of the weather station be as close to sea level as possible for the reasons that elevation has additional effects on temperature, wind, and precipitation. Since the weather is already complex enough, it would be best to keep the amount of factors at play to a minimum. In the end, extremely few stations qualified all those prerequisites. One such station which proved promising was the lighthouse at Strømtangen, in the municipality of Fredrikstad, which is located south east in Norway. This station had reliable measurements dating back to January 2004, which meant that at least a decade of factors could be used. Because precipitation measurements were readily available from this weather station, I decided to include precipitation as the sixth weather factor to work on forecasting.

## 4.2 Evaluation

This thesis evaluates how well the practical implementation of HTM called NuPIC can forecast the weather using the analogue method compared to both the actual occurring weather conditions and another less sophisticated analogue method. Additionally, it evaluates the HTM's ability to detect anomalies in the weather data used for forecasting. While it would be advantageous to compare NuPIC's forecast accuracy to actual weather forecasts that use NWP methods

such as in the news, it should be clear by now why it cannot be the case. The intent is to use the unmodified version of NuPIC to attempt to predict the various weather factors discussed for the next four days. Despite NuPIC remaining unmodified, there are numerous parameter values in the encoder, SP and TM algorithms that can be tweaked to adapt and optimize NuPIC for the task.

## 4.3 Related work

### 4.3.1 ANNs used in weather forecasting

The study of meteorology dates back millennia and ANNs have existed for quite a few decades now, so it should come as no surprise that a good deal of research has already been done on the use of ANNs in weather forecasting. A variety of different ANNs have been used, such as multilayer perceptron networks [28], radial basis function networks [29], recurrent neural networks, and ensemble-based networks [30], which outperformed all other solitary networks on prediction accuracy. ANNs have been applied in different ways to forecast various weather factors, including fog [31] and tornadoes [32]. The experiments used data from one or more locations to predict one or more factors. Likewise, the span of the data sets ranged from several days to years, with most factor measurements being collected on an x-hour(s) basis. Summarized, ANNs have been found capable to forecast various weather factors several hours into the future, and in particular when used as a post processing tool [33] for NWP models. Despite the ingenuity of ANNs, numerical weather prediction still remains the best forecasting technique to this day, which indicates that ANNs are no holy grail.

Even though the aforementioned ANNs were very different from HTM, there were some valuable practices that could be applied in this thesis. The first one was using an equation for measuring the success of prediction called the root mean square error (RMSE) defined as:

$$RMSE = \sqrt{\frac{(p_1 - a_1)^2 + (p_2 - a_2)^2 + ... + (p_n - a_n)^2}{n}} \tag{4.1}$$

where $p_t$ and $a_t$ is the predicted and actual value respectively at time $t$, and $n$ is the total amount of predictions. The second one was normalizing the various weather measurements from their different numerical ranges to values between 0 and 1. This normalization would have the benefit of removing the need to fine-tune the parameters of the encoder for every weather factor. The final beneficial practice was removing records that had some missing measurements, instead of doing linear interpolation between the previous and next complete record to fill out the missing records. Given the process that HTM uses to learn, removing records altogether could lead to wrong sequences being learned between values each time such a cut is performed. However, performing interpolation between two values that might be a month apart to fill in the gaps,

would lead to a much larger number of wrong sequences being learned. Therefore, the former option was was adopted.

### 4.3.2 Applications that utilize HTM

Since the development of HTM, several applications have been created both by Numenta as example applications and their commercial partners. HTM for Stocks[1] analyzes a company's stock prices and Twitter feed to detect anomalies in real time and gives alerts when something unusual is happening. Likewise, Grok[2] detects anomalies in servers and applications by monitoring various factors such as CPU, RAM, and Internet connection usage. The last example application is Retina API and Retina Spark [3], which use HTM by converting words from texts to SDRs in order to create 'semantic fingerprints'. These 'semantic fingerprints' can then be used to do things such as compare files for semantic similarity, and search, filter and index texts based on meaning. Despite the HTM's compelling ability to perform predictions, the majority of applications focused on using its anomaly detection capabilities. Since no documented research has been conducted into the utilization of HTM for weather forecasting, this thesis will be the first to do so.

[1] http://numenta.com/htm-for-stocks/
[2] grokstream.com
[3] cortical.io

# Chapter 5

# Running NuPIC

## 5.1 My setup

All research conducted for this thesis was performed on a personal computer with the following specifications, using the NuPIC version 0.5.5.

Host machine

| | |
|---|---|
| CPU | Intel i5-4670K, 4 cores, 3.40GHz |
| RAM | 16 GB |
| SSD | 256 GB |
| OS | Windows 10 Pro, 64-Bit |

Virtual machine

| | |
|---|---|
| VM | VMware Worstation 12 Player, v.12.0.5 |
| CPU | 2 cores, 3.40GHz |
| RAM | 6 GB |
| SSD | 20 GB |
| OS | Ubuntu 14.04 LTS, 64-Bit |

Although NuPIC was mostly developed in Python, it uses a collection of third-party libraries. Since the overwhelming majority of Numenta employees use Mac OS X, those dependencies were mainly managed for that OS. Since OS X has UNIX roots, the management of dependencies was easily adapted to Linux OSes, which led to the Ubuntu OS being supported. Later on, more OSes were able to run NuPIC, including Windows, although users would mostly have to resolve dependency issues on their own. At the beginning of my thesis research, I used the Java implementation of NuPIC called HTM.java, although I stopped because it lacked the swarming algorithm. This problem led me to use the official NuPIC implementation. First, I tried to run it on a Windows OS, but was ultimately unable. Later, I managed to run NuPIC in a virtual machine on a Ubuntu OS.

## 5.2 Trial and error

A good deal of trial and error was needed to successfully run NuPIC. Most of my problems were probably due to incomplete understanding of the NuPIC code and not being able to speak with the original developers of the software. At the start of my research, I ran a great deal of tests with various data sets, fully believing that the swarming algorithm would find the best values for the parameters in the `model_params` file described in Section 3.1.2. While it may have been naive of me to do so, the NuPIC documentation did not warn me of any issues. Unfortunately, the parameters obtained from the swarm resulted in bad and inconsistent predictions, while the anomaly scores were either too high or too low throughout the data. Although I was using the Online prediction Framework (OPF), I still had to manually input certain properties pertaining to the swarming algorithm and manage the extraction of results from the OPF. To automate all these tasks and make testing more effective, I modified a management script created by a Numenta employee, which takes care of an entire HTM run from start to finish. Such a run includes providing the necessary properties to the swarm and running it, feeding the input data to the OPF with the parameters provided by the swarm, extracting output predictions and anomaly scores from the OPF, and finally running a performance test on the results.

After having carried out a multitude of tests, I came to the conclusion that the swarming algorithm often used the values 21 and 22 for the encoder parameters $w$ and $n$ respectively. So although the size of $w$ was fine, the size of $n$ meant that according to Equation 2.1, the encoder used only two buckets. This meant that half of all input values were encoded to one representation, while the other half were encoded to another representation. Consequently, HTM was learning and predicting an interpreted pattern that was very different from the intended one. While increasing the value of $n$ was fairly easy, determining the best value was not. Testing revealed that having exactly one bucket for every value within the data range gave quite poor results, while having 4 to 5 times as many buckets compared to values yielded the best results, which was achieved with an $n$ of approximately 500.

Despite an improvement in predicted results after fixing the issue above, there was still something awry with the predictions. It seemed as if the predictions became increasingly better in the first couple of hundred inputs, until they slowly began to degrade, even though the same simple input pattern was repeated endlessly. After more testing, I located the problem to a parameter called maxBoost. Briefly explained, maxBoost was an implementation specific process used by the SP to increase the activity of columns that were unused, in order to maintain the sparsity of SDRs. Deactivating this process improved prediction results considerably, and was later confirmed to be a known issue by Numenta employees.

The final issue only appeared halfway through the thesis year when the CLA classifier (CLAc) was updated to a newer version called the SDR classifier, which creates likelihood distributions

using a simple feedforward ANN. Although it performed better than the CLAc, it had an issue that caused all predictions to be identical to the input value at that time step. I found the issue to be caused by a classifier parameter called alpha, which simply described, determines how fast the classifier should learn new values at the cost of forgetting older ones. When this alpha was below a certain threshold, it caused the SDR classifier to malfunction. I avoided this issue by simply using the CLAc when using alpha values below the threshold and using the SDR classifier when above it. Discovering and resolving these small issues provided much insight into NuPIC, though they invalidated many of my prior tests and forced me to redo a lot of work multiple times.

## 5.3 Simple data sets

Learning the theory of HTM was the first step to understanding the algorithms. Learning about NuPIC and its architecture was the second logical step in gaining an understanding of the practical side of HTM. Running NuPIC was the third and final step in comprehending it all. Although I cannot show everything that I have tested, I can demonstrate some NuPIC runs using diverse data sets under controlled conditions. The following demonstrations and experiments have been executed with all the previously described issues resolved. In addition, to minimize the amount of potential errors, I reused the same encoder parameters by simply normalizing the data sets to values between 0 and 1 and used no more than 100 different values. For the demonstrations, I devised of two ways to assess the prediction accuracy of a pattern. The first is the most strict and requires calculating the amount of times a pattern needs to be repeated (i.e the number of pattern iterations) for NuPIC to perfectly predict an entire pattern. The second way is similar but more forgiving, by calculating the amount of pattern iterations needed to correctly predict 90% of all values in a pattern. To be fair, there was no scientific reason for choosing the 90% mark, I simply believed it was a high enough success rate to warrant satisfactory results for most cases. Furthermore, an argument against solely using the former metric was that waiting for NuPIC to perfectly learn an entire pattern would be both unnecessary for most tasks, and subject to some issues that will be discussed later.

**Linear pattern**

The simplest pattern that I could create was a linear pattern: 0, 1, 2,..., 100, 0, 1, 2, etc repeated 40 times. For this example I performed the test predicting only 1 step into the future. The graph in Figure 5.1 shows that at the very beginning, when the pattern was seen for the first time, NuPIC had no idea about any future values and predicts seemingly random values that were previously seen. On the next pattern iteration, the predictions are much better, though obvious prediction errors can be witnessed later on. The anomaly score was unsurprisingly high throughout the entire first iteration of the pattern, and fell off very quickly afterwards. The

anomaly likelihood however begins always at 0.5 and needs a couple of hundred records to process before it can start producing actual anomaly likelihood scores. Like the former score, it drops down, though not completely. In total, it took 11 pattern iterations for NuPIC to correctly predict 90% of the pattern, and 20 iterations to learn it perfectly with an anomaly score of 0 throughout it all in the end. The erroneous predictions that can be seen between time step 500 and 900 happen randomly but not very often and they always eventually cease to happen as NuPIC learns a pattern more and more. An explanation of these erroneous predictions is provided in the next chapter due to the complexity of the problem. While it is inconvenient to display more than one prediction at a time, NuPIC successfully managed to make multiple simultaneous predictions for 1, 2, 3, and 4 time steps into the future. Since the CLAc is responsible for creating all predictions irrespective of the number of time steps, it took just as many iterations as previously to correctly predict 90% of the pattern with all four predictions simultaneously.

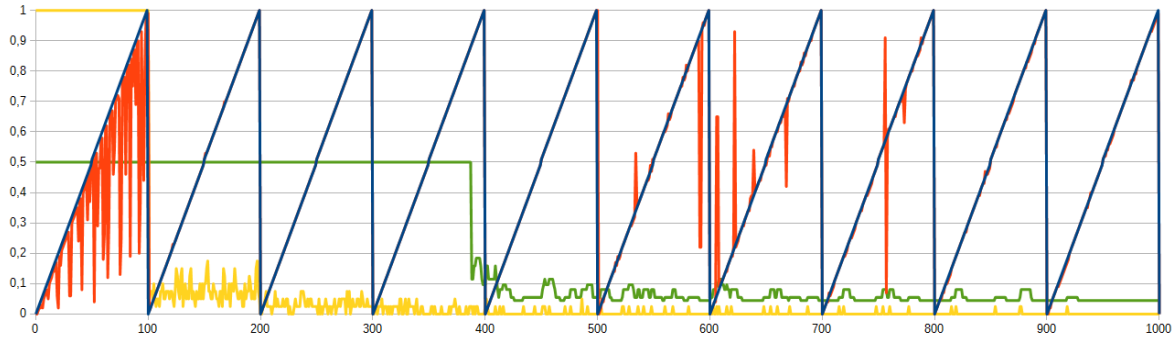



**Figure 5.1:** This graph shows how NuPIC handles a linear pattern. The x-axis represents the time steps, and the y-axis represents the values at the corresponding time step. The blue line represents the linear pattern, and the red line represents NuPIC's predictions. The values have been aligned so that at any given time step, the predictions can be compared to the actual values at that time step. The yellow line represents the anomaly score and the green line represents the anomaly likelihood score.

### Random patterns

The next demonstration was intended to show NuPIC's ability to learn randomly generated patterns. It is reasonable to assume that longer patterns with more convoluted values would take longer to learn. So to show this, random patterns were created using different lengths and different value ranges to vary the concentration of values. The results have been gathered in Figure 5.2, and show that there was no apparent connection between the length of a pattern and the amount of iterations necessary to learn it. Likewise, there was no obvious connection between the concentration of values and number of iterations needed. The only apparent thing was that approximately 20 pattern iterations of the same pattern were required to learn it using the 90% metric.

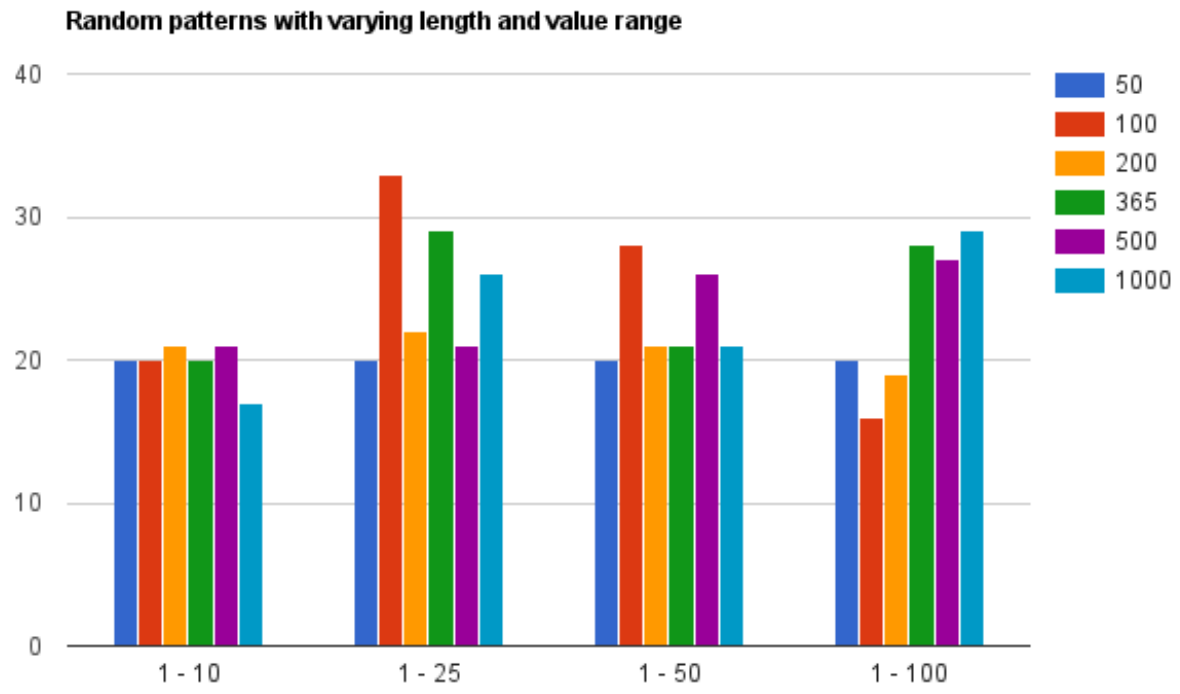



**Figure 5.2:** A graph demonstrating the amount of pattern iterations necessary to learn a pattern based on the 90% metric, denoted by the y-axis. The x-axis shows the range of the random values, while the colored bars represent the length of each pattern.

### Generalization

Despite the fact that NuPIC is all about learning patterns, it cannot generalize its knowledge about learned patterns, given how cells in the TM represent specific values and make connections to other distinct cells. For example, if NuPIC was to learn a pattern consisting of several numbers and then be given a series of new numbers arranged in a similar pattern to the previous one, but with either higher, lower, or stretched values (i.e. multiplied by common a factor), it would be unable to correctly predict any of them. Additionally, it would output very high anomaly scores for each new value, and more while it would be learning their pattern.

When contemplating about this issue, I came up with the idea of making NuPIC's anomaly detection less sensitive to new values, and focus more on anomalous patterns irrespective of the specific numbers. I would achieve this by abstracting over the distinct numbers in a pattern, by calculating both the differences and ratios between subsequent values. This would ensure that any previously learned pattern would still be recognised no matter if it were shifted higher, lower, or stretched. To accomplish this in practice, I would simply pre-process a data set by performing the aforementioned calculations and store the results into their own separate files. Next, I would have to run NuPIC over the original data and the new data files. Finally, I would gather the anomaly scores of all three data sets, and calculate their average, producing a new

encompassing anomaly score for each record. My theory was that such a composite anomaly score would still retain the anomaly detection capabilities of the original one. However, it would now need the support of at least one of the other two anomaly scores, to become high enough to warrant interest. This theory will be tested in the next section.

## 5.4 Oil prices

This section is both a demonstration and an experiment performed using a real world data set, consisting of daily crude oil dollar prices spanning from January 1986 to December 2015. The purpose of this experiment was to observe how HTM would work with real data that is both messy and quite unpredictable. The unpredictability is caused by many factors that influence the price of oil, ranging from small daily fluctuations to economic and geopolitical events [34]. I presumed that NuPIC would not be able to predict any of the sudden drops or increases in oil prices due to the previously mentioned reasons, since it would have no way of receiving information from other sources. However, I was very interested in exploring the anomaly detection capabilities, and seeing if it would raise any red flags at the moment of, or better yet, prior to any such big event. For this experiment, I ran NuPIC through the entire data set only once.

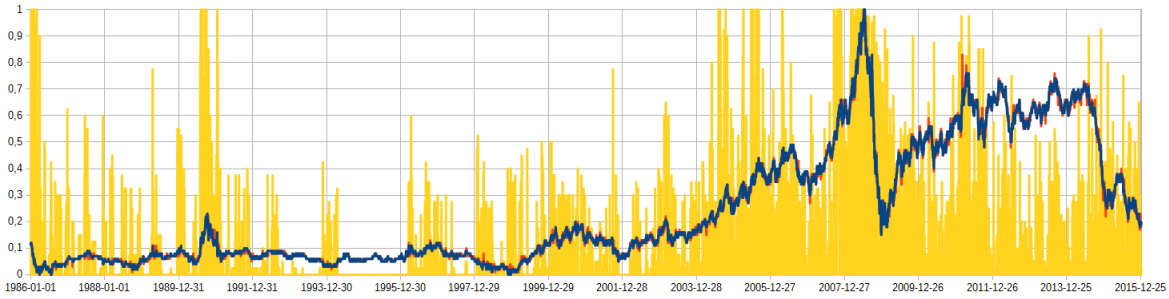



**Figure 5.3:** An overview of the oil price from 1986 to 2015. The blue line represents the actual oil price, the red line represents next day predictions, and the yellow spikes represent anomaly scores.

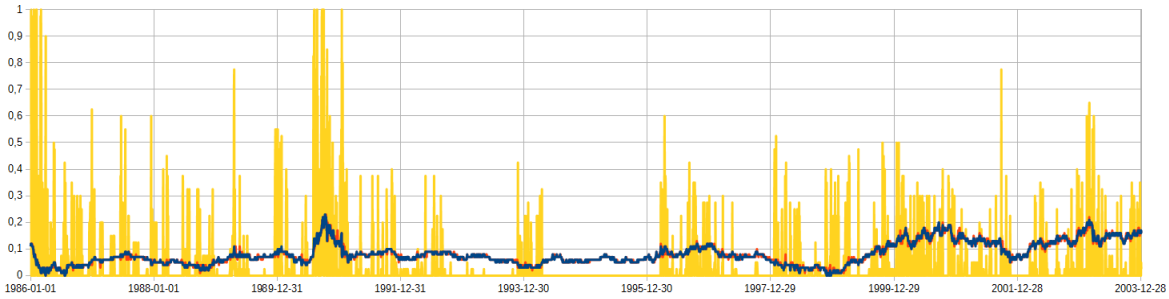



**Figure 5.4:** A close up of the graph between 1986 and 2003.

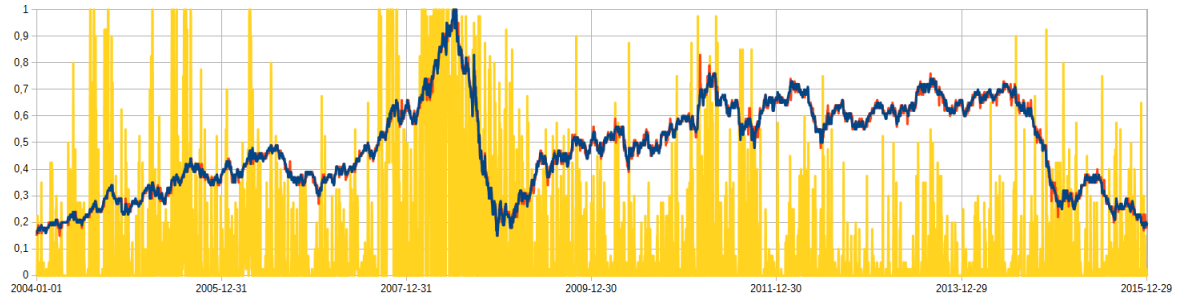



**Figure 5.5:** A close up of the graph between 2004 and 2015.

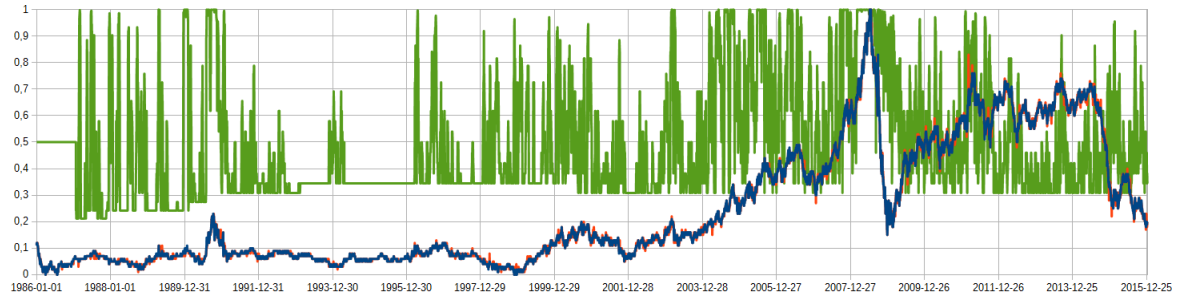



**Figure 5.6:** An overview similar to Figure 5.3, displaying the anomaly likelihood score results.

The results in Figure 5.3, 5.4, and 5.5 revealed that real life was indeed messy, with anomaly spikes throughout most years, with particularly many spikes from 2004 til the end of 2015. Figure 5.6 showed that using the anomaly likelihood metric resulted in even more spikes making it virtually useless in this experiment. The reason that there were so many spikes during the latter period was as I hypothesised and discussed, the fact that NuPIC could not generalize any of its previously learned patterns. This meant that every new value that had never been seen before was regarded automatically as an anomaly, even if it had occurred in a pattern that was previously learned before, albeit with lesser values. In addition to all that, the oil prices between the years 2004 and 2015 were very unstable compared to 1987 and 2003, which led to additional anomaly spikes. For this reason, I decided to divide the data into those two periods. This way, it would give a more correct picture of NuPIC's ability to detect anomalies. The year 1986 was also artificially anomalous due to the data still being very new to NuPIC, so I decided to ignore everything prior to 1987.

I collected the anomaly data into Table 5.1 by separating it into the two mentioned periods. Each period was then divided into the amount of anomalies that had an anomaly score $\geq 0.5$, $\geq 0.75$ and, $\geq 0.9$. The first two rows contained the anomaly and anomaly likelihood scores, while the last two contained the composite anomaly and composite anomaly likelihood scores. In the first row, there were 403 anomalies between 1987 and 2015 with a score $\geq 0.5$, which was far too many for me to investigate. However the corresponding amount for the anomaly

likelihood row was far bigger with no less than 4320 anomalies. I therefore turned my focus on the anomalies with scores $\geq 0.75$, and although there were still too many in total, there was a manageable amount in the first period. I researched the dates of the 24 anomalies by doing both some general history research and searching in a series of Wikipedia pages [35] with information about major historical economic and geopolitical events that affected the price of oil. According to my research, 13 of the 24 anomalies coincided directly to the dates of such major events. All of them were caused by events resulting from the Gulf war with the exception of the last one which was caused by fears of an incoming economic recession. Four anomalies seemed to be caused by the aftermath of major events, because they all happened the day after one. Of the remaining seven, only one anomaly could potentially be considered an anomaly foreshadowing event, as it happened the day before the Oil Pollution Act of 1990 was enacted into law in the US. Of the 14 with scores $\geq 0.9$ between 1987 and 2003, only seven coincided with economical and/or geopolitical events that happened that day. Amongst the remaining seven, three were aftermath events.

| | 1987-2003 | | | 2004-2015 | | | 1987-2015 | | |
|---|---|---|---|---|---|---|---|---|---|
| | $\geq$0.50 | $\geq$0.75 | $\geq$0.90 | $\geq$0.50 | $\geq$0.75 | $\geq$0.90 | $\geq$0.50 | $\geq$0.75 | $\geq$0.90 |
| A. | 61 | 24 | 14 | 342 | 185 | 126 | 403 | 209 | 140 |
| A.L. | 1740 | 552 | 312 | 2580 | 1179 | 737 | 4320 | 1731 | 1049 |
| C.A. | 51 | 16 | 5 | 262 | 62 | 11 | 313 | 78 | 16 |
| C.A.L. | 1332 | 272 | 91 | 2472 | 975 | 477 | 3804 | 1247 | 568 |

**Table 5.1:** A. = Anomaly scores, A.L. = Anomaly likelihood scores, C.A. = Composite anomaly scores, C.A.L = Composite anomaly likelihood scores. Each column in the table represents the amount of anomalies with scores greater than or equal to X using the corresponding anomaly metric. The total period between 1987 and 2015 has been divided into the two intermediate periods of 1987 to 2003 and 2004 to 2015 for distinctness.

In the period between 2004 to 2015, NuPIC detected 185 anomalies with an anomaly score $\geq 0.75$. While I could have painstakingly checked each one of them, I knew that many of them were caused by NuPIC reacting to both completely new values, and learning their patterns for the first time. Therefore, I decided it was time to see how well my composite anomaly score method would fair in this experiment by running it through the steps described earlier on. Looking at the results for the first period from 1987 to 2003 once more, only 16 anomalies were recorded to have a composite score $\geq 0.75$. Again, eight coincided directly with major events and one was an anomaly potentially foreshadowing an event. The "red flag" happened the day prior to an official announcement from the American Petroleum Institute about a decrease in the nation's oil inventory. Amongst the five anomalies with a composite score $\geq 0.9$, four coincided

with major events. In the second period between 2004 and 2015, even though there were only 62 anomalies compared to 185 with scores $\geq 0.75$, there were still far too many anomalies for me to investigate. I therefore decided to only look at the anomalies with scores $\geq 0.9$, of which there were only 11. I found that five were coinciding with major events, and one was the aftermath of an event. The majority of the coinciding events were caused by the 2008 global financial crisis. I added the score results of both the anomaly likelihood and composite anomaly likelihood for perspective and completeness. It became obvious however, that there were far too many anomalies for me to research all of them.

This experiment revealed that the plain anomaly metric was rather decent at finding anomalies in the data that could be linked to actual real world events. However, the constraint was that the data must remain in its original range for a while to give NuPIC enough time to learn the patterns within. The composite anomaly method was also quite successful when used on the first period between 1987 and 2003. Although it was less successful during the second period between 2004 and 2015, it still managed to give some relevant results when the data landscape was very new and unstable. The anomaly likelihood and composite anomaly likelihood metrics however were essentially useless unfortunately, which led me to abandon them.

## 5.5 The weather

Given that NuPIC could be used for both prediction and anomaly detection purposes, I decided to divide the running of the weather data into those two main categories. I would first research NuPIC's ability to forecast the weather using its different available techniques and with differently composed data sets. Subsequently, I would research to what extent NuPIC could detect anomalies in the weather. I already discussed the weather data that I would utilize for this thesis in Section 4.1, so I will now give some more details about it. The weather data was collected from a lighthouse located south east of Norway and spans the period from the 1st of January 2003 to the 31st of December 2015. Although the period contains 4747 days in total, I had to remove 214 records from the data set due to missing measurements or erroneous values, for a final total of 4533 records. The data consists of six weather factors that were recorded by calculating the daily mean from the corresponding amount of measurements.

| Factor | Unit | Measurements/day | Max value | Min value |
|---|---|---|---|---|
| Wind direction | degrees | 3 | 360 | 0 |
| Wind speed | m/s | 4 | 18.2 | 0.5 |
| Atmospheric pressure | hPa | 24 | 1047.4 | 966.0 |
| Precipitation | mm | 24 | 84.4 | 0.0 |
| Temperature | ºC | 24 | 24.1 | -15.0 |
| Humidity | % | 24 | 100 | 32 |

**Table 5.2:** This table shows information about the recorded data of each weather factor. The unit of measurement shows which metric was used to record each factor. The measurements per day column shows how many times each factor was measured during a day. The columns for the max and min values show what the maximum and minimum recorded values were for each weather factor.

Table 5.2 shows that there is a disparity between the amount of wind related measurements per day and all the other measurements, which is unfortunate for the sake of consistency. Despite each factor's unit and range of values, I also discussed that I would follow the helpful practices of previous researchers by normalizing all the measurements to values between 0.00 and 1.00 (both inclusive). Looking at the max and min values of each factor, there was no issue normalizing them, apart from the wind direction which lost some of its original resolution. It should not be an issue however since the eight cardinal directions are usually enough for most forecasts.

### 5.5.1 Predicting the weather

Originally, my intentions were to first run NuPIC individually for each weather factor, and then run NuPIC by inputting multiple factors simultaneously, albeit only to predict one factor at a time. Unfortunately, my latter objective was unachievable because NuPIC could not correctly process multiple simultaneous inputs. It produced worse predictions with multiple inputs than single inputs even when its own calculations were telling otherwise. My discovery was later tested and confirmed by Numenta employees. I would therefore have to keep my research to only one metric per run.

In Section 4.1, I explained about using the Root Mean Square Error (RMSE) as the performance metric for assessing predictions. The smaller the RMSE score, the better the predictions, since it means that the differences between the predictions and the actual values were small. Since I normalized the measurement values, the RMSE score would appear artificially small, so I decided that I would multiply the scores by 100 to make them more easily readable and comparable to each other. To better evaluate the performance of NuPIC's predictions, I concluded

in Section 4.2 that I would use another less sophisticated analogue method as comparison. This method would be to simply use the average of the last $x$ days of measurements as the basis of its next four days of forecasts.

Finally, the last detail was to determine how many records would be used as the training set and test set. The length of the data spanned 12 years, with each year representing the four seasonal weather cycles. This meant that the minimum test set should contain one year of data. As for the training set, Numenta's recommendation was to use at least 3000 values for proper training, so the training set would have to consist of at least nine years. I decided to use an even number and increased the training set to ten years which left one spare year that was added to the test set for a total of two years. Summarized, it meant that each separate data set would be input into NuPIC from the first year to the last. The predictions from the first ten years would be ignored, while the predictions for the last two years would be used as the basis for evaluating the RMSE score. One way to expand the training set using what I learned when testing the random patterns, was that HTM needed multiple pattern iterations to predict them better. I therefore decided to run two additional separate series of tests that use 25 and 50 consecutive iterations of training sets, which I will refer to as 'training set iterations' or TSI. In other words, each data set would consist of the first ten years repeated 25 or 50 times consecutively, followed by the last two test set years.

Figures 5.7–5.12 show the (normalized) recordings of the last two years of each weather factor in blue. The red line shows NuPIC's predictions for the next day and has been shifted to show how well the predictions overlap or differ with the actual values.

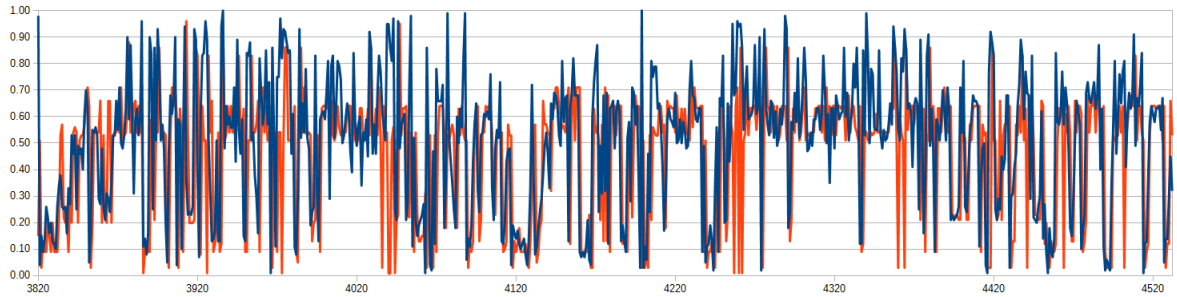



**Figure 5.7:** Wind direction. All these graphs were based on the results from running only one training set iteration. All these figures show the results from the last two years of the data sets, i.e the test sets. The x-axis denotes the time steps into the data set, while the y-axis denotes the values.

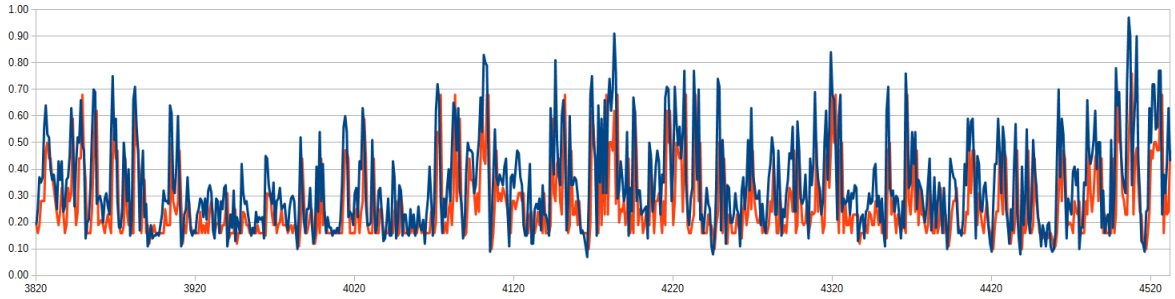



**Figure 5.8:** Wind speed.

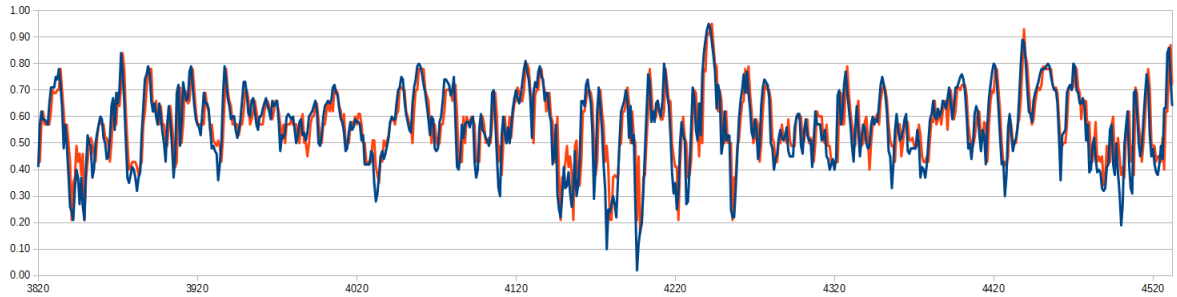



**Figure 5.9:** Pressure.

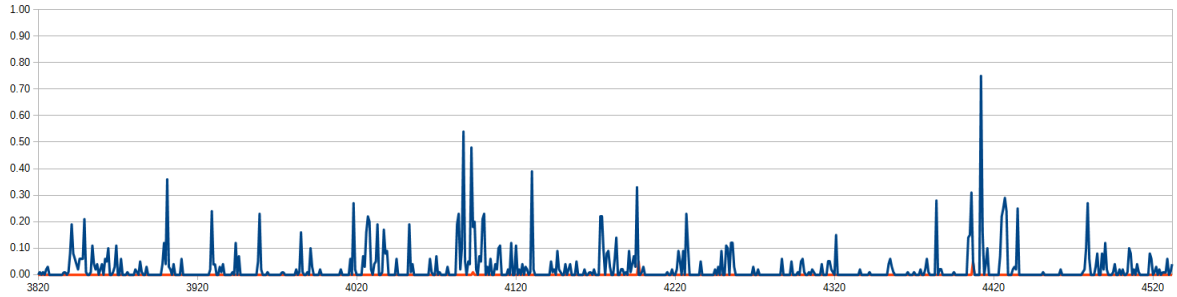



**Figure 5.10:** Precipitation.

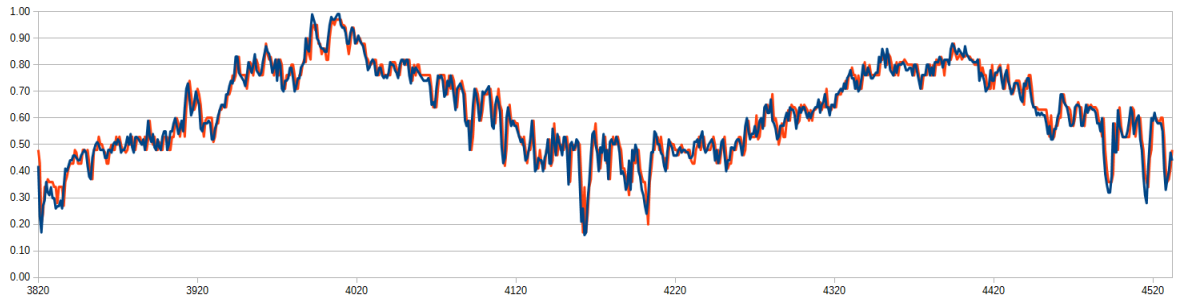



**Figure 5.11:** Temperature.

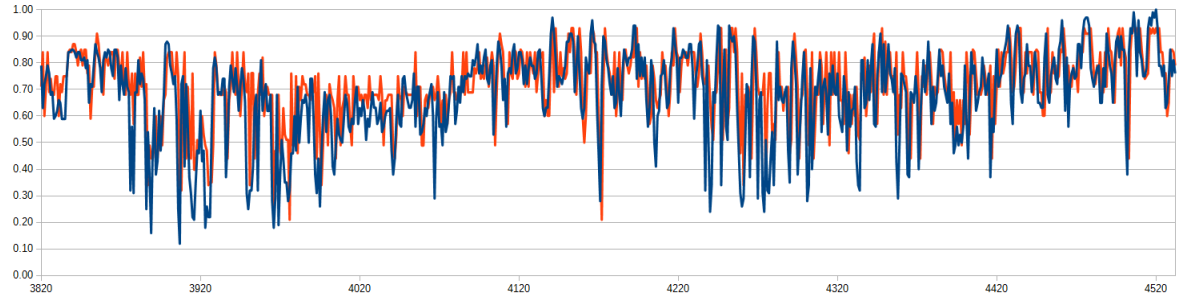



**Figure 5.12:** Humidity.

| | | RMSE scores from using NuPIC | | | | | |
|---|---|---|---|---|---|---|---|
| | | Wind direction | Wind speed | Pressure | Precipitation | Temperature | Humidity |
| 1 TSI | 1D | 27.54 | 17.16 | 8.31 | 7.15 | 4.50 | 14.85 |
| | 2D | 29.32 | 21.44 | 11.83 | 7.16 | 6.46 | 18.08 |
| | 3D | 32.24 | 21.78 | 13.57 | 7.16 | 7.40 | 18.73 |
| | 4D | 32.38 | 21.63 | 14.29 | 7.16 | 8.03 | 18.61 |
| 25 TSI | 1D | 28.50 | 18.40 | 8.90 | 7.23 | 4.80 | 15.90 |
| | 2D | 31.03 | 21.23 | 12.89 | 7.22 | 6.59 | 19.43 |
| | 3D | 32.09 | 21.68 | 14.89 | 7.17 | 7.62 | 18.95 |
| | 4D | 32.65 | 21.86 | 15.41 | 7.16 | 8.46 | 19.51 |
| 50 TSI | 1D | 29.54 | 19.56 | 10.67 | 7.91 | 5.53 | 17.95 |
| | 2D | 32.39 | 20.14 | 13.08 | 7.95 | 6.50 | 18.43 |
| | 3D | 34.62 | 21.92 | 15.21 | 7.76 | 7.95 | 20.70 |
| | 4D | 35.57 | 22.19 | 17.71 | 8.30 | 8.84 | 21.13 |

**Table 5.3:** TSI = training set iteration, XD = scores for the predictions $X$ days ahead. The columns represent the RMSE values of each distinct weather factor, while the three sections of rows contain the results for the various iterations over the training set. Within each TSI, each smaller row shows the RMSE scores for forecasting one to four days ahead.

I collected the RMSE scores from all the aforementioned tests in Table 5.3. Looking at it, one of the most surprising things was that the RMSE scores were better for almost all weather factors when NuPIC iterated only once over the training set rather than 25 or even 50 times. The only exception seemed to be the wind speed, which had a better one and four day forecast with one training set iteration but slightly better two and three day forecast using 25 iterations. The next point was the clear distinction in score performance for each weather factor, with the temperature scores having the lowest RMSE values and wind direction the highest values. While the RMSE scores on their own were compelling, they did not reveal much on their own.

So I created predictions for the next four days using the averaging technique discussed earlier and noted the RMSE scores in Table 5.4.

| | | RMSE scores from using the average of the last $X$ days | | | | | |
|---|---|---|---|---|---|---|---|
| | | Wind direction | Wind speed | Pressure | Precipitation | Temperature | Humidity |
| last 1 day | 1D | 27.04 | 16.34 | 8.08 | 8.05 | 4.33 | 14.30 |
| | 2D | 31.35 | 20.76 | 12.47 | 8.90 | 6.16 | 18.43 |
| | 3D | 32.64 | 22.18 | 15.05 | 9.16 | 7.23 | 19.26 |
| | 4D | 34.15 | 22.77 | 16.74 | 9.20 | 7.98 | 19.92 |
| last 3 days | 1D | 25.54 | 17.01 | 10.86 | 7.26 | 5.36 | 14.89 |
| | 2D | 28.22 | 19.29 | 13.79 | 7.71 | 6.62 | 16.90 |
| | 3D | 29.68 | 20.10 | 15.63 | 7.71 | 7.40 | 17.84 |
| | 4D | 30.04 | 20.41 | 16.84 | 7.58 | 7.92 | 18.30 |
| last 7 days | 1D | 25.48 | 17.08 | 13.13 | 6.76 | 6.36 | 15.17 |
| | 2D | 26.70 | 18.24 | 14.87 | 6.92 | 7.01 | 16.22 |
| | 3D | 27.40 | 18.76 | 16.07 | 6.97 | 7.43 | 16.67 |
| | 4D | 27.40 | 18.94 | 16.94 | 6.95 | 7.73 | 16.82 |

**Table 5.4:** Last X days = average based on last X days. The columns represent the RMSE values of each distinct weather factor, while the rows are divided into the amount of last X days. The 'last X days' refers to the amount of past day values used to calculate an average value used to predict the next one to four days. Within each main row, a smaller row shows the RMSE scores for forecasting one to four days ahead.

The averaging technique generated forecasts by calculating the average of the last one, three or seven days and using the result as the predicted conditions for the next one to four days. Table 5.4 shows the RMSE scores from using the averaging technique. The score distribution amongst the weather factors displayed many similar features to the previous table. Here however, there is a distinction between the factors as to how many past days the average should be calculated from to produce the most accurate forecasts. While in Table 5.3, it was clear which TSI row generated the best RMSE results for each weather factor, it was not as easy to see which 'last X days' main row generated the best RMSE results in Table 5.4. Since RMSE scores are better the lower they are, for each column I added up the scores in each main row and chose the main row with the lowest total as the one with the four best forecasts. This meant that the averaging technique produced the best forecasts based on the average of the last seven days for the wind direction, wind speed, precipitation, and humidity factors. The pressure and temperature factors on the other hand, were best predicted by using the average of the last day, i.e forecast the next four days to have equal conditions to the current one.

| | NuPIC's RMSE scores compared to the best average RMSE scores | | | | | |
|---|---|---|---|---|---|---|
| | Wind direction | Wind speed | Pressure | Precipitation | Temperature | Humidity |
| 1D | +8.08% | +0.47% | +2.85% | +5.77% | +3.93% | -2.11% |
| 2D | +9.81% | +17.54% | -5.13% | +3.47% | +4.87% | +11.47% |
| 3D | +17.66% | +16.10% | -9.83% | +2.73% | +2.35% | +12.36% |
| 4D | +18.18% | +14.20 | -14.64% | +3.02% | +0.63% | +10.64% |

**Table 5.5:** The percentage values denote how much higher or lower NuPIC's RMSE scores were, compared to the RMSE scores of the averaging technique. Positive values mean that NuPIC's RMSE score was higher and thus performed worse than the averaging technique, while negative values mean that NuPIC's RMSE score was lower and thus performed better.

Table 5.5 shows how NuPIC's RMSE scores compare to the scores from the average technique. They indicate that all weather factors except for the pressure and humidity were better predicted using the much simpler averaging technique, than the sophisticated HTM algorithms. NuPIC's results on the atmospheric pressure increasingly outperformed the averaging technique by a fair amount when forecasting two, three, and four days ahead. Humidity was the second factor where NuPIC was able to outperform the averaging technique, albeit by a smaller amount than previously and only for forecasts one day ahead. As for the rest of the factors, the averaging technique yielded consistently better results, with NuPIC's scores being +0.47% higher at the least to +18.18% at the most.

### 5.5.2 Detecting anomalies in the weather

To fully test for anomalies, I decided to make use of both the normal anomaly detection approach and my own composite anomaly detection method. I would run NuPIC on each weather factor individually with both methods, and then average their results using multiple combinations of factors, to see if any of them yield interesting results. The reasoning for this was that a weather anomaly could present itself through multiple factors instead of just one. Extreme weather events and phenomena such as storms, tornadoes, and hurricanes could be prime examples of anomalies, even though tornadoes and hurricanes are rare in Norway. Researching extreme weather events, I found that between 2003 and 2015, eight extreme weather events were recorded in south eastern Norway. Of those eight, only one hit the area of Strømtangen particularly hard.

After having run NuPIC through several combinations of factor anomalies, and looking only at anomalies with scores equal to or above 0.75, I found that the number of anomalies changed from 2 to 91 depending on the exact combination. Figure 5.13 shows such a graph that was made by taking the average of the anomaly scores from all six weather factors. None of its

anomalies with scores $\geq 0.75$ corresponded to any extreme weather events. Regardless of the weather factor combinations, none of the anomalies matched any of the dates of the eight extreme weather events. There were also no signs of any aftermath or foreshadowing anomalies surrounding those dates. Lastly, my composite scores method did not provide better results. This can be seen in Figure 5.14 as barely any anomalies with scores $\geq 0.75$ are present.

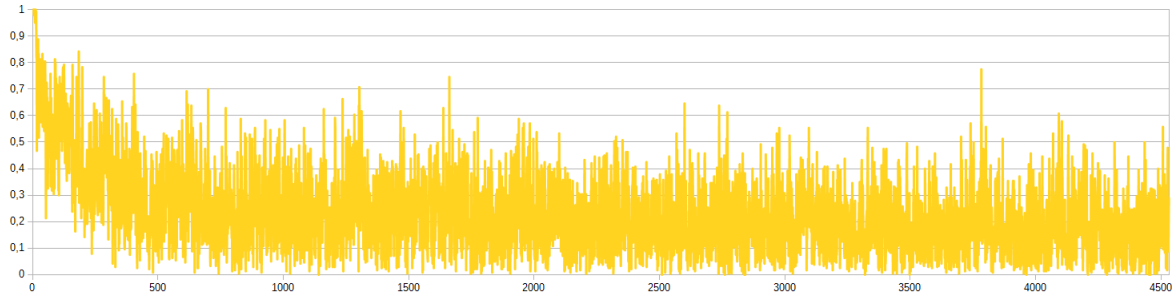



**Figure 5.13:** Average of anomaly scores from all weather factors.

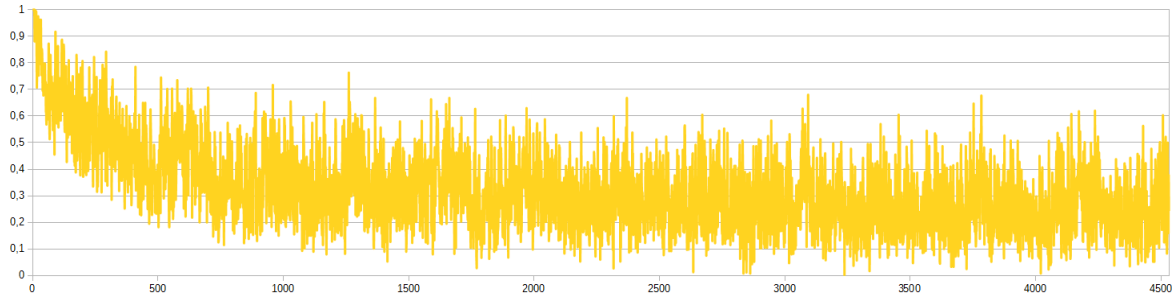



**Figure 5.14:** Average of composite anomaly scores from all weather factors.

## 5.6 Summary

The results in this chapter have shown that NuPIC is capable of learning complex patterns in varying real world data sets and both provide decent predictions and compelling anomaly detections. However, the results also show that it has clear limitations which affects the data sets that it can reliably work with. While it was able to create modest weather forecasts, it was predominantly outperformed by a much simpler analogue technique, although not completely. It was not able to detect any noteworthy anomalies in the weather data, though it did detect very interesting anomalies in the oil data that were connected to real world events. All these findings indicate that while NuPIC is a competent algorithm, the quality of its results are highly dependant on whether the properties of a data set conform to its strengths.

# Chapter 6

# Analysis and discussion

## 6.1 Peculiar predictions

The linear pattern discussed in Section 5.3 was supposed to demonstrate NuPIC performing the very simple task of predicting a linear pattern. One of the things that really caught my attention were the blatantly wrong predictions that appeared at random places throughout the pattern. It was in part because of those errors that I had to create a distinction between the 90% accuracy mark and 100% mark. Looking at the linear pattern test, this issue clearly appears multiple times. What was even more surprising to me was the fact that it looked like NuPIC managed to learn the pattern several times and generate very accurate predictions, yet repeatedly produce such wrong predictions. Looking at time step 591 for example, NuPIC received the value 0.92, and should have predicted the value 0.93. Instead it predicted 0.22, even though the sequence ..., 0.92, 0.22,... had obviously never occurred in the past. Looking at the code, I found that the TM algorithm was predicting the correct cells located in the columns that would represent the next SDR value of 0.93. It was in fact the CLA classifier (CLAc) who was to blame for generating the incorrect prediction. In its likelihood distribution, it had several probable predictions. These were the values 0.34, 0.22, 0.92 and 0.94, with the respective likelihoods of 0.11%, 99%, 0.6% and 0.24% (rounded off).

To understand the following explanations, it is crucial that the reader understands how the CLAc works (explained in Section 3.2). Additionally, because I often use the term contextual SDRs, I will refer to them as CSDRs. At first I was most surprised to find the value 0.92 in the CLAc's likelihood distribution of the CSDR 0.92. I then found that all new CSDRs that have no likelihood distribution in the CLAc, are automatically provided with a prediction probability to the value that they represent, so that the CLAc would have at least something to output as its prediction. However, the first pattern iteration in the linear pattern showed that quite a bit of new CSDRs did have predictions for values other then their own representations in the CLAc,

even though it should not have been possible.

The second surprise was the fact that there were so many possible predictions in the likelihood distribution of the CLAc's CSDR representation of 0.92, despite none of those values ever occurring after 0.92 in the pattern. I suspected that the reason for this was because some of the 'cells' in the CLAc that represent the CSDR 0.92 (at time step 591) were shared by other CSDRs. This meant that some 'cells' in the CLAc would be part of multiple different CSDRs, and thus learn different sequences of values that would be added to their likelihood distribution. This would ultimately result in the CLAc calculating wrong predictions for certain CSDRs. However, given that the CLAc binds present values to past CSDRs, it would mean that it was not the CSDRs 0.34, 0.22, (ignoring 0.92) and 0.94 that shared cells with the CSDR 0.92, but the CSDR 0.33, 0.21 and 0.93. This is because in the CLAc, the CSDRs representing 0.33, 0.21 and 0.93 would learn to predict the respective values 0.34, 0.22 and 0.94.

To determine what really happened, I first looked at the columns representing the SDRs for the values 0.92, 0.33, 0.21, and 0.93. The columns representing the SDR 0.92 did indeed share columns with the other three SDRs. Since it could have been a coincidence or that only the columns were shared, but no the cells within, I proceeded to dig deeper. I examined the active cells of the CSDRs that represented the four values, and successfully found a subset of cells shared amongst 0.92's CSDR and the other values' CSDRs. So in simple terms, some of the 'cells' in the columns that represented the value 0.92 were also shared in the representations of the values 0.33, 0.21 and 0.93. So when the CLAc had calculated the total likelihood distribution of 0.92's CSDR, the likelihood distribution of the shared 'cells' contributed to the total with probabilities that should not have been there.

Ultimately this meant that since 0.92's CSDR learned the next sequence to be 0.93 but still had 0.22 as a possible prediction, so did 0.21's CSDR have 0.93 as a possible prediction. The phenomenon of CSDRs sharing predictions with other CSDRs in the CLAc can be most clearly seen when the CLAc must output predictions from never before seen CSDRs that have no likelihood distributions yet. In the linear pattern for example, this happens in the entire first iteration of the pattern, where some CSDRs predict themselves while others give predictions of values that have in fact been learned by previous CSDRs. It should be noted however that the TM is dynamic and may change the specific cells it chooses from the columns that different SDRs may share, which explains why this issue is not continual throughout the entire pattern.

Once I had analyzed the issue, I wished to resolve it. However, the fundamental reason that SDRs shared columns to begin with, was because of the SP algorithm. Its entire purpose was to create SDRs and represent similar values with similar representations. While the values 0.92 and 0.21 were admittedly very different, SP would still represent values similar to each other using shared columns. In addition, since SP uses elements of randomness when creating

representations, there will never be any guarantee that no columns between differing values would ever be shared. Despite this, I performed many attempts at separating SDRs to have no shared columns using a variety of different parameter values, but to no avail. While this issue may not seem very problematic, I believe that it is a fundamental problem that hampers NuPIC's potential, especially in the cases when the HTM algorithms predict one thing, while the CLAc predicts another. For although it was easy to detect this issue for linear patterns it is impossible to do so for highly non-linear patterns.

## 6.2 Discussing random patterns and generalization

Earlier in my thesis, I hypothesised that two major pattern properties would play pivotal roles, the length of a pattern and the concentration of values within. The idea was that longer patterns and higher concentrations of values would force cortical columns to learn more sequences and different contexts, ultimately resulting in a need for more iterations for a pattern to be learned. My analogy would have been to ask a person to try to remember a short or long pattern, with each value either occurring often or rarely. The graph in Figure 5.2 showed that my hypothesis was not correct. Longer patterns took about just as many iterations than their shorter counterparts, no matter the concentration of values. As I learned more about the theory of HTM I understood that the process of learning was not centralized to one unit, but spread amongst countless SDRs. This meant that instead of just one set of cortical cells learning all sequences, different subsets of them learned different transitions. For example, while one CSDR representing 0.31 learned to predict 0.45, an other CSDR representing 0.31 learned to predict 0.89. The sparsely distributed representations insured that learning was distributed amongst an entire cortical layer. This explained why NuPIC did not need more pattern iterations to learn patterns with concentrated values as opposed to patterns with lower concentrations of values. The same answer could be applied as to why pattern length did not affect learning. One thing that I have not been able to find an answer to however, was why NuPIC needed to be repeatedly given the same pattern about 20 times for it to output predictions with a 90% accuracy. My assumption is that it takes an average of 20 repetitions for a pattern to be confidently learned by the HTM. A final note about the matter, was why there were some irregularities with some patterns suddenly taking more or less iterations to learn. Given that the HTM algorithms do have some elements of randomness at the time of initialization, the CLAc's imperfections and its statistical nature, there could never be complete consistency in the process of learning.

I have already talked about the subject of generalization, so I will only add a few more thoughts. Generalizing is a very complex task, which HTM unfortunately cannot do as of yet. The consequence is that it limits which data sets NuPIC can effectively work with. While NuPIC will learn everything eventually once it has seen enough values and patterns, it is largely useless

when seeing new values or when seeing previously seen patterns with different values. This shortcoming was what drove me to attempt to give NuPIC a very basic ad-hoc generalization capability to its anomaly detection feature.

## 6.3 Reviewing the oil runs

In the oil data experiment, a clear example of the consequences of NuPIC's lack of generalization could be seen in Figure 5.3. An overabundance of high anomaly scores seen in Figure 5.5 caused the second period of the data set to be practically impossible to examine. In addition, I was disappointed by the anomaly likelihood's (Figure 5.6) clear inferiority to the more plain anomaly score, given what NuPIC's documentation suggested. When I first looked at the anomaly graph, what caught my attention the most was how relatively few anomaly spikes with scores $\geq 0.75$ there were between 1987 and 2003 (Figure 5.4). I knew that oil was a highly valuable commodity that had deep roots in both the economy and politics of numerous countries. So I hypothesised that while day-to-day fluctuations in oil prices were normal, abnormally large fluctuations or irregular sequences of oil prices might indicate that more serious matters were to blame. My research revealed that a fair number of the higher anomaly scores did in fact either coincide with major economical/political events or happen the day after. The reason for including aftermath events was because the impact of an event would be dependant upon its severity and consequences. In addition, while some nations have daytime others have nighttime which means that some countries will naturally react to the news of major events a day later than others, thus bolstering the argument for using aftermath events. In total, 70.8% (13 + 4 / 24) and 71.4% of anomalies with respective scores of $\geq 0.75$ and $\geq 0.9$ seemed to be caused by major events.

Concerning the so called 'foreshadowing' anomaly which happened the day before the enactment of the Oil Pollution Act. The reason that I included a foreshadowing anomaly is due to the HTM's ability to detect anomalies caused by events that may over time become major events. Looking over the values leading up to the date of the foreshadowed anomaly with the naked eye revealed nothing of interest. Dates prior to the foreshadowed anomaly exhibited behaviors of both higher and lower price changes between subsequent days. Likewise, on the day of the foreshadowed anomaly, the actual price of 0.14 (normalized) had already occurred in the past, meaning that it was not new to the HTM. I found no evidence suggesting that the enactment of the Oil Pollution Act influenced the oil price in any way whatsoever. This indicates that the occurrence of the anomaly was likely due to pure coincidence instead of some correlation to the policy enactment.

The composite anomaly method seemed to accomplish its intended job. In the first period, 50.0% of its anomalies with scores $\geq 0.75$ were linked to events. Likewise, 80.0% were linked

to events if only looking at scores $\geq 0.9$. The foreshadowing anomaly of the composite method was quite intriguing. I found multiple past reports of surges in the oil price as a direct result of public announcements reporting decreases in the oil inventory of countries. Although NuPIC did not report any anomalies on the actual or following day of the event, it may be that speculation amid the announcement lead to irregular price fluctuations. Although I once again could not find any evidence supporting my theory, I would have to leave it as plausible correlation. Finally in the second period, the composite method was indeed able to filter out a lot of anomalies with high scores, though not enough to warrant examining anomalies with composite anomaly scores $\geq 0.75$. Amongst those with scores $\geq 0.9$ however 54.5% were linked to events.

Considering all the results, NuPIC managed to find anomalies in the data that were ostensibly linked to real world events slightly over 70% of the time if considering only the stable first period. My method did not seem particularly successful for the first period for scores $\geq 0.75$, but did quite well for scores $\geq 0.9$. Used in the second period, it only managed to find anomalies linked to events 54% of the time. However, this examination so far was based on checking if NuPIC's anomalies coincided with major events, but not the other way around. When doing my research, I found many more recorded events that were related to the oil market than there were anomalies with scores $\geq 0.75$ in the first period for example. This implies that NuPIC's detected anomalies always had a certain chance of happening on a certain day through statistical luck. A counter argument could be that do determine this, every single anomaly with a score $\geq 0.5$ should be checked. While this would resolve the problem, it would require far too much work, especially considering the second period. One way to increase the success rate of such anomalies would have been to add an additional source of information so as to provide NuPIC with multiple simultaneous inputs as was done with the app HTM for Stocks (described in Section 4.3.2). While the app monitors the stock and twitter feed of the chosen company as an additional information source, doing so would be more complex with the oil price since no single company affects it. In other words, a multitude of companies and organizations would have to be monitored in order to receive adequate extra information.

## 6.4 The weather results

NuPIC's weather forecasts were in general disappointing. The algorithm did not manage to beat the much more simplistic averaging technique in most cases. However, NuPIC did outperform the results of the former technique when forecasting the pressure and humidity, but not for all four days. So the questions that I asked myself were why did it perform generally worse the more times it was given the training set, and why did it perform worse than the averaging technique, yet better for some weather factors. I suspected that all these questions were involved with each other.

The answer to the first question, revealed itself in Table 5.3 and was tied to the way NuPIC worked on a conceptual level. NuPIC is a sophisticated pattern remembering algorithm. It remembers patterns and tries to predict them when seeing segments of them. The more times it sees a pattern, the more confident it becomes it its predictions. The results showed that the more times NuPIC iterated over the training set, the worse it performed on the test set. So if NuPIC remembered the patterns in the training set increasingly better, yet performed increasingly worse, than it must mean that the patterns in the test set were not similar enough to the ones that it learned. This fact can be clearly seen given that the forecasts for later days were consistently worse than for the immediate ones, since the actual values deviated increasingly more from what NuPIC previously learned.

The next question was why NuPIC performed generally worse than the averaging technique. The answer is revealed by asking why the averaging technique performed better than NuPIC. The averaging technique although very simple, is quite good at capturing current trends in the data. While there is no doubt that the weather is a chaotic system, and can be quite abrupt, it changes for the most part progressively over days. So although for example the temperature throughout a day may vary quite a bit, the average between days varies considerably less. In addition, the weather changes according to trends depending on current and upcoming events such as high or low pressure systems and incoming warm or cold fronts. This gives the averaging technique an edge by always giving an answer that is somewhere in the middle between the preceding and current conditions. Although the patterns that NuPIC previously learned included their trends, the results indicate that trends are much more specific and reliant on the actual conditions occurring at the time.

And finally, the last and hardest question was why the pressure factor was so successfully forecast by NuPIC. While I had several theories, I did not manage to prove any one of them. I believed that the most likely reason would be tied to NuPIC's strength, meaning that the patterns in the pressure data were the most consistent throughout the years in the training set, and thus made it possible for NuPIC to correctly predict them in the test set. However, I did not find an empirical way to measure the similarity between patterns located in different data sets. In conclusion, despite NuPIC's unprecedented ability to learn patterns and make predictions based on them, it unfortunately did not outperform the much simpler averaging technique. While it was superior in one particular case, there was no way to determine why. Given that the experiment was performed on data gathered from only one location, it would be wise to do multiple more from several differing locations in order to conclusively determine NuPIC's capability or inability to forecast the weather.

Anticlimactically, the detected anomaly results from NuPIC were also very negative. Despite its quite successful results with the oil data, it was unable to find any worthy anomalies despite

there being reason to expect eight clear ones. The only explanation that I conceive of is that my definition for what would constitute an anomaly was different from NuPIC's. In conclusion, nothing of interest was found, which may further reinforce the argument that weather data is unsuited for NuPIC.

# Chapter 7

# Conclusion

The goal of this thesis was to evaluate how well the official practical implementation of HTM called NuPIC could forecast diverse weather data sets. In addition, it would evaluate the HTM's ability to detect relevant anomalies in the weather data. The results from chapter 5 have shown that NuPIC is indeed able to forecast the weather. Unfortunately, despite its sophistication, its prediction results were for the most part outperformed by a much simpler analogue method that used the average of the last few days as its forecast basis. The analysis concluded that due to the weather's sensitivity to initial conditions, weather patterns rarely repeat themselves exactly as before. The weather follows trends since it is affected by current and upcoming events such as pressure systems and fronts. This means that although NuPIC was able to learn the weather's past patterns, its predictions could not account for current trends which were different from the past, despite their similar prior conditions. The reason NuPIC was unable to factor current trends in its predictions is because of its fundamental inability to generalize. The averaging technique however, was predicting values that were always in-between the immediate preceding and current conditions, which meant it was naturally factoring in weather trends into its predictions. It should be noted that NuPIC's ability to be input multiple simultaneous metrics was malfunctioning at the time of the thesis work. Furthermore, due to the CLA classifier's current internal issues, it sometimes leads NuPIC to make predictions that are different and sometimes worse than what the core HTM algorithms predict. This implies that if all these issues were to be fixed, NuPIC's forecast performance could have potentially surpassed the averaging technique. All these points indicate that the current version of NuPIC is unsuited for weather forecasting, although not conclusively.

NuPIC was sadly completely unable to detect any noteworthy anomalies in the weather data, which again is most likely connected to the weather data's chaotic nature. The anomalies with high scores did not coincide with any recorded extreme weather events, while the amount of anomalies with lower scores was far too great to be investigated. However, the oil data results

revealed that NuPIC can detect anomalies related to major economic and/or geopolitical events. Going solely by oil prices, it managed to detect anomalies linked to major events 70% of the time when looking at scores equal to or above 0.75. The lack of generalization once again proved to be a major hindrance when oil prices started to rise above the usual values in 2004, leading to too many anomalies. My ad hoc composite anomaly score method helped to somewhat alleviate this problem, decreasing the amount of anomalies across the board, although it was not as successful at finding anomalies linked to major events as the original method. While the particular case of detecting anomalies linked to major real world events is admittedly not very useful since anyone following the news would be aware of such events, it is a proof of concept. Used on the right data, NuPIC could detect major world events with decent accuracy in real time.

In conclusion, NuPIC is a proficient algorithm that can learn patterns in complex real world data to both make predictions, and detect anomalies in the data. However, the quality of NuPIC's results are highly dependant on whether the properties of the data set conforms to its strengths or weaknesses. If Numenta is ever able to implement functional hierarchies that enables generalization, it could potentially become one of the most powerful learning algorithms made by mankind.

# References


[1] A. W. Kosner. *Why Is Machine Learning (CS 229) The Most Popular Course At Stanford?* URL: http://www.forbes.com/sites/anthonykosner/2013/12/29/why-is-machine-learning-cs-229-the-most-popular-course-at-stanford/#2da7a45b61ba.

[2] R. Waters. *Investor rush to artificial intelligence is real deal*. URL: https://www.ft.com/content/019b3702-92a2-11e4-a1fd-00144feabdc0#axzz4AEw4Dgv5.

[3] *About Jeff Hawkins*. URL: https://en.wikipedia.org/wiki/Jeff_Hawkins.

[4] T. M. Mitchell. *Machine Learning*. McGraw-Hill Education, 1997.

[5] R. Swenson. *The Cerebral Cortex*. URL: https://www.dartmouth.edu/~rswenson/NeuroSci/chapter_11.html.

[6] P. Rakic. "Evolution of the neocortex: a perspective from developmental biology". In: *Nature Reviews Neuroscience* 10.10 (2009), pp. 724–735.

[7] F. H. Martini et al. *Anatomy and Physiology*. Pearson Education, 2007.

[8] J. Hawkins et al. *Biological and Machine Intelligence (BAMI)*. Initial online release 0.4. 2016. URL: http://numenta.com/assets/pdf/biological-and-machine-intelligence/0.4/BaMI-HTM-Overview.pdf.

[9] J. Horton and D. L. Adams. "The Cortical Column: A Structure without a Function". In: *Philosophical Transactions of the Royal Society B* 360.1456 (2005), pp. 837–862.

[10] J. Hawkins and S. Ahmad. "Why Neurons Have Thousands of Synapses, a Theory of Sequence Memory in Neocortex". In: *Frontiers Media* 10.23 (2016).

[11] J. Hawkins and S. Blakeslee. *On Intelligence*. St. Martin's Press, 2005.

[12] *Numenta history*. URL: http://numenta.com/mission-and-history/.

[13] *Numenta history*. URL: http://numenta.com/applications/.

[14] *Principles of Hierarchical Temporal Memory (HTM): Foundations of Machine Intelligence*. Stated in their Youtube video by Jeff Hawkins. URL: https://youtu.be/6ufPpZDmPKA?t=11m29s.

[15] S. Ahmad and J. Hawkins. "Properties of Sparse Distributed Representations and their Application to Hierarchical Temporal Memory". In: *Cornell University Library* (2015). URL: https://arxiv.org/abs/1503.07469.

[16] S. Purdy. "Encoding Data for HTM Systems". In: *Cornell University Library* (2016). URL: https://arxiv.org/abs/1602.05925.

[17] S. Ahmad and S. Purdy. "Real-Time Anomaly Detection for Streaming Analytics". In: *Cornell University Library* (2016). URL: https://arxiv.org/abs/1607.02480.

[18] S. Ahmad. *Anomaly Detection and Anomaly Scores*. URL: https://github.com/numenta/nupic/wiki/Anomaly-Detection-and-Anomaly-Scores.

[19] S. Ahmad. *Anomaly Detection in CLA*. URL: https://youtu.be/nVCKjZWYavM?t=8m55s.

[20] S. Ahmad. *Anomaly Detection in CLA*. URL: https://youtu.be/nVCKjZWYavM?t=38m20s.

[21] *UK's national Meteorological Office*. URL: http://www.metoffice.gov.uk/about-us/who/accuracy/forecasts.

[22] *Forecasting the Weather by the Australian Government's Bureau of Meteorology*. URL: http://www.bom.gov.au/info/ftweather/contents.shtml.

[23] *Global inhibition boosts performance by 60 times*. URL: https://github.com/numenta/nupic/blob/master/src/nupic/research/spatial_pooler.py.

[24] *The CLA CLassifier*. URL: https://github.com/numenta/nupic/wiki/CLA-Classifier.

[25] *Network API*. URL: https://github.com/numenta/nupic/wiki/Network-API.

[26] *Online Prediction Framework*. URL: https://github.com/numenta/nupic/wiki/Online-Prediction-Framework.

[27] *Using NuPIC with multiple metrics*. URL: https://github.com/subutai/nupic.subutai/tree/master/swarm_examples.

[28] A. Kaur, J. K. Sharma, and S. Agrawal. "Artificial neural networks in forecasting maximum and minimum relative humidity". In: *International Journal of Computer Science and Network Security* 11.5 (2011), pp. 197–199.

[29] T. Santhanam and A.C. Subhajini. "An Efficient Weather Forecasting System using Radial Basis Function Neural Network". In: *Journal of Computer Science* 7.7 (2011), pp. 962–966.

[30] I. Maqsood, M. R. Khan, and A. Abraham. "An ensemble of neural networks for weather forecasting". In: *Neural Computing and Applications* 13.2 (2004), pp. 112–122.

[31] D. Fabbian and R. D. Dear. "Application of Artificial Neural Network Forecasts to Predict Fog at Canberra International Airport". In: *Weather and Forecasting* 22.2 (2006), pp. 372–381.

[32] C. Marzban and G. J. Stumpf. “A Neural Network for Tornado Prediction Based on Doppler Radar-Derived Attributes”. In: *Journal of Applied Meteorology* 35.5 (1996), pp. 617–626.

[33] Yuval and W. W. Hsieh. “An Adaptive Nonlinear MOS Scheme for Precipitation Forecasts Using Neural Networks”. In: *Weather and Forecasting* 18.2 (2003), pp. 303–310.

[34] *What drives crude oil prices?* URL: `https://www.eia.gov/finance/markets/spot_prices.cfm`.

[35] *Oil market timelines*. URL: `https://en.wikipedia.org/wiki/Category:Oil_market_timelines`.