hochschule mannheim

Intelligent Predictions: an empirical study of the Cortical Learning Algorithm

Michael Galetzka

Master Thesis
for the acquisition of the academic degree Master of Science (M.Sc.)
Course of Studies: Computer Science

Department of Computer Science
University of Applied Sciences Mannheim

31.10.2014

Tutors
Prof. Lutz Strüngmann, Hochschule Mannheim
Christian Weber, SAP AG

Abstract

Intelligent Predictions: an empirical study of the Cortical Learning Algorithm
The theory of Hierarchical Temporal Memory (HTM) created a new approach to machine learning for time-series prediction and anomaly detection. A subset of the theoretical framework was implemented in the open source framework *nupic* by Numenta. With the help of this framework, an empirical study was conducted to assess the capabilities and limitations of HTM. The results indicate that the performance is comparable to state-of-the-art machine learning algorithms. Furthermore, the HTM models were able to form accurate predictions from very noisy and irregular data sets. Another finding was that HTM models are not very prone to overfitting.

Intelligent Predictions: an empirical study of the Cortical Learning Algorithm
Die Hierarchical Temporal Memory (HTM) Theorie hat neue Ansätze für maschinelles Lernen bei der Vorhersage zeitlicher Verläufe und Anomaliedetektion ermöglicht. Ein Teil des theoretischen Frameworks wurde im Open-Source Framework *nupic* von Numenta implementiert. Mit Hilfe dieses Frameworks wurde eine empirische Studie durchgeführt, mit dem Zweck die Stärken und Schwächen von HTM zu finden. Die Ergebnisse weisen darauf hin, dass die Leistungsfähigkeit vergleichbar ist mit den aktuell besten Algorithmen für maschinelles Lernen. Außerdem waren die HTM Modelle in der Lage, genaue Vorhersagen aus verrauschten und irregulären Daten zu erstellen. Eine weitere Erkenntnis ist, dass HTM Modelle nicht sehr anfällig sind für das Overfitting-Problem.

Contents

Chapter 1

Introduction

The mammalian brain's ability to express something we call intelligence is not only one of the greatest evolutionary successes, but also still one of the greatest mysteries. The human neocortex in particular is highly interesting, because humans are able to deal with very complex patterns and layers of abstractions – a feat that no other organism exhibits. Naturally, we want to understand what the secret behind the neocortex is.[1]

Of course, as technology advanced, the capabilities to measure, visualize and analyze different parts of the neocortex increased. In addition, the steady, exponential rise of computing power opened up the possibility of simulating ever larger parts of the brain. However, the general consensus is that all these advances have led to little more than big piles of data – there are no unifying theories or general artificial intelligences.

However, there is a new concept called Hierarchical Temporal Memory (HTM), which is an algorithm based on the biological mechanisms of the neocortex. The aim of HTM is to achieve similar pattern matching and prediction capabilities as the neocortex. This thesis intends to empirically evaluate a framework based on HTM by applying the framework to the data of several different use-cases and comparing it to the performance of known approaches, whereby valuable insight about the capabilities of HTM should be gained. The topic of HTM has so far not been widely recognized by researchers and there have been no performance studies on actual data sets [44, p.17]. The few experiments that have been conducted show varying results, such as accuracies ranging from 90% to 55% for classification use-cases [41, p.372, 9, p.260].

The term "intelligence" itself is rather difficult to define and often cause for debates, especially in computer science [40]. From the many different existing definitions of intelligence the following one was chosen according to [33]: intelligence is defined as the capability to solve hard problems.

The lack of unifying theories regarding intelligence does not translate to a lack of applications using artificial intelligence algorithms; for example, neural networks help doctors detect cancer [11], Bayesian networks filter out spam mail [2] and it is even possible to

[1]However, some scientists think this will ultimately lead to the death of billions of people [29].

autonomously steer a vehicle in traffic, as demonstrated by Google's driverless car [15]. However, as impressive as these examples may seem, for many significant real-world problems the algorithms underlying these systems are learning too slowly and require too much training [50, p.326].

Another problem is that each of these systems would fail horribly if used for other, even trivial, tasks [46, p.6]. If, for example, a spam filter is tasked to describe the taste of a banana or to differentiate between pictures of cats and dogs, it is highly unlikely that it will succeed. One might argue that a spam filter lacks a semantic model of the world, which is why things like taste or cats have no meaning for it. However, the problem is more profound, namely that these algorithms are not intended to work like the human mind – they were each created and optimized to solve a single, individual problem [42, p.268].

Imagine for a moment being one of the first explorers to come in contact with Native Americans and being exposed to an object nobody outside of the native tribe has ever seen before: a dreamcatcher[2]. There are no two dreamcatchers alike, because they are all individually made and vary in size, technique and decoration. Despite this uniqueness, one can identify other dreamcatchers easily after only seeing one or two examples; no previously defined semantic model of dreamcatchers or large amount of training data is necessary. This is a strong hint that the pattern recognition and generalization algorithm used by the human brain is working differently than, for example, a spam filter.

Jeff Hawkins describes similar pattern recognition problems in his book *On Intelligence* [20]. His algorithmic solution is a pattern recognizer based on a hierarchical memory conceptually close to the brain's biology. The novelty of his concept is that it is not a static pattern recognizer, but also accounts for spatial and temporal variability [44, p.7]; therefore the name *Hierarchical Temporal Memory*. This idea was refined and extended to the CLA and later implemented in the open source project *NuPIC*[3].

1.1 Memory Prediction Framework

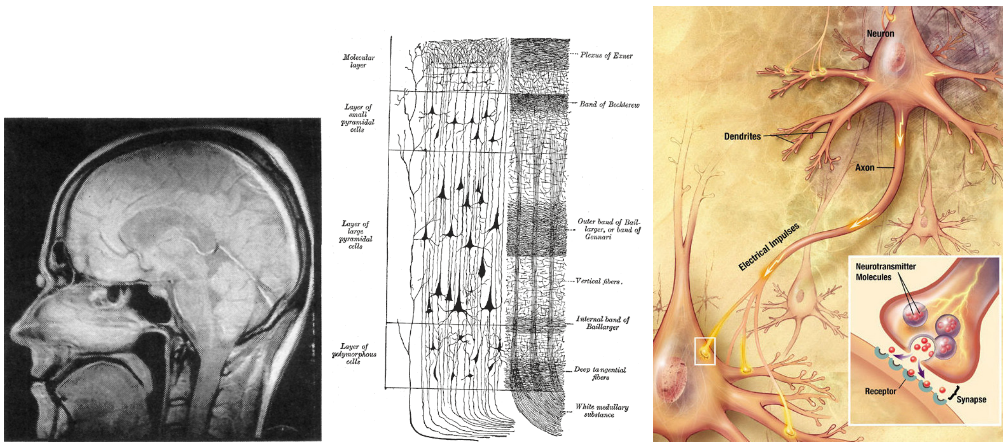
The following is a short introduction to the idea behind HTM to help understand the research setting and the decisions made during the study. A more detailed explanation of the HTM technology can be found in chapter 3.2.

In *On Intelligence* [20], Hawkins develops the idea to create a simple model of the neocortex not by simulating every part of it, but by reducing it to its core functions.[4] This approach has been tried before, for example with Neural Networks that use artificial neurons and synapses as building blocks [22]. The problem with every algorithm trying to model a

[2]A dreamcatcher is a handmade circular object with a woven net in the middle and decorated with items like beads or feathers.

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

[4]Example projects attempting to simulate every aspect of the brain down to the molecular level are the *China Brain Project* [10] or the *Blue Brain Project*. See http://bluebrain.epfl.ch/ for more information.


Molecular layer
Layer of small pyramidal cells
Layer of large pyramidal cells
Layer of polymorphous cells
Plexus of Exner
Band of Bechterew
Outer band of Baillarger, or band of Gennari
Vertical fibers.
Internal band of Baillarger
Deep tangential fibers
White medullary substance
Neuron
Dendrites
Axon
Electrical Impulses
Neurotransmitter Molecules
Receptor
Synapse

(a) MRI head scan [18, p.87]. **(b)** Cerebral cortex [16]. **(c)** Synaptic transmission [52].

Figure 1.1: Example visualizations of different parts of the human brain.

part of the brain is that there are too many anatomical and physiological details to determine what exactly is required from a computational point of view, and what is merely necessary to allow proper functioning in an actual living being [14, p.4]. Because of this difficulty, the derived models often lack some important biological features like feedback connections [1] or inhibitory neurons. Even advanced concepts like Recurrent Neural Networks (RNN) are often used only for static problems, such as classification, because they have trouble with values varying over time [6, p.112]. Figure 1.1 shows visualizations of the brain with varying magnifications, each with many different details and structures.

In Hawkins' opinion, the neocortex is responsible only for memorizing and making predictions based on those memories. Consequently, he named his model *Memory Prediction Framework*. It basically works as follows:

1. The model represents the part of the world it is exposed to by memorizing patterns. These patterns can occur both temporally (as a chain of events often occurring in the same order) and spatially (as a group of events often occurring together).

2. The model uses its stored temporal and spatial patterns to predict the future state of the world it is observing. Through the use of a hierarchy, patterns stored in a lower level can be reused by hierarchically higher levels.

3. Information flows up and down the hierarchy to disambiguate between different possible patterns.

4. The model learns in an unsupervised way by incorporating the actual observed event into the stored patterns.

An example of this is how the human brain understands spoken sentences. In the physical sense, a spoken sentence is a series of rapid air movements that are translated to a stream of electrical signals by the ear. First of all, these signals are routed to the lowest levels of the brain’s hierarchy for processing, which search for simple patterns, such as vowels and consonants. The output of the lower levels is the input for the superordinate levels, which try to find and predict more complex patterns, such as words and punctuation. Figure 1.2 shows an example of this process.

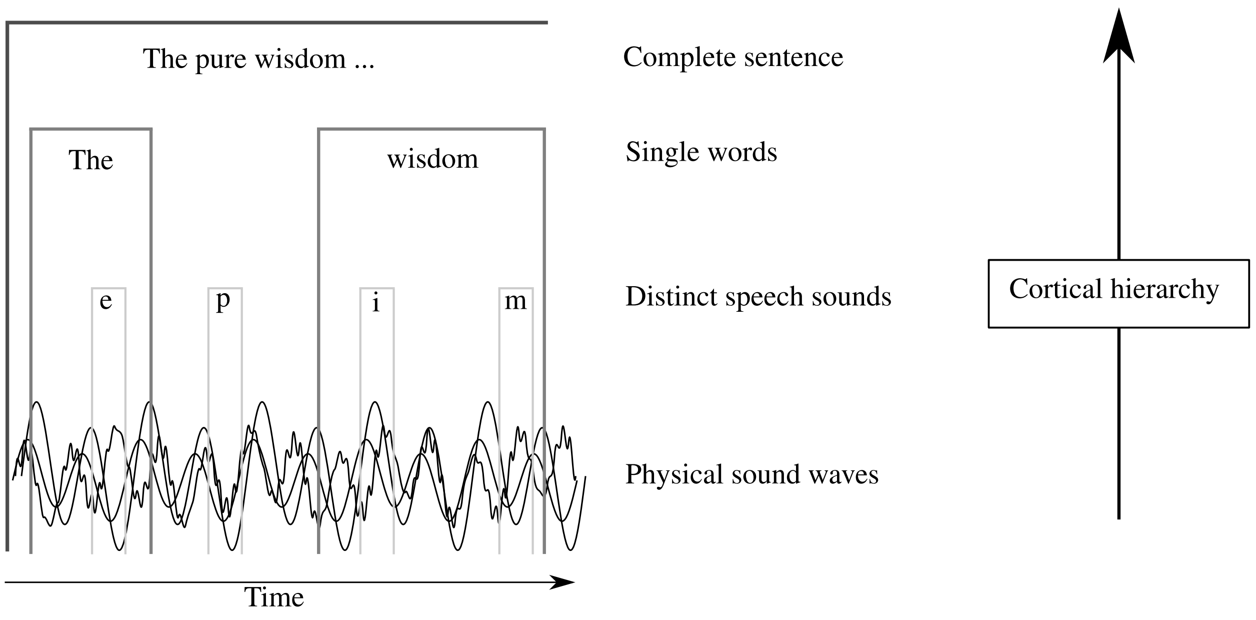


Figure 1.2: Speech recognition in the cortical hierarchy.

Interestingly, as indicated by step three of the description, the higher levels also send information back to the lower levels, influencing their pattern recognition to match the predictions of the higher levels. For example, if a certain word is predicted by a higher level, the lower levels are more likely to recognize the sounds present in that word. This mechanism not only explains how the brain is able to disambiguate very similar words, but also how auditory illusions like Sheppard-Risset tones[5] are possible.

This shows that the algorithm assumes that spatial and temporal patterns in the sensory data exist and tries to exploit these for the actual pattern recognition. Therefore, HTM yields the best results when used for "problems where the data to be modeled is generated by a hierarchy of causes unfolding in time" [38, p.2]. Many real-world scenarios fit this description, such as weather predictions, car monitoring systems or music classifications[6] [38, p.1-6].

The use-cases and data gathered for this study have to fit this description of a hierarchical model as well. Therefore, biometric key event analysis, network event detection and user browse behavior have been chosen as study use-cases. A detailed description of each use-case can be found in chapter 4.1.

[5]"Sheppard-Risset tones are sounds that seem to go up (or down) indefinitely" [4].
[6]There are even special HTM adaptions for musical applications [32].

Chapter 2

Research Setting

This chapter describes the research setting of the conducted empirical study. The scientific problem of artificial intelligence with emphasis on pattern recognition and anomaly detection is a very broad one. Therefore, to narrow the scope, the focus of this work lies on the HTM technology as implemented in the NuPIC framework by Numenta. The overall aim of this research is to explore the capabilities and performance of HTM and the Cortical Learning Algorithm as they are applied to three different use-cases.

2.1 Research Questions

The following questions should be answered by the data collected during the study.

1. *How good are the predictions made by an HTM model?*

   Recognizing a known pattern and predicting future values based on that pattern is one of the major challenges of artificial intelligence. It is a necessary skill for any intelligent agent operating in a changing environment, be it a robot trying to catch a ball or an automatic trader on a stock exchange. The better the prediction rate of a system is, the better it is able to plan ahead and choose its actions.

   Of course, the accuracy of the predictions depends on many variables, e.g. the amount and quality of training data. Furthermore, the problem domain must be considered, as some algorithms depend on the underlying structure inherent in domain-specific data. In addition, with HTM the structure of the used hierarchy as well as the configuration of the model parameters has to be considered. Being able to give an estimate for the prediction quality of an algorithm is very valuable when deciding which algorithm to use to solve a problem.

   Currently, there is little public data describing the prediction quality of HTM, so even a partial answer to this question will be helpful to assess the capabilities of this technology.

2. *How good are the anomaly detection results from an HTM model?*

   Due to the way that HTM works, it does not only make various predictions about future inputs, but also states how reliable each of these predictions is. This makes it possible to recognize novel or unusual inputs by identifying those that were not predicted. Not only is this an essential part of learning and training, but it can also be used for a great variety of use-cases, e.g. fraud detection or data sanitization.

   The ability to detect anomalies is intimately tied to the pattern detection ability. Since the CLA is an online learning algorithm, one aspect of the question is how stable the recognition of an anomaly is, e.g. how often it has to occur to be seen as a normal occurrence.

3. *How much time does it take to find good HTM model parameters?*

   Besides the layout of HTM regions in a hierarchy, there a many different parameters to adjust the CLA and its corresponding model. For example, the learning rate and synaptic thresholds can be adjusted as well as the bit density of the SDR encoders. Optimizing these parameters is a difficult task in itself, as there are too many possibilities to systematically try all of them. *Numenta*'s approach to solve this problem is to use a heuristic swarming algorithm, basically running several models in parallel and choosing the one with the best results. It is important to assess how much time this or similar approaches take and how they scale with the model size.

4. *How prone to overfitting is an HTM model?*

   Eventually, every statistical model in machine learning is prone to overfitting. This means that the model too closely fits the training data, unable to abstract randomness and errors away, leading to a poor prediction performance. Especially models with a great number of parameters, like the HTM models, are vulnerable to overfitting. Under what circumstances and to what extent this overfitting occurs with HTM models is important to answer.

5. *How does the algorithm's performance scale with the amount of data?*

   The human brain has a vast input capacity and is able to perform about 10^{12} to 10^{14} parallel computations per second.[1] The CLA approach mimics the working of the human brain to achieve pattern recognition and make intelligent predictions. It is therefore interesting to know if it is also able to cope with a large amounts of data. This question has two different aspects that have to be considered: the amount of data that is processed in every single time step and the amount of data that is processed overall.

[1]For more details, see [34, p.57,163].

6. *How much influence do the model parameters have on the execution time?*

   All previous research questions work on HTM models with different, optimized parameters. Naturally, a change in model parameters also impacts the execution time of the CLA. To assess the contribution to the error rate of the previous research questions and to find the most influential model parameters, this change in execution time must be measured.

2.2 Research Environment

The environment, in which the study was conducted, was as follows:

```
Host machine
   i5 CPU (M540 @ 2.53GHz, 2 Cores)
   8GB RAM
   500 GB disc (no SSD)
   Windows 7 - 64 Bit, all updates
   VMWare Workstation 10

Virtual Machine (VM) settings
   1 Processor (2 Cores)
   2 GB RAM
   25 GB disc
   Ubuntu 14.04 - 64 Bit, all updates

Software installed on VM
   Python 2.7
   NuPIC (installation as described in the official wiki)
   Git repository:
     URL: https://github.com/numenta/nupic
     Commit ID: 97ef00d00c583e3f9553a4335e61552eab4e0097
     Time: 7 May 2014 15:19
```

Chapter 3

Theoretical Principles

This chapter is meant to give a brief overview of the theoretical concepts involved in this study. This includes the HTM technology and related concepts of artificial intelligence. As this thesis focuses mainly on the empirical study, the description of these concepts will not go into details.

With the first computers, the field of artificial intelligence emerged. At first, researchers had high hopes of creating a General Artificial Intelligence (GAI). However, those hopes quickly vanished as more and more approaches failed and the problem of creating intelligent machines soon came to be deemed impossible [7, p.394]. Lately, many researchers have tried to tackle the problem anew [51]; this has led to some interesting approaches, e.g. the HTM technology as proposed by Hawkins.

Strictly speaking, HTM is a prediction framework based on pattern recognition, not a GAI. A true GAI would have to achieve more than merely correct predictions. For example, it would have to plan ahead to achieve a goal, make trade-offs such as time versus efficiency, understand language and even be able to perform tasks involving metacognition [42, p.269-273]. However, tackling the problem of creating predictions is a good starting point to understanding human intelligence.

3.1 Machine Learning

Any algorithm trying to model and predict the real world has to deal with uncertainty. For example, most of the time the input shows only the result (e.g. wet grass) and not the underlying cause (e.g. rain, sprinkler). Furthermore, the input signal may be noisy, have a low resolution or have parts of it missing. An algorithm could simply make a single assumption about the real state of the world and base its calculations thereon; however, it is far more effective to make multiple assumptions and assign a likelihood to each of them.

Therefore, many machine learning algorithms use statistical methods to model the world in a probabilistic way. Among the best known are the graphical models that represent the world as a graph of random variables and their dependencies; for example, a Bayesian network can be modeled as a directed acyclic graph. The following gives an overview of the most common approaches to Machine Learning:

Bayesian Network (BN) Also called "belief network", the BN models the causal relationships of a problem domain by building a directed acyclic graph from random variables and probability distributions. Figure 3.1 shows an example network where a visible event (in this case, a traffic jam) is connected to its causes (e.g. accident) which themselves can have other causes (such as the weather).

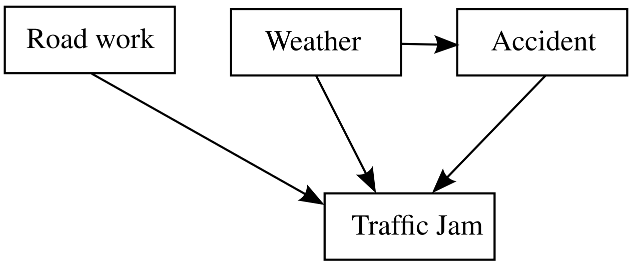


Figure 3.1: Example Bayesian network.

Because they rely heavily on causality, BNs are also called "causal networks" [43, p.241], although an edge of the network graph does not necessarily imply a causal relationship. Despite their static knowledge representation, BNs have a wide range of applications, such as computer vision, medical diagnosis, robotics and manufacturing [21, 24]. The more general Dynamic Bayesian Network (DBN) is even able to deal with time series and dynamic systems by relating variables over adjacent time steps with each other [35].

One of the drawbacks of BNs is that the engineer of the network has to specify the topology and variables of the network before being able to use it [26, p.297]. The automated construction of networks continues to be a research topic.

Hidden Markov Model (HMM) A special form of DBN [35, p.15], the Hidden Markov Model uses a Markov Chain to represent a dynamic system as a sequence of states and observations. It works under the assumption that the modeled system states are hidden and can not be measured directly, and that each subsequent state depends only on the previous one [43, p.267]. Figure 3.2 shows an example HMM with a sequence of five states S_i and their corresponding observations O_i. It shows that each state S_i depends solely on the previous state S_{i-1}.

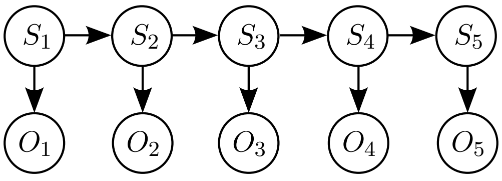


Figure 3.2: Example Hidden Markov Model. S_i = states, O_i = observations.

HMMs are especially suited for temporal pattern recognition, such as speech and gesture recognition [49] or computational molecular biology [3]. However, the models have the same drawbacks as DBNs: it is hard to automatically adjust model parameters and find a meaningful state sequence to match given observations [45, p.8]. In addition, they require large amounts of training data and are computationally expensive in terms of processing time and memory consumption [35, p.9, 48, p.2].

Artificial Neural Network (ANN) The algorithm for ANNs was inspired by the architecture and function of neurons in the brain. As a result, it uses a highly simplified model of neurons and synapses. Figure 3.3 shows an example of a simple feed-forward network with three layers.

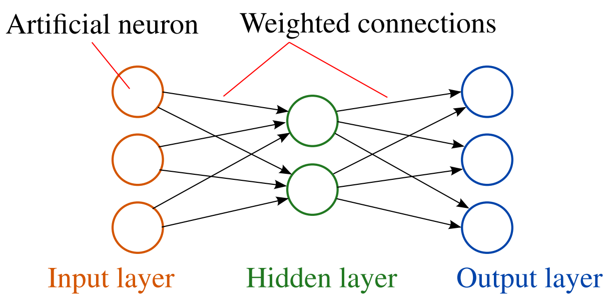


Figure 3.3: Example Artificial Neural Network.

There are many varieties of ANNs because of their great flexibility. Yet, the original biological model has been mostly discarded. It was replaced by more statistical approaches, because they yield better results. Despite their varieties, most ANNs share the following commonalities:

- The neurons are organized in layers.
- The connections between the neurons have a weight attached to them.
- The output of a neuron is a function of its weighted input.
- The weights and parameters must be adjusted with training data before being used.

ANNs are very flexible and have many applications from regression and classification problems to pattern matching. It is even possible to achieve time series prediction with a feed-forward network by using a sliding window approach [13]. The drawback of this flexibility is that, as with HMMs, the topology of the network has to be adapted for every problem domain. In addition, the weights must be adjusted with lots of training data. Self-organizing and unsupervised networks are possible, but very limited in their capabilities [31]. Another drawback of ANNs is their sensitivity towards outliers in the training data [8, p.240]. Further, they do not scale well to large implementations [44, p.2].

One solution to overcome the problems of the sliding window approach for time-series prediction is to use a Recurrent Neural Network (RNN). An example network is displayed in figure 3.4. The feedback neurons help to overcome the problem of working with time-series by feeding the output of previous calculations back as input into the remaining hidden layer. There exist many different architectures of RNNs with varying performance depending on the problem domain [23]. In general, RNNs are robust and powerful for time-related tasks. However, they are hard to train and their network structure increases the computational complexity [36, p.582-583].

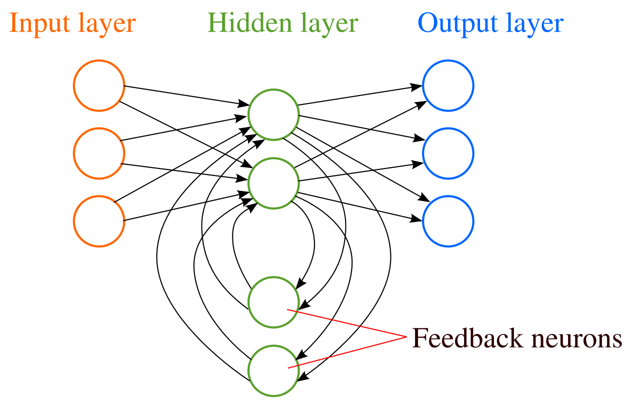


Figure 3.4: Example Recurrent Neural Network. The hidden layer forms a directed graph able to feed back information from previous calculations.

3.2 Hierarchical Temporal Memory (HTM)

While HTM was inspired by classical approaches like BN or ANN, it tries to be consistent with the underlying biological model [37, p.7]. The actual algorithm behind this concept changed a lot since its first proposal. Therefore, many of the descriptions found in research literature differ from one another. For example, HTMs are described as "Multi-stage Hubel-Wiesel Architectures" [30, p.1], "tree-shaped" Bayesian Networks [41, p.370] or "hierarchical Hidden Markov Models" [47, p.10]. Irrespective of the changes made during its development, the reason why HTM is described as a variation of known algorithms is that it was in fact designed to be a "synthesis of these ideas" [30, p.2].

The following are the main elements of HTM architecture as displayed in figure 3.5:

Layer The layers are the largest organizational unit and are comparable to the layers in the organization of ANNs. Contrary to ANNs, however, it is not necessary for HTMs to have multiple layers to work effectively with complex data. Each layer is comprised of one or more regions. This hierarchy of columns, regions and layers is also depicted in figure 3.6.

Region A region consists of a fixed set of columns. The columns of one region influence each other in such a way that the output of a region is a SDR.

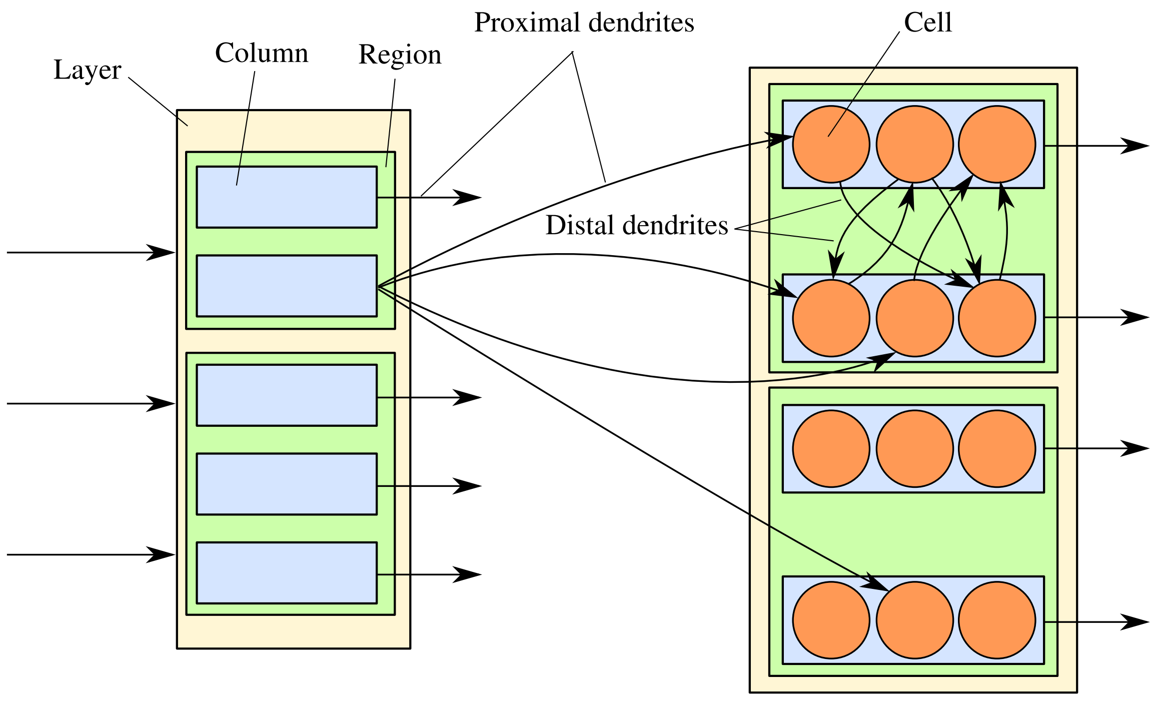


Figure 3.5: Elements of HTM architecture.

Column Each column contains several cells and combines their output to either activate itself or not. Active columns inhibit columns around them, effectively creating the SDR of a region. The columns are modeled after the columnar organized structures of neurons in the neocortex [37, p.58].

Dendrite The dendrites connect different HTM parts similar to the synaptic connections in ANNs. The difference is that the dendrite model used in HTM is more complex as it tries to be closer to the biological original. There exist two types of dendrites in HTM: The *distal dendrites* carry the output of cells to other cells within the same region while *proximal dendrites* carry the unified output of columns to individual cells in the next highest layer. Proximal dendrites are also used to connect the lowest layer of a HTM model with the actual input.

Cell The cells are the smallest elements of the hierarchy and responsible for most of the computation. They are comparable to the neurons of ANNs. A cell computes its output with a threshold function using the input from distal and proximal dendrites.

3.2.1 Sparse Distributed Representation (SDR)

One of the main concepts used in HTM is Sparse Distributed Representation (SDR), which is a way to encode data so that values with semantic similarity are also share parts of their

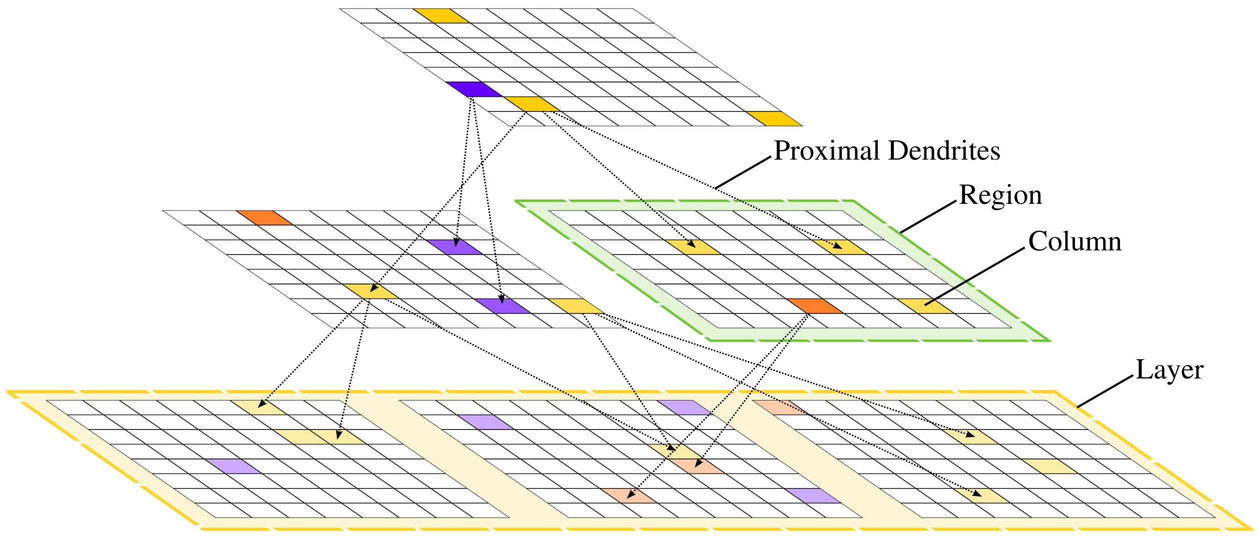


Figure 3.6: HTM hierarchy example. Note that neither cells nor distal dendrites are shown - the smallest visible units are the columns.

representation. Usually, information is encoded using a *dense* representation where the information is distributed over all available bits. For example, twelve bits are able to represent 4096 unique values in a dense representation. In this dense representation a single active bit has no semantic meaning, because the represented value is spread to all bits equally.

Figure 3.7 shows an example of using twelve bits to encode values in the range of 0 to 100 in a sparse representation. In this example, each value is encoded by activating three out of twelve possible bits. This encoding actually loses some of the original information. As can be seen, the values 1 and 7 share the same sparse representation. This kind of generalization can be avoided by increasing the number of bits available in the SDR. However, this is only feasible if the value range is small enough. The example also shows that the number of possible encodings has been drastically reduced from the original 4096, but the sparse representation has a number of advantages:

- Similar values have similar representations. Usually, values in a natural system do not change abruptly, but tend to do so gradually. SDRs are able to do the same as can be seen in figure 3.7: as the values increase only very few bits of the sparse representation change.

- The similarity of two values is easily measured by the percentage of overlapping active bits. For example, the value 7 in figure 3.7 is more similar to 15 than to 36, because it has more bits overlapping with 15 than with 36.

- It is fault tolerant to some degree since single bit flips are not able to drastically change the represented value.

- It is possible to represent multiple values at once. Since most of the bits are usually inactive, it is possible to merge multiple values with a bitwise *OR* operator into a single representation. For example, the combined representation of the values 1 and 99 in figure 3.7 would be 1110 0000 0111.

- The memory footprint of a sparse representation can be greatly reduced by storing just a small subset of the active bits. This is a technique called *sub-sampling* and works because the probability of starting with different values and ending up with the same subset of active bits is negligible [37, p.30].

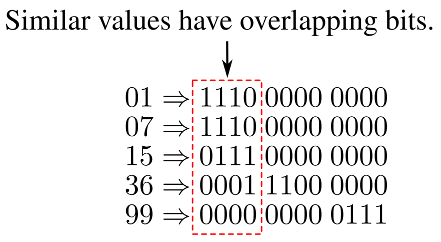


Figure 3.7: Sparse Distributed Representation (SDR) example. Encoding of values in the range 0 to 100 with 3 active bits in a total of 12 bits.

The output of each region in a HTM model can be seen as a SDR value, because the output of each column is a single bit, depending on whether that column is active or not. Accordingly, the result of a HTM model, which is the output of the model's top most region, is a SDR value. Consequently, this output shares all the advantages of SDRs; for example, a single output can contain multiple predictions at once.

The source data for HTM problems such as prediction or anomaly detection can come in a great variety of formats, such as numbers, texts, dates, pictures or audio. As figure 3.6 shows, the output of one region is also the input for other regions and therefore the input of a region must also be a valid SDR. Thus the input for a HTM's lowest layer must first be encoded as SDR, because almost no kind of source data meets this requirement. Deciding on an encoding scheme is an important step and not trivially done for a number of reasons:

- To ensure that the active bits in a SDR have semantic meaning, the semantics of the source data has to be known to some extent by the encoder.

- The encoder has to make sure that each encoded value only has very few active bits, for example ten out of a thousand. For some types of data this means that the encoding has to value some parts of the source data over others or split it into equally weighted parts. For example, a web url can be encoded by splitting it into its different parts like domain, path and query and encode each part separately.

- Data values that are semantically similar should have a similar encoded representation with overlapping active bits.

- The number of total and active bits of the SDR representation must be chosen carefully. These two factors determine many key properties of a HTM model, such as the number of possible learnable patterns, abstraction capabilities, noise tolerance, size and execution speed. As can be seen in the example in figure 3.7, they also determine the amount of information loss caused by the encoding of the source data.

The NuPIC framework already comes with a number of encoders for various kinds of data, such as scalar numeric values, discrete categories, dates or bitmaps. An example of three different encoders is shown in table 3.1. Since the full-length bit representation of the SDR would consume too much space, only the indices of active bits are shown.

Encoded value	Active Bits in SDR (indices)		
	Scalar Encoder	Category Encoder	Random Scalar Encoder
1	1, 2, 3	1, 2, 3	5, 15, 19
2	1, 2, 3	3, 4, 5	5, 15, 19
3	2, 3, 4	6, 7, 8	1, 15, 19
5	3, 4, 5	12, 13, 14	1, 10, 19
10	5, 6, 7	27, 28, 29	4, 10, 22

Table 3.1: Examples of different SDR encoders. Please note that for each encoder, instead of a full representation as in figure 3.7, only the index numbers of active bits are shown. Total number of bits: 30; number of active bits: 3.

The *Scalar Encoder* works like the example shown in figure 3.7: a consecutive number of active bits moves around as the encoded value gets bigger. The resulting SDR has the property that values share active bits depending on how close they are to each other.

The *Category Encoder* is used to encode values that have no semantic similarity between each other, e.g. database ids or category names. It works almost in the same way the scalar encoder does, but the step width between two values is big enough to ensure that no two values share any active bits.

The *Random Scalar Encoder* activates random bits in the SDR for each encoded value. However, it does this in such a way that similar values still share a number of active bits.

Of course, the results for each of the standard encoders depends greatly on the configuration of the encoder. This configuration always includes the number of total output bits and the number of active bits in the output. Also, most encoders require additional parameters, such as the value range for a Scalar Encoder or a list of categories for the Category Encoder.

3.2.2 Cortical Learning Algorithm (CLA)

The goal for each HTM region is to learn patterns from the input data while accounting for spatial and temporal variability. This is achieved with an algorithm called CLA which conceptually works as follows:

1. The region receives input from sensor stimulus or lower levels. Sensor input has to be encoded into a sparse representation.

2. The following sub-steps are often aggregated into a single unit called Spatial Pooler (SP) [37, p.34], since their purpose is to find a single representation for patterns that are spatially similar. The result is a sparse vector of active columns in the region.

   a) Each column of the region is connected with a proximal dendrite to a unique subset of the input bits. Similar to the biological model, each of these bits is connected to the proximal dendrite using a synapse. Each of these synapses has a permanence value which is used to determine if the synapse is active. The overlap score is computed for each column by counting all synapses with a high enough permanence value that are connected to active input bits.

   b) Columns with a high overlap score are activated and inhibit the activation of columns near them. If a column is not activated for some time, then its overlap score will get a boost to prevent "dead" columns in the region.

   c) Learning occurs by adjusting the synapse permanence values of all active columns. The permanence of synapses connected to active input bits are increased while the permanence of the remaining synapses are decreased.

3. As with the SP, the following sub-steps are aggregated into a unit called Temporal Pooler (TP) [37, p.39]. The purpose of the TP is to represent the input's context and form predictions.

   a) Each active column activates those cells that are in predictive state from the previous input. This way, the same input signal to a column can be represented in different ways depending on the input's context [37, p.22]. If no cell is in a predictive state then all cells are activated to represent a novel or unexpected input.

   b) The predictive state for each cell in the region is calculated by checking if it is connected to enough active cells with dendrite connections to pass a threshold. The cells that are already active from the previous step do not switch to a predictive state. If taken as a whole, the cells in the predictive state represent the regions prediction for the next input.

   c) Learning occurs by adjusting the permanence values of the synapses connected to cells in the predictive state. The permanence of synapses is increased if they are connected to active cells, otherwise it is decreased. The actual implementation is more complex with temporary updates and multiple steps [37, p.31].

4. The regions output consists of all active columns. A column is active if it contains at least one cell activated in step 3a or at least one cell in a predictive state from step 3b.

The function of a cell and its states within the TP as described in step 3 of the CLA is further describes in figure 3.8. As can be seen, the cell model used in HTM is more complex than the neurons in ANN as it tries to incorporate the biological features of different types of dendrites.

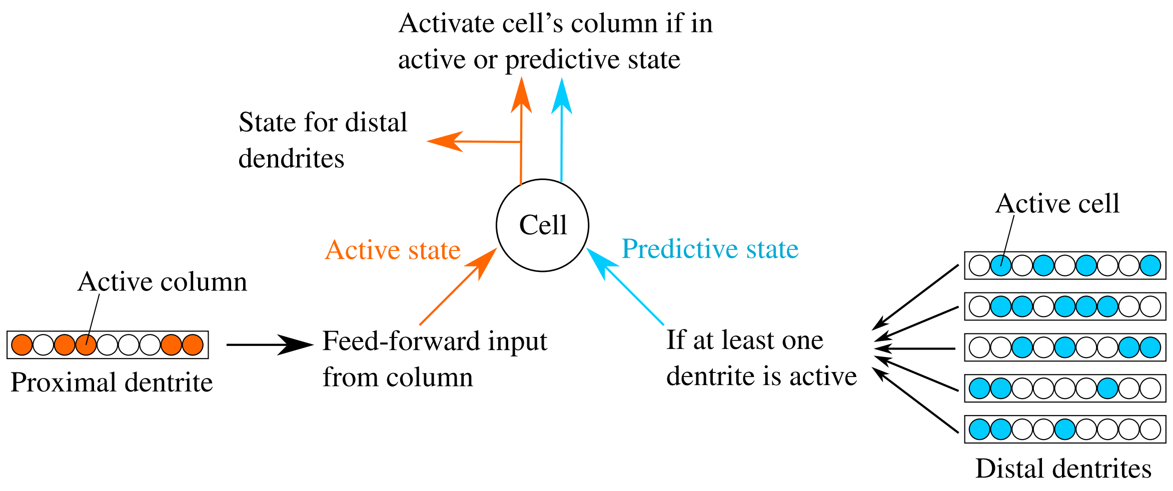

Activate cell's column if in active or predictive state
State for distal dendrites
Cell
Active state
Predictive state
Active cell
Active column
Proximal dentrite
Feed-forward input from column
If at least one dentrite is active
Distal dendrites

Figure 3.8: Depiction of cell states. A cell can be in one of the following three states: inactive, active, predictive. The cell's active state takes precedence over the predictive state, so a cell can only enter the predictive state if it was not activated by the columns feed-forward input.

Chapter 4

Study Design

This chapter is intended to describe the details of the study's use cases. In addition, the methods uses to collect the data are described.

4.1 Use cases

In the recent trend of Big Data, the number of freely available information databases is steadily growing. For example, many companies are creating public APIs for data crawling[1] and governmental organizations are increasingly sharing their data publicly[2]. In addition, the data is also becoming more and more structured, so that it can be better used for machine learning algorithms [28, p.2].

However, most of the publicly available data is best used for classification problems. To find use-cases where the data fits HTM's prediction and anomaly detection capabilities was one of the major challenges of this study, because of the following problems:

- To evaluate the results, some kind of performance benchmark must be defined. When predicting time-series values, such as stock market prices, the benchmark would be the actual recorded price. However, many interesting prediction use-cases involve a certain degree of user input. In these cases it is almost impossible to know what the optimal input would be.
- Many applications do not allow the tracking or modification of user input without actually changing the source code of the application. This has not been feasible on the scale of applications developed by SAP.
- Access to many of SAP's datasets, such as customer support databases, was not granted due to legal reasons.
- Usually, datasets do not annotate anomalies, which makes it hard to benchmark any results of anomaly detection algorithms. While adding this information manually is

[1]See for example the Twitter API: https://dev.twitter.com/docs/api/streaming
[2]One example is the U.S. government's open data website: http://www.data.gov/

often hardly possible without expert knowledge, it is downright infeasible for large datasets.

4.1.1 Biometric keystroke analysis

The use case of biometric keystroke analysis is also known as keystroke dynamics or key event biometrics. Its aim is to gather and analyze information about a user's typing behavior to create a unique biometric identity for each user. As with other biometric approaches, such as fingerprints or voice recognition, this can be used for authentication and verification purposes.

For example, voice recognition biometrics can help to verify that an audio recording has been created by a certain person or they can be used to authenticate a user by comparing a spoken passphrase with a prerecorded version. Similarly, keystroke biometrics can be used to verify that a text has been written by a certain person. This is very useful for fraud detection, for example in exams of online education courses [17].

Furthermore, keystroke biometrics can be used to authenticate a user by having the user type a sample text and comparing the keystroke timings with the user's biometric profile. It can also be used as an additional security feature where a user is authenticated not only by typing a password, but also by typing it with the correct timings. The biggest advantage keystroke analysis has over other biometric approaches is that it is the easiest to collect, because no additional hardware is required.

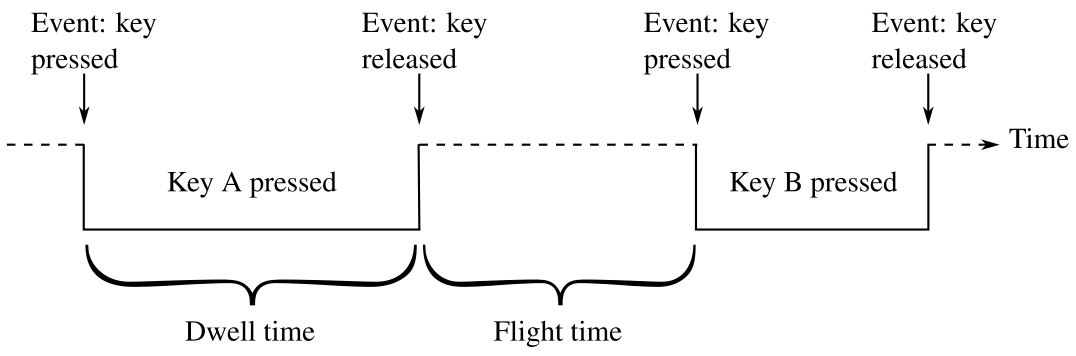


Figure 4.1: Depiction of keystroke timings and events collected for biometric analysis.

The data collected while a user is typing is displayed in figure 4.1. Every time a user presses or releases a key, a corresponding event is sent from the keyboard to the computer. Two values can be calculated from the timestamps of these events:

- Dwell time: This is the total time a key was pressed. This value is calculated by subtracting the timestamp value of a *pressed* event from the next *released* event of the same key.

- Flight time: This is the total time that elapsed without a key being pressed. It is called flight time because this is the time the hand takes to move from one key to the next. The flight time value is calculated by subtracting the timestamp value of a *released* event from the next *pressed* event of a different key.

The best possible data source for the study would be a recording of keystroke data from applications in a production environment. However, computer applications usually discard the timing data of user input and keep only the text for their calculations; it is also difficult to extend existing applications and add this functionality.

It would be possible to collect the data by installing a system-wide keylogger on a user's computer. The problem with this approach is that a record of a user's keystrokes contains potentially sensitive information like passwords. The recording of such data is a breach of security which is prohibited in any production environment.

To collect the data used in this study, a dedicated Java application was developed. The application consists of a single *Swing JTextArea* where each key event is intercepted and recorded with a nanosecond timestamp. The following is an excerpt from the recorded data:[3]

```
Timestamp         Key ID  Flight Time  Dwell Time
2429607084263029  65      197047       150864
2429607234369276  82      150106       72277
2429607379937127  32      145567       94859
2429607566445771  69      186508       102932
```

Although the timestamps provide nanosecond precision, the actual accuracy is in the range of microseconds (μs) to milliseconds (ms). This is due to the fact that keyboard events are gathered by the operating system, which then delivers them to an application. A recent study found that this update-cycle is executed 64 times per second on a typical Windows XP computer [25, p.332]. This means that the actual accuracy of each recorded key event might be close to 15 ms.

As can be seen above, the HTM model for this use case takes three values as input: the key ID, flight time and dwell time. Each value is encoded using a Scalar Encoder with varying parameters. Flight time is clipped at 2000 ms, meaning that all values higher than that are treated like the value 2000 ms by the encoder. This is a necessary trade-off between possible input range and the bits available in the encoded value. Likewise, dwell time is clipped at 500 ms. This clipping value is lower because it allows for a greater resolution of the small time scales in the encoded input.

The training and test data consist of excerpts from the German fairy tale "Snow White." The training data consist of about 5000 keystrokes, which is equivalent to about four A4-pages. This data was collected from a single person in one sitting; subsequently, the test

[3]The unit of flight time and dwell time is microseconds.

data was collected from four people, with each person performing about 1000 keystrokes, the equivalent of approximately one A4-page of text.

The test data from the person who also contributed the training data are considered a *positive* test, whereas the three other data sets are considered to be *negative* tests. It is the aim of this use case to measure the HTM model's ability to identify which input has not been created by the same person who produced the training data. Therefore the quality of the model is measured by its ability to separate the positive from the negative test cases. This ability is expressed by the model score s^{mod}, which is calculated in the following way:

$$s^{mod} = a^{neg} - a^{pos} \tag{4.1}$$

where a^{pos} is the average anomaly score over all n keystroke inputs of the positive test case:

$$a^{pos} = \frac{1}{n} \sum_{i=1}^{n^{pos}} a_i \tag{4.2}$$

and a^{neg} is the average anomaly score over all m keystroke inputs of the 3 negative test cases:

$$a^{neg} = \frac{1}{m} \sum_{k=1}^{3} \sum_{i=1}^{n_k^{neg}} a_i \tag{4.3}$$

The variable a_i in the equations 4.2 and 4.3 is the anomaly score for the i-th input as calculated by the HTM model. Note that equation 4.3 is only valid in this case because all three negative test cases are of the same size. If one of the test cases were considerably larger than the other ones, that test case's overall anomaly score would outweigh the other scores in a^{neg}.

4.1.2 Network data analysis

The aim of this next use case is to assess the anomaly detection capabilities of an HTM model by analyzing recorded network data. The number of malicious network attacks on businesses has rapidly increased over the past years. In addition, these attacks are becoming more and more sophisticated with the use of complex tools, which require little to no technical knowledge [19, p.2]. Being able to identify these attacks and distinguish them from normal network activity is a major challenge. If an attack goes unnoticed, it is impossible to assess the damage it has caused and analyze it for future prevention measures.

A common approach to identify attacks is to use a rule-based system where each attack scenario is stored in a database. The network activity is then compared to the stored scenarios in order to identify ongoing attacks. This approach requires a large amount of manual labor and expert knowledge to create and maintain the stored scenarios. Therefore, more adaptive intrusion detection systems have been created with the use of machine learning techniques

[27, p.1]. Anomaly detection is the most important part of an Intrusion Detection System (IDS), as it enables the IDS to identify even novel attacks which have not occurred before.

The data used for the analysis in this use case consists of an excerpt from a network monitoring database of SAP's internal network. Here, activities on the network are monitored and logged as events, for example 'Business Transaction' or 'Http Server'. Each event is stored with 22 attributes, such as timestamp or client ID. The database excerpt used in this study consists of roughly 8 million distinct events, which were observed over the course of two weeks. The following shows an example of the data; note that not all of the 22 attributes are shown, but only the ones used by the HTM model:

```
ID,     Timestamp,           User ID,     Event Type
5F3A13, 2014-04-01 17:26:18, JC8uCREF,    ReadAccessLog
19F3FD, 2014-04-04 11:47:03, 4+Qtz807XgH, BusinessTransactionLog
```

The data had to be pre-processed, because it would have been too much input for the model otherwise. For this purpose, the data was aggregated every 300 seconds by summing up the events based on user ID and event type. The resulting output consisted of about 1600 files, one for each unique user; each of these files contained the number of events in the corresponding 300 second intervals. The following shows an excerpt from one of these files:

```
Start of 300s interval  Sums for different event types
2014.03.23 15:07:06;    0;0;6;0;173;0;9
2014.03.23 15:12:06;    0;0;6;0;10;0;0
2014.03.23 15:17:06;    0;0;7;0;12;0;0
```

The excerpt shows the sums for the following seven different event types (in that order):

1. UserChangeLog
2. SYSLOG
3. SecurityAuditLog
4. ReadAccessLog
5. BusinessTransactionLog
6. HttpServerLog
7. SystemLog

There were a number of reasons for not using all of the resulting 1600 files:

- The anomalies in each file have to be marked manually for the test and training data. This is not feasible for such a large number of files.

- A separate HTM model has to be created and executed for each file. This is due to the fact that each model is supposed to fit a single user's profile to better find anomalies in that user's network activity. Creating and executing 1600 HTM models every time a parameter is changed during the study would exceed the possibilities of this thesis.

- Many files do not contain enough data points. Some users have generated only one or two events over the course of the two weeks, which is not enough to carry out a meaningful calculation. The number of simultaneously active users at any given time ranges from 10 to 150, meaning that only a handful of users have generated most of the events.

- If the data in a given file does not contain any anomalies, it cannot be used to compute a meaningful model score. For example, some users show regular, recurring network activity with no deviation from the pattern. Artificial anomalies could be inserted into these files, but this would render the point of using authentic data useless.

Considering these reasons, it became clear that a selection had to be made; therefore, ten of these 1600 files have been manually selected to be used as training and test cases. The data in these files was carefully analyzed for anomalies, which were then annotated to be excluded from the HTM model's training phase. An additional part of each file was annotated to be used as positive test data and was also excluded from the training phase. Figure 4.2 shows an example of this separation.

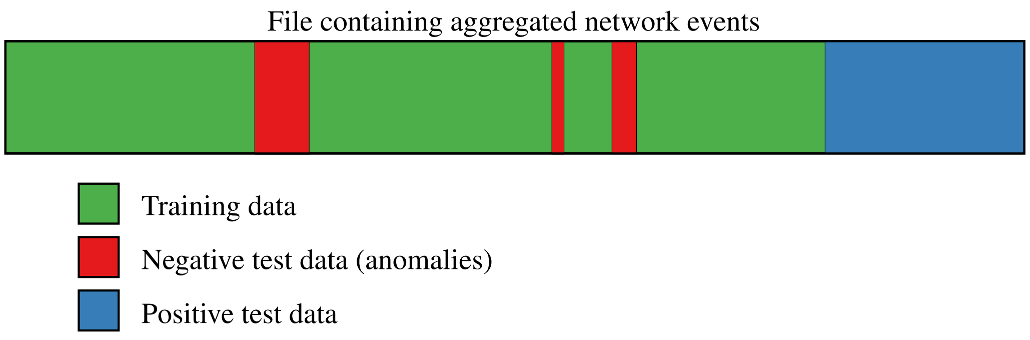

File containing aggregated network events
Training data
Negative test data (anomalies)
Positive test data

Figure 4.2: Example of the separation of training and test data for the network event analysis files.

Similarly to equation 4.1, a model score s^{mod} was calculated for each HTM model with the test data mentioned above. Since a separate HTM model was created for each network user, the final model score s^{mod} used for evaluation and comparison purposes is the average of the individual model scores s_i^{mod}:

$$s^{mod} = \frac{1}{u} \sum_{i=1}^{u} s_i^{mod} \tag{4.4}$$

where u is the total number of selected user files and s_i^{mod} is the model score calculated for the i-th file.

The difference between the two use-cases is that in the case of the network event analysis, multiple positive test cases can be used instead of just one. Therefore, a^{pos} is calculated in the following way, where m is the number of positive test cases, n_k is the number of inputs for the k-th test case and a_i is the anomaly score for a particular input:

$$a^{pos} = \frac{1}{m} \sum_{k=1}^{m} \left(\frac{1}{n_k} \sum_{i=1}^{n_k} a_i\right) \tag{4.5}$$

Note that a^{neg} is calculated similarly to a^{pos} in equation 4.5, as it, too, can contain the result of multiple test cases. As can be seen in figure 4.2, the length of the test cases can vary greatly. However, the score of each test case has an equal weight when calculating a^{pos} or a^{neg}. This approach was chosen because although some test cases, like short bursts of activity, consist of fewer time steps, their result is no less important than that of longer test cases.

4.1.3 Website visitor analysis

This use case analyzes a publicly available data set generated from visits to the *msnbc.com* website.[4] The aim is to predict the navigation behavior of the website users with the help of an HTM model.

The data had already been analyzed for behavior patterns by previous research with varying approaches. For example, users were clustered into different types depending on their visits [5]; another study used a pattern mining algorithm to create a graph showing the user's traversal through the website [39, 53]. Where applicable, the results of the HTM model are compared to the results of these previous studies.

Being able to predict user behavior on the internet has many advantages and possible use cases:

- It is important for website developers and usability experts, because it allows them to facilitate the website transitions for the user. For example, the URLs a user is most likely to click could be highlighted or displayed at the top of a page.
- Content providers are able to cache content based on the prediction of user requests.

[4]Data set URL: https://archive.ics.uci.edu/ml/machine-learning-databases/msnbc-mld/msnbc.html.

- Information can be made more accessible to users by automatically retrieving those parts that are interesting to them.

The data from the msnbc website was recorded in 1999 by logging the requested URLs from each user for an entire day, resulting in about six million recorded requests from one million users. Each URL a user requested was assigned a category according to the part of the msnbc website that it belonged to. For example, a user request for the msnbc.com landing page was assigned the category ’frontpage’.

In total, there are 17 different categories available, each of which is abbreviated with an integer: frontpage (1), news (2), tech (3), local (4), opinion (5), on-air (6), misc (7), weather (8), msn-news (9), health (10), living (11), business (12), msn-sports (13), sports (14), summary (15), bbs (16), travel (17).

Figure 4.3 shows an example for the sequence of requests generated by several users over the course of the day.

User	Recorded data	Requested pages (category)
1	2, 12, 3, 4	news $\longrightarrow$ business $\longrightarrow$ tech $\longrightarrow$ local
2	1, 1, 12, 2	frontpage $\longrightarrow$ frontpage $\longrightarrow$ business $\longrightarrow$ news
3	6, 8	on-air $\longrightarrow$ weather
4	4, 3, 4	local $\longrightarrow$ tech $\longrightarrow$ local

Figure 4.3: Examples for the sequence of website requests sent by the users.

As with the other two use cases, a model score s^{mod} was calculated for the HTM model. However, since the focus in this case was prediction rather than anomaly detection, the score was calculated somewhat differently. For each prediction the HTM model makes, it also provides the estimated probability for each prediction. This is taken into account when calculating the model score in the following way, where n is the number of inputs and p_i is the predicted probability for the i-th input:

$$s^{mod} = \frac{1}{n}\sum_{i=1}^{n} p_i \tag{4.6}$$

If an input was not predicted by the model at all, then a probability value of 0 is used.

4.2 Data collection

Using a framework like *nupic*, which has a large amount of different configuration parameters, is challenging because of the amount of data generated during its execution. To answer the research question posed in chapter 2, it is not sufficient to merely record the results provided by the framework. Additional values required are the runtimes of different model phases (creation, training, test), the influence of model parameters on the results and the training results in addition to the test results.

To gather these values in a controlled manner, a test bed was developed which automatically creates and executes HTM models with varying parameters while recording all necessary data. The *nupic* framework already includes a swarming algorithm that works in a similar fashion. However, the difference is that the swarming algorithm was designed to find the best set of parameters for a single model – not to systematically explore all parameters and record the results.

Chapter 5

Results

This chapter contains the model score and runtime results gathered during the evaluation of the use cases. The values for each of the three use cases are displayed in the charts below. The equations used for calculating the model scores can be found in section 4.1. A model's runtime is the sum of the training and testing times.

A notable difference can be found between the runtimes of the different use cases. For example, the web prediction model usually only runs for a few seconds, whereas the network analysis runs for several minutes. This is due to the fact that, while the web prediction use case employs a single HTM model with a simple input encoder, the network analysis requires ten separate models with complex input encoding.

Every chart below describes how the model scores of the use cases change in response to the modification of one particular HTM model parameter. The diamond-shaped data points in each chart show the default values used for the calculations of the other results.

5.1 Training Data

The amount of training data used has a direct influence on the quality of the results in any machine learning algorithm. Additionally, more training data requires more time for the training phase, increasing the overall runtime, which can be seen in figure 5.2. Figure 5.1 shows how the amount of training data influences the model score s^{mod} in each use case; while more training data has a positive effect on the key dynamics and network analysis use case, the additional data slightly reduces the model score in the web prediction case.

While there was more than enough training data for the web prediction and key dynamics use cases, the network analysis use case only provided about 3000 data points for training. However, these data points are counted separately for each of the ten HTM models used to calculate the model score, resulting in the higher overall runtime.

5.2 Model Parameters

To create a new CLA model, a number of different parameters have to be specified. These parameters can be roughly divided into three categories related to the parts of the CLA client: encoders, spatial pooler and temporal pooler. The configuration files for the tested CLA models typically contained around 80 different parameters, most of which were uninteresting for the purpose of this study.[1] For each parameter tested, a short description of its purpose and value range will be given along with the test results.

Column count The number of columns available to an HTM model's region defines the number of columns available to the spatial and temporal pooler as well as the input width. Although each of these values can be adjusted separately in the configuration, they have to be the same number or the model cannot be generated.

Figure 5.4 shows how the model runtime steadily increases in each use case with the amount of available columns. The results displayed in figure 5.3 show how the model score if affected by a change in column count. While the key dynamics and web prediction scores are hardly affected by a change in column count, the network analysis case shows a large spread.

5.2.1 Encoders

The values for the encoder parameters are dependent on the type of encoder used. The most adaptable type of encoder is the *ScalarEncoder*, because it simply turns an integer within a given range into an SDR. Many other encoders, for example the *CategoryEncoder*, use a *ScalarEncoder* with adjusted parameters internally. The minimum and maximum of the encoder's range is specific to the use case data and was not tested as part of this study.

w – the number of "on" bits This determines the number of output bits that should be active when a value is encoded. The value must be at least 1, but the recommended minimum is 21.

Figure 5.6 shows that the parameter's influence on the model runtime is minimal for each use case. The influence on the model score is displayed in figure 5.5. While the web prediction scores are hardly affected, the other two use cases benefit from a higher amount of bits. This might be due to the fact that, in order to generalize input values, a certain amount of active bits must overlap in the SDR created by the encoder. After a certain overlap threshold is reached, there is no more benefit from adding more active bits. The web prediction use case does not show this behavior since its category encoder is not able to create an SDR with overlapping active bits for different input values.

n – the number of total bits This determines the total number of output bits available to the encoder. The value must be greater than the value of parameter *w*, but there is no restriction

[1]For example, these parameters include the seed for the random number generator and information on the aggregation of sensor input.

otherwise. Note that the web prediction use case is not represented here since it uses a category encoder where the number of total bits is directly derived from the parameter of active bits. This is due to the fact that encoded categories must have no overlapping bits, so the number of total bits is equal to the number of active bits multiplied by the number of categories.

Figure 5.8 shows how the model runtime increases linearly with the number of encoder bits. The parameter also significantly influences the model score, as displayed in figure 5.7. Similar to the results of previous parameters, the spread of model score values is much greater in the network analysis use case. Both use cases displayed in figure 5.7 do not benefit from more encoder bits after a certain amount has been reached.

5.2.2 Spatial Pooler

As detailed in section 3.2.2, the spatial pooler is a step in the CLA algorithm. Its role is to find spatially similar patterns and represent them as similar SDRs to the temporal pooler. For this purpose, the spatial pooler takes the output of the encoders as input.

numActiveColumnsPerInhArea – the inhibition area size This parameter defines the number of columns affected by the inhibition described in step 2b of the CLA algorithm in section 3.2.2. Figure 5.9 shows the parameter's influence on the model score, while figure 5.10 illustrates the influence on the model runtime. The runtime increases with a larger inhibition area since it requires more columns to be checked during every pass of the spatial pooler.

potentialPct – the column's receptive field size This parameter defines the size of a column's receptive field that is available for potential synapses. A lower value means that fewer connections are available between columns. The parameter has a value range greater than zero and smaller or equal to one. The value is a percentage of the size of the cortical region; for example, a value of 0.4 means that a column's receptive field size constitutes 40% of the whole region.

Figure 5.11 shows the parameter's influence on the model score. Likewise, figure 5.12 shows the influence on the model runtime. As can be seen, the parameter has no influence on the prediction quality or runtime of the web prediction use case. Similarly, the network activity use case shows some spread in model score, but is otherwise also not improved or worsened. The key dynamics use case, on the other hand, shows a steady improvement in both runtime and model score with a larger receptive field.

synPermConnected – the synapse connection threshold This parameter governs which synapses count as being connected to a given cell. Any synapse with a permanence greater than the threshold parameter is seen as connected to a cell and is able to contribute to the cell's firing. Synapses with a value lower than the threshold are seen as potential synapses and can become active after weight adjustments during the learning phase.

Figure 5.14 shows that the parameter has no influence on the model runtime. Likewise, figure 5.13 shows that model score is also hardly affected by a parameter change. Even the spread of model score values usually seen in the network analysis use case is minimal for this parameter.

synPermActiveIncrease – The increase in active synapse strength This parameter determines how much strength is added to an active synapse's connection. During the learning phase of the spatial pooler, the synapses of all firing columns are reinforced by the value of this parameter. The details of the learning algorithm are described in section 3.2.2.

Figure 5.16 shows that the parameter's influence on the model runtime is minimal for the network analysis and web prediction use cases. The runtime for the key dynamics use case, on the other hand, increases linearly with the value of the parameter.

Figure 5.15 shows that the parameter only influences the model scores of the two anomaly detection use cases.

synPermActiveDecrease – The decrease in inactive synapse strength This parameter is very similar to the *synPermActiveIncrease* parameter; however, instead of strengthening connections, it determines how much inactive synapse's connections are weakened. During the learning phase of the spatial pooler, the synapses of all columns connected to inactive input are weakened by the value of this parameter. As with the previous parameter, the details of the learning algorithm are described in section 3.2.2.

Figure 5.17 shows the parameter's influence on the model score, where higher parameter values correlate with very low scores in the network analysis use case. In addition, figure 5.18 shows the influence on the model runtime, which is minimal in all cases – except for a sharp increase for very small values. The reason for this sharp increase is unknown.

5.2.3 Temporal Pooler

As detailed in section 3.2.2, the temporal pooler is a step in the CLA algorithm and follows the execution of the spatial pooler. It's role is to represent temporally similar patterns in context and make predictions based on these patterns.

cellsPerColumn – The number of cells in each column This parameter defines the number of available cells for each column of the temporal pooler. The more cells a column has, the more contexts it can represent a given input in, by activating different cells for each context. The minimum for the parameter value is two, but there is no maximum for the value.

Figure 5.19 shows the parameter's influence on the model score, which is minimal for all use cases. Furthermore, figure 5.20 shows the influence on the model runtime. As can be seen in this figure, the drawback of more cells is the increased runtime caused by a more complex model.

newSynapseCount – The number of new synapses formed during training This parameter defines the number of synapses that are newly created in the temporal pooler during training phases. Figure 5.21 shows the parameter's influence on the model score. Similarly, figure 5.22 shows the influence the parameter has on the model runtime.

The parameters influence on the model's prediction capability, as represented by the web prediction use case, is minimal. However, the model's anomaly detection capabilities are apparently greatly affected. They show an unusual model score of zero and a sharply increased runtime for parameter values smaller than 12. The reasons for this are unknown, especially since the web prediction values are unaffected.

initialPerm – The starting permanence for model synapses This parameter defines the permanence value assigned to all synapses in a newly created model. Figure 5.24 shows that the parameter has little influence on the model runtime. The parameter's influence on the model score, as displayed in figure 5.23, is also minimal except for a sudden sharp drop after the value 0.5.

permanenceInc – The increase in synapse permanence This parameter defines how much the active synapse permanence is increased during the training phase. As displayed in the figures 5.25 and 5.26, the parameter has little influence on either the model score or runtime. The only exception is the sharp increase of model runtime in the network analysis case for very small values of the parameter.

permanenceDec – The decrease in synapse permanence Correspondingly to the previous parameter, this parameter defines how much the inactive synapse permanence is decreased during the training phase. As displayed in the figures 5.27 and 5.28, the parameter's value does not influence either model score or runtime of any use case.

minThreshold – The segment search threshold This parameter defines the threshold for the number of active synapses required in a segment for it to be considered during the best-matching segment search. Figure 5.29 shows that a higher parameter value has a positive influence on the model score in the two anomaly detection use cases, whereas it has no influence on the web prediction's model score. The runtime displayed in figure 5.30 is not affected by a change of the parameter value.

activationThreshold – The segment activation threshold This parameter defines the threshold for the number of connected synapses required for a segment to be activated. Figure 5.31 shows the parameter's influence on the model score. While the score for the web prediction case is unaffected, the scores for the other two use cases show hill-shaped curves with different optimal values. Figure 5.32 shows that the model runtime is unaffected by the parameter value, aside from a sharp runtime increase for the value zero.

pamLength – The learned sequence length This parameter defines how many elements the temporal pooler appends to the end of a learned sequence. The parameter's documentation states that smaller values are better suited for data-sets containing short sequences, whereas bigger values should be used for data-sets with long sequences.

Although all of the three use cases contain sequences of variable length, a change of the parameter's value showed little effect on the model score aside from a decrease in the key dynamics and web analysis cases. Figure 5.33 displays the results for the model scores. The model runtimes, as displayed in figure 5.34 increased noticeably with the parameter value. This stems from the fact that longer sequences in the temporal pooler also require more calculations for each iteration of the HTM model.

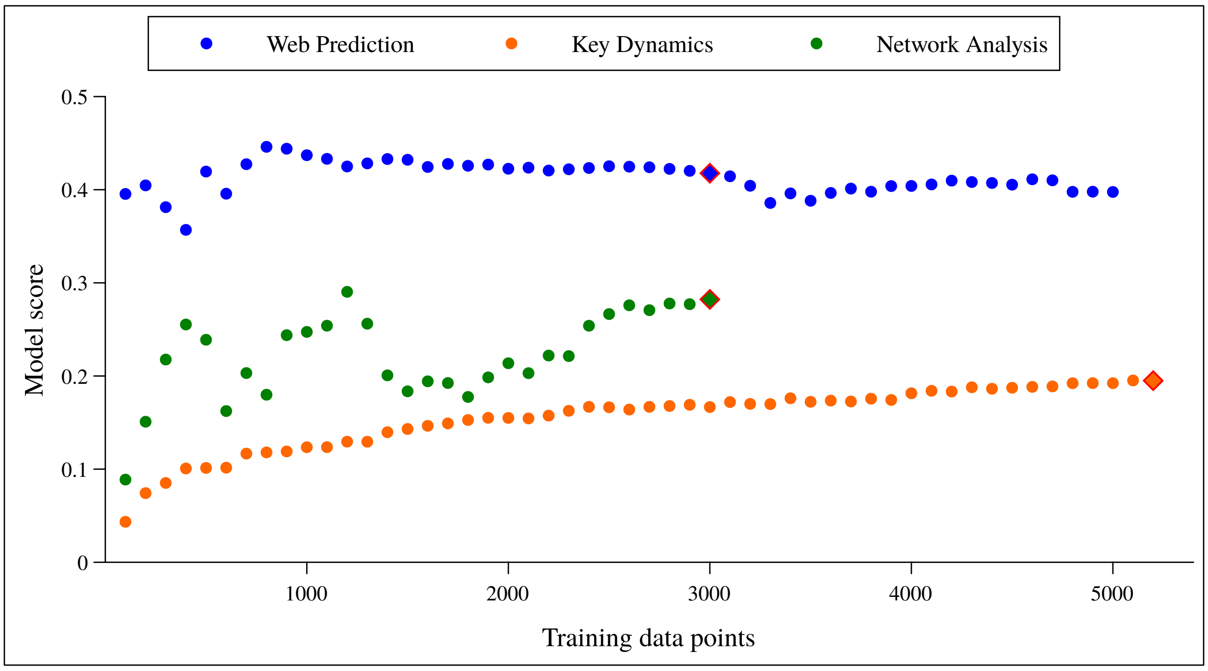


Figure 5.1: Shows the relationship between the amount of training data and model score. Marked data points show the default value for each use case.

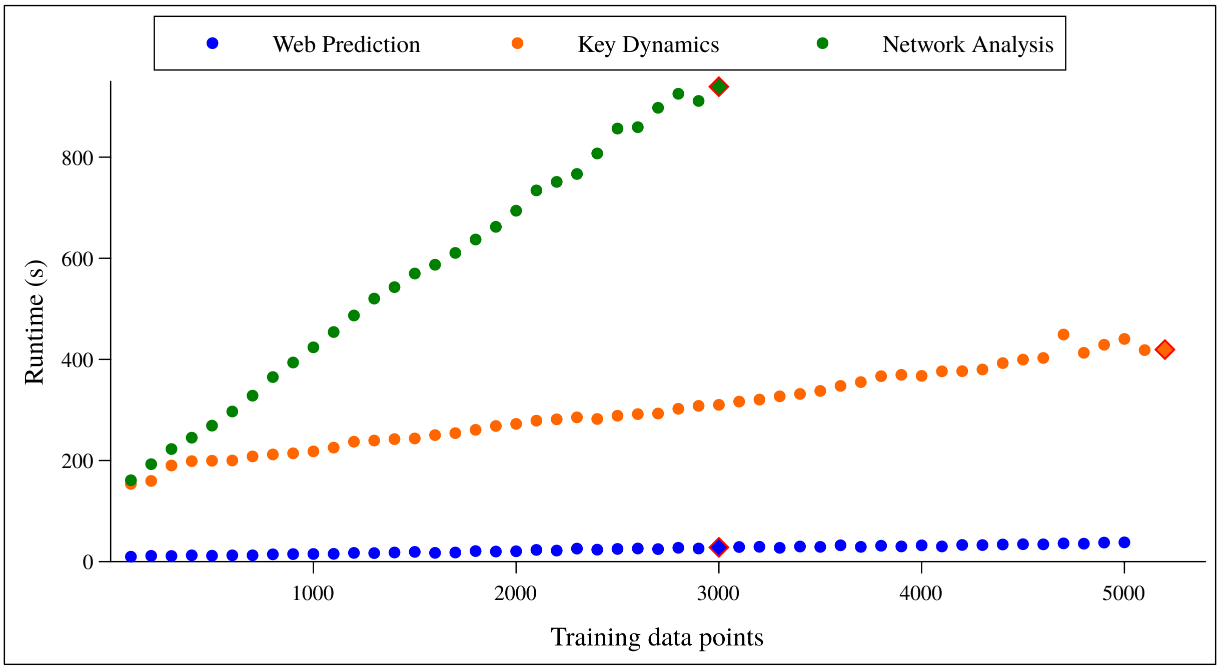


Figure 5.2: Shows the relationship between the amount of training data and model runtime. Marked data points show the default value for each use case.

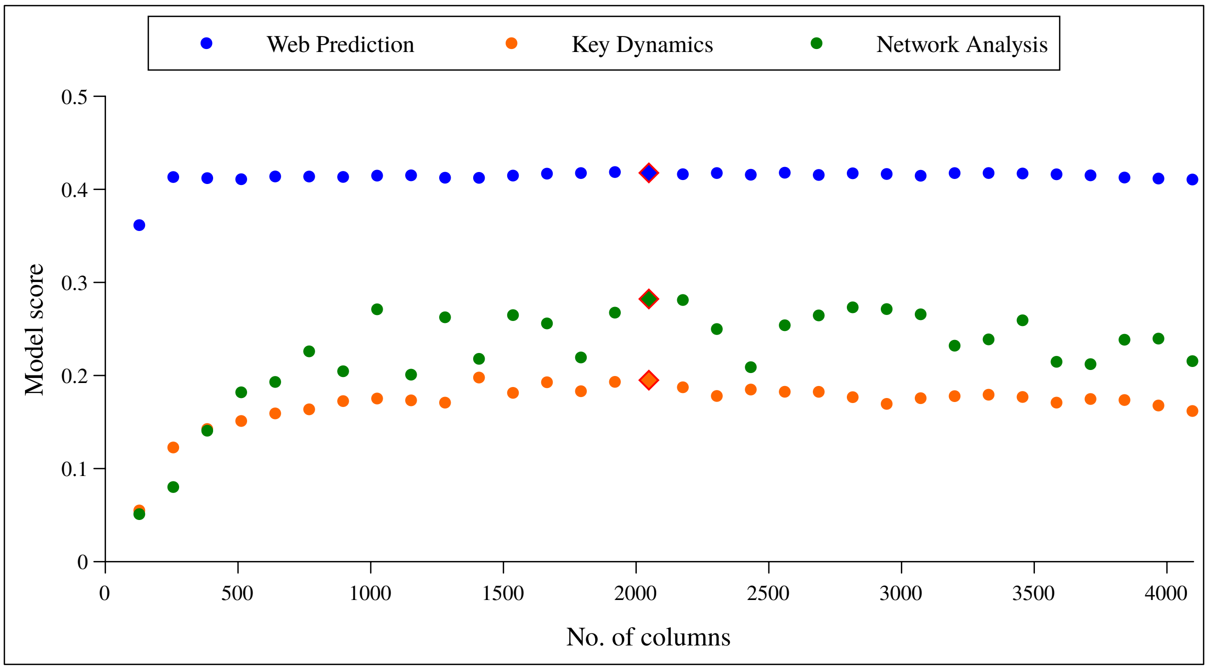


Figure 5.3: Column count parameter influence on the model score.

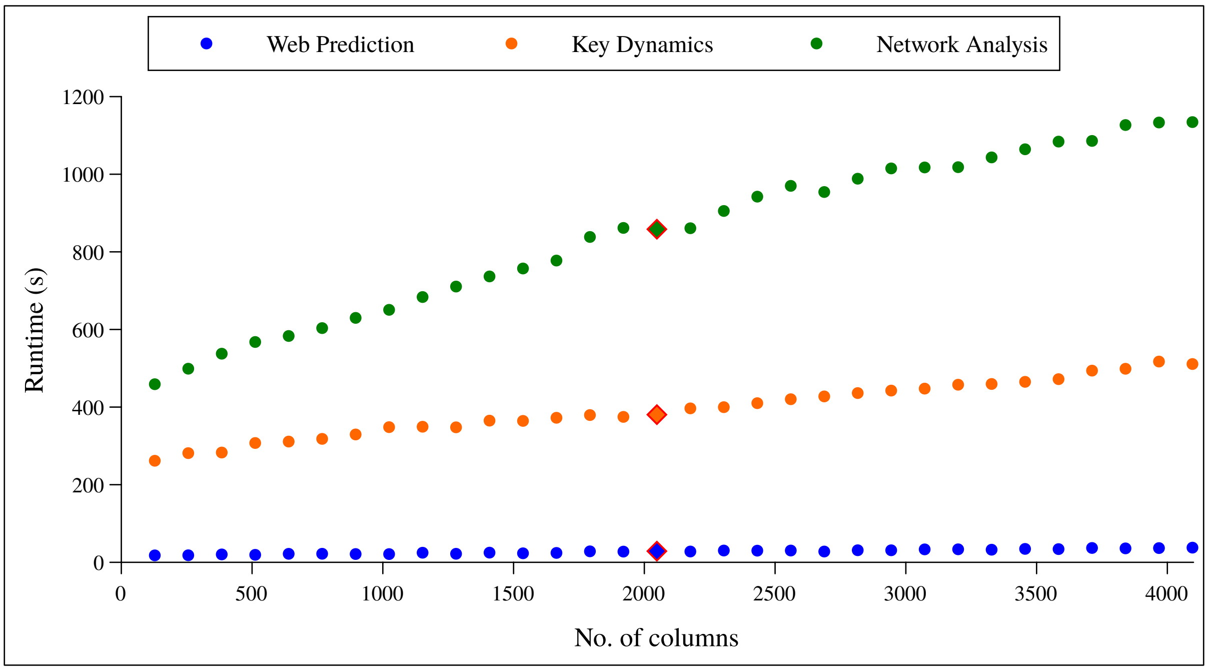


Figure 5.4: Column count parameter influence on the model runtime.

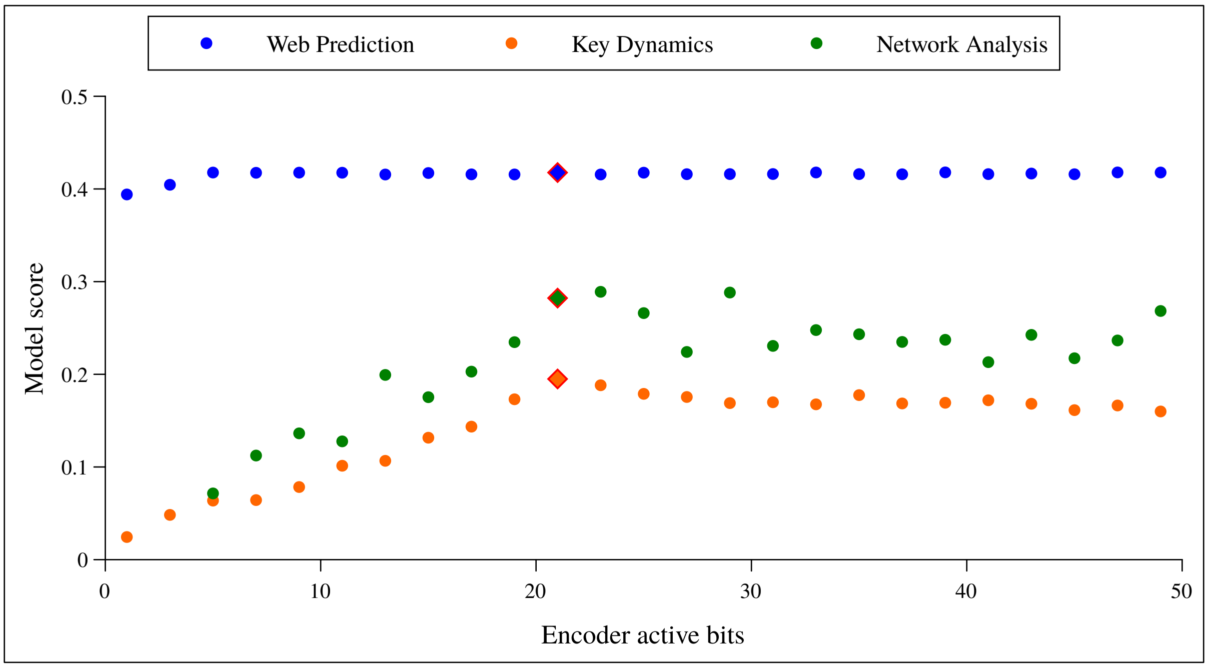


Figure 5.5: Active encoder bits parameter (w) influence on the model score.

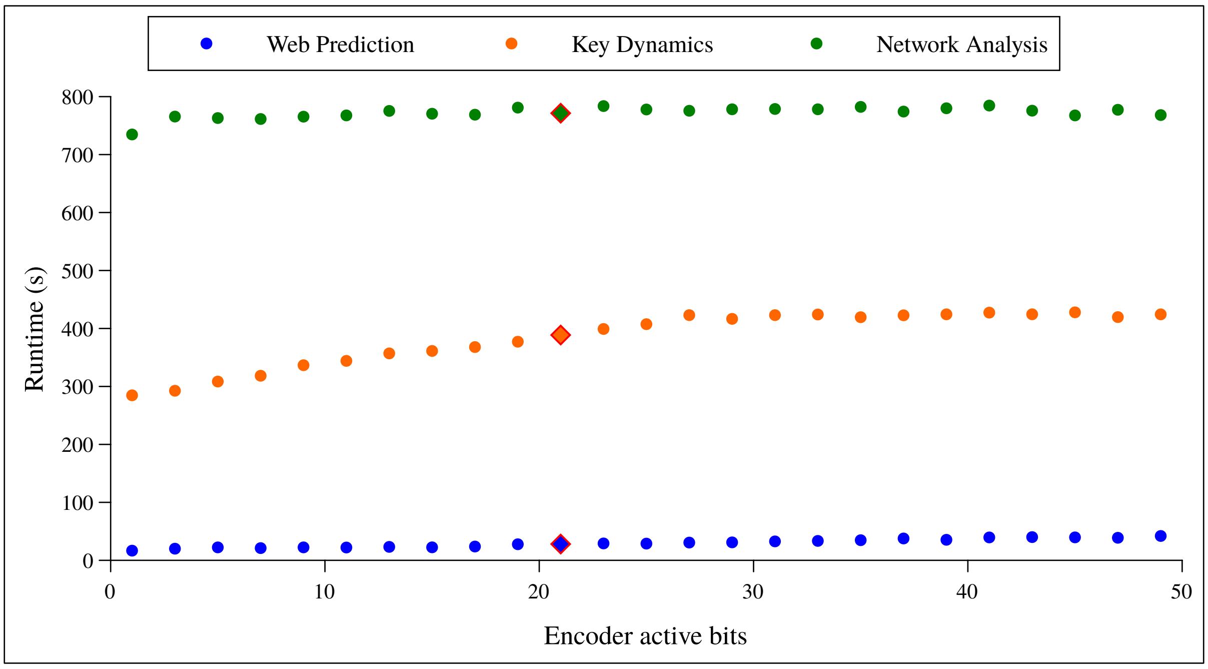


Figure 5.6: Active encoder bits parameter (w) influence on the model runtime.

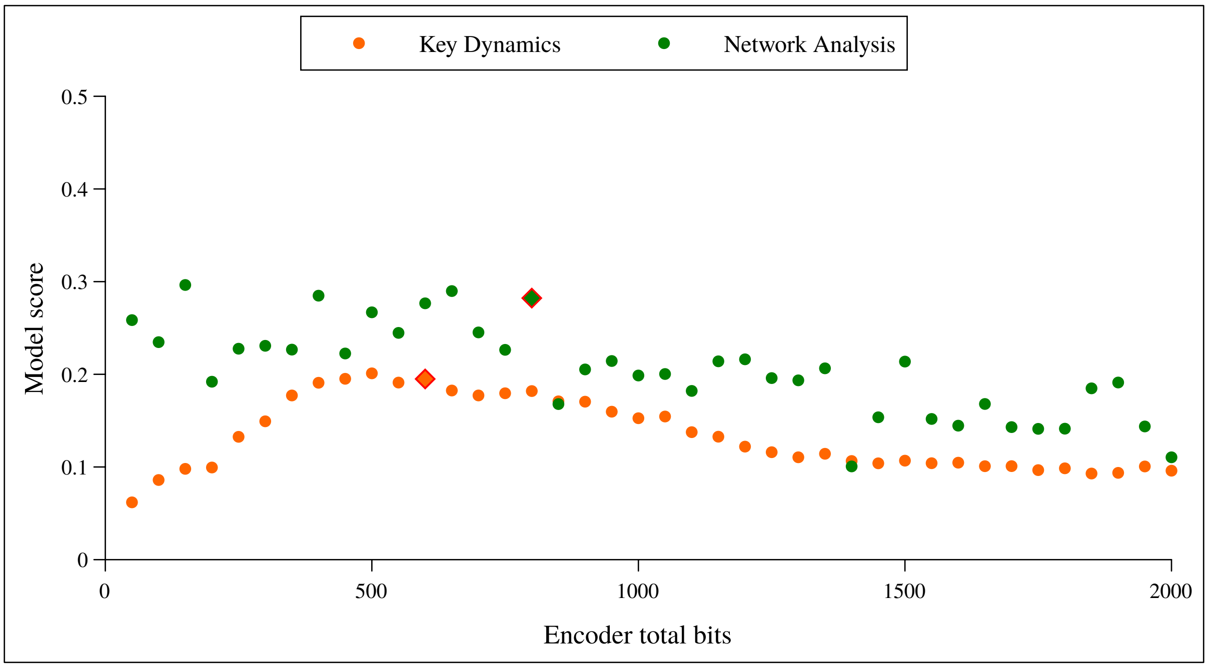


Figure 5.7: Total encoder bits parameter (n) influence on the model score.

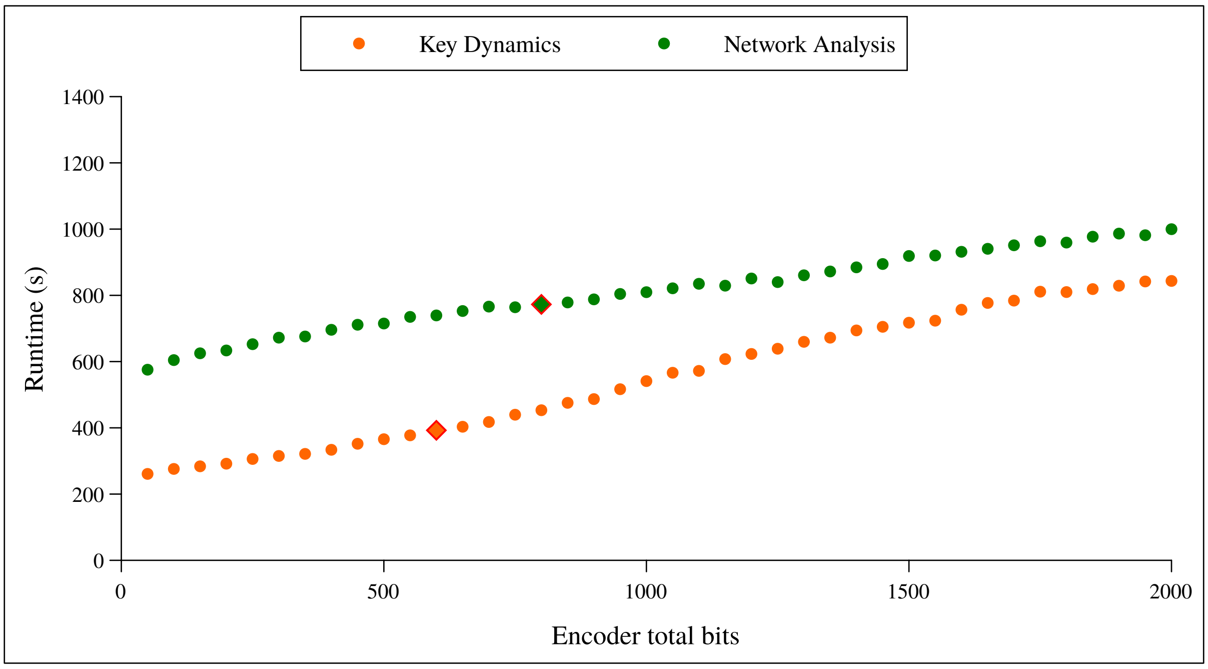


Figure 5.8: Total encoder bits parameter (n) influence on the model runtime.

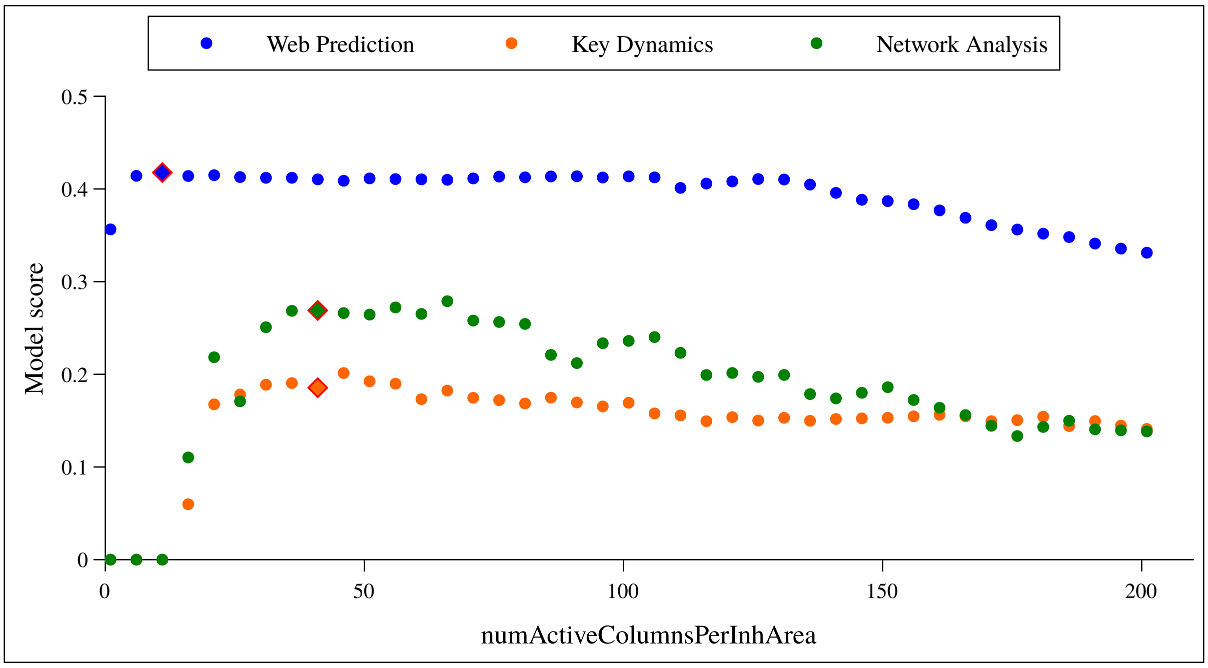


Figure 5.9: Shows the influence of the spatial pooler's inhibition area size on the model score.

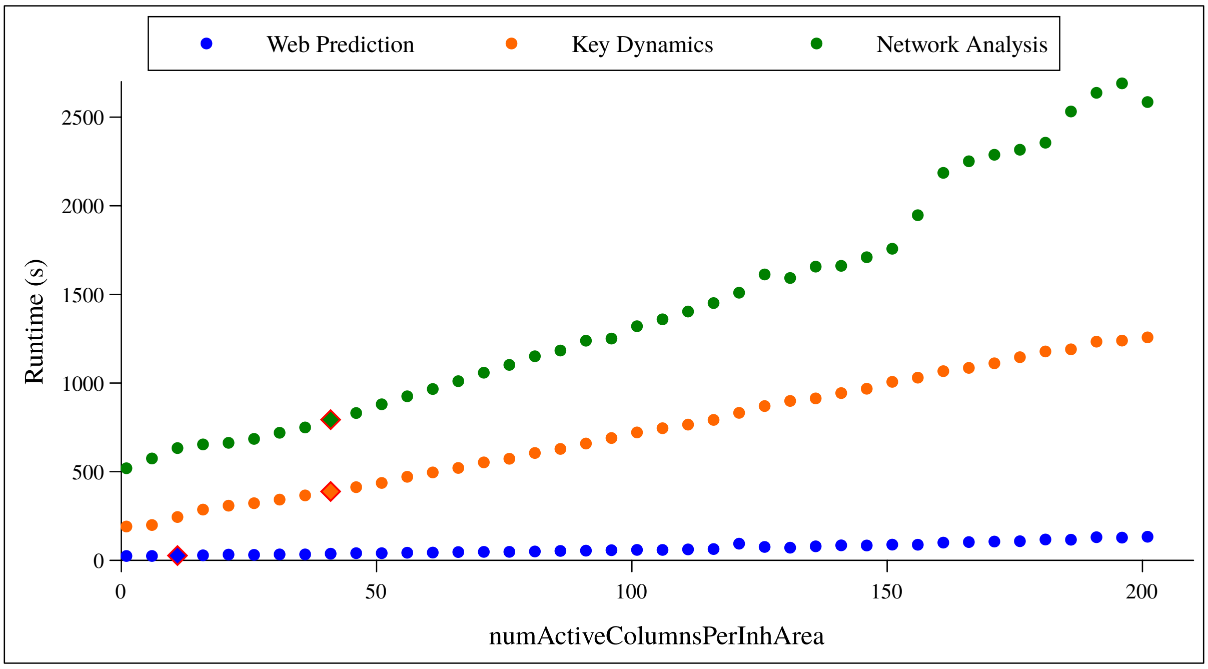


Figure 5.10: Shows the influence of the spatial pooler's inhibition area size on the model runtime.

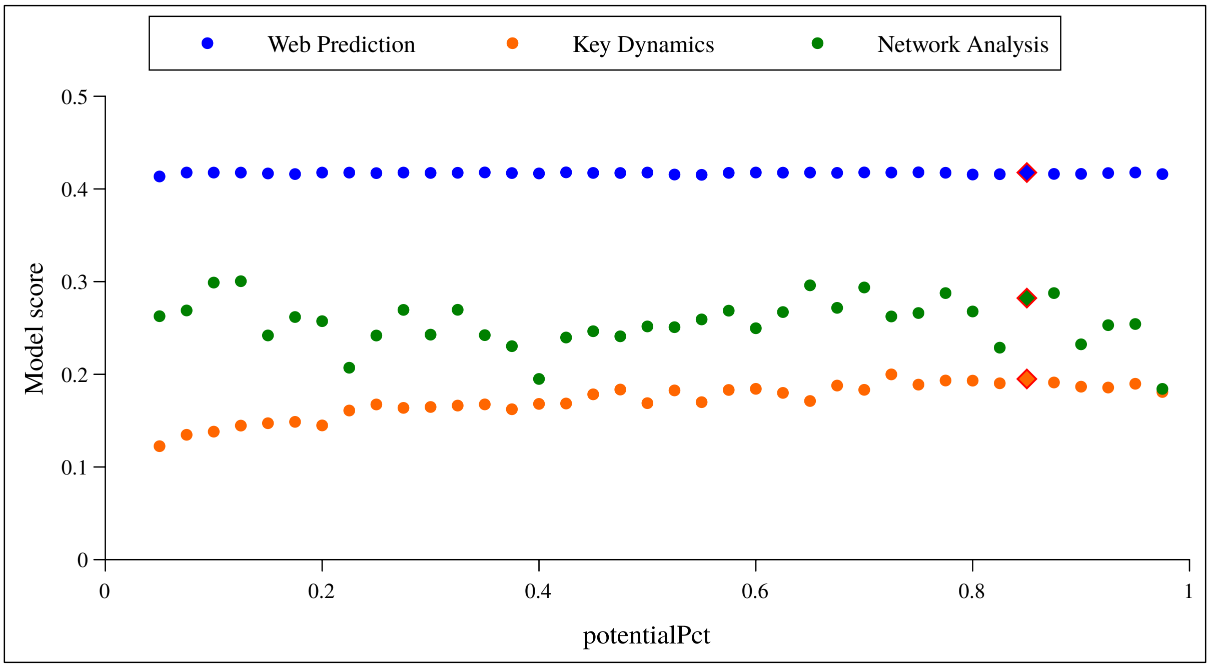


Figure 5.11: Shows the influence of a column’s receptive field size on the model score.

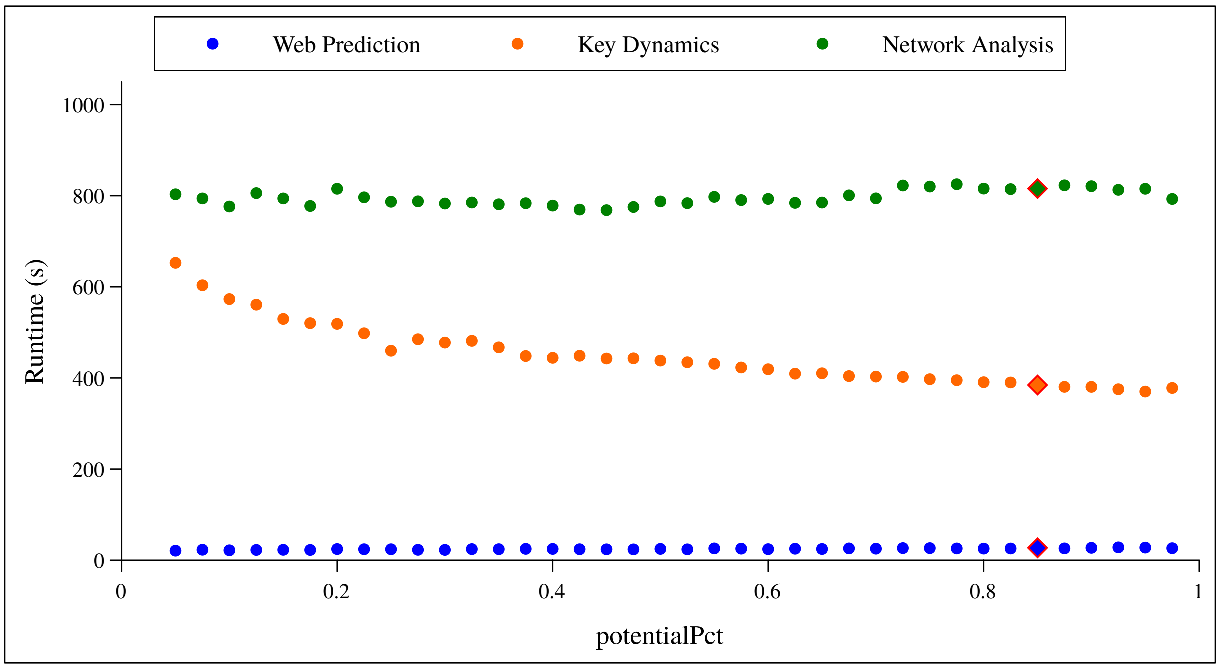


Figure 5.12: Shows the influence of a column’s receptive field size on the model runtime.

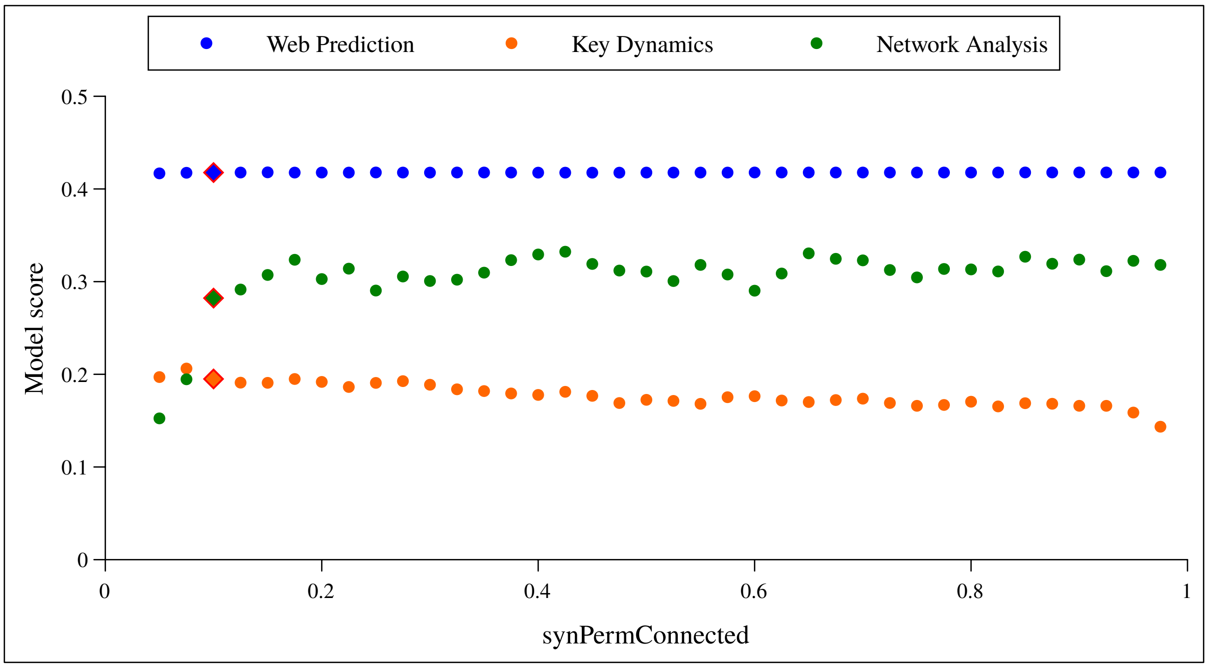


Figure 5.13: Shows the influence of the connection threshold parameter on the model score.

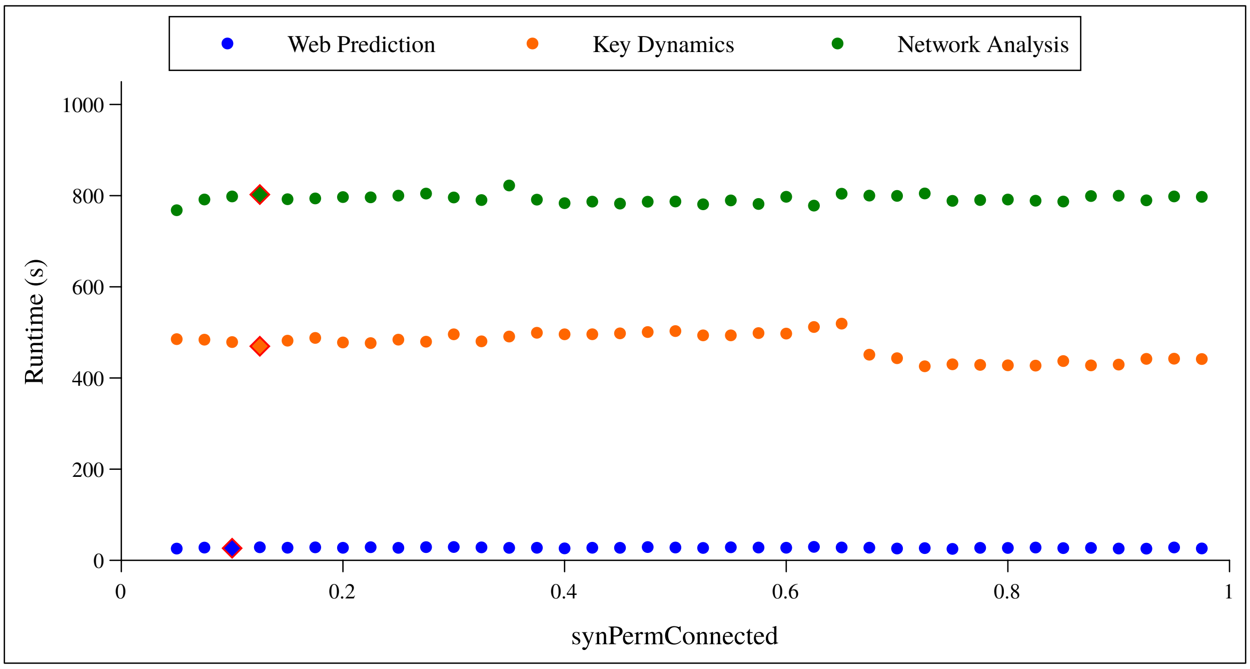


Figure 5.14: Shows the influence of the connection threshold parameter on the model runtime.

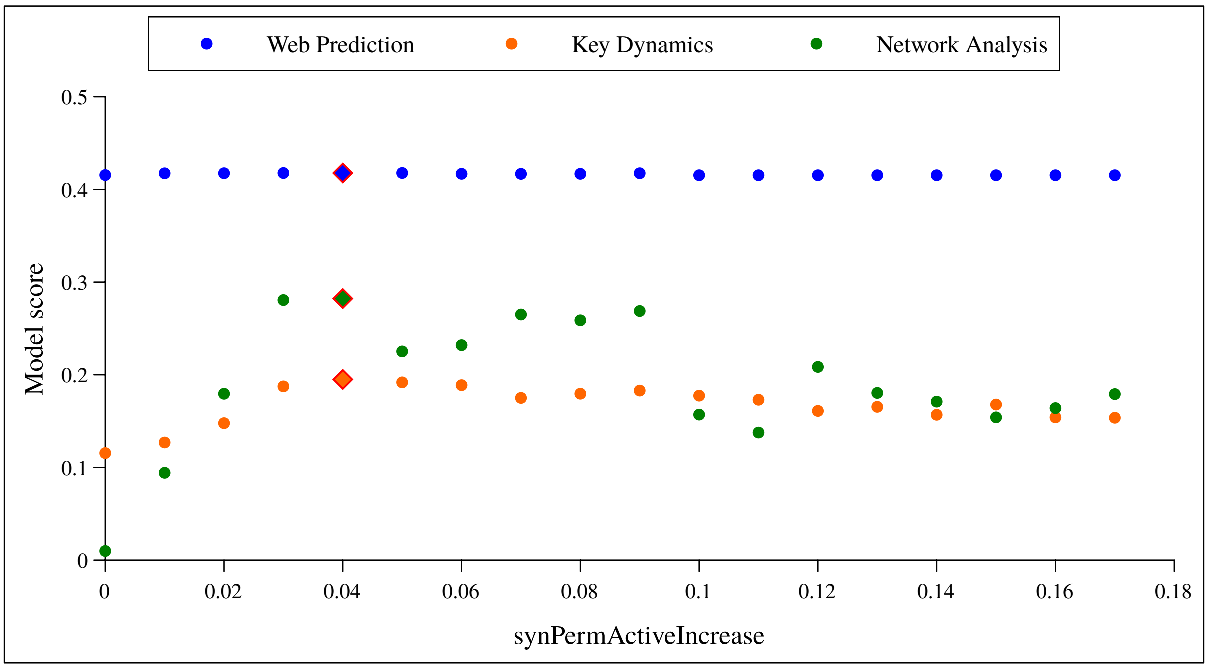


Figure 5.15: Shows the influence of the connection reinforcement parameter on the model score. The parameter is used to increase the connection strength on active synapses during the learning phase.

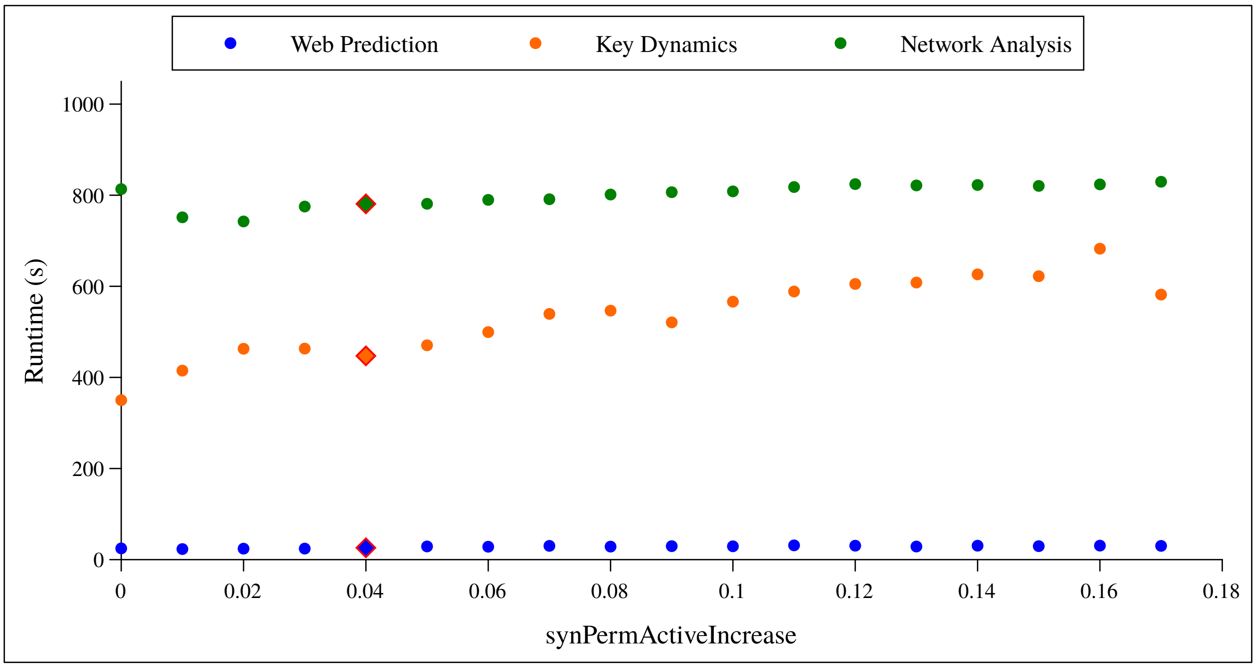


Figure 5.16: Shows the influence of the connection reinforcement parameter on the model runtime. The parameter is used to increase the connection strength on active synapses during the learning phase.

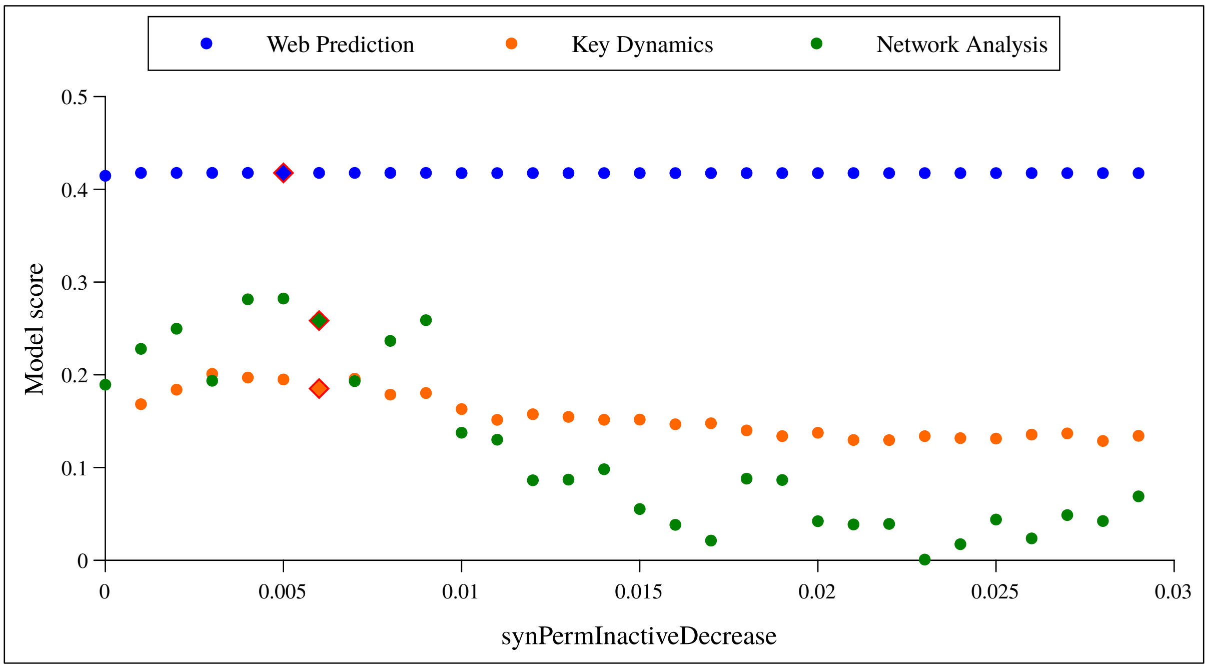


Figure 5.17: Shows the influence of the connection weakening parameter on the model score. The parameter is used to reduce the connection strength on inactive synapses during the learning phase.

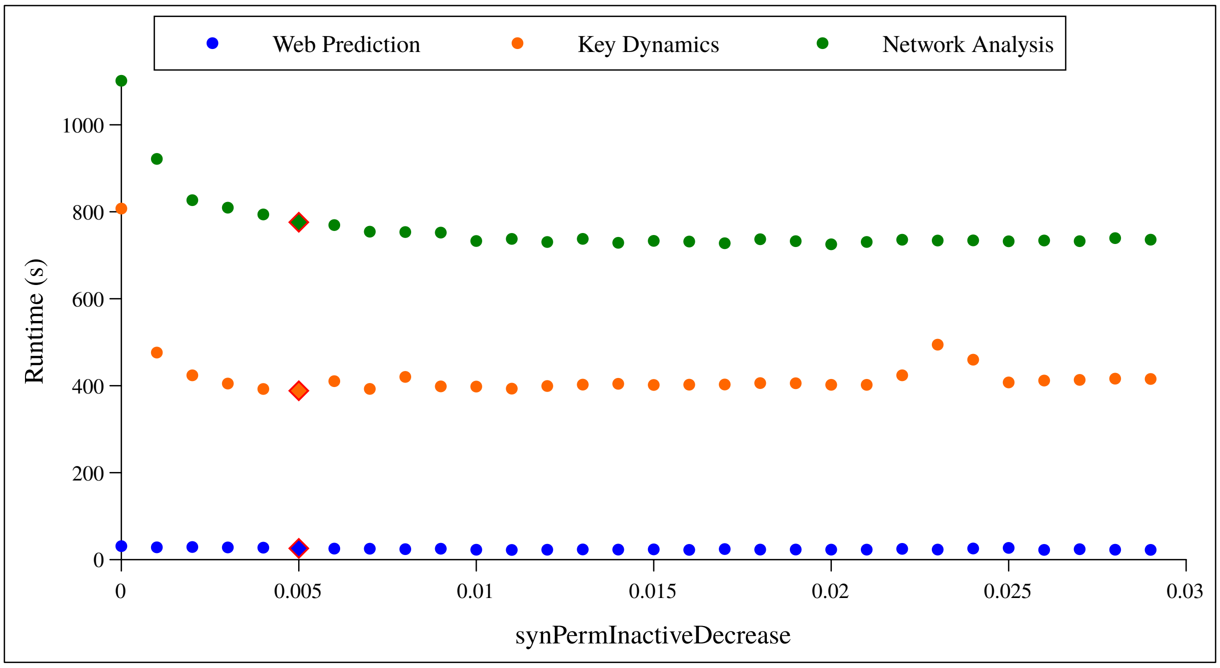


Figure 5.18: Shows the influence of the connection weakening parameter on the model runtime. The parameter is used to reduce the connection strength on inactive synapses during the learning phase.

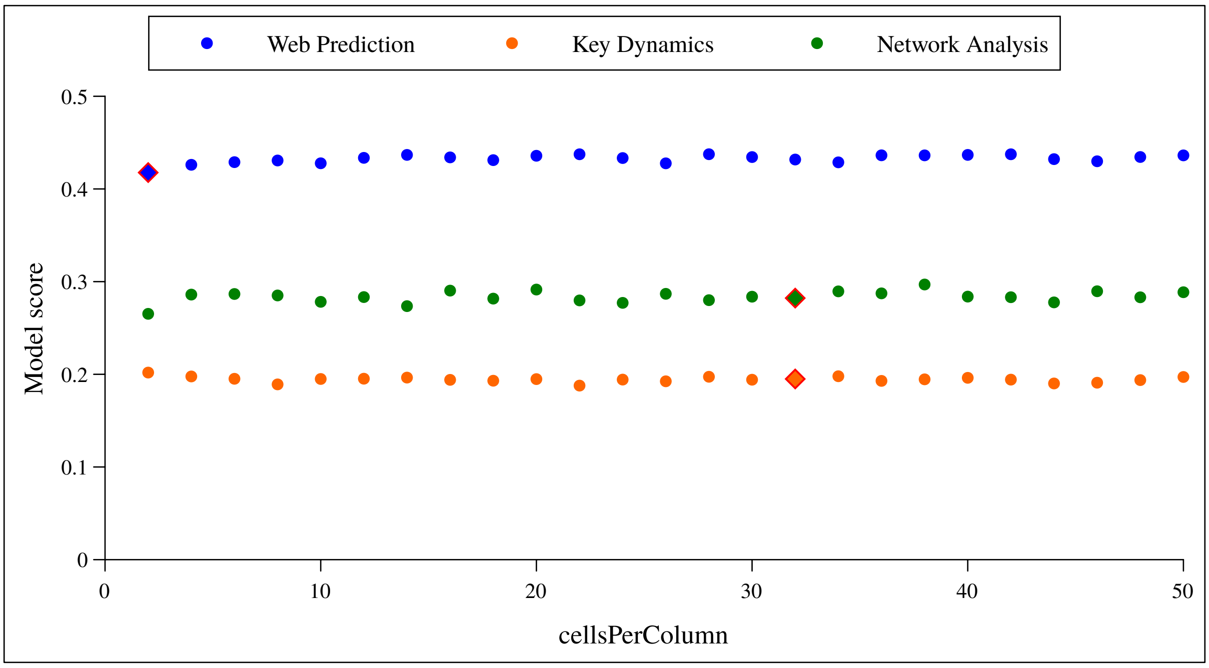


Figure 5.19: Shows the influence of the column's cells count on the model score.

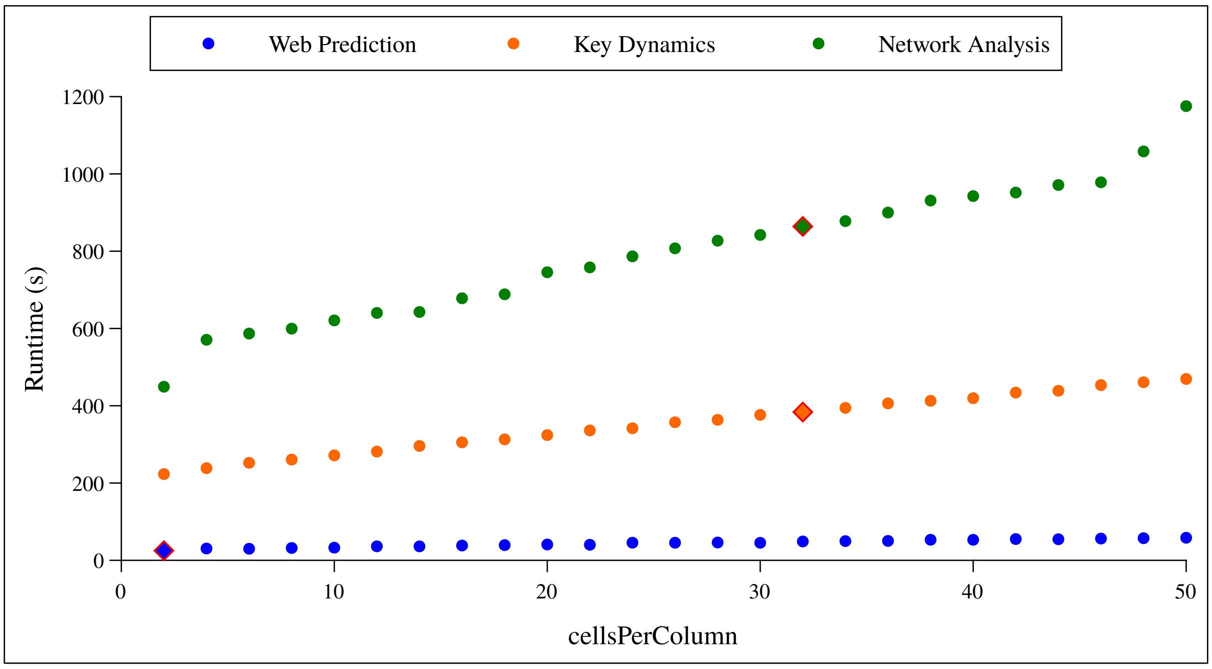


Figure 5.20: Shows the influence of the column's cells count on the model runtime.

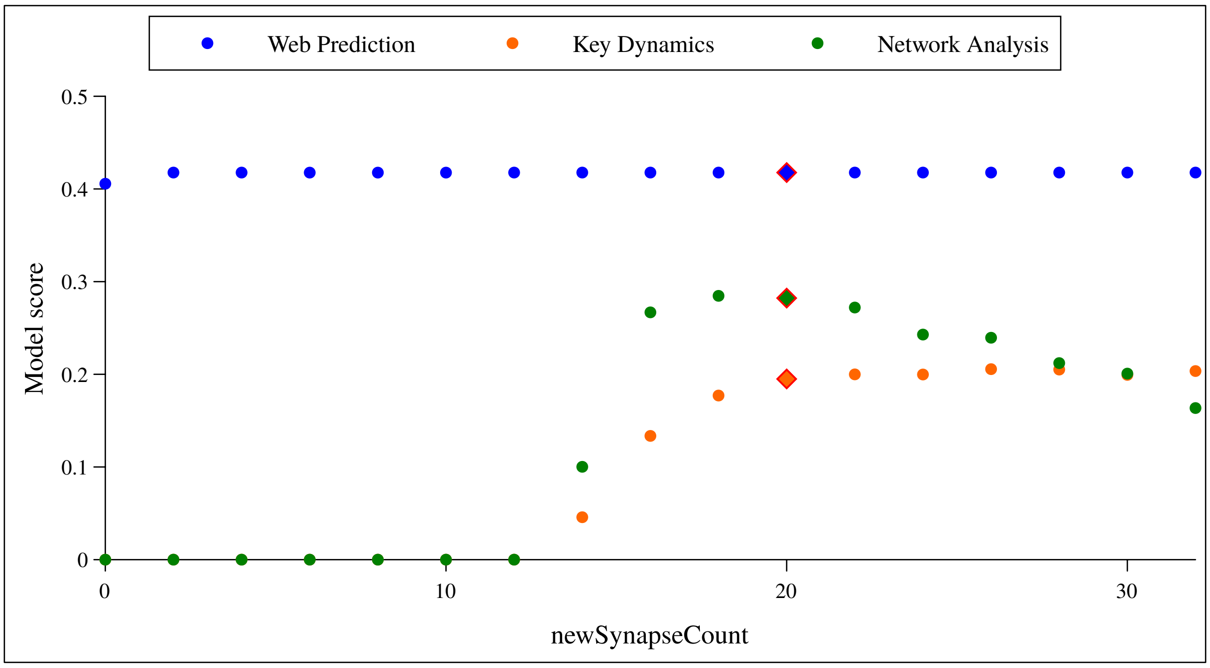


Figure 5.21: Shows the influence the number of new synapses has on the model score.

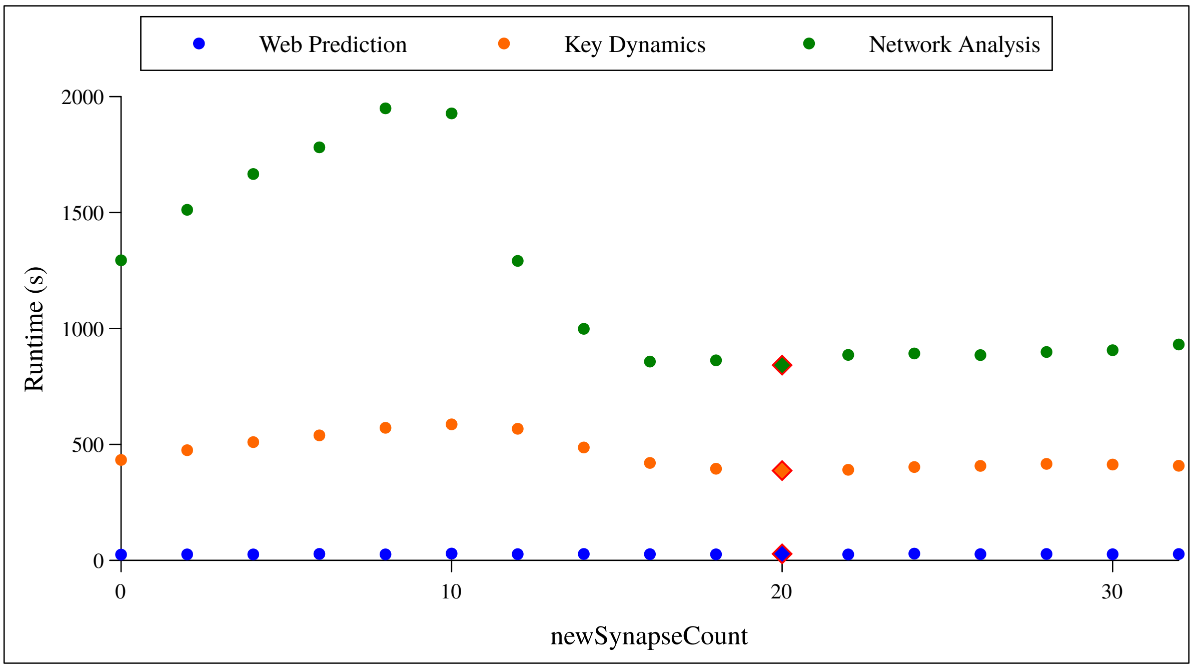


Figure 5.22: Shows the influence the number of new synapses has on the model runtime.

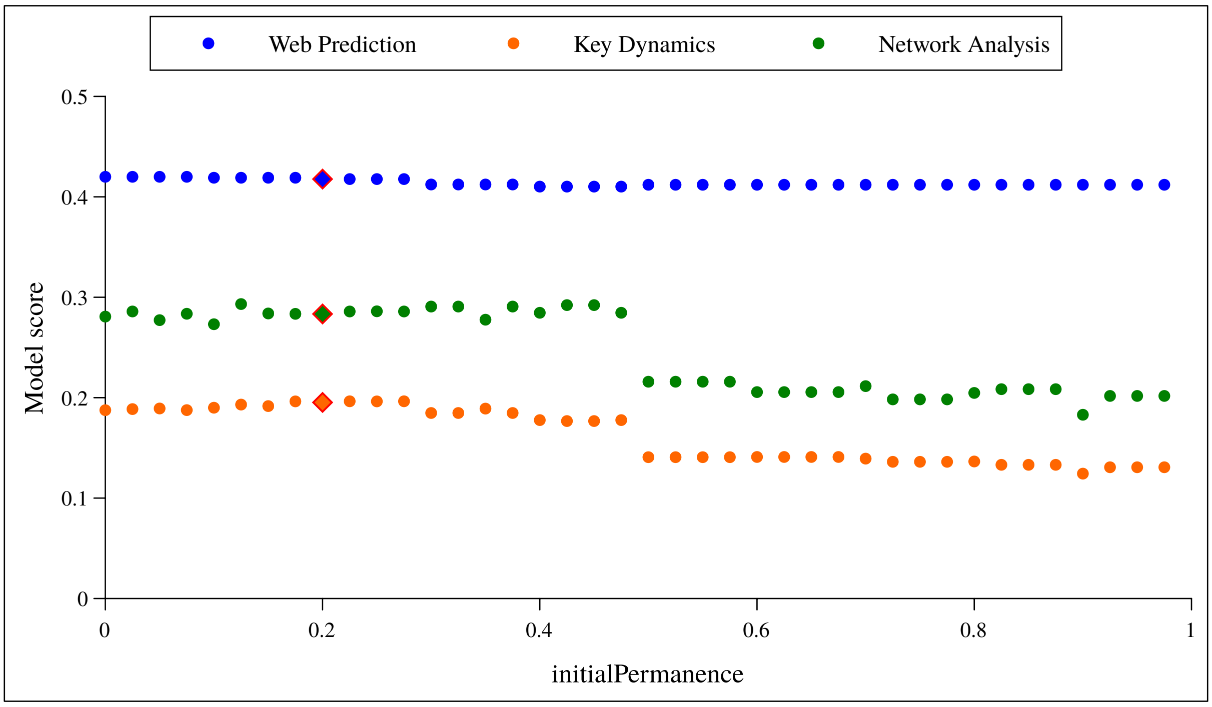


Figure 5.23: Shows the influence the initial synapse permanence has on the model score.

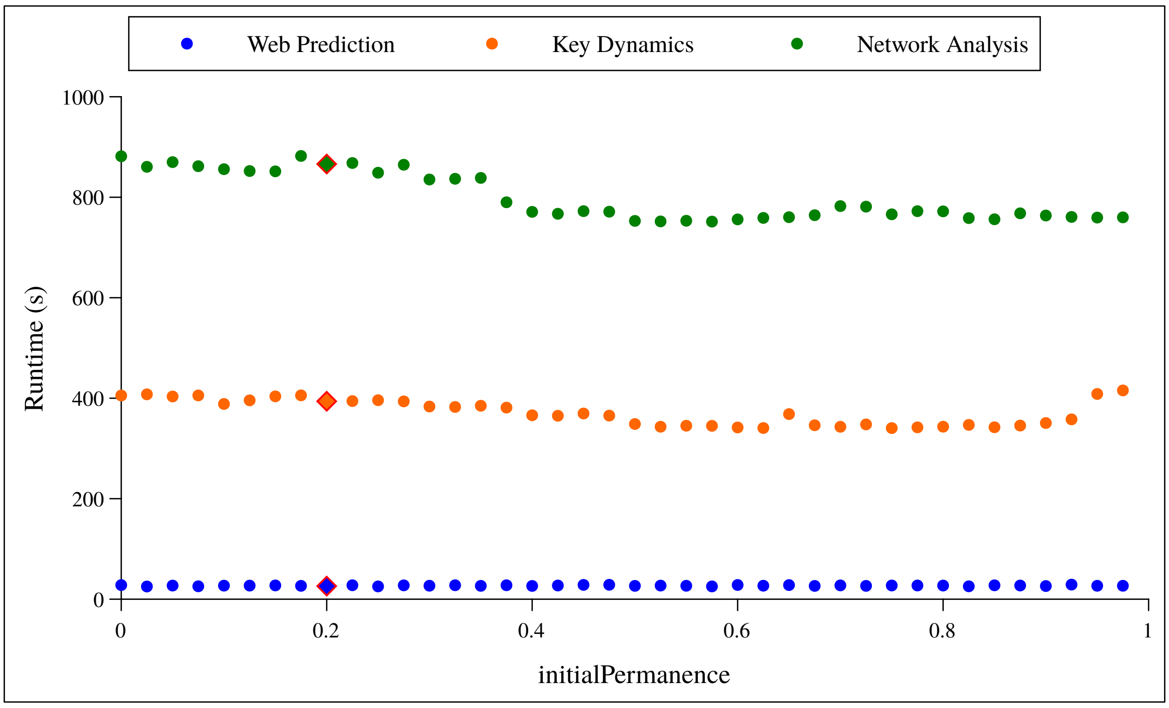


Figure 5.24: Shows the influence the initial synapse permanence has on the model runtime.

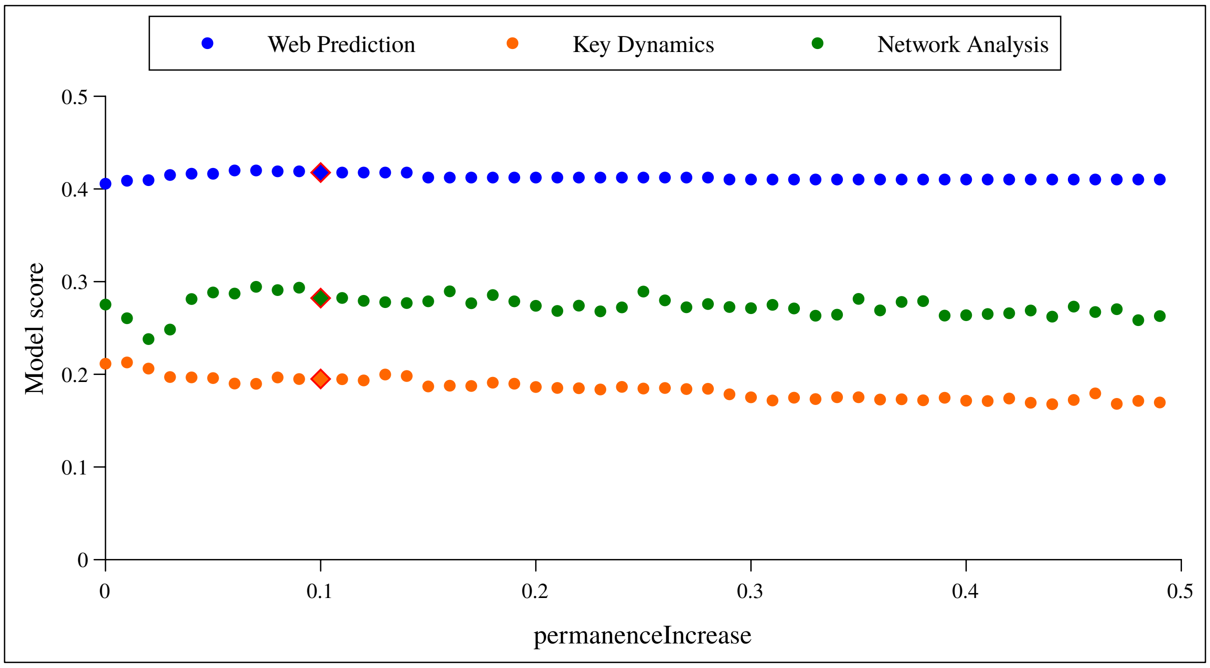


Figure 5.25: Shows the influence the increase of synapse permanence has on the model score.

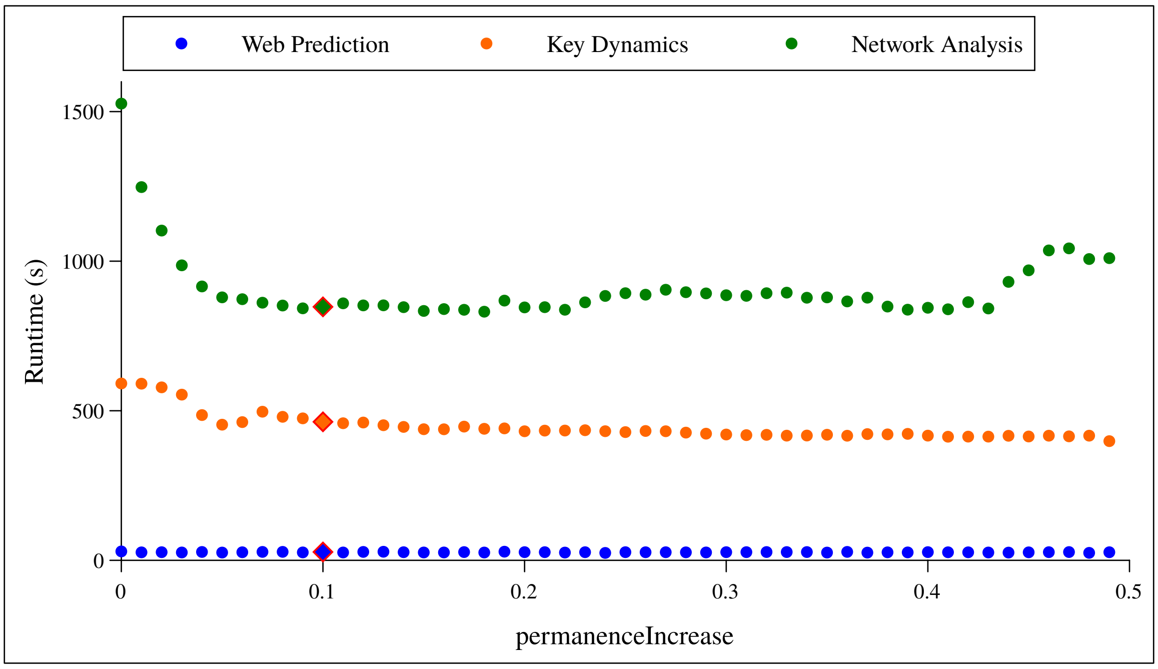


Figure 5.26: Shows the influence the increase of synapse permanence has on the model runtime.

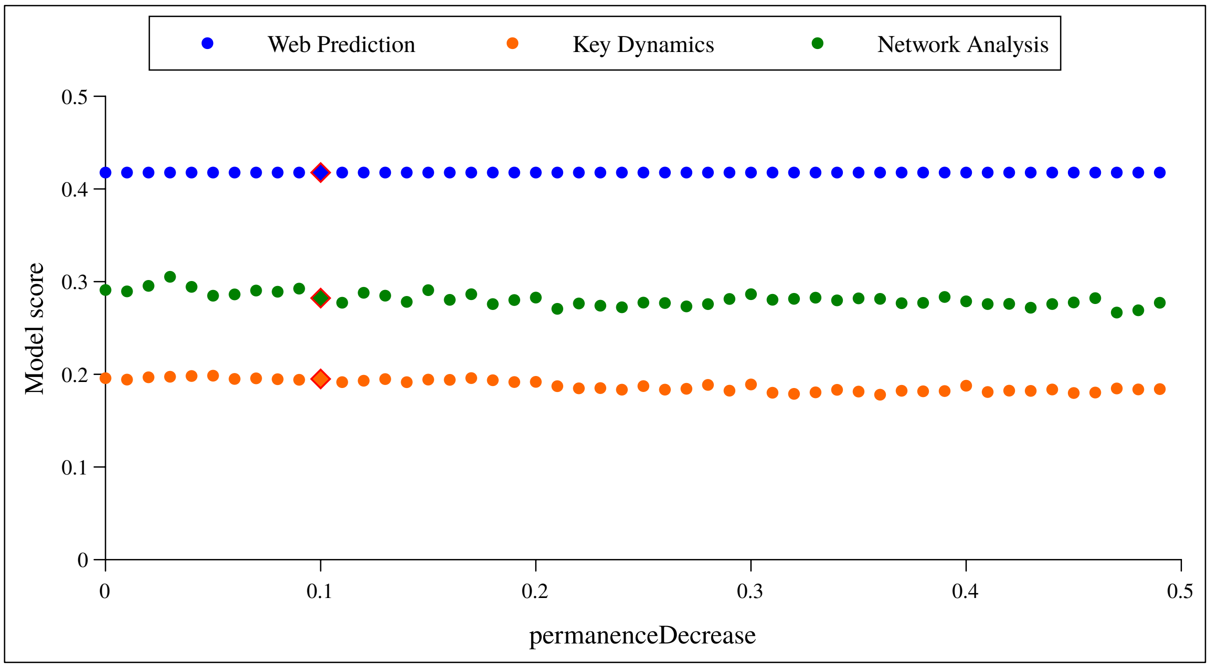


Figure 5.27: Shows the influence the decrease of synapse permanence has on the model score.

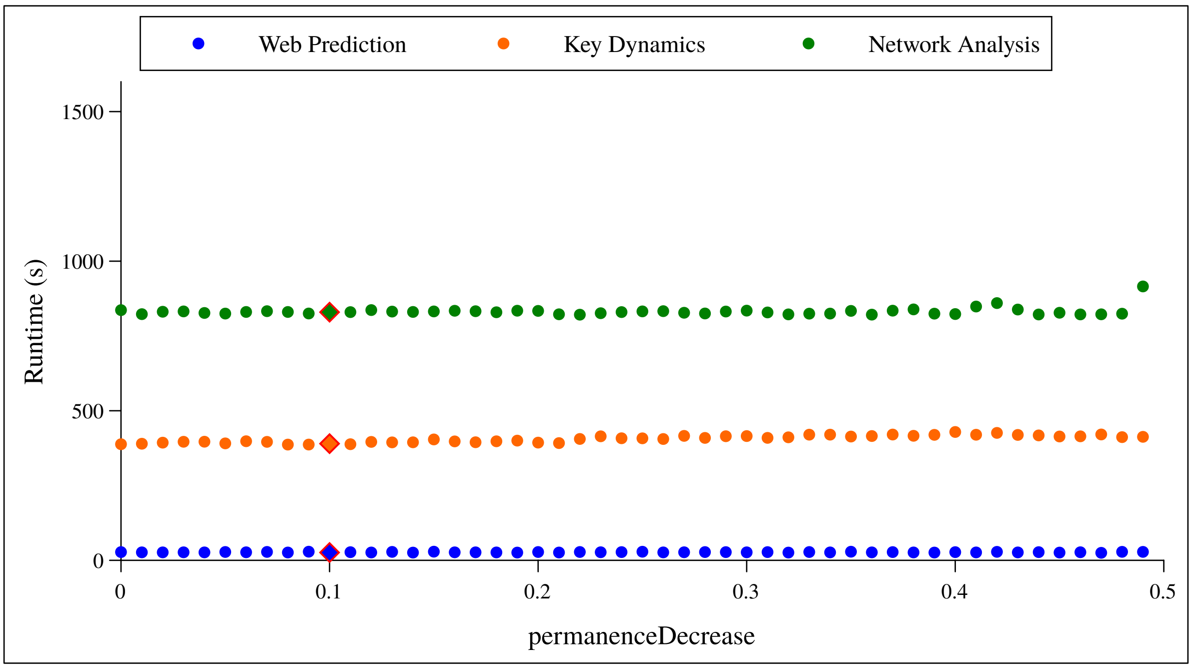


Figure 5.28: Shows the influence the decrease of synapse permanence has on the model runtime.

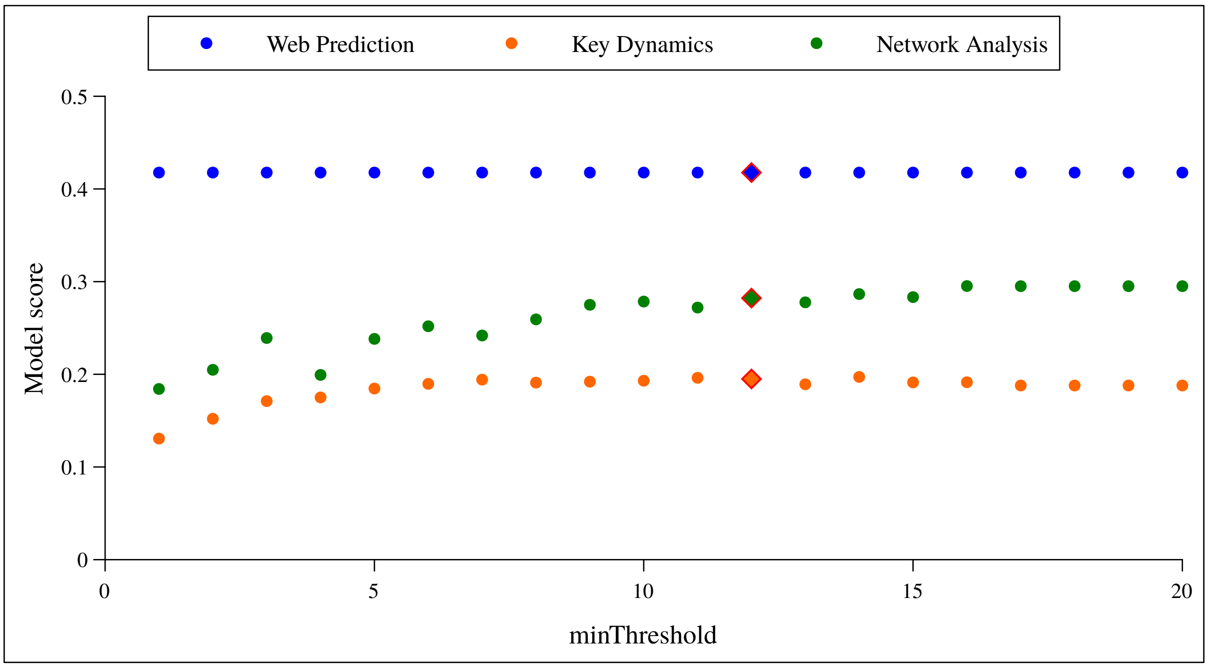


Figure 5.29: Shows the influence the threshold of active segment synapses has on the model score.

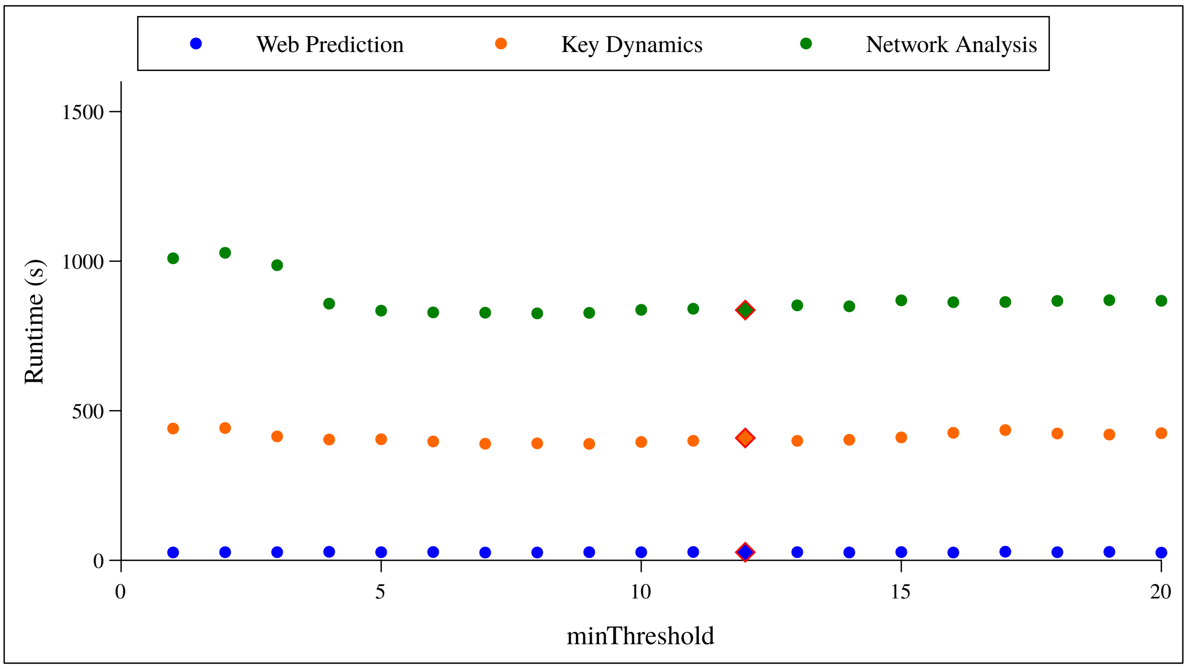


Figure 5.30: Shows the influence the threshold of active segment synapses has on the model runtime.

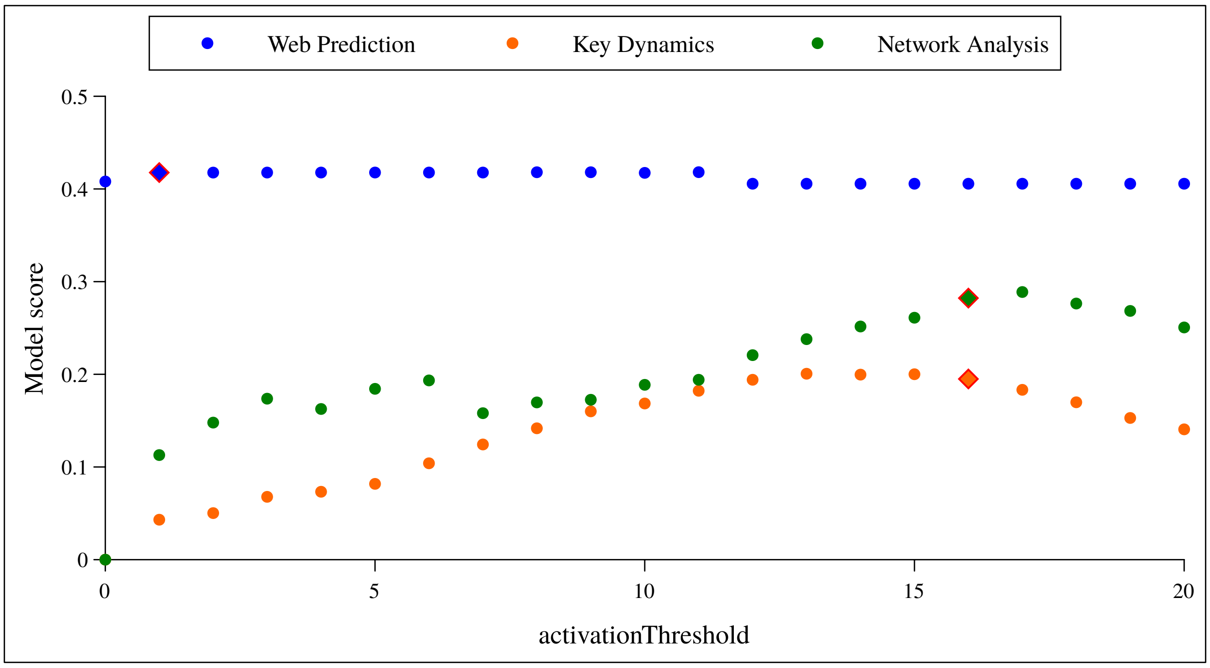


Figure 5.31: Shows the influence the threshold of connected segment synapses has on the model score.

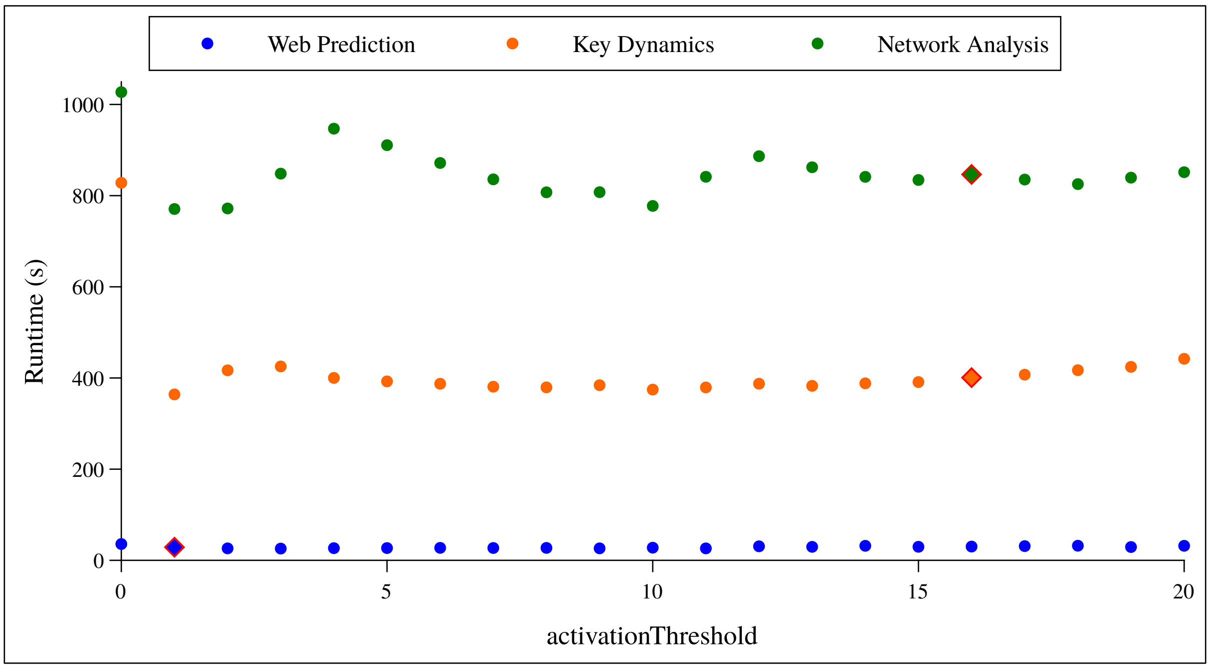


Figure 5.32: Shows the influence the threshold of connected segment synapses has on the model runtime.

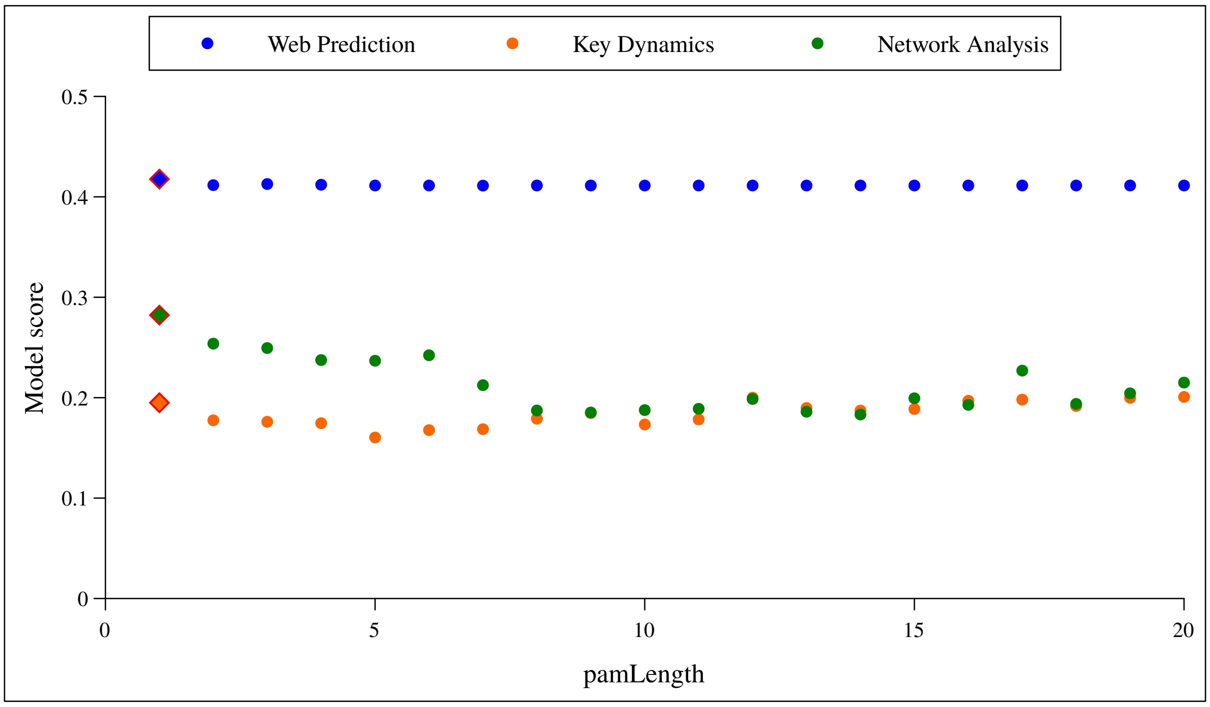


Figure 5.33: Shows the influence the length of newly learned sequences has on the model score.

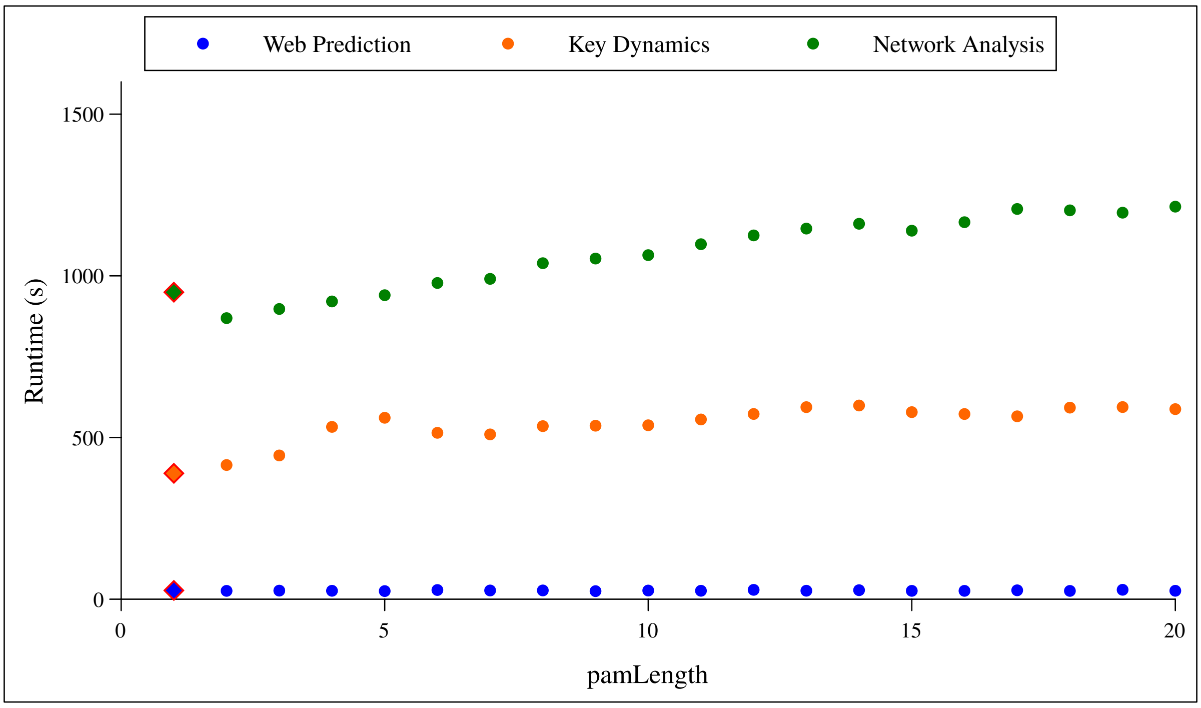


Figure 5.34: Shows the influence the length of newly learned sequences has on the model runtime.

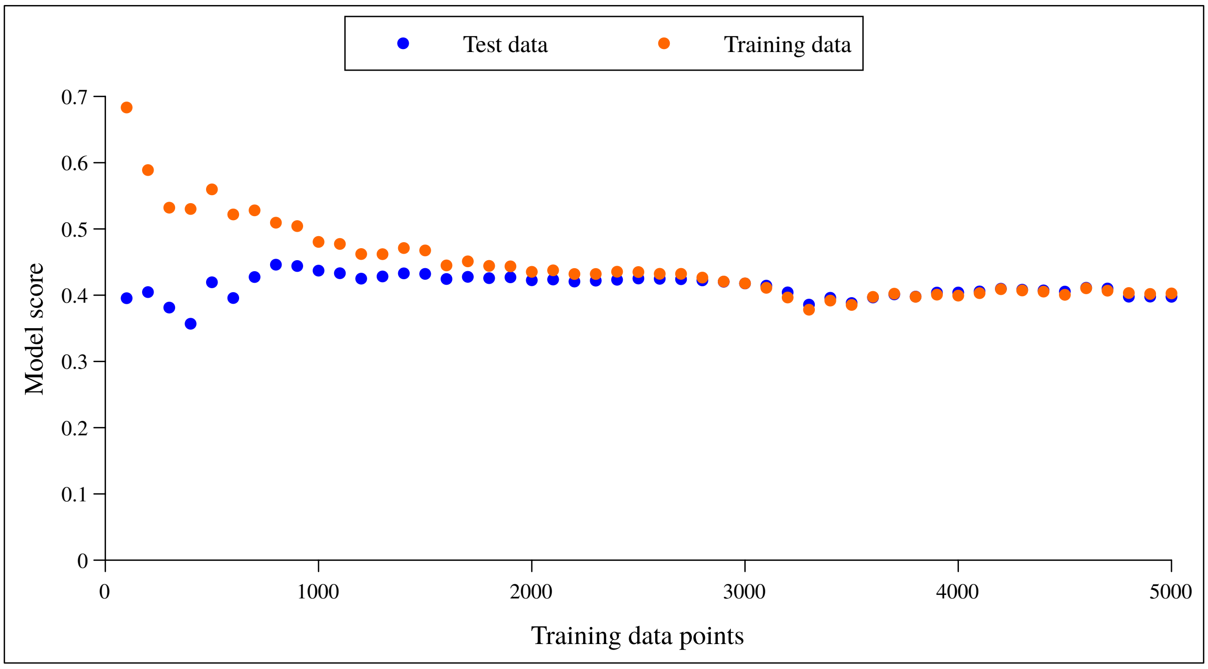


Figure 5.35: Shows the model scores of the web prediction use case for the training and test data sets. A gap between model scores indicates an overfitting problem.

Chapter 6

Discussion

In this chapter, the results shown in chapter 5 are analyzed to answer the research questions posed in chapter 2.1. In addition, the results are complemented by contextual information, such as difficulties obtaining them, where applicable.

6.1 Research Questions

An analysis for each research question is detailed below:

1. *How good are the predictions made by an HTM model?*

   All HTM models used in the three use cases performed a prediction task. However, the predictions for the web analysis and key dynamics use cases were not used directly as a result, but rather to internally calculate an anomaly score. Therefore, the web prediction use case is better suited to answer this question.

   The model score for the web prediction was almost always slightly higher than 0.4, meaning that the corresponding HTM model had a prediction accuracy of about 40%. This is surprisingly high for the irregular and noisy kind of data from the msnbc.com website. In addition, this result could most certainly be improved by adjusting the model parameters with nupic's swarming algorithm.

   Comparing the result to the research literature is somewhat difficult, as the way to measure prediction accuracy is not uniform among papers. For example, using an approach where the top-3 predictions from a markov model were all considered a hit, the accuracy reached about 50% [12, p.24]. If, instead of the model score, the same approach of accuracy measurement was used with the HTM model, an accuracy of 77.8% was reached.

   These results indicate that the predictions made by HTM models are clearly comparable, if not better, than the ones of other state-of-the-art approaches.

2. *How good are the anomaly detection results from an HTM model?*

   The model scores of the key dynamics and network analysis use cases were not as high as in the web prediction case with values in the range of 0.2 to 0.3. However, that does not mean that the anomaly detection capabilities of the HTM models are any worse than the prediction capabilities. Since the model scores for the two anomaly detection cases are calculated by measuring the spread between normal and abnormal test cases, any score greater than 0 indicates a capability to find anomalies.

   In the case of the key dynamics case, the rate of anomalies should be quite high, since the model cannot possibly predict what the user is going to type next most of the time. In addition, even if the user types the same text twice, the model input is never the same due to small differences in the key timings. Despite this, the HTM model was able to detect anomalies in this kind of noisy and irregular data stream with great reliability.

   The models used in the network analysis case often showed a greater spread of model score compared to the key dynamics use case, as it is for example displayed in figure 5.7. The model score of 0.3 is just an average over the individual scores of the models used for each of the ten network users. The spread seen in the result charts stems from the spread of model scores between the different models.

   For some users, the models detected over 90% of network anomalies, whereas the models for other users detected close to 0% of anomalies. Although these low performing models could very likely be improved by adjusting their parameters with the swarming algorithm, their performance is very unlikely to match that of the best models.

   Taking these things into account, the anomaly detection results from an HTM model are dependent on the actual data, but on average they are very good and useful.

3. *How much time does it take to find good HTM model parameters?*

   In all of the three use cases, the default parameters used to create the models created good first results. This took a few minutes at most to adjust things like the encoders, which are unique for each use case. As can be seen in the charts of the previous chapter, the parameters often did not even affect the results in the web prediction case. The parameters do, however, greatly impact the runtime of the models.

   Finding optimal parameters for a given use case, on the other hand, is a very hard problem and takes a lot of time. This is due to the fact that a change in one parameter also changes the model score curves for almost all other parameter ranges. This means that the search space for the optimal parameter combination grows exponentially with every additional parameter.

Nupic provides a swarming algorithm that uses a heuristic approach to approximate the optimal parameters. However, this algorithm takes hours or even days for more complex models to complete.

4. *How prone to overfitting is an HTM model?*

   Measuring overfitting is difficult and often not very accurate. One way is to examine the difference in error rate when using a cross-validation approach. A high spread in error rate indicates an overfitting problem.

   Another approach, as used in this study, is to plot the model performance on the training and test data sets. Usually, a model performs better on the training than on the test data, but this effect should become smaller with more training. If the model perform significantly better on the training data no matter how much the model is trained, then this might indicate an overfitting problem.

   Figure 5.35 shows that the model performs significantly better on the training data set when only little training was performed. However, this difference in model score quickly becomes smaller as the model learns to generalize with more training and after about 3000 training rounds there is hardly any difference in model score. This result might not be valid for all possible parameter configurations, but it indicates that HTM models are not prone to overfitting.

5. *How does the algorithm's performance scale with the amount of data?*

   As can be seen from figure 5.2, the runtime of the HTM models increases linearly with the amount of data. Depending on the complexity of the model used, the gradient of this runtime curve can vary.

   The time required to process a single training data value ranged from 2ms in the web prediction case to 8ms in the network analysis case, which also did not change when more data was being processed. This means that between 100 to 600 values were processed by a single model every second.

6. *How much influence do the model parameters have on the execution time?*

   While some parameters have a large effect on the model's runtime, others have minimal to no effect. Table 6.1 shows the runtime changes caused by each parameter.

   The different influence of each parameter on the model runtime should be considered when searching for an optimal parameter configuration. The parameters with no effect on the runtime can be chosen freely, whereas the trade-off between runtime and prediction accuracy should be considered for all other parameters.

Parameter	Figure reference	Runtime effect
Column count	5.4	linear increase
Active bits (w)	5.6	minimal
Total bits (n)	5.8	linear increase
Inhibition area size	5.10	linear increase
Receptive field size	5.12	no effect or decrease
Synapse connection threshold	5.14	no effect
Increase in synapse strength	5.16	no effect or slight increase
Decrease in synapse strength	5.18	large increase for small values
Number of column cells	5.20	linear increase
Number of new synapses	5.22	increased for values < 16
Synapse starting permanence	5.24	minimal
Synapse permanence increase	5.26	increased for extreme values
Synapse permanence decrease	5.28	no effect
Segment search threshold	5.30	no effect
Segment activation threshold	5.32	large increase for value = 0
Learned sequence length	5.34	linear increase

Table 6.1: Shows the runtime changes caused by different model parameters.

6.2 Challenges

During the course of the study, some challenges arose and had to be worked around. One problem was the acquisition of suitable data for possible use cases. Numerous use cases were discussed prior to the study and were eventually ruled out, because the data could not be gathered within SAP. For example, legal problems prevented the analysis of financial data gathered from SAP customers.

Another problem faced during the study was the state of the analyzed *nupic* framework. The framework was created only recently is should be considered to be in an alpha status, because the implementation changes very rapidly and the documentation is poor at best for most parts. For example, model parameters are often missing documentation on purpose and value ranges, resulting in a try-and-error approach to find suitable values.

It would have been interesting to include the frameworks swarming algorithm in the study and measure its effect on model scores. However, this would have required for the swarming to be executed dozens of times for every use case, with every run taking several hours to days. This was not possible in the time available for the study.

Chapter 7

Conclusion and Future Work

The study described in this thesis investigated the viability of the HTM technology as implemented by the *nupic* framework. To do this, data from three different use cases was collected and analyzed with the help of HTM models. A *model score* metric was defined for every use case to evaluate the results created by the HTM models. In addition, the impact of different model configuration parameters on the model score was measured and plotted.

Prior to the study, a number of research questions were posed. The design of the study and the type of data collected was then adjusted to answer these questions. Where applicable, the results of previous research was included in the answer for comparison purposes.

The results show that the HTM technology is able to provide good predictions and anomaly detection, even without an extensive parameter optimization. The prediction accuracy was comparable to other state-of-the-art approaches like hidden markov models. It was surprising to see how good the HTM models were able to handle noisy and irregular data, for example in the case of network data analysis.

One of the drawbacks discovered in the study was the fact that the *nupic* framework is still in an early stage of development and therefore often hard to use and poorly documented. Due to the frequent development changes, some results from this study might not be valid for future versions of the framework. On the other hand, this means that the framework's performance and usability is very likely increasing with new versions.

This study can be extended in many ways. For example, it would be interesting to explore how the different parameter affect each others performance impact when changed. Although this is very likely a problem with exponential runtime for an optimal solution, it might be possible to find a better approximation algorithm than the swarming approach currently used. Another possible way to extend the presented results would be a direct comparison between different machine learning algorithms. The comparisons done in this study are not very accurate, because the performance measurement and methodology is not the same among scientific papers.

The models used in this study were created with a single cortical region. It would be interesting to extend the models with a hierarchy of regions and evaluate them with the same use case data to see if additional regions make a difference in model score.

List of Abbreviations

ANN	Artificial Neural Network
API	Application Programming Interface
BN	Bayesian Network
CLA	Cortical Learning Algorithm
DBN	Dynamic Bayesian Network
GAI	General Artificial Intelligence
HMM	Hidden Markov Model
HTM	Hierarchical Temporal Memory
IDS	Intrusion Detection System
RNN	Recurrent Neural Network
SDR	Sparse Distributed Representation
SP	Spatial Pooler
TP	Temporal Pooler
URL	Uniform Resource Locator
VM	Virtual Machine

List of Tables

List of Figures

Bibliography

[1] Tsvi Achler and Eyal Amir. “Input feedback networks: Classification and inference based on network structure”. In: *Frontiers in artificial intelligence and applications* 171 (2008), pp. 15–26.

[2] Ion Androutsopoulos et al. “An experimental comparison of naive Bayesian and keyword-based anti-spam filtering with personal e-mail messages”. In: *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*. ACM. 2000, pp. 160–167.

[3] Pierre Baldi et al. “Hidden Markov models of biological primary sequence information”. In: *Proceedings of the National Academy of Sciences* 91.3 (1994), pp. 1059–1063.

[4] Michel Beaudouin-Lafon and Stéphane Conversy. “Auditory illusions for audio feedback”. In: *Conference Companion on Human Factors in Computing Systems*. ACM. 1996, pp. 299–300.

[5] Igor Cadez et al. “Model-Based Clustering and Visualization of Navigation Patterns on a Web Site”. In: *Data Mining and Knowledge Discovery* 7.4 (2003), pp. 399–424.

[6] Jean-Cédric Chappelier, Marco Gori, and Alain Grumbach. “Time in connectionist models”. In: *Sequence Learning*. Springer, 2001, pp. 105–134.

[7] Jonathan Connell and Kenneth Livingston. “Four paths to AI”. In: *Frontiers in artificial intelligence and applications* 171 (2008), pp. 394–398.

[8] Jerome T Connor, R Douglas Martin, and Les E Atlas. “Recurrent neural networks and robust time series prediction”. In: *IEEE Transactions on Neural Networks* 5.2 (1994), pp. 240–254.

[9] Adám B Csapó, Péter Baranyi, and Domonkos Tikk. “Object categorization using vfa-generated nodemaps and hierarchical temporal memories”. In: *IEEE International Conference on Computational Cybernetics, 2007. ICCC 2007.* IEEE. 2007, pp. 257–262.

[10] Hugo De Garis et al. “The China-Brain Project: Building China’s Artificial Brain Using an Evolved Neural Net Module Approach”. In: *Frontiers in artificial intelligence and applications* 171 (2008), pp. 107–121.

[11] Bob Djavan et al. “Novel artificial neural network for early detection of prostate cancer”. In: *Journal of Clinical Oncology* 20.4 (2002), pp. 921–929.

[12] Magdalini Eirinaki and Michalis Vazirgiannis. “Web site personalization based on link analysis and navigational patterns”. In: *ACM Transactions on Internet Technology (TOIT)* 7.4 (2007), p. 21.

[13] Ray J Frank, Neil Davey, and Stephen P Hunt. “Time series prediction and neural networks”. In: *Journal of Intelligent and Robotic Systems* 31.1-3 (2001), pp. 91–103.

[14] Dileep George. “How the brain might work: A hierarchical and temporal model for learning and recognition”. PhD thesis. Stanford University, 2008.

[15] Google. *Google Official Blog*. 2014. URL: http://googleblog.blogspot.de/2014/04/the-latest-chapter-for-self-driving-car.html.

[16] Henry Gray. *Anatomy of the Human Body*. 2000. URL: http://www.bartleby.com/107/.

[17] Biometrics Research Group. *Coursera looks to verify online student identity with photo, keystroke dynamics*. 2013. URL: http://www.biometricupdate.com/201301/coursera-looks-to-verify-online-student-identity-with-photo-keystroke-dynamics.

[18] Matti Hämäläinen et al. “Magnetoencephalography—theory, instrumentation, and applications to noninvasive studies of the working human brain”. In: *Reviews of modern Physics* 65.2 (1993), p. 413.

[19] Simon Hansman and Ray Hunt. “A taxonomy of network and computer attacks”. In: *Computers and Security* 24.1 (2005), pp. 31–43.

[20] Jeff Hawkins and Sandra Blakeslee. *On Intelligence*. Henry Holt and Company, 2007.

[21] David Heckerman, Abe Mamdani, and Michael P Wellman. “Real-world applications of Bayesian networks”. In: *Communications of the ACM* 38.3 (1995), pp. 24–26.

[22] John J Hopfield. “Artificial Neural Networks”. In: *Circuits and Devices Magazine, IEEE* 4.5 (1988), pp. 3–10.

[23] Bill G Horne and C Lee Giles. “An experimental comparison of recurrent neural networks”. In: *Advances in neural information processing systems* (1995), pp. 697–704.

[24] Xia Jiang and Gregory F Cooper. “A Bayesian spatio-temporal method for disease outbreak detection”. In: *Journal of the American Medical Informatics Association* 17.4 (2010), pp. 462–471.

[25] Kevin Killourhy and Roy Maxion. “The Effect of Clock Resolution on Keystroke Dynamics”. In: *Recent Advances in Intrusion Detection*. Ed. by Richard Lippmann, Engin Kirda, and Ari Trachtenberg. Vol. 5230. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2008, pp. 331–350.

[26] K.B. Korb and A.E. Nicholson. *Bayesian Artificial Intelligence*. Chapman & Hall / CRC Computer Science and Data Analysis. CRC Press, 2011.

[27] Aleksandar Lazarevic et al. “A Comparative Study of Anomaly Detection Schemes in Network Intrusion Detection.” In: *Proceedings of the 2003 SIAM International Conference on Data Mining*. 2003. Chap. 3, pp. 25–36.

[28] Steve Lohr. “The age of big data”. In: *New York Times* 11 (2012).

[29] A Bitter Controversy Concerning Whether Humanity Should Build Massively Intelligent Machines. “The Artilect War”. In: *Artificial General Intelligence, 2008: Proceedings of the First AGI Conference*. Vol. 171. IOS Press. 2008, pp. 437–447.

[30] Davide Maltoni. *Pattern recognition by hierarchical temporal memory*. Technical Report. University of Bologna, 2011.

[31] Christian Mannes. “A neural network model of spatio-temporal pattern recognition, recall, and timing”. In: *Neural Networks, 1992. IJCNN., International Joint Conference on*. Vol. 4. IEEE. 1992, pp. 109–114.

[32] James Maxwell, Philippe Pasquier, and Arne Eigenfeldt. “Hierarchical Sequential Memory for Music: A Cognitive Model”. In: *ISMIR*. 2009, pp. 429–434.

[33] Marvin Minsky. *Society of mind*. Simon and Schuster, 1988.

[34] Hans Moravec. *Mind Children: The Future of Robot and Human Intelligence*. Harvard University Press, 1988.

[35] Kevin Patrick Murphy. “Dynamic bayesian networks: representation, inference and learning”. PhD thesis. University of California, 2002.

[36] Ali Nouri and Hooman Nikmehr. “Hierarchical bayesian reservoir memory”. In: *Computer Conference, 2009. CSICC 2009. 14th International CSI*. IEEE. 2009, pp. 582–587.

[37] Numenta. *Hierarchical Temporal Memory*. 2011. URL: http://numenta.org/cla-white-paper.html.

[38] Numenta. *Problems that Fit HTM*. Technical Report. 2006.

[39] George Pallis, Lefteris Angelis, and Athena Vakali. “Validation and interpretation of Web users’ sessions clusters”. In: *Information processing & management* 43.5 (2007), pp. 1348–1367.

[40] Wang Pei. “What Do You Mean by ’AI’?” In: *Artificial General Intelligence, 2008: Proceedings of the First AGI Conference*. Vol. 171. IOS Press. 2008, pp. 362–373.

[41] AJ Perea, JE Meroño, and MJ Aguilera. “Application of Numenta® Hierarchical Temporal Memory for land-use classification”. In: *South African Journal of Science* 105.9-10 (2009), pp. 370–375.

[42] Marc Pickett, Don Miner, and Tim Oates. “Essential phenomena of general intelligence”. In: *Frontiers in artificial intelligence and applications* 171 (2008), pp. 268–274.

[43] David Poole and Alan Mackworth. *Artificial Intelligence: Foundations of Computational Agents*. Artificial Intelligence: Foundations of Computational Agents. Cambridge University Press, 2010.

[44] Ryan William Price. “Hierarchical temporal memory cortical learning algorithm for pattern recognition on multi-core architectures”. PhD thesis. Portland State University, 2011.

[45] Lawrence Rabiner and Biing-Hwang Juang. “An introduction to hidden Markov models”. In: *ASSP Magazine, IEEE* 3.1 (1986), pp. 4–16.

[46] Erik Rehn. “On the Slowness Principle and Learning in Hierarchical Temporal Memory”. PhD thesis. Berlin, Germany: Bernstein Center for Computational Neuroscience, 2013.

[47] Manuel Scheele. *Hierarchical Temporal Memory for Speech Recognition Tasks*. Interim Report. Imperial College London, 2014.

[48] Kristie Seymore, Andrew McCallum, and Roni Rosenfeld. “Learning hidden Markov model structure for information extraction”. In: *AAAI-99 Workshop on Machine Learning for Information Extraction*. 1999, pp. 37–42.

[49] Thad Starner and Alex Pentland. “Real-time american sign language recognition from video using hidden markov models”. In: *Motion-Based Recognition*. Springer, 1997, pp. 227–243.

[50] Matthew E Taylor, Gregory Kuhlmann, and Peter Stone. “Transfer Learning and Intelligence: an Argument and Approach.” In: *Frontiers in artificial intelligence and applications* 171 (2008), pp. 326–337.

[51] Pei Wang, Ben Goertzel, and Stan Franklin. *Artificial General Intelligence, 2008: Proceedings of the First AGI Conference*. Frontiers in artificial intelligence and applications. IOS Press, 2008.

[52] Wikipedia. *Chemical synapse schema*. 2009. URL: http://en.wikipedia.org/wiki/File:Chemical_synapse_schema_cropped.jpg.

[53] Zhenglu Yang, Yitong Wang, and Masaru Kitsuregawa. “An Effective System for Mining Web Log”. In: *Frontiers of WWW Research and Development - APWeb 2006*. Ed. by Xiaofang Zhou et al. Vol. 3841. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2006, pp. 40–52.