
Using Numenta's hierarchical temporal memory to recognize CAPTCHAs

Yensy James Hall

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
yjhall@cmu.edu

Ryan E. Poplin

Center for the Neural Basis of Cognition
Carnegie Mellon University
Pittsburgh, PA 15213
poplinre@cnbc.cmu.edu

Abstract

We attempted to break CAPTCHA letters by using Numenta's Hierarchical Temporal Memory (HTM). CAPTCHAs are an automated test that humans can pass, but current computer programs cannot pass [3]. Numenta's HTM, on the other hand, is a new computing technology that replicates the structure and function of the human neocortex [6]. This makes CAPTCHAs an ideal problem for HTM. We describe experiments and show the accuracy of HTM and Support Vector Machine (SVM) in trying to classify single CAPTCHA letters.

1 Introduction

CAPTCHAs are an example of problems that humans can easily solve yet computers cannot. Numenta Inc, is working on a technology called Hierarchical Temporal Memory (HTM) that replicates the structural and algorithmic properties of the human neocortex and promises to build machines that approach or exceed human level performance for many cognitive tasks [1]. This makes HTM an ideal candidate for breaking CAPTCHAs. In this project, we trained HTM to recognize CAPTCHA characters. This paper describes the experiments we performed, shows the results of HTM classifications, and compares performance against SVM.

2 The HTM learning algorithm

In this section, we are giving a quick overview of HTM's algorithm. For a thorough and very clear explanation, we encourage the reader to review [5].

HTM is a memory system that learns a hierarchical representation of the world by using one algorithm for all types of problems. At the top of this hierarchy, an HTM network forms invariant representations of the world, determined by the sensory data it has been exposed to. These sensory data must constantly flow through time, while the HTM assumes that the causes of these sensory data remains relatively stable. Time is the fundamental component of an HTM, and can be thought of as a learning supervisor.

HTM networks are made of nodes. Each node receives as input a temporal sequence of

patterns. The goal of each node is to group input patterns that are likely to have the same cause, therefore forming invariant representations of extrinsic causes.

An HTM node uses two grouping mechanisms to form invariants. The first one is called spatial pooling, which creates a group of different patterns seen by the node that differ by a configurable maximum distance. This mechanism is similar to a quantization process that maps a potentially infinite number of input patterns to a finite number of quantization centers [5].

The second mechanism is called temporal pooling, which groups together patterns that are temporally close. This way, patterns that are very different, but that have a common cause, can be in the same group.

Both, the spatial and temporal poolers, switch from learning to inference mode at some point. In the case of the spatial pooler, its output is a vector of length equal to the number of patterns pooled by the node, and the i_{th} position in this vector corresponds to the i_{th} pattern inside this spatial pooler. This output is a probability distribution of the similarity between the input pattern and the stored patterns, measured in terms of Euclidean distances. An assumption commonly made by the designers of HTM is that the probability that a pattern is closest to another pattern falls off as a Gaussian function of the Euclidean distance, therefore it can be calculated as proportional to $e^{-\frac{d_i^2}{\sigma^2}}$.

In a node, the outputs of the spatial pooler are the inputs of the temporal pooler. As mentioned before, the temporal pooler forms groups of patterns that are likely to follow each other in time, since it would indicate that they are likely to have the same cause in the world. The designers of HTM used a time adjacency matrix partitioned with a greedy algorithm. This algorithm creates groups by finding the most-connected pattern that is not part of a group and picking the N most-connected patterns to this pattern recursively [5].

For every input from the spatial pooler, the temporal pooler outputs a probability distribution over its groups, propagating the uncertainties up in the hierarchy in a Bayesian Belief Propagation way. The ambiguous information propagated from the bottom of the hierarchy is resolved higher up in the hierarchy.

3 Our approach

We first tried to solve relatively simple problems that helped us gain familiarity with the Numenta Platform for Intelligent Computing (NuPIC). Numenta provides examples of how to create and use HTMs in various scenarios. One of these examples trains an HTM to recognize black and white pictures (one bit per pixel) with different levels of deformations. Another example uses an HTM to classify fruit images (grayscale, 8 bits per pixel) distorted by noise, brightness, blurredness, and occlusion, among others. We adapted these examples to address problems related to the reading of CAPTCHA letters.

To simplify the problem, we focused on classifying single CAPTCHA letters. For an HTM to be able to read CAPTCHAs, it had to be able to read individual characters that are relatively easy to identify by humans. Since humans generalize very well from a few examples, and HTM is an attempt to emulate the method that the human neocortex uses, we generated a synthetic training dataset, very different from the actual CAPTCHA letters that we are trying to classify. At the end, we hoped to achieve higher levels of accuracy than we would with other more main-stream algorithms, like SVM.



Figure 1: Example letters generated for this experiment. The category names for these images, from left to right, are ‘clean’, ‘polynomial distortion’, and ‘radial distortion’.

4 Experiments

Proof of concept

As a proof of concept, the first experiment we performed was the classification of moderately distorted typographic characters. We generated 1500 images that were random moderate distortions of the letters “a”, “b”, and “c”. We trained a four-layer HTM on 30 examples of each letter and achieved 99.8% accuracy when testing on the remaining characters. The distortions used were random second degree polynomial transformations of the pixels. We chose coefficients that were low enough so as to still produce recognizable images. Although the task was extremely simple when compared to actual CAPTCHA letters, this result was a ‘sanity check’ and began to demonstrate the feasibility of this approach.

CAPTCHA letters

Next, we embarked on the classification of individual CAPTCHA letters. The training data we used for this experiment was as follows: For each letter in the English alphabet we generated 120 randomly distorted, rotated, and scaled characters, covering three different fonts. Three types of distortions were used; we called these “slightly distorted,” “polynomial distortion,” and “radial distortion.” Figure 1 shows an example of the different distortions we used. We trained a four-layer HTM on a randomly selected subset of this dataset and tested on the different distortion categories as well as segmented CAPTCHA characters. To obtain the test dataset, we cut out 64 by 64 pixel regions containing a letter from actual CAPTCHA images. To reduce the number of unknowns and decrease the computation time, we converted these new images to grayscale, 8 bits per pixel (fig. 2). In the end, the HTM returns the letter that it believes is the most probable explanation of the input image (26 classes in total).

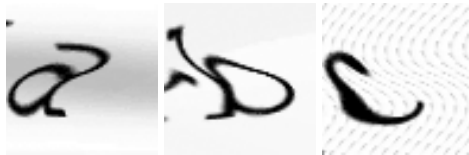


Figure 2: Example of CAPTCHA letters used for testing (a, b, and c).

For the results given in figure 3 (left), we chose a set of specific HTM parameters that remained fixed throughout the experiment, and varied the number of examples of each letter that was given as training data. The first thing to note is that with more training examples the HTM’s performance improves. The second thing to note is that the data in the different distortion categories are not actually that different from each other, as the performance is approximately the same in all three categories.

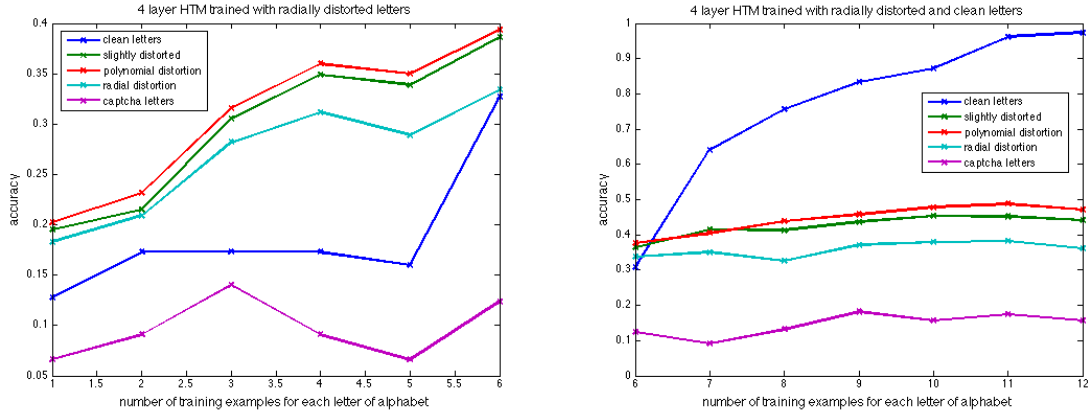


Figure 3: The image on the left shows the results for the first large scale training experiment. The image on the right shows the results for the second large scale training experiment, where we used both radially distorted and clean letters for training. In both experiments we used a four-layer HTM, with $\sigma = 100$, $maxD = 16000$, $topN = 3$, $tM = 3$, $mGS = 32$, random sweeps, and 155000 training iterations. The different colors represent different categories of test data. Note the change in scale between the two graphs.

Adding clean letters

Finally, we ran one more experiment with our four-layer HTM (results given in figure 3, right). Here we used the same idea of successively adding examples of each letter but we changed it so that after training on six radially distorted examples of each letter, we then add clean examples of each letter, one at a time. As would be expected, the performance on the clean letters improved dramatically as we used more clean letters during training. The hypothesis was that this would have a positive effect on the performance in the distorted categories as well; however it turned out not to be that way. The best accuracy we obtained for the CAPTCHA letters was 18%.

To put the performance of the HTM in context, we asked four people to classify all the CAPTCHA characters individually and then we calculated the average accuracy, which turned out to be 89% with a standard deviation of 4%.

Tweaking HTM’s parameters for layer 1

Our next experiments focused on tweaking the parameters of the HTM to improve the performance of the first layer. We created a one-layer HTM and trained it while varying many of the possible parameters. The parameters we varied for this experiment were `maxDistance`, `topNeighbors`, `transitionMemory`, and `maxGroupSize`. These are parameters which control the spatial pooler’s sensitivity as well as the properties of the graph cut algorithm employed by the temporal pooler. The training set used here was the same as our previous experiments, 12 examples per letter (six clean and six radially distorted). Our goal was to find a set of parameters which generate spatial and temporal groups that “make sense” at the lowest level. In other words, the 4x4 pixel images of the quantization centers should be such that these patterns tend to follow each other when our training letters are swept by an HTM sensor. Figure 4 gives an intuitive idea of what good and bad temporal groups are.

Running the complete matrix of possible parameters required a week of solid computation time. Unfortunately there was no quantitative way to evaluate the quality of the quantiza-

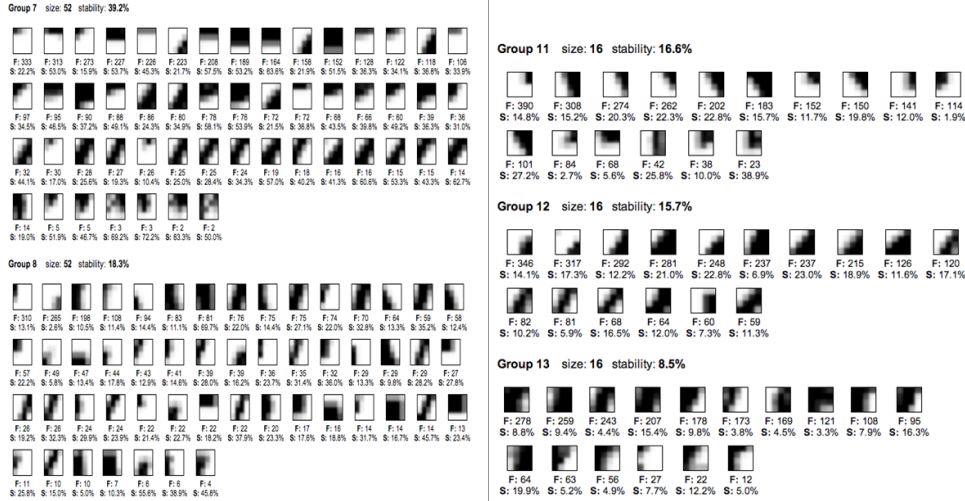


Figure 4: The left half is an example of “bad” groups. These groups contain quantization centers that probably should not go together (parameters: $\sigma = 100$, $maxD = 16000$, $topN = 3$, $tM = 3$, $mGS = 32$). The right half shows “good” groups. They are smaller and contain related patterns like diagonals or corners followed by lines (parameters: $\sigma = 100$, $maxD = 22000$, $topN = 6$, $tM = 6$, $mGS = 16$).

tion centers so we resorted to looking at them as images. After finding a set of parameters yielding a reasonably good set of spatial and temporal groups, we repeated our main experiment from figure 3. Now we achieved 20% accuracy, a 5% improvement over the previous experiment when using 12 examples per letter (six clean and six radially distorted) as training data. The full list of parameters used are given in figure 4.

Using support vector machines

For our next experiment we implemented SVM, using LIBSVM [7], and trained it on exactly the same dataset as in our previous experiment. The value of the slack penalty parameter C did not have an appreciable effect on the test error. SVM achieved 9% accuracy on the test dataset, versus 20% accuracy with the HTM.

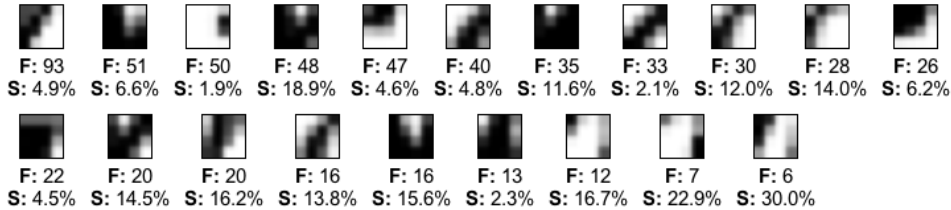
Adding segmented CAPTCHA letters to the training set: HTM vs. SVM

At this point we had no reason to believe that HTM was going to dramatically generalize from the synthetic dataset that we generated. Therefore, we decided to add a single CAPTCHA example of each letter to the training set. These new training data were segmented from the provided CAPTCHAs as described above.

Surprisingly, the performance of HTM went downhill with the inclusion of CAPTCHA letters, yielding 11% accuracy. SVM yielded 14% accuracy under these conditions. One explanation for the reduced accuracy for HTM is that we optimized the layer one quantization centers and temporal groups for the training dataset that didn’t include the CAPTCHA letters. There are some quite distinct differences between the two types of images, most notably the inclusion of background patterns in the CAPTCHA examples, and therefore another round of optimizing layer 1 would probably have been useful. Figure 5 shows an example of the temporal groups that arose from training with six clean, six radially distorted, and a single CAPTCHA of each letter. Qualitatively speaking, it is somewhat worse

than the results on the right side of Figure 4. Once again, there are diagonal lines, corners, and other patterns that do not seem like they should be grouped together. Also notice the distinct inclusion of varying shades of gray.

Group 23 size: 20 stability: 8.7%



Group 24 size: 20 stability: 25.0%

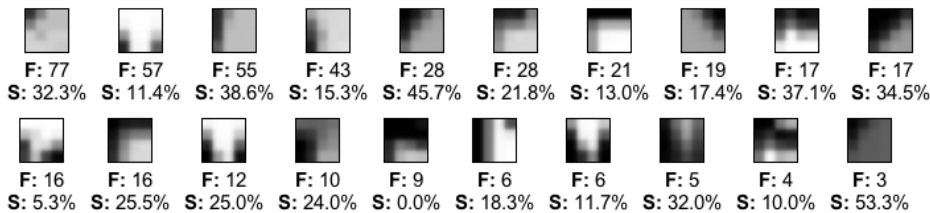


Figure 5: Example layer one temporal groups generated after including a single CAPTCHA of each letter as training data. Unfortunately, they are not as nice as our previously optimized groups. The parameters used here were: $\sigma = 100$, $maxD = 22000$, $topN = 6$, $tM = 6$, $mGS = 20$.

5 Conclusions

Our HTM performed well on this problem but not resoundingly so. There are many things that could have helped the performance of HTM, which we did not try for lack of time. For example, training on a wide spectrum of letters with more severe distortions, or allowing quantization centers to belong to more than one temporal group might help our HTM to generalize over the possible deformations of a given letter. Unfortunately, training an HTM on 312 grayscale, 8 bit, 64x64 letters took us around 20 hours for each combination of parameters tried. This significantly narrowed the width of results we could provide.

Hierarchical Temporal Memory is an elegant idea. We think that more research in this direction will lead to the creation of revolutionary technologies. The challenge of CAPTCHAs turned out to be too difficult given the time constraints and scope of this project. However, we still believe that it is the right problem to be solved with technologies like HTM.

6 Acknowledgments

We would like to thank Joseph Gonzalez, our project advisor, for his support and good ideas.

We also would like to express our gratitude to Dr. Subutai Ahmad, from Numenta, for his advices on fine-tuning HTM.

References

- [1] Hawkins J, George D, “Hierarchical Temporal Memory, Concepts, Theory, and Terminology”
- [2] Hawkins J “On Intelligence”
- [3] Von Ahn L, Blum M, and Langford J, The CAPTCHA Project. <http://www.captcha.net>.
- [4] Chellapilla K, Simard P, “Using Machine Learning to Break Visual Human Interaction Proofs (HIPs)”
- [5] George D, Jaros B, “The HTM Learning Algorithms”
- [6] “Numenta Technology Web Page: <http://www.numenta.com/about-numenta/numenta-technology.php>, 2007”
- [7] “LIBSVM – A Library for Support Vector Machines, <http://www.csie.ntu.edu.tw/~cjlin/libsvm/>, 2007”