



Spoken Digit Recognition using a Hierarchical Temporal Memory

Joost van Doremalen, Lou Boves

Centre for Language and Speech Technology, University of Nijmegen, the Netherlands {j.vandoremalen, l.boves}@let.ru.nl

Abstract

In this paper we explore the feasibility of the Memory-Prediction Theory, implemented in the form of a Hierarchical Temporal Memory (HTM), for automatic speech recognition. Up to now HTMs have almost exclusively been applied to image processing. However, the underlying theory can also be used as an approach to active perception of audio signals. Using the software platform under development by NUMENTA[®] we implemented a system for isolated digit recognition, the speech recognition task that can be most easily cast in a form similar to image recognition. Our results show that the HTM approach holds promises for speech recognition. At the same time it is clear that the present implementation is not ideally suited for processing signals that encode information mainly in dynamic changes.

Index Terms: Memory-Prediction Theory, Hierarchical Temporal Memory, Speech Recognition

1. Introduction

Recently it has been argued that the prediction of future sensory input from salient features of the current input is the keystone of intelligence [1]. Current sensory input patterns activate stored traces of previous inputs that then generate top-down expectations. These expectations are verified against the bottom-up input signals. If the verification succeeds, the predicted pattern is recognized. This theory explains how animals can cope with previously unseen inputs in a latency-free manner that is needed for survival in the real world. Parts of this theory known as the *Memory-Prediction Theory* (MPT) are modelled in the *Hierarchical Temporal Memory* or HTM technology developed by a company called NUMENTA[®] [2].

In this model, spatial and temporal relations between features of the sensory signals are formed in a hierarchical memory architecture during a learning process. Learning can be supervised or unsupervised. When a new pattern arrives, the recognition process can be viewed as choosing the stored representation that best predicts the pattern. HTMs have been successfully applied to the recognition of relatively simple images [2]. The system shows invariance across several transformations and is robust with respect to noisy patterns.

It has long been believed that *Hidden Markov Modelling* (HMM) in automatic speech recognition will fail to reach the performance levels that are needed for a wide range of applications [3]. HMM approaches have problems dealing with the unexpected events that abound in natural speech. Therefore, the generalization capabilities of the model presented in [2] also seem interesting for automatic speech recognition (ASR) systems, because these systems have to be robust against unexpected behaviour of the speakers and background-noise.

In this research we applied the concept of HTM as implemented by NUMENTA[®] to speech recognition. Since this soft-

ware had only been applied to image recognition, we designed a task that resembles image processing as closely as possible. Therefore, we built and tested a system that learned to recognize the 11 words in the TIDIGITS corpus [4].

2. Hierarchical Temporal Memory

An HTM is a collection of linked nodes, organized in a treeshaped hierarchy (cf. Fig.1). HTMs consist of several layers or *levels* of nodes, with one node at the top level. HTMs operate in two stages: the *learning* stage and the *inference* stage. During the learning stage, the network is exposed to training patterns and it builds a model that maps these patterns to the categories. During inference the network will generate a *belief distribution* over these categories for usually unseen test patterns. All of the nodes (except the top node) process information in roughly the same way and consist of two components: a *spatial pooler* and a *temporal pooler*. Understanding an HTM node boils down to understanding the operation of these poolers during both the learning and training stage.

2.1. Operation of nodes during learning

During the learning stage, the spatial pooler learns to map input data to a number of quantization centers or *coincidences*. The output of the spatial pooler (and input to the temporal pooler) is in terms of its coincidences and as such can be seen as a preprocessing step for the temporal pooler, simplifying its input. The temporal pooler learns *temporal groups*, which are groups of coincidences that frequently occur close in time.

2.1.1. Spatial pooler

Spatial poolers of input nodes receive raw data from the sensor, spatial poolers of higher nodes receive the outputs from their child nodes. The input of the spatial pooler in higher layers is the fixed order concatenation of the output of its children. This input is represented by row vectors and the role of the spatial pooler is to build a matrix (the *coincidence matrix*) from input vectors that occur frequently. There are multiple spatial pooler algorithms, i.e. *Gaussian* and *Product*. The Gaussian spatial pooler algorithm is used for nodes at the input layer, whereas the nodes higher up the hierarchy use the Product spatial pooler.

Gaussian spatial pooler The Gaussian spatial pooler algorithm compares the raw input vectors to the existing coincidences in the coincidence matrix. If the Euclidean distance between this input vector and an existing coincidence is small enough, the input is considered to be the same coincidence and the count for that coincidence is incremented and stored in memory. The distance between an input vector \vec{x} and an existing coincidence



 \vec{w} is computed using

$$d^{2}(\vec{x}, \vec{w}) = \sum_{i=1}^{D} (x_{i} - w_{i})^{2}$$
(1)

where D is the dimensionality of the vectors. The threshold for pooling an input vector with an existing coincidence is the parameter MAXDISTANCE. In other words, if $\forall \vec{w} (d^2(\vec{x}, \vec{w}) > MAXDISTANCE)$ the input vector \vec{x} is stored as a new coincidence, otherwise it is pooled with the closest existing coincidence. Thus, the operation of a Gaussian spatial pooler is similar to conventional vector quantization and MAXDISTANCE can be regarded as a vigilance parameter.

Product spatial pooler Because the Product spatial pooler is always part of a node higher up the hierarchy, it receives the concatenation of the outputs of its child nodes. This vector is divided up into N portions, which is the number of children of the node. These portions are belief distributions over the temporal groups formed by the child nodes. The Product spatial pooler sets the highest value in each of these N distributions to 1. The other values are set to 0. These new vectors are stored in the coincidence matrix, and the counts of the coincidences that already exist are incremented.

2.1.2. Temporal pooler

The temporal pooler tries to find groups of coincidences that occur frequently together close in time, so called temporal groups. To this goal, it builds a time-adjacency matrix, from which after learning can be derived how likely certain transitions between coincidences are. When a new input vector is presented during learning, the spatial pooler represents it as one of its learned coincidences *i*. It increments element (j, i) of the time-adjacency matrix with (TRANSITIONMEMORY-t + 1) (when > 0) if coincidence *j* was seen *t* timesteps back. TRANSITIONMEMORY is a parameter that can be varied. After the learning stage, the temporal pooler uses this matrix to create the temporal groups. The following algorithm is used for creating these groups:

- 1. Pick the most frequently seen coincidence *i* and pool it with a temporal group.
- 2. Pick the *N* coincidences *j* on row *i* of the time-adjacency matrix having the highest value and pool them to the

Figure 1: A layer of input nodes and a top node are depicted for a two-layer network. The spatial pooler of Node 1 has learned 4 coincidences. During inference, it communicates a belief distribution over these coincidences to the temporal pooler. The temporal pooler has learned 2 temporal groups. It outputs a belief distribution over these groups using the belief distribution over the coincidences. All the other nodes in this layer are similiar. The outputs of these nodes together form the input to the top node. Its spatial pooler has learned 3 coincidences and outputs a belief distribution to the supervised mapper. The supervised mapper computes a belief distribution over the categories it has seen during training.

same temporal group. N is a parameter called TOP-NEIGHBORS and thus governs how many coincidences are added here each time.

3. Call the coincidences *j i* and repeat step 2 for each of the coincidences *i*.

It is likely that in step 2 coincidences that are most temporally connected are already part of the same temporal group. When no coincidences can be added the process is terminated and repeated until no coincidences are left ungrouped. Additionally, a weight matrix is formed. It has as many rows as temporal groups and as many columns as coincidences. Every element (i, j) represents the row-normalized frequency of coincidence j in group i. This matrix is used during inference.

2.1.3. Training procedure

The nodes are trained from bottom to top. That is, first the bottom nodes are trained on the whole training set, then these nodes are set to *inference mode* (which we will explain below) and the nodes one level up in the hierarchy are trained in a similar way. The top node is trained differently because it has a *supervised mapper* instead of a temporal pooler. For every training pattern, the supervised mapper receives two inputs during learning: the coincidence found by the spatial pooler and the category of the input pattern. It has a mapping matrix, which stores how many times a coincidence *i* belongs to a category *c* by incrementing element (c, i) everytime it receives these inputs together.

2.2. Operation of nodes during inference

After training a node, it is set to inference mode. When the whole network is trained, all nodes are in inference mode and the network is able to perform inference. When an input pattern arrives, it will generate a belief distribution over the categories it has seen during learning.

2.2.1. Spatial pooler

The Gaussian and Product spatial pooler work differently during inference, but they both convert an input vector to a belief vector over coincidences.

Gaussian spatial pooler In the Gaussian spatial pooler algorithm, the distance between an input vector \vec{x} and every one of

the learned coincidences is computed using Eq. 1. This distance is converted into a belief vector by seeing \vec{x} as a random sample drawn from a set of multi-dimensional Gaussian probability distributions all centered on one of the learned coincidences. All these probability distributions have the same variance uniform across all dimensions, the parameter σ , which is the square root of the variance. Each element *i* of the belief vector \vec{y} representing the belief that the input vector \vec{x} was generated from the same cause as coincidence *i*, is computed using

$$y_i = \exp\left\{-\frac{d^2(\vec{x}, W_i)}{2\sigma^2}\right\}$$
(2)

where d^2 is defined in Eq. 1 and W_j is the *j*th coincidence in the coincidence matrix W.

Product spatial pooler The Product spatial pooler algorithm portions the input vector into the outputs of each of its children, takes the dot product with the corresponding portions of coincidence i and then calculates the product of these numbers to give element b_i of the belief vector over all coincidences in the coincidence matrix.

2.2.2. Temporal pooler and Supervised Mapper

The temporal pooler receives a belief vector over coincidences from the spatial pooler. It will then calculate a belief distribution over groups using the weights matrix formed during learning. If we call this weights matrix W and the input vector \vec{y} , then the belief vector \vec{b} is computed using

$$b_i = \sum_j W_{ij} y_j \tag{3}$$

In the top node, the supervised mapper also receives a belief vector over coincidences from the spatial pooler. It calculates a belief distribution over these categories using the mapping matrix formed during learning. First, the mapping matrix is column-normalized. Then, if we call this new matrix C and the input vector \vec{y} , the belief vector \vec{b} is computed using

$$b_i = \sum_j C_{ij} y_j \tag{4}$$

3. Method and Materials

3.1. Test- en Train data

The dataset used in this research is obtained from the TIDIG-ITS corpus [4]. We extracted the utterances of isolated digits sampled at 8kHz. The data was already separated in proper train- and test sets. The train set consists of 2412 utterances of each digit from 1206 male and 1206 female speakers. The test set consists of 1144 utterances, also equally divided into female and male speakers.

3.2. Auditory Preprocessing

We produced auditory feature vectors from the raw signal which we used to train and test the HTM. Fundamentally, HTM is a model of the neocortex. Therefore, we want our feature vectors to have some form of physiological and psychological validity. At the least, they must be tonotopical mappings of the speech signal. That is, sounds which are close to each other in frequency must be represented in topologically neighbouring features. This is necessary to infer spatial relations apparent in the pattern. To produce the auditory feature vectors, we processed the data in the following stages using a MATLAB toolbox (the AUDITORY TOOLBOX [5]):

- ERB-spaced Gammatone Filterbank
- Averaging energy over a time window
- Decimation
- Quantization

Gammatone filter modelling is a physiologically motivated method. It models the cochlea by a bank of gammatone filters. The impulse response of the gammatone filter resembles the impulse response of cochlea filters [6]. We used a gammatone filterbank consisting of 16 filters. The filters are equally placed on the ERB or Equivalent Rectangular Bandwidth scale from 100Hz to 4000Hz. The ERB is a nonlinear rescaling in the frequency domain designed to resemble human frequency selectivity [7]. Furthermore, we averaged energy over an exponential time window after the signal has gone through the filterbank. Next, the signal is decimated to lower the amount of data. To decimate the signal, it is low-pass filtered to maintain the Nyquist criterion and downsampled with a factor 100. Finally, all values in the feature vectors are quantized with 4 bits to lower the amount of data, resulting in values between 0 and 15. Furthermore, we also calculated deltas by simply subtracting consecutive preprocessed feature vectors. In this way we obtained feature vectors with 32 elements.

3.3. HTM Design and Implementation

We used NUPIC, an API for implementing HTMs, developed by NUMENTA[®] [2], to implement our HTM network. To implement an HTM two steps have to be taken: creating the architecture and training it with a set of training patterns. After we created an architecture and trained the network on the TIDIG-ITS train set, we tested the HTM with the test set.

Architecture Our HTM consists of 3 levels. The input level consists of 16 nodes, each receiving a feature and the corresponding delta. Level 2 consists of 4 nodes, each receiving the output of 4 input level child nodes. Level 3 consists of one top level node.

Training During the training stage we fed the feature vectors of all training utterances to the HTM, separated by a zero vector. The HTM uses this zero vector to detect the end of an utterance and resets the history of the temporal pooler. In addition to the feature vector, we also fed the category label of the utterance to the supervised mapper of the top level node.

Testing After training, the feature vectors of all the test utterances were fed to the HTM, but without the category label of the utterance. The HTM output consists of belief distributions (or belief vectors) over the 11 categories for every feature vector calculated from every test utterance. We calculated the 'winning digit' by normalizing every belief vector of the utterance so that its elements sum to one, adding the logs of these vectors and taking the element index with the largest value of the resulting vector.

4. Results

We investigated the effect of the parameters MAXDISTANCE and SIGMA on the Word Error Rate (WER) and the average number of coincidences and temporal groups learned in the bottom level nodes. The other parameters (TRANSITIONMEM-ORY and TOPNEIGHBOURS) were set to 5 and 1 respectively.



Figure 2: Inproducts of the mean belief vectors of /six/ and the other digits for timepoints 1 to 10.

MAXDISTANCE	SIGMA	WER(%)	#coincs	#groups
1	1.00	10.00	50.00	25.00
3	1.73	8.57^{*}	34.31	20.00
6	2.45	15.47	17.94	11.88
9	3.00	18.86	12.20	7.45

Table 1: WER and average number of coincidences and temporal groups learned in the 16 bottom nodes for different values of MAXDISTANCE and SIGMA.

These are default values and other values had a negative effect on the performance of the system. We varied across different values for MAXDISTANCE and set SIGMA to the square root of MAXDISTANCE. This is a reasonable starting value for SIGMA, because distances between coincidences are calculated as the squared Euclidean distance instead of the standard Euclidean distance. Of course, these values for SIGMA could be optimized. The results are in Table 1. The lowest WER (8.57%) was obtained with an intermediate value for MAXDISTANCE of 3. This might indicate that with a lower value of MAXDISTANCE, the HTM will see variations in input patterns due to noise as different coincidences. On the other side, when MAXDISTANCE is higher than the optimal value, the spatial pooler will pool together patterns that have different causes. The confusion matrix for this best performing system is in Table 2.

If we denote the unit length mean belief vector for the known category c label (0-11, where 0 corresponds to /oh/, 1 to /one/ etc. and 11 to /zero/) at time t as \vec{b}_c^t (derived from the inference results on the test set), we can define an 11×11 matrix Φ^t with elements $\Phi_{ij}^t = \vec{b}_i^t \cdot \vec{b}_j^t$. We would expect these inproducts to be high at points in time t where similar phones are uttered. For example, we would expect the inproducts of the mean belief vectors of /six/ and /seven/, both beginning with /s/, to be high at beginning time points b. In other words, Φ_{67}^b will be high. Looking at Figure 2 this seems to be the case. The overlap of /six/ and /seven/ is relatively high in the beginning of the utterance when, approximately, /s/ is uttered, but later becomes less and less apparent. Furthermore, the overlap with a digit starting with a similar phone like /th/ in /three/ also overlaps with /six/ when /s/ is spoken.

	oh	one	two	three	four	five	six	seven	eight	nine	zero
oh	.90	.01	.00	.00	.03	.01	.00	.02	.00	.01	.02
one	.00	.92	.00	.00	.07	.00	.00	.00	.00	.01	.00
two	.00	.00	.93	.01	.00	.00	.01	.00	.00	.00	.04
three	.01	.00	.10	.90	.00	.00	.00	.00	.00	.00	.00
four	.04	.03	.00	.00	.93	.00	.00	.00	.00	.00	.00
five	.03	.02	.00	.00	.00	.89	.00	.00	.00	.07	.00
six	.00	.00	.03	.00	.00	.00	.90	.03	.00	.00	.03
seven	.00	.03	.00	.00	.01	.02	.00	.89	.00	.02	.05
eight	.01	.01	.02	.00	.00	.00	.00	.00	.94	.00	.01
nine	.02	.06	.00	.00	.00	.02	.00	.00	.00	.89	.01
zero	.00	.01	.00	.00	.00	.00	.01	.00	.00	.01	.97

Table 2: Confusion matrix of the best performing system.

5. Discussion and Conclusion

Our results show that the HTM approach holds promises for speech recognition. The system we developed was previously only applied to image recognition, but with a few minor changes can be applied to a simple speech recognition task. This system has reasonable results and behaviour. Furthermore, it could be further optimized in several ways with respect to the the input representation, HTM parameters and architecture.

At the same time it is clear that the present implementation is not ideally suited for processing signals that encode information mainly in dynamic changes. Some design choices made in the development of the algorithms could be suboptimal. In the learning stage, these could be the algorithms for learning coincidences and learning (unordered) temporal groups. During inference, it is likely previously seen input data should be taken into account in the form of top-down feedback. In the MPT top-down feedback and previous sensory input is used to generate predictions. This type of feedback is not yet implemented, while these predictions could be crucial for succesfully learning and recognizing temporal patterns.

6. References

- [1] J. Hawkins & S. Blakeslee, *On Intelligence*. New York: Henry Holt, 2004.
- [2] D. George & B. Jaros, *The HTM Learning Algorithms*, Numenta, 2007.
- [3] R.K. Moore, "A comparison of the data requirements of automatic speech recognition systems and human listeners," in *Proceedings of the 8th European Conference* on Speech Communication and Technology, Eurospeech 2003: Genève, pp 2582-2584, 2003.
- [4] R.G. Leonard, "A database for speaker-independent digit recognition," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 3, pp 42114214, 1984.
- [5] Malcolm Slaney, Auditory Toolbox: A MATLAB toolbox for sound, 1994, Apple Technical Report #45.
- [6] E. de Boer and de H.R. Jongh, "On cochlear encoding: potentialities and limitations of the reverse-correlation technique," *JASA*, vol. 63, pp. 115-135, 1978.
- [7] B.C.J. Moore, R.W. Peters and B.R. Glasberg, "Auditory filter shapes at low center frequencies," *JASA*, vol. 88, pp. 132-140, 1990