Augmented Spatial Pooling

Author

Thornton, John, Srbic, Andrew, Main, Linda, Chitsaz, Mahsa

Published

2011

Journal Title

Lecture Notes in Computer science

DOI

10.1007/978-3-642-25832-9_27

Augmented Spatial Pooling

John Thornton, Andrew Srbic, Linda Main, and Mahsa Chitsaz

Institute for Integrated and Intelligent Systems, Griffith University, QLD, Australia
j.thornton@griffith.edu.au
{andrew.srbic,linda.main,mahsa.chitsaz}@griffithuni.edu.au

Abstract. It is a widely held view in contemporary computational neuroscience that the brain responds to sensory input by producing sparse distributed representations. In this paper we investigate a brain-inspired spatial pooling algorithm that produces such sparse distributed representations by modelling the formation of proximal dendrites associated with neocortical minicolumns. In this approach, distributed representations are formed out of a competitive process of inter-column inhibition and subsequent learning. Specifically, we evaluate the performance of a recently proposed binary spatial pooling algorithm on a well-known benchmark of greyscale natural images. Our main contribution is to augment the algorithm to handle greyscale images, and to produce better quality encodings of binary images. We also show that the augmented algorithm produces superior population and lifetime kurtosis measures in comparison to a number of other well-known coding schemes.

1 Introduction

Advances in computational neuroscience over the last twenty years have produced increasingly realistic and viable models of the functioning of the mammalian neocortex. These advances provide a compelling evidence-based picture of the kinds of physical processes and structures that underpin natural intelligence – a picture that suggests various computational realisations. In the current paper, we investigate a computational model of the neocortex proposed by Jeff Hawkins, known as *hierarchical temporal memory* (HTM) [3]. Hawkins first published his ideas in 2004, but only recently developed a practical computational description of its low-level functioning [8]. Our task is to evaluate the spatial pooling component of this algorithm in terms of its ability to robustly and efficiently encode Willmore and Tolhurst's well-known benchmark of greyscale natural scene images [11].

Research suggests that the neocortex uses a *sparse coding* strategy to represent information within a hierarchal structure of layers [9]. A sparse code is one where a relatively small proportion of code elements are active at any one time. If we take the cortical column to be the basic unit of neocortical activation [7], this implies that only a small proportion of columns connected to a given input will be active when the input is present. In addition, for a code to be representationally useful, differing inputs must activate different subsets of columns. This can

be achieved by maximising the *statistical independence* of the generated codes [4]. The advantage of sparse coding over local coding is that sparse codes have greater representational capacity while still being able to encode simultaneous inputs without interference.

The HTM model extends existing work by explicitly handling temporal sequences of input within a hierarchical Bayesian framework [2]. This is achieved by localised collections of cortical minicolumns learning to predict sequences of feed-forward input arriving either from sensory receptors or from other regions of the neocortex, and having the entire hierarchy learn and exchange inferences about temporal sequences (i.e. events) rather than spatial patterns. It is this temporal *predictive* function of groups of minicolumns that sets the HTM model apart from other hierarchical Bayesian approaches (e.g. [1, 6]).

However, it is only recently that the computational details of the HTM model have explicitly incorporated a sparse coding strategy into the *spatial pooler* component of the architecture [8]. To date there has been no published evidence evaluating the performance of the new spatial pooler, or of the new cortical column architecture. To address this, we implemented the HTM spatial pooler and evaluated it on a set of static image benchmarks. In the remainder of the paper we provide a description of this implementation and explain the principles upon which it works. We then introduce a number of modifications that were necessary to make the pooler operate efficiently on our benchmark problems and present an empirical study comparing the HTM spatial pooler with our modified pooler and with the various techniques presented in Willmore and Tolhurst's paper on characterising the sparseness of neural codes [11]. As part of this empirical study we investigate a range of measures to capture the important dimensions of the spatial pooler's behaviour.

2 Spatial Pooling

Basic Principles: The latest HTM architecture [8] introduces a more sophisticated and biologically plausible neural model than is typically employed in artificial neural network research. This model is structured as a hierarchy of regions, where each region consists of a set of columns and each column consists of a set of neurons and their associated dendrites and synapses. According to HTM theory, these neurons control which columns in a region are currently active, and which are currently *predicting* they will be active. The first function is determined by a procedure known as *spatial pooling* and the second by a procedure known as *temporal pooling.*

The basic task of the spatial pooler is to form a sparse distributed representation of the input. This is required by the temporal pooler in order to learn and predict the sequential order of particular input streams. However, to be biologically plausible as well as practically useful, the spatial pooler must also be able to *efficiently* form a relatively *stable* representation of a *continuous* stream of input. These requirements rule out existing solutions, such as independent

components analysis [4], as these lack the flexibility and efficiency to adjust to online data streams.

As the internal structure of an HTM column and its associated neurons is only relevant to the implementation of temporal pooling, we shall not discuss these details further. To understand spatial pooling, we need only consider a column as a unified entity with an associated set of proximal dendrites that synapse directly with the input (see [8]). These synapses are *not* associated with weights that multiplicatively determine the strength of the signal. Instead, each dendrite is associated with a *potential* synapse and each synapse is associated with a *permanence value.* If the permanence value of a synapse passes a certain threshold then the synapse is connected and the dendrite will transmit the input to which it is connected, otherwise the synapse remains potential and inactive.

To justify the use of potential synapses, Hawkins argues that the traditional artificial neural network approach of learning by adjusting the strength or weight of individual synapses is not biologically realistic. He acknowledges that synapses have differing strengths, but argues that the synaptic release of neurotransmitters is too stochastic to explain the fine distinctions that are made between differing inputs. Instead, he points to recent research that shows how synapses can rapidly form and un-form [10] and argues that this provides a better mechanism for synaptic learning.

A second important aspect of the operation of the spatial pooler is the use of inhibition between columns to produce sparse distributed representations. It is this feature that produces the *self-organising* capacity of the system to adjust itself to the structure of the input data. As with Kohonen's self-organising maps [5], the spatial pooler performs learning on the basis of how well the synapses from a particular column match (or overlap) the input to which the synapses are connected. However, instead of altering the relative weights of the synapses of neighbouring columns, a strongly activated column will compete with and *inhibit* its less active neighbours [10]. At the end of this process, only the potential synapses belonging to the winning columns that best represent the current input will be able to learn. Here learning entails increasing the permanence values of potential synapses that are connected to active input and decreasing the permanence values of those connected to inactive input. This implements the forming and un-forming of synaptic connections discussed above.

Binary Spatial Pooling: The basic functioning of spatial pooling is described in [8]. Here we only provide a brief outline of the algorithm and explain those areas which deviate from or extend the original proposal. Firstly, each HTM column has a set of potential synapses that are randomly connected with probability $P(connect)$ to each input coordinate. Every potential synapse s is then initialised with a randomly generated permanence value $perm(s)$ bounded within a small range of a threshold $permThreshold$, such that the probability of connection varies inversely and linearly with the distance of the column from the input coordinate. Potential synapses with a permanence value greater than $permThreshold$ are now defined as *connected*.

Algorithm 1 performLearning($columns$)

for each potential synapse s in each active column c **do**
  if s has active input **then** $perm(s) = min(perm(s) + pInc, 1)$
  else $perm(s) = max(perm(s) - pDec, 0)$
end for
for each c in $columns$ **do**
  if $activity(c) < minActivity(c)$ **then** $boost(c) = boost(c) + bInc$
  else $boost(c) = 1$
  if $overlapSum(c) < minActivity(c)$ **then**
    for each potential synapse s in c **do** $perm(s) = min(perm(s) \times pMult, 1)$
  end if
end for

The system then calculates the activity level of each column's response to each *image*. For binary images, this column activity (or overlap) is a simple count of the number of connected synapses that are receiving active input. However, each column's overlap ($overlap(c)$) must also exceed a $minOverlap$ threshold, in which case $overlap(c)$ is multiplicatively boosted by a factor $boost(c)$ (determined by the learning procedure), otherwise it is set to zero. A column c then becomes active if it is one of the n most active columns within the $meanInhibitionArea$ of c, where $meanInhibitionArea$ is the mean size of the receptive fields of all columns and n is set by the parameter $desiredActivity$.

Algorithm 1 ($performLearning$) implements the basic learning strategy. Firstly, the permanence values of all synapses belonging to active columns are adjusted by incrementing those connected to active input and decrementing those connected to inactive input. Then two strategies are used to increase the activity of insufficiently active columns. This first involves counting how often a column c has been active over the last i iterations ($activity(c)$) and how often it has exceeded the $minOverlap$ threshold ($overlapSum(c)$). These values are compared with $minActivity(c)$, calculated by $maxActivity \times minActivityThreshold$, where $maxActivity$ is the maximum activity of any column falling within the $meanInhibitionArea$ of c and $minActivityThreshold$ is a user defined parameter. If column c's $activity(c)$ falls below $minActivity(c)$ then $boost(c)$ is incremented by $bInc$ and if $overlapSum(c)$ falls below $minActivity(c)$ then the permanence values of all c's potential synapses are increased by a factor of $pMult$. The first strategy ensures all columns maintain a minimum level of activity and the second ensures they maintain a minimum level of synapse connectivity.

Finally, the system converges when no changes have been made to the permanence value of any synapse since the last iteration through the entire set of images. The end result is a sparse, distributed encoding of each image presented to the pooler, comprising of the set of columns that are active when an image is present. The sparse distributed nature of the encoding is produced by the self-organising interaction of inhibition, which focuses activity on a small subset of columns (sparsifying), and learning, which ensures all columns become at least minimally active (distributing).

3 Modifications

Handling Greyscale Images: The HTM specifications only handle binary input. Hence we term the original algorithm *binary spatial pooling* (BSP). Our first extension was to redefine the notion of overlap so that synapse inputs can take on integer values. To achieve this, a column's overlap becomes the sum of the integer input values at each connected synapse rather than a simple count of active bits. The main alteration occurs in the updating of the permanence values of potential synapses of active columns (compare lines 1–4 of Algorithm 1 with lines 1–9 of Algorithm 2). Previously a permanence value was incremented whenever a potential synapse is associated with an active input. Now we redefine the notion of an active input to be an input that is greater than the mean activation level of the current image. In addition, *minOverlap* is adjusted by being multiplied by the mean value of all non-zero pixels in the current image. This preserves the original value of *minOverlap* for binary images (as the mean value of non-zero binary pixels is one) while ensuring that (on average) for each column at least *minOverlap* synapses are connected to active (above mean) non-binary inputs.

Accelerating Convergence: The convergence behaviour of the pooler can be accelerated by switching off the basic learning function in lines 1–4 of Algorithm 1 at the point where the two boosting strategies become inactive. This is achieved by keeping count of the number of columns that are either boosted or have their potential synapses incremented during a single iteration through the entire set of images (or over a sufficiently long period of time). If this count is zero then all columns will have attained a sufficient level of activation over the entire data set and there is no further need to adjust the synapse connections. The advantage of this approach is that the pooler can have its main learning function suspended and yet still remain responsive to new input, i.e. if new input cannot be represented by the existing pattern of synapses, some form of boosting will occur and learning will be resumed. The system is still not considered to have finally converged until a complete iteration through all images has occurred such that the permanence values of all synapses remain the same at the end of the iteration as they were at the start.

Augmented Spatial Pooling: The main contribution of the paper, aside from evaluating the current HTM spatial pooler, is the development of a more robust learning strategy. This strategy was suggested by observing that the existing boosting strategy often fails to sufficiently alter the pattern of connected synapses: although boosting succeeds in elevating an inactive column into activity, because the boost value is then immediately reset to one, the column does not remain active long enough for any of its currently inactive synapses to become connected. If no new connections are made in the first iteration of activity, the column can immediately fall into inactivity and again have to wait for its boost value to increment to a point where it becomes active. If a large

Algorithm 2 performAugmentedLearning($image, columns$)

for each potential synapse s in each active column c **do**
  if $input(s) > meanInput(image)$ and $perm(s) >= connectThreshold$ **then**
    $perm(s) = min(perm(s) + pInc, 1)$
  else if $input(s) > meanInput(image)$ and $perm(s) < connectThreshold$ **then**
    $perm(s) = min(perm(s) + pInc, connectThreshold - pInc)$
  else
    $perm(s) = max(perm(s) - pDec, 0)$
  end if
end for
for each c in $columns$ **do**
  if $activity(c) < minActivity(c)$ and $(boost(c) = boost(c) + bInc) > bMax$ **then**
    $boost(c) = 1$
    for each disconnected synapse s in c in ascending distance order from c **do**
      if $perm(s) > maxPerm$ **then** $maxPerm = perm(s)$ and $maxS = s$
    end for
    $perm(maxS) = connectThreshold + pInc$
  end if
end for

number of columns are in this position, then an escalating boosting competition can occur where, although each column is slowly gaining new connections, so are its competitors, meaning none remain active long enough to form stable representations. The end result is that the pooler can fail to converge, especially on complex (high entropy) images.

Algorithm 2 details the augmented learning procedure. Here the updating of active column synapses is altered so that disconnected synapses only have their permanence values incremented to a point just below *connectThreshold* (lines 4–5). Now, the only place where synapses can become connected is in the boosting procedure (line 16). As before, if a column's activity is below the $minActivity(c)$ threshold its boost value is increased (line 11). However, if $boost(c)$ exceeds a $bMax$ threshold then the closest synapse to c ($maxS$) is selected from the set of *disconnected* synapses with the greatest permanence value ($maxPerm$) and this synapse has its permanence value set so that it is connected. In this way the connection of synapses is controlled entirely within the boosting procedure and the earlier ineffective escalating boosting behaviour is remedied.

4 Experimental Evaluation and Discussion

In order to evaluate the HTM spatial pooler we used the 64 greyscale images from Willmore and Tolhurst's influential study on measures of sparsity [11], and similarly generated ten sets of $10,000$ 16×16 image patches selected randomly from the 64 full images. In the original paper, sparseness was characterised according to two statistics: population kurtosis and lifetime kurtosis. Population sparseness is defined as the average kurtosis of the distribution of the activities

of the complete set of N columns for each *image* in the set of input images. The population kurtosis of a single image i is given by:

$$populationKurtosis_i = \left\{ \frac{1}{N} \sum_{c=1}^{N} \left[\frac{a_c - \bar{a}}{\sigma_a} \right]^4 \right\} - 3 \quad (1)$$

where $a_1 \dots a_N$ are the post-inhibition overlap activities of columns $1 \dots N$ for image i, and $\bar{a}$ and σ_a are the mean and standard deviation of these activities. To allow for comparison between methods, and again following [11], we standardised the activities to have a mean of zero and a standard deviation of one, giving an averaged population kurtosis of $\frac{1}{M} \sum_{i=1}^{M} populationKurtosis_i$ for an entire set of M images. This statistic measures the infrequency or *sparseness* of collective column activity in response to individual images but fails to measure how the responses are *distributed* between columns, i.e. how infrequently individual columns become active. To capture this second dimension, we need the average *lifetime* kurtosis of the columns, which measures the averaged kurtosis of each column's responses to an entire set of input images. Lifetime kurtosis is defined in the same way as population kurtosis except that $a_1 \dots a_N$ are now the activities of column c in response to the entire set of N images and the average kurtosis is taken over $i = 1 \dots M$ columns.

Comparisons with Greyscale Images: Given these measures we can now compare the sparseness and distribution of the spatial pooler representations with the results reported in [11] (see Table 1). Here, following [11], we generated a spatial pooler column for each image pixel (256 in all), and then set the pooler parameters as follows: $P(connect) = 0.15$, $connectThreshold = 0.2$, $pInc = pDec = 0.02$, $bInc = 0.005$, $bMax = 4$, $minActivityThrehold = 0.01$, $desiredActivity = 0.05 \times meanInhibitionArea$, and decay $d = 100$. In addition, $minOverlap$ was dynamically set to be the product of the mean pixel intensity of the current image and the mean number of connected synapses for an individual column. These values proved fairly robust for the augmented spatial pooler (ASP) and were subsequently used as ASP defaults.

In contrast, despite extensive parameter tuning, the original binary spatial pooler (BSP) was unable to converge to a stable representation on any of the $10 \times 10,000$ greyscale image sets (after allowing 500 cycles through each image set). This reflects that BSP was not developed to process greyscale images. If we binarise the input by setting each pixel with an intensity greater than the mean intensity for a given image to one and all others to zero, then BSP can successfully converge. However, as the Willmore study was concerned with greyscale coding schemes, we cannot fairly compare BSP with the other coding schemes, and so we only report statistics for ASP in Table 1.

Overall, the results show that ASP significantly outperforms all the coding schemes considered in [11], having a lifetime kurtosis on the raw images 4.6 times greater than the best alternative (ICA) and 3.9 times greater than Gabor on the whitened images. The population kurtosis improvements were less pronounced on

the raw images (but still 1.5 times greater than PCA) but even more pronounced on the whitened images (8.31 times greater than Gabor).

ASP was also able to *efficiently* converge on stable representations, requiring, on average, 13.62 cycles through each of the ten raw data sets (where each cycle processes all $10,000$ images in a set) and 10.67 cycles through the whitened data. This took an average 8.01 seconds per convergence on the raw data and 6.98 seconds on the whitened data (all ASP and BSP experiments were run on an Apple MacBook Pro 2.93 GHz Intel Core 2 Duo processor with 4 GB of 1067 MHz DDR3 RAM and running Mac OS X version 10.6.7).

Comparisons with Binary Images: As the original spatial pooling algorithm (BSP) was unable to converge on the greyscale images, we ran a separate experiment using the same set of natural image patches but after performing a binary conversion. BSP still found these binary images challenging in comparison to simpler binary encodings and was unable to converge using ASP's default parameter settings. We found that BSP will only converge on these images if the effect of decrementing the permanence value of a synapse is much stronger than the effect of an increment, making it easier for a synapse to become disconnected than for it to become connected. ASP achieves a similar effect by not incrementing a permanence value past $connectThreshold$ unless the associated column is sufficiently inactive (see Algorithm 2). To similarly influence BSP we set $pInc$ ($= 0.0005$) to be five times weaker than $pDec$ ($= 0.0025$). In addition, to enable BSP to *reliably* converge within 500 cycles we reduced $P(connect)$ to 0.1 and limited $minOverlap$ to range between 3.0 to 4.0.

Table 2 compares BSP at these adjusted settings with ASP using the standard defaults. The results first show the significant effect of altering $minOverlap$ on the convergence behaviour of BSP, i.e. a reduction of from 4.0 to 3.0 causes a tenfold speedup in convergence, making BSP 3.0 the fastest of the three algorithms. However, this superior convergence is bought at the cost of longer codes, as shown by the mean code length and the distribution of code lengths in the graph. Here a code is the set of columns C_i that become active when an image patch i is present, and a distribution of code lengths is the set of code lengths $|C_i|$ for each image patch $i = 1 \ldots M$. All else being equal, shorter codes are preferred over longer codes because they are more efficient. On this measure, and on the measures of lifetime and population kurtosis, ASP is clearly better than either BSP 3.0 or 4.0. This means ASP reliably produces shorter codes that involve fewer columns and that are more evenly distributed across all columns (as shown by the sharp peak for ASP on the graph in Table 2).

However, an additional dimension is the degree to which a code can distinguish between different inputs. Again, all else being equal, a code that produces finer distinctions is to be preferred. To measure this, we looked at the proportion of image patches that were encoded using common sets of columns (% duplicates and % zero length in Table 2). We used two measures because the duplicate percentage cannot represent the difference between an encoding that captures 1000 images using one set of columns and one that captures 1000 images using 500

sets of columns, where each column set encodes a pair images. In practice, the majority of duplicates only involved column sets encoding image pairs, except for zero length encodings. Such encodings occur when an image fails to make any column active, i.e. the image is ignored or remains unencoded. Clearly, duplicates involving a high proportion of zero length codes (BSP 3.0 and 4.0) make poorer distinctions than encodings where all duplicates are made up of column sets encoding pairs of images (ASP). We can therefore conclude that ASP produces better encodings, both in terms of efficiency, and in terms of making finer distinctions. The price is that ASP converges more slowly than BSP 3.00. However, if speed of convergence is an issue, the ASP parameter defaults can be altered to produce results equivalent to BSP 3.0, whereas we could find no BSP settings that could improve upon the BSP 4.0 encodings.

5 Conclusions

Firstly, we have shown that augmented spatial pooling significantly outperforms the coding schemes presented in Willmore and Tolhurst's original study, both in terms of population and lifetime kurtosis. Secondly, we can conclude that augmented spatial pooling is better than binary spatial pooling for encoding the natural images in the Willmore and Tolhurst data set. The results hold most strongly for the greyscale encodings of the images, where BSP is unable to converge on any of the data sets. It also holds on the binary encodings, where ASP produces better quality sparse representations, both in terms of efficiency (code lengths) and discrimination (duplicates and zero length codes). More generally, we conjecture that the reason BSP performs poorly on natural images is because it forms synapses too easily. This behaviour comes out in relation to natural images because such images have relatively poorly defined structure (i.e. they have high entropy), meaning synapses will tend to form uniformly across the entire image. ASP controls this behaviour by more tightly constraining the situations where new synapses will form.

In future work, we intend to compare ASP and BSP on a wider range of images to confirm our conjecture concerning the complexity of the encodings. We also intend to investigate greyscale spatial pooling using two forms of synapse, one responsive to darker shades and the other responsive to light.

Acknowledgments: We thank David Tolhurst and Ben Willmore for supplying the images used in their original paper.

References

1. Chikkerur, S., Serre, T., Tan, C., Poggio, T.: What and where: A Bayesian inference theory of attention. Vision Research 50(22), 2233–2247 (2010)
2. George, D., Hawkins, J.: A hierarchical Bayesian model of invariant pattern recognition in the visual cortex. In: Proceedings of the International Joint Conference on Neural Networks (IJCNN-05). pp. 1812–1817 (2005)

3. Hawkins, J., Blakeslee, S.: On intelligence. Henry Holt, New York (2004)
4. Hyvärinen, A., Karhunen, J., Oja, E.: Independent Components Analysis. John Wiley and Sons, Inc., New York (2001)
5. Kohonen, T.: Self-organization and associative memory. Springer, Berlin (1989)
6. Lee, T.S., Mumford, D.: Hierarchical Bayesian inference in visual cortex. Journal of the Optical Society of America A 20(7), 1434–1448 (2003)
7. Mountcastle, V.B.: Introduction to the special issue on computation in cortical columns. Cerebral Cortex 13(1), 2–4 (2003)
8. Numenta Inc.: Hierarchical temporal memory including HTM cortical learning algorithms. Tech. rep., Numenta, Inc, Palto Alto (2010), www.numenta.com/htm-overview/education/HTM_CorticalLearningAlgorithms.pdf
9. Olshausen, B.A.: Sparse codes and spikes. In: Rao, R.P.N., Olshausen, B.A., Lewicki, M.S. (eds.) Probabilistic Models of the Brain: Perception and Neural Function, pp. 257–272. MIT Press, Cambridge, Mass (2002)
10. Stuart, G., Spruston, N., Häusser, M.: Dendrites. OUP, New York (2008)
11. Willmore, B., Tolhurst, D.J.: Characterizing the sparseness of neural codes. Network: Computational Neural Systems 12, 255–270 (2001)

Table 1. Comparison of augmented spatial pooling (ASP) averaged lifetime and population kurtosis measures with results published in [11] where Gabor = Gabor filters, ICA = independent components filters, O&F = Olshausen-Field bases, PCA = principal components filters, Sinu = sinusoids, Walsh = Walsh functions, Gaus = Gaussian filters, Pixel = single pixel, Raw = unprocessed images, and White = whitened images.

Kurtosis Measure	Image Process	Coding Scheme ASP	Gabor	ICA	O&F	PCA	Sinu	Walsh	Gaus	Pixel
Lifetime	Raw	87.10	18.50	18.74		8.24	10.33	10.69	7.37	6.76
	White	71.23	18.47		17.21	8.13	10.05	10.91	8.93	11.13
Population	Raw	50.54	21.66	6.42		32.64	27.12	27.75	0.21	1.66
	White	44.64	5.37		2.17	3.07	4.62	4.01	0.52	2.68

Table 2. Comparison of augmented spatial pooling (ASP) with binary spatial pooling (BSP) on the complete set of binary scaled natural images taken from [11].

	Algorithm ASP	BSP	BSP
minOverlap setting	auto	4.00	3.00
Converge Time (secs)	8.48	27.88	2.73
Converge Cycles	15.70	56.40	4.70
% Duplicates	10.96	14.94	8.19
% Zero Length	0.00	9.22	4.40
Lifetime Kurtosis	61.94	47.35	26.98
Population Kurtosis	36.95	27.98	20.37
Mean Code Length	15.59	19.02	22.64