International Journal of Machine Learning and Cybernetics manuscript No.
(will be inserted by the editor)

Spatial Pooling for Greyscale Images

John Thornton · Andrew Srbic

Received: date / Accepted: date

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 sparse distributed representations of spatial images 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. In the process, we 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 well-known coding schemes and explain how the augmented coding scheme can be used to produce high-fidelity reconstructions of greyscale input.

This paper extends work previously published as conference proceedings [22].

Keywords Hierarchical Temporal Memory · Sparse distributed representations · Spatial clustering · Sparse coding · Self-organisation

J. Thornton
Institute for Integrated and Intelligent Systems,
Griffith University, Queensland, 4222, Australia
E-mail: j.thornton@griffith.edu.au

A. Srbic
School of Information and Communication Technology,
Griffith University, Queensland, 4222, Australia
E-mail: andrew.srbic@griffithuni.edu.au

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 connect directly with the original inspiration of artificial intelligence: to build machines that capture and exhibit the principles on which natural intelligence operates. Neurological evidence concerning the hierarchical structure of bottom-up and top-down processing in sensory cortex has already produced a number of promising machine learning applications, including Hinton's work on multilayer generative models [10], and Serre and Poggio's neuromorphic approach to computer vision [20].

In the current paper, we investigate a more general model of neocortical function known as *hierarchical temporal memory* (HTM) [8]. HTMs were first proposed by Jeff Hawkins in 2004, but a practical computational description of their low-level functioning has only recently been developed [7]. 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 [23].

The HTM model is grounded in the view that the broadly uniform structure of the neocortex implies a correspondingly uniform function that the neocortex implements. This structure is organised into six histologically distinct horizontal layers of cells which in turn are locally organised into vertical *minicolumns,* each consisting of between 80-100 neurons (with some exceptions) [14]. The vertically aligned cells in a given minicolumn form a functional unit that shares the same input and tends to become uniformly active or inactive

according to the feedforward input it receives from sensory receptors or from other regions of the neocortex.

Experimental observations have shown that different regions of the neocortex are connected together to form *hierarchical structures* [5]. Current theory suggests that cortical columns play an analogous role to that of random variables in a *Bayesian network* [17], and that the neocortex's basic function is to enact a form of hierarchically-structured probabilistic inference [13, 4, 2]. The powerful aspect of the Bayesian model is that it allows for *feedback* within the neocortical hierarchy while containing the potential combinatorial explosion of possible interpretations. This feedback provides contextual information that in turn can resolve the ambiguities that typically appear in feedforward models when given noisy, realistic input [19].

Research also suggests that the neocortex uses a *sparse coding* strategy to represent information within the hierarchy [15, 16]. A sparse code is one where a relatively small proportion of the code elements are active at any one time. If we take the cortical column to be the basic unit of neocortical activation, this means 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 [1]. The advantage of sparse coding over local coding (where a single input is encoded by a single column) 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 [6, 9]. 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. [13, 18, 3, 2]).

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 [7]. This was needed to extend the representational capacity of the system to the point where realistic sequences of input patterns could be stored and recalled. To the best of our knowledge (with the exception of the conference proceedings [22] which this paper extends), there has been no published evidence evaluating the performance of the new spatial pooler, or of the new cortical column architecture. We therefore decided to implement the HTM spatial pooler and evaluate it on a set of static image benchmarks.

In the remainder of the paper we provide a description of the HTM spatial pooler 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. To evaluate this work, we provide 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 [23]. 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

2.1 Basic Principles

The latest HTM architecture [7] 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. An HTM column currently only implements the functionality of the layer three and four neurons found in the neocortex. 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 [11], 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 [7]). 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 directly relay the input to which it is connected, otherwise the synapse remains potential and inactive. The column then sums the inputs from all its connected synapses to determine its level of activity.

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 [7]. He acknowledges that synapses have differing strengths, but argues that the synaptic release of neurotransmitters is too unreliable or stochastic to explain the fine distinctions that are made between differing inputs. Instead, he points to more recent research that shows how synapses can rapidly form and unform [21] 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 [12], 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, implementing the function of short-range inter-columnar inhibition neurons [21]. 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 unforming of synaptic connections discussed above.

2.2 Implementation Details

The top level structure of an HTM spatial pooler is presented in Algorithm 1. Here the pooler takes as input a set of binary input vectors and set of columns (represented as *images* and *columns*). For the purposes of explanation, we shall assume the vectors represent two

Algorithm 1 spatialPooler($images, columns$)

```
initialiseSynapses(columns, inhibitionArea)
while not converged do
  for each image = readInput(images) do
    calculateOverlap(image, columns)
    inhibitColumns(columns, inhibitionArea)
    learnSynapseConnections(columns, inhibitionArea)
  end for
  converged = testForConvergence(columns)
end while
```

dimensional bitmap images, where each image I is specified by a set of pixel values $I(x, y)$. The current HTM specifications assume that all inputs are binary-valued, hence we term this algorithm as *binary* spatial pooling or BSP (we present modifications that handle greyscale images in section 3.1).

In procedure *initialiseSynapses,* the potential synapses of each column are randomly connected with probability $P(connect)$ to each (x, y) input coordinate, such that no coordinate is shared by synapses belonging to the same column. A mapping is then generated from each column to an (x, y) input coordinate, such that no two columns share the same coordinate. This ensures each column is uniquely centred over an input coordinate, and specifies a Euclidean distance d_{ij} between a column i and the coordinates of each of its synapses j. Using this distance, every potential synapse is allocated a randomly generated permanence value bounded within a small range of the threshold value, such that the probability of connection varies inversely and linearly with d_{ij}. In the current implementation, the permanence value has a potential range from 0 to 1 and is bounded on initialisation to be within 0.1 of the 0.2 $connectThreshold$. Those potential synapses with a permanence value > 0.2 are now defined as *connected* synapses.

The *readInput* procedure randomly selects an image without replacement from *images* and *calculateOverlap* works out the initial activity level of each column's response to the given image. For binary images, this activity level (or overlap) is a simple count of the number of its connected synapses that are receiving active input. However, a column must also pass a *minOverlap* threshold and have its overlap boosted before being considered for inhibition, as follows:

for each column c **do**
  if $overlap(c) < minOverlap$ **then** $overlap(c) = 0$
  else $overlap(c) = overlap(c) \times boost(c)$
end for

where $boost(c)$ is determined by the learning procedure. *inhibitColumns* is then implemented as follows:

Algorithm 2 learnSynapseConnections(*columns*, *inhibitionArea*)

```
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 if
end for
for each c in columns do
  if activity(c) < minActivity(c) then
    boost(c) = boost(c) + bInc
  else
    boost(c) = 1
  end if
  if overlapSum(c) < minActivity(c) then
    for each potential synapse s in c do
      perm(s) = min(perm(s) × pMult, 1)
    end for
  end if
end for
inhibitionArea = updateInhibitionArea(columns)
```

```
for each column c do
  active(c) = false and activitySum = 0
  for each neighbour n of c within inhibitionArea do
    if overlap(n) > overlap(c) then
      activitySum = activitySum + 1
    end if
  end for
  if activitySum < desiredActivity then active(c) = true
end for
```

Here, *desiredActivity* defines the number of columns that should be active within the *inhibitionArea*, and *inhibitionArea* is the mean size of the receptive fields of all columns. In the current implementation, we calculate the size of a column's receptive field as $\pi d^2/16$, such that:

$d = max(c, x) - min(c, x) + max(c, y) - min(c, y) + 2$

where *max* and *min* return the the maximum and minimum x and y values of all synapses belonging to column c. This calculation determines the rectangle that bounds a column's synapses, and returns the area of the circle that is enclosed by the square whose sides equal the average of the two dimensions of the bounding rectangle. We developed this procedure to compensate for the distorting edge effects of projecting synapses into rectangular images (the published HTM specifications leave the calculation of the inhibition area undefined).

The basic learning strategy is then implemented in *learnSynapseConnections* (see Algorithm 2). This is the most complex procedure and forms the focus of the modifications of the next section. Firstly, the permanence values of all synapses belonging to active columns are adjusted: those connected to currently active input are incremented and those connected to inactive input are decremented. Then, in the second for loop, 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 $minActivity(c)$, which is defined as follows:

```
for each active column c do
  maxActivity = 0
  for each column n within inhibitionArea of c do
    if activity(n) > maxActivity then
      maxActivity = activity(n)
  end for
  minActivity(c) = maxActivity × minActivityThreshold
end for
```

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, *testForConvergence* (see Algorithm 1) calculates whether any changes have been made to the permanence value of any synapse since the last iteration through the entire set of *images*. If no changes have occurred the spatial pooler terminates.

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

3.1 Handling Greyscale Images

The HTM specifications only handle binary input. 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 (see lines 1–7 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. So, for example, given a greyscale 16×16 pixel image, the mean activation would be the sum of the individual pixel values divided by 256. In this case, only potential synapses connected to pixel values greater than the mean have their permanence values incremented. 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.

3.2 Improving Efficiency

Preliminary empirical testing showed it was unnecessary to evaluate the activity of every column in every iteration (lines 8–19 in Algorithm 2). The significant factor is whether the permanence value of a synapse is incremented past the *connectThreshold* so that it becomes connected, or decremented so that it becomes disconnected. Only then can a change in permanence have any influence on the behaviour of a column, as only then does it alter the overlap calculation. Consequently, we introduced a decay mechanism such that whenever a synapse crosses the *connectThreshold* and becomes connected then the column to which the synapse belongs is not evaluated for boosting (in lines 8–19 in Algorithm 2) for the next d iterations. This gives the column a chance to respond to the next d images, during which time it may become active without further boosting. In addition, if a column is already strongly active then boosting is unnecessary and we should avoid testing it. This can be achieved by calculating the difference f between a column's activity and the threshold at which boosting is triggered, where $f = activity(c) - minActivity(c)$, and enforcing that the column is not evaluated for boosting in the next $f \times d$ iterations. These two modifications approximately halved the execution time of our spatial pooler, without significantly altering its convergence behaviour.

A second time consuming calculation is the updating of the inhibition area of each column after each iteration (in *updateInhibitionArea*). This procedure can again be simplified by keeping track of the synapses that become connected or disconnected during a single iteration. If there is no change for a column during an iteration then its inhibition area necessarily remains the same and need not be updated.

Finally, the convergence behaviour of the pooler can be accelerated by switching off the basic learning function in lines 1–7 of Algorithm 2 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. We can therefore turn off the basic learning function (lines 1–7 of Algorithm 2). This does not mean the pooler has attained a stable representation: it can still happen in the next iteration through the same images that the (now fixed) pattern of connected synapses is unable to maintain sufficient activity in all columns, in which case a column will still be boosted or its potential synapses incremented. If this occurs then basic learning is immediately resumed. 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 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.

3.3 Augmented Spatial Pooling

The main contribution of the paper, aside from evaluating the current HTM pooler, is the development of a more robust learning strategy. This strategy was suggested by observing that the existing HTM 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 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 3 learnSynapseConnections(*image*, *columns*, *inhibitionArea*)

```
1  for each potential synapse s in each active column c do
2    if input(s) > meanInput(image) and perm(s) >= connectThreshold then
3      perm(s) = min(perm(s) + pInc, 1)
4    else if input(s) > meanInput(image) and perm(s) < connectThreshold then
5      perm(s) = min(perm(s) + pInc, connectThreshold − pInc)
6    else
7      perm(s) = max(perm(s) − pDec, 0)
8    end if
9  end for
10 for each c in columns do
11   if activity(c) < minActivity(c) and (boost(c) = boost(c) + bInc) > bMax then
12     boost(c) = 1
13     for each disconnected synapse s in c in ascending distance order from c do
14       if perm(s) > maxPerm then
15         maxPerm = perm(s) and maxS = s
16       end if
17     end for
18     perm(maxS) = connectThreshold + pInc
19   end if
20 end for
21 inhibitionArea = updateInhibitionArea(columns)
```

Algorithm 3 details the augmented learning procedure. Here the updating of synapses of active columns is altered so that disconnected synapses can only have their permanence values incremented to a point just below the *connectThreshold* (in lines 5–8). Now, the only place where synapses can become connected is in the boosting procedure (line 23). As before, if a column's activity is below the $minActivity(c)$ threshold its boost value is increased (lines 14–15). 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 (line 23). 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 and our various modifications we chose the 64 greyscale images used in Willmore and Tolhurst's influential study on measures of sparsity [23]. Following [23], we generated ten sets of $10,000$ 16×16 image patches selected randomly from the original 64 greyscale images.

The two primary dimensions on which we evaluate the spatial pooling algorithms are firstly the speed and reliability of convergence, and secondly, the quality of the representations produced. Measuring representational quality again has two dimensions: 1) the degree of sparseness; and 2) the degree of fidelity to the original images. In Willmore and Tolhurst's original paper, sparseness was further broken down in two statistics: 1) population sparseness; and 2) lifetime sparseness, both of which measure the kurtosis of distributions of activity units. Population sparseness is calculated by averaging the kurtosis of the distribution of the activities of the complete set of 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 $a_1 \ldots a_N$ are the post-inhibition overlap activities of columns $1 \ldots N$ 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 [23], 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*. Here the averaged population kurtosis measures the infrequency or sparseness of column activity in response to individual images. This is only a partial measure of sparseness, as it does not consider how the responses are *distributed*. For example, if the same column is strongly responsive to *every* image while every other column is inactive, the average population kurtosis will be the same as when each image causes a *different* column to become active. To balance this, we also need to consider the average lifetime kurtosis of the columns. This is the averaged kurtosis of each column's responses to an entire set of input *images* and measures how infrequently a particular column is active. The lifetime kurtosis of a single column c is given by:

$$lifetimeKurtosis_c = \left\{ \frac{1}{M} \sum_{i=1}^{M} \left[\frac{a_i - \bar{a}}{\sigma_a} \right]^4 \right\} - 3 \quad (2)$$

where $a_1 \ldots a_M$ are the activities of column c to the entire set of M *images*. As before, the mean activity $\bar{a}$ is standardised to zero, σ_a is standardised to one, and the averaged lifetime kurtosis is calculated over the entire set of N *columns* as $\frac{1}{N} \sum_{c=1}^{N} lifetimeKurtosis_c$.

4.1 Comparisons with Greyscale Images

Given these measures we can now directly compare the sparseness and distribution of the spatial pooler representations with the results reported in [23] for the

same 64 greyscale images and against a range of other coding schemes (see Table 1). Here, to correspond with [23], we generated a spatial pooler column for each image pixel (256 in all), and set the pooler parameters as follows: $P(connect) = 0.15$, $connectThreshold = 0.2$, $pInc = pDec = 0.02$, $bInc = 0.005$, $bMax = 4$, $minActivityThrehold = 0.01$, $desiredActivity = 0.05 \times inhibitionArea$, and decay $d = 100$. $minOverlap$ 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 of the 10 image sets). This reflects the fact 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.

Table 1 Comparison of augmented spatial pooling (ASP) averaged lifetime and population kurtosis measures with results published in [23] where 'Gabor' = Gabor filters, 'ICA' = independent components filters, 'O&F' = Olshausen-Field bases, 'PCA' = principal components filters, 'Walsh' = Walsh functions, 'Gaussian' = Gaussian filters, 'DoG' = difference of Gaussians, 'Pixel' = single pixel, 'Raw' = unprocessed images, and 'White' = whitened images.

	Lifetime Kurtosis		Population Kurtosis	
	Raw	White	Raw	White
ASP	87.10	71.23	50.54	44.64
Gabor	18.50	18.47	21.66	5.37
ICA	18.74		6.42	
O&F		17.21		2.17
PCA	8.24	8.13	32.64	3.07
Sinusoid	10.33	12.05	27.12	4.62
Walsh	10.69	10.91	27.75	4.01
Gaussian	7.37	8.93	0.21	0.52
DoG	9.67	11.20	1.70	1.74
Pixel	6.76	11.13	1.66	2.68

Overall, the results show that ASP significantly outperforms all the coding schemes considered in [23], 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).

Whenever such a large improvement is found in a particular metric it is typically purchased at the expense of worse performance in another area. In this case, and in comparison with other biologically plausible algorithms (such as independent components analysis (ICA) [11] and Olshausen-Field filters (O&F) [15]), the price is paid in terms of the reconstruction error. For example, both ICA and O&F explicitly search for a set of basis functions ϕ such that the ν error term in $I(x,y) = \sum_i a_i \phi_i(x,y) + \nu(x,y)$ is minimised. In contrast, spatial pooling concentrates on generating codes that have high population and lifetime sparseness, and achieves this by limiting the number of columns that can be active at any one time while ensuring that all columns are active at least some of the time. Rather than minimising reconstruction error, spatial pooling is concerned with finding *stable* codes, while also remaining realistically responsive to new input patterns.

On this basis, a strict spatial pooler reconstruction of an input would be the linear superposition of the connected synapses of the columns that become active when the input is present. We show such a reconstruction in the bottom left image of Figure 1. Here we took $14,641$ 16×16 pixel patches from an individual Willmore and Tolhurst image and used a linear superposition of ASP column responses to represent each pixel. The root mean squared error (RMSE) between the original image and this raw reconstruction (using an 8-bit greyscale mapping) is 48.0%, although this can be trivially reduced to 21.6% by selecting a narrower mapping that lightens the image.

To obtain further improvements we rescaled the synapse responses according to the mean intensity of the inputs. Firstly, for each input patch, if no active synapse is connected to an input pixel then the response to that pixel is set to the mean value of all the below mean pixels in the input patch. Next, if one or more synapses are connected to an input pixel then the existing response to that pixel is multiplied by the mean value of all the

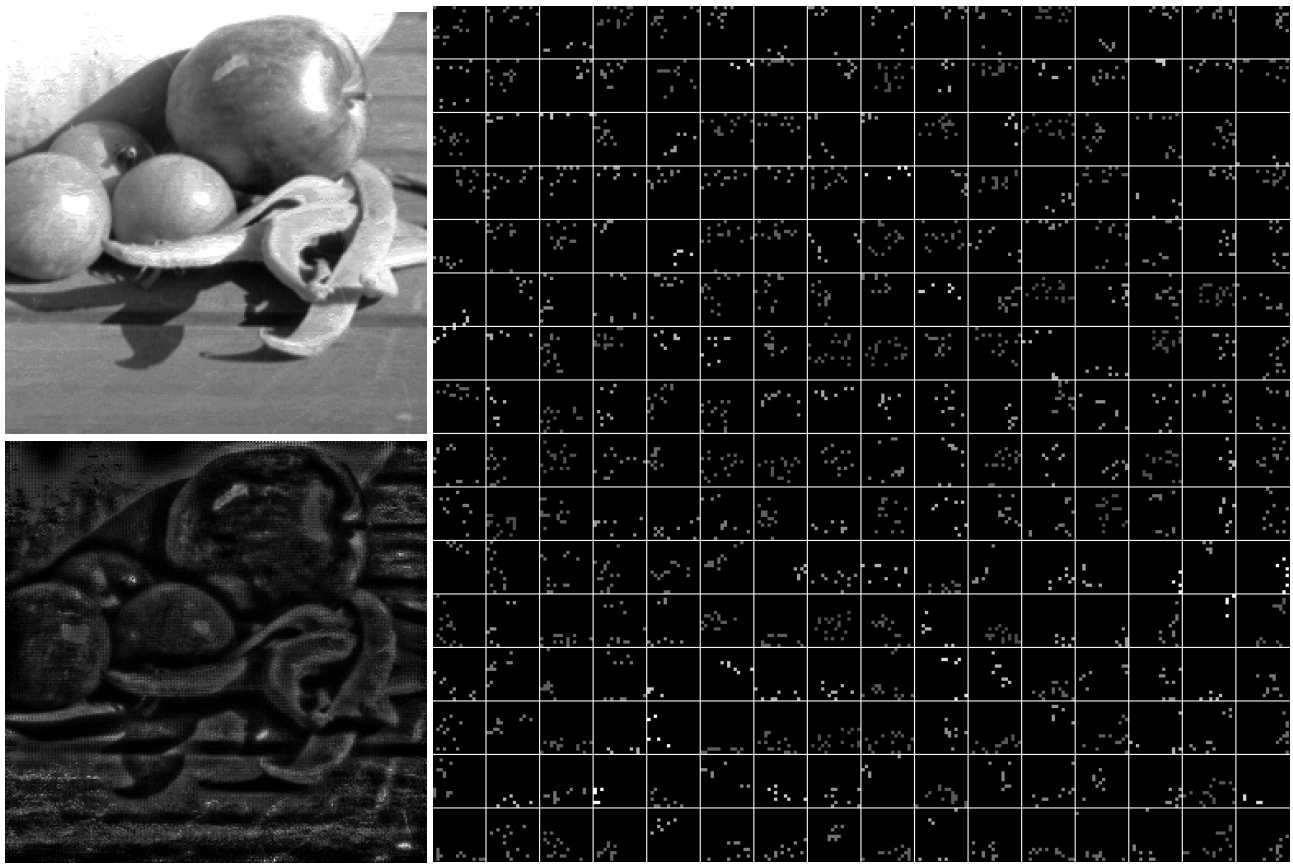


Fig. 1 ASP response to a 256×256 pixel greyscale natural image (taken from [23]). Top left: original image. Right: Diagram showing the synapse connections learnt after exposure to $14,641$ 16×16 pixel patches taken from the original image (each square represents a column and each dot represents a connected synapse of that column, where a lighter shaded synapse represents a stronger boost value). Bottom left: Superposition of the connected synapses after convergence on the complete set of patches.

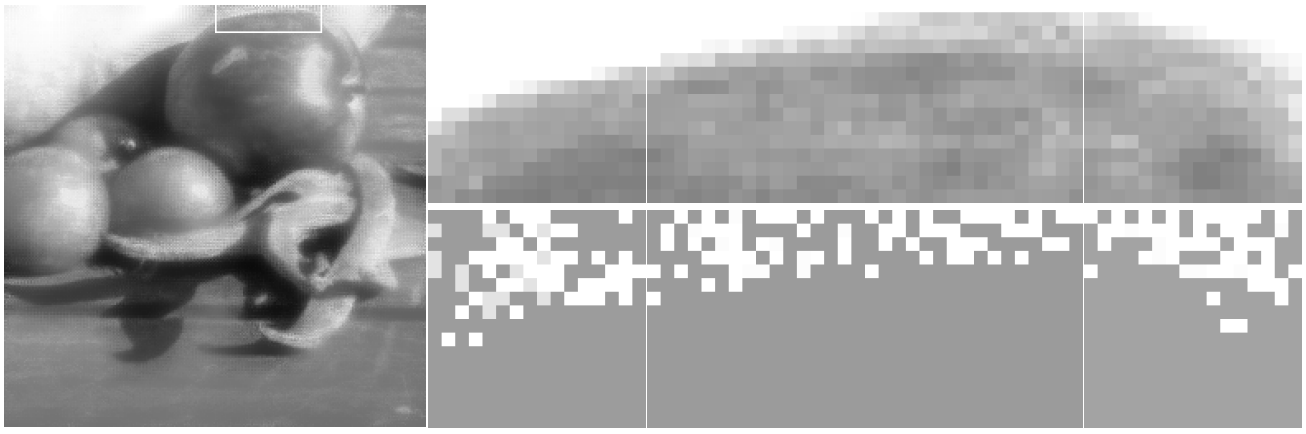


Fig. 2 ASP reconstruction of the original image. Left: Final reconstruction after scaling column responses according to the mean pixel intensity of each patch (see text). Right: Diagram showing the active column responses to four 16×16 pixel patches (indicated by the white bordered box on the reconstructed image). The upper row of the diagram reproduces the original patches and the bottom row shows the rescaled responses of the active synapses to the corresponding upper row patch. As in Figure 1, synapse activity is represented by lighter shaded dots, only here synapse activity is a superposition of all columns that are activated by the patch.

above mean input pixels divided by the mean value of the unscaled responses to the input patch. This rescales the synapse responses to take into account the average intensity of the input. The left hand image in Figure 2

shows a reconstruction of the original image from Figure 1 using this procedure. Here the RMSE is reduced to 6.7%. The high fidelity of this reconstruction demonstrates that the synapse connections learnt by the spatial pooling algorithm have successfully captured the underlying structure of the input *independently* of the particular conditions of illumination, i.e. as adding the mean illumination conditions back into the superposed synaptic representation accurately reproduces the original image.

4.2 Comparisons with Binary Images

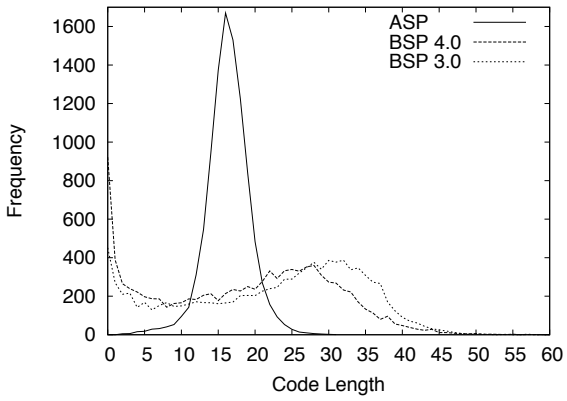
As the original spatial pooling algorithm (BSP) was unable to converge on the greyscale images, we ran a separate experiment using the same set of natural image patches but after performing a binary conversion. This conversion sets all pixels in a given image patch to zero whose greyscale values are less than the mean greyscale value for that patch, otherwise a pixel is scaled to one.

BSP still found these binary images challenging in comparison to simpler binary encodings and was unable to converge using ASP's default parameter settings. After testing, we found BSP would only converge if the effect of decrementing the permanence value of a synapse is made much stronger than the effect of an increment, making it easier for a synapse to become disconnected than for it to become connected. ASP already achieves this by not incrementing a permanence value past *connectThreshold* unless the associated column is sufficiently inactive (see Algorithm 3). 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 *pConnect* 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 previously described 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 producing 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 presented with an image patch i, and a distribution of code lengths is the set of code lengths $|C_i|$ for each image patch $i = 1 \dots M$. All else being equal, shorter codes are to be 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.

Table 2 Comparison of augmented spatial pooling (ASP) with binary spatial pooling (BSP) on the complete set of binary scaled natural images taken from [23].

	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



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 distinc-

tions. 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 have demonstrated that while a raw ASP image reconstruction has a relatively high error, this can be largely eliminated by taking the mean illumination conditions into account. Thirdly, 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 natural and artificial 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.

Acknowledgements We thank Ben Willmore and David Tolhurst for supplying the images used in their original paper on sparse neural codes [23].

References

1. Bell, A.J., Sejnowski, T.J.: Edges are the 'independent components' of natural scenes. In: M.C. Mozer, M.J. Jordan, T. Petsche (eds.) Advances in neural information processing systems, vol. 9, pp. 831–837. MIT Press, Cambridge, Mass (1997)
2. Chikkerur, S., Serre, T., Tan, C., Poggio, T.: What and where: A Bayesian inference theory of attention. Vision Research **50**(22), 2233–2247 (2010)
3. Dean, T.: A computational model of the cerebral cortex. In: Proceedings of the 20th National Conference on Artificial Intelligence (AAAI-05), pp. 938–943 (2005)
4. Doya, K., Ishii, S., Pouget, A., Rao, R.P.N.: The Bayesian Brain. MIT Press, Boston (2007)
5. Felleman, D., van Essen, D.: Distributed hierarchical processing in the primate cerebral cortex. Cerebral Cortex **1**(Jan/Feb), 1–47 (1991)
6. 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)
7. Hawkins, J., Ahmad, S., Dubinsky, D.: Hierarchical temporal memory including HTM cortical learning algorithms. Tech. rep., Numenta, Inc, Palto Alto (2010). URL www.numenta.com/htm-overview/education/HTM_CorticalLearningAlgorithms.pdf
8. Hawkins, J., Blakeslee, S.: On intelligence. Henry Holt, New York (2004)
9. Hawkins, J., George, D.: Hierarchical temporal memory: Concepts, theory and terminology. Tech. rep., Numenta, Inc, Palto Alto (2006). URL www.numenta.com.Numenta.HTM.Concepts.pdf
10. Hinton, G.: Learning multiple layers of representation. Trends in Cognitive Sciences **11**(10), 428–434 (2007)
11. Hyvärinen, A., Karhunen, J., Oja, E.: Independent Components Analysis. John Wiley and Sons, Inc., New York (2001)
12. Kohonen, T.: Self-organization and associative memory. Spriger-Verlag, Berlin (1989)
13. Lee, T.S., Mumford, D.: Hierarchical Bayesian inference in visual cortex. Journal of the Optical Society of America A **20**(7), 1434–1448 (2003)
14. Mountcastle, V.B.: Introduction to the special issue on computation in cortical columns. Cerebral Cortex **13**(1), 2–4 (2003)
15. Olshausen, B.A.: Sparse codes and spikes. In: R.P.N. Rao, B.A. Olshausen, M.S. Lewicki (eds.) Probabilistic Models of the Brain: Perception and Neural Function, pp. 257–272. MIT Press, Cambridge, Mass (2002)
16. Olshausen, B.A.: Principles of image representation in visual cortex. In: L. Chalupa, J. Werner (eds.) The Visual Neurosciences. MIT Press, Cambridge, Mass (2003)
17. Pearl, J.: Probabilistic reasoning in intelligent systems. Morgan Kaufman Publishers, San Francisco, California (1988)
18. Rao, R.P.N.: Bayesian computation in recurrent neural circuits. Neural Computation **16**(1), 1–38 (2004)
19. Serre, T., Oliva, A., Poggio, T.: A feedforward architecture accounts for rapid categorization. Proceedings of the National Academy of Sciences of the USA **104**(15), 6424–6429 (2007)
20. Serre, T., Poggio, T.: A neuromorphic approach to computer vision. Communications of the ACM **53**(10), 54–61 (2010)
21. Stuart, G., Spruston, N., Häusser, M.: Dendrites. Oxford University Press, New York (2008)
22. Thornton, J., Srbic, A., Main, L., Chitsaz, M.: Augmented spatial pooling. In: Proceedings of the 24th Australasian Joint Conference on Artificial Intelligence, pp. 261–270. Perth, Australia (2011)
23. Willmore, B., Tolhurst, D.J.: Characterizing the sparseness of neural codes. Network: Computational Neural Systems **12**, 255–270 (2001)