Scaling up Spike-and-Slab Models for Unsupervised Feature Learning

Ian J. Goodfellow, Aaron Courville, and Yoshua Bengio

Abstract—We describe the use of two spike-and-slab models for modeling real-valued data, with an emphasis on their applications to object recognition. The first model, which we call *spike-and-slab sparse coding* (S3C), is a pre-existing model for which we introduce a faster approximate inference algorithm. We introduce a deep variant of S3C which we call the *partially directed deep Boltzmann machine* (PD-DBM) and extend our S3C inference algorithm for use on this model. We describe learning procedures for each. We demonstrate that our inference procedure for S3C enables scaling the model to unprecedently large problem sizes, and demonstrate that using S3C as a feature extractor results in very good object recognition performance, particularly when the number of labeled examples is low. We show that the PD-DBM generates better samples than its shallow counterpart, and that unlike DBMs or DBNs, the PD-DBM may be trained successfully without greedy layerwise training.

Index Terms—Neural nets, pattern recognition, computer vision

1 Introduction

It is difficult to overstate the importance of the quality of the input features to supervised learning algorithms. A supervised learning algorithm is given a set of examples $V = \{v^{(1)}, \ldots, v^{(m)}\}$ and associated labels $\{y^{(1)}, \ldots, y^{(m)}\}$ from which it learns a mapping from v to y that can predict the labels y of new unlabeled examples v. The difficulty of this task is strongly influenced by the choice of *representation*, or the feature set used to encode the input examples v. The premise of unsupervised feature discovery is that, by learning the structure of V, we can discover a feature mapping $\phi(v)$ that renders standard supervised learning algorithms, such as the support vector machine, more effective. Because $\phi(v)$ can be learned from unlabeled data, unsupervised feature discovery can be used for semi-supervised learning (where many more unlabeled examples than labeled examples are available) or transfer learning (where the classifier will be evaluated on only a subset of the categories present in the training data).

When adopting a *deep learning* (Bengio, 2009) approach, the feature learning algorithm should discover a ϕ that consists of the composition of several simple feature mappings, each of which transforms the output of the earlier mappings in order to incrementally disentangle the factors of variation present in the data. Deep learning methods are typically created by repeatedly composing together shallow unsupervised feature learners. Examples of shallow models applied to feature discovery include sparse coding (Raina *et al.*, 2007), restricted Boltzmann machines (RBMs) (Hinton *et al.*, 2006; Courville *et al.*, 2011a), various autoencoder-based models (Bengio *et al.*, 2007; Vincent *et al.*, 2008), and hybrids of autoencoders and sparse coding (Kavukcuoglu *et al.*, 2010).

In this paper, we describe how to use a model which we call *spike-and-slab sparse coding* (S3C) as an efficient feature learning algorithm. We also demonstrate how to construct a new deep model, the *partially directed deep Boltzmann machine* (PD-DBM) with S3C as its first layer. Both are models of real-valued data, and as such are well-suited to modeling images, or image-like data, such as audio that has been preprocessed into an image-like space (Deng *et al.*, 2010). In this paper, we focus on applying these models to object recognition.

Single-layer convolutional models based on simple thresholded linear feature extractors are currently among the state-of-the-art performers on the CIFAR-10 object recognition dataset (Coates and Ng, 2011; Jia and Huang, 2011). However, the CIFAR-10 dataset contains 5,000 labels per class, and this amount of labeled data can be inconvenient or expensive to obtain for applications requiring more than 10 classes. Previous work has shown that the performance of a simple thresholded linear feature set degrades sharply in accuracy as the number of labeled examples decreases (Coates and Ng, 2011).

We introduce the use of the S3C model as a feature extractor in order to make features more robust to this degradation. This is motivated by the observation that sparse coding performs relatively well when the number of labeled examples is low (Coates and Ng, 2011). Sparse coding inference invokes a competition among the features to *explain* the data and therefore, relative to simple thresholded linear feature extractors, acts as a more regularized feature extraction scheme. We speculate that this additional regularization is responsible for its improved performance in the low-labeled-data regime. S3C can be

• *Département d'Informatique et de recherche opérationelle, Université de Montréal, Montréal, QC, H3C 3J7, Canada*
E-mail: goodfeli@iro.umontreal.ca

considered as employing an alternative regularization for feature extraction where, unlike sparse coding, the sparsity prior is decoupled from the magnitude of the non-zero, real-valued feature values.

The S3C generative model can be viewed as a hybrid of sparse coding and the recently introduced spike-and-slab RBM (Courville *et al.*, 2011b). Like the spike-and-slab RBM (ssRBM), S3C possesses a layer of hidden units composed of real-valued *slab* variables and binary *spike* variables. The binary spike variables are well suited as inputs to subsequent layers in a deep model. However, like sparse coding and unlike the ssRBM, S3C can be interpreted as a *directed* graphical model, implying that features in S3C compete with each other to explain the input. As we show, S3C can be derived either from sparse coding by replacing the factorial Laplace prior with a factorial spike-and-slab prior, or from the ssRBM, by simply adding a term to its energy function that causes the hidden units to compete with each other.

We hypothesize that S3C features have a stronger regularizing effect than sparse coding features due to the greater sparsity in the spike-and-slab prior relative to the Laplace prior. We validate this hypothesis by showing that S3C has superior performance when labeled data is scarce. We present results on the the CIFAR-10 and CIFAR-100 object classification datasets. We also describe how we used S3C to win a transfer learning challenge.

The major technical challenge in using S3C is that exact inference over the posterior of the latent layer is intractable. We derive an efficient structured variational approximation to the posterior distribution and use it to perform approximate inference as well as learning as part of a variational Expectation Maximization (EM) procedure (Saul and Jordan, 1996). Our inference algorithm allows us to scale inference and learning in the spike-and-slab coding model to the large problem sizes required for state-of-the-art object recognition.

Our use of a variational approximation for inference distinguishes S3C from standard sparse coding schemes where maximum *a posteriori* (MAP) inference is typically used. It also allows us to naturally incorporate S3C as a module of a deeper model. We introduce learning rules for the resulting PD-DBM, describe some of its interesting theoretical properties, and demonstrate how this model can be trained jointly by a single algorithm, rather than requiring the traditional greedy learning algorithm that consists of composing individually trained components (Salakhutdinov and Hinton, 2009). The ability to jointly train deep models in a single unified learning stage has the advantage that it allows the units in higher layers to influence the entire learning process at the lower layers. We anticipate that this property may become essential in the future as the size of the models increases. Consider an extremely large deep model, with size sufficient that it requires sparse connections. When this model is trained jointly, the feedback from the units in higher layers will cause units in lower layers to naturally group themselves so that each higher layer unit receives all of the information it needs in its sparse receptive field. Even in small, densely connected models, greedy training may get caught in local optimal that joint training can avoid.

2 MODELS

We now describe the models considered in this paper. We first study a model we call the spike-and-slab sparse coding (S3C) model. This model has appeared previously in the literature in a variety of different domains (Lücke and Sheikh, 2011; Garrigues and Olshausen, 2008; Mohamed *et al.*, 2012; Zhou *et al.*, 2009; Titsias and Lázaro-Gredilla, 2011). Next, we describe a way to incorporate S3C into a deeper model, with the primary goal of obtaining a better generative model.

2.1 The spike-and-slab sparse coding model

The spike-and-slab sparse coding model consists of latent binary *spike* variables $h \in \{0,1\}^N$, latent real-valued *slab* variables $s \in \mathbb{R}^N$, and real-valued visible vector $v \in \mathbb{R}^D$ generated according to this process:

$\forall i \in \{1,\ldots,N\}, d \in \{1,\ldots,D\}$,

$$\begin{aligned} p(h_i = 1) &= \sigma(b_i) \\ p(s_i \mid h_i) &= \mathcal{N}(s_i \mid h_i \mu_i, \alpha_{ii}^{-1}) \\ p(v_d \mid s, h) &= \mathcal{N}(v_d \mid W_{d:}(h \circ s), \beta_{dd}^{-1}) \end{aligned} \tag{1}$$

where σ is the logistic sigmoid function, b is a set of biases on h, μ and W govern the linear dependence of s on h and v on s respectively, α and β are diagonal precision matrices of their respective conditionals, and $h \circ s$ denotes the element-wise product of h and s.

To avoid overparameterizing the distribution, we constrain the columns of W to have unit norm, as in sparse coding. We restrict α to be a diagonal matrix and β to be a diagonal matrix or a scalar. We refer to the variables h_i and s_i as jointly defining the i^{th} hidden unit, so that there are a total of N rather than $2N$ hidden units. The state of a hidden unit is best understood as $h_i s_i$, that is, the spike variables gate the slab variables[1].

2.2 The partially directed deep Boltzmann machine model

As described above, the S3C prior is factorial over the hidden units ($h_i s_i$ pairs). Distributions such as the distribution over natural images are rarely well described by simple independent factor models, and so we expect that S3C will likely be a poor generative model for the kinds of data that we wish to consider. We now show one way of incorporating S3C into a deeper model, with the primary goal of obtaining a better generative model. If we assume that μ becomes large relative to α, then the primary structure we need to model is in h. We therefore propose placing a DBM prior rather than a factorial

1. We can essentially recover h_i and s_i from $h_i s_i$ since $s_i = 0$ has zero measure.

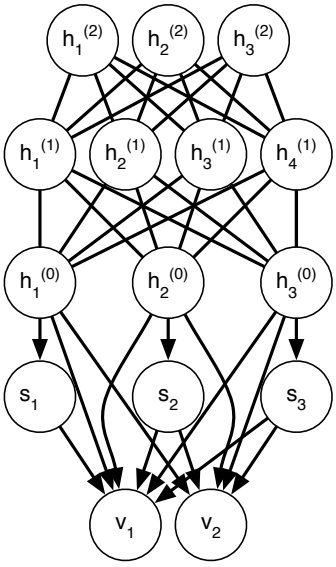

Fig. 1. A graphical model depicting an example PD-DBM.

prior on h. The resulting model can be viewed as a deep Boltzmann machine with directed connections at the bottom layer. We call the resulting model a partially directed deep Boltzmann machine (PD-DBM).

The PD-DBM model consists of an observed input vector $v \in \mathbb{R}^D$, a vector of slab variables $s \in \mathbb{R}^{N_0}$, and a set of binary vectors $\boldsymbol{h} = \{h^{(0)}, \ldots, h^{(L)}\}$ where $h^{(l)} \in \{0,1\}^{N_l}$ and L is the number of layers added on top of the S3C model.

The model is parameterized by β, α, and μ, which play the same roles as in S3C. The parameters $W^{(l)}$ and $b^{(l)}$, $l \in \{0, \ldots, L\}$ provide the weights and biases of both the S3C model and the DBM prior attached to it.

Together, the complete model implements the following probability distribution:

$$P_{\mathrm{PD-DBM}}(v, s, \boldsymbol{h}) = P_{\mathrm{S3C}}(v, s | h^{(0)}) P_{\mathrm{DBM}}(\boldsymbol{h})$$

where

$$P_{\mathrm{DBM}}(\boldsymbol{h}) \propto \exp\left(- \sum_{l=0}^{L} b^{(l)T} h^{(l)} - \sum_{l=1}^{L} h^{(l-1)T} W^{(l)} h^{(l)} \right).$$

A version of the model with three hidden layers ($L = 2$) is depicted graphically in Fig. 1.

Besides admitting a straightforward learning algorithm, the PD-DBM has several useful properties:

- The partition function exists for all parameter settings. This is not true of the spike-and-slab restricted Boltzmann machine (ssRBM), which is a very good generative model of natural images (Courville *et al.*, 2011b).
- The model family is a universal approximator. The DBM portion, which is a universal approximator of binary distributions (Le Roux and Bengio, 2008), can implement a one-hot prior on $h^{(0)}$, thus turning the overall model into a mixture of Gaussians, which is a universal approximator of real-valued distributions (Titterington *et al.*, 1985).
- Inference of the posterior involves feedforward, feedback, and lateral connections. This increases the biological plausibility of the model, and enables it to learn and exploit several rich kinds of interactions between features. The lateral interactions make the lower level features compete to explain the input, and the top-down influences help to obtain the correct representations of ambiguous input.

3 Learning Procedures

Maximum likelihood learning is intractable for both models. S3C suffers from an intractable posterior distribution over the latent variables. In addition to an intractable posterior distribution, the PD-DBM suffers from an intractable partition function.

We follow the variational learning approach used by Salakhutdinov and Hinton (2009) to train DBMs: rather than maximizing the log likelihood, we maximize a variational lower bound on the log likelihood. In the case of the PD-DBM we must do so using a stochastic approximation of the gradient.

The basic strategy of variational learning is to approximate the true posterior $P(h, s \mid v)$ with a simpler distribution $Q(h, s)$. The choice of Q induces a lower bound on the log likelihood called the negative variational free energy. The term of the negative variational free energy that depends on the model parameters is

$$\begin{aligned} &\mathbb{E}_{s,h\sim Q}[\log P(v, s, h)] \\ &= -\mathbb{E}_{s,h\sim Q}[\log P(v \mid s, h^{(0)}) + \log P(s \mid h) + \log P(h)] \end{aligned}$$

In the case of S3C, this bound is tractable, and can be optimized in a straightforward manner. It is even possible to use variational EM (Saul and Jordan, 1996) to make large, closed-form jumps in parameter space. However, we find gradient ascent learning to be preferable in practice, due to the computational expense of the closed-form solution, which involves estimating and inverting the covariance matrix of all of the hidden units.

In the case of the PD-DBM, the objective function is not tractable because the partition function of the DBM portion of the model is not tractable. We can use contrastive divergence (Hinton, 2000) or stochastic maximum likelihood (Younes, 1998; Tieleman, 2008) to make a sampling-based approximation to the DBM partition function's contribution to the gradient. Thus, unlike S3C, we must do gradient-based learning rather than closed-form parameter updates. However, the PD-DBM model still has some nice properties in that only a subset of the variables must be sampled during training. The factors of the partition function originating from the S3C portion of the model are still tractable. In particular, training does not ever require sampling real-valued variables. This is a nice property because it means that the gradient estimates are bounded for fixed parameters and data. When sampling real-valued variables, it is possible for the sampling procedure to make gradient estimates arbitrarily large.

We found that using the "true gradient" (Douglas *et al.*, 1999) method to be useful for learning with the norm constraint on W. We also found that using momentum (Hinton, 2010) is very important for learning PD-DBMs.

3.1 Avoiding greedy pretraining

Deep models are commonly pretrained in a greedy layerwise fashion. For example, a DBM is usually initialized from a stack of RBMs, with one RBM trained on the data and each of the other RBMs trained on samples of the previous RBM's hidden layer.

Any greedy training procedure can obviously get stuck in a local minimum. Avoiding the need for greedy training could thus result in better models. For example, when pretraining with an RBM, the lack of explaining away in the posterior prevents the first layer from learning nearly parallel weight vectors, since these would result in similar activations (up to the bias term, which could simply make one unit always less active than the other). Even though the deeper layers of the DBM could implement the explaining away needed for these weight vectors to function correctly (ie, to have the one that resembles the input the most activate, and inhibit the other unit), the greedy learning procedure does not have the opportunity to learn such weight vectors.

Previous efforts at jointly training even two layer DBMs on MNIST have failed (Salakhutdinov and Hinton, 2009; Desjardins *et al.*, 2012; Montavon and Müller, 2012). Typically, the jointly trained DBM does not make good use of the second layer, either because the second layer weights are very small or because they contain several duplicate weights focused on a small subset of first layer units that became active early during training. We hypothesize that this is because the second layer hidden units in a DBM must both learn to model correlations in the first layer induced by the data and to counteract correlations in the first layer induced by the model family. When the second layer weights are set to 0, the DBM prior acts to correlate hidden units that have similar weight vectors (see Section 5.2).

The PD-DBM model avoids this problem. When the second layer weights are set to 0, the first layer hidden units are independent in the PD-DBM prior (essentially the S3C prior). The second layer thus has only one task: to model the correlations between first layer units induced by the data. As we will show, this hypothesis is supported by the fact that we are able to succesfully train a two layer PD-DBM without greedy pre-training.

4 Inference Procedures

The goal of variational inference is to maximize the lower bound on the log likelihood with respect to the approximate distribution Q over the unobserved variables. This is accomplished by selecting the Q that minimizes the Kullback–Leibler divergence:

$$\mathcal{D}_{KL}(Q(h,s)\|P(h,s|v)) \quad (2)$$

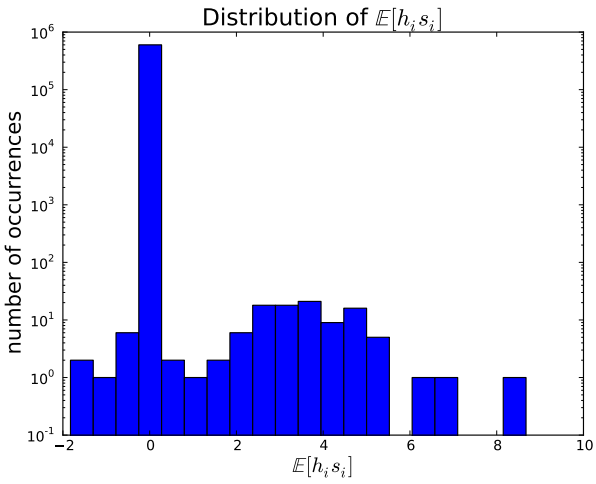

Fig. 2. This example histogram of $\mathbb{E}_Q[h_i s_i]$ shows that Q is a sparse distribution. For this 6,000 hidden unit S3C model trained on 6×6 image patches, $Q(h_i = 1) < .01$ 99.7% of the time.

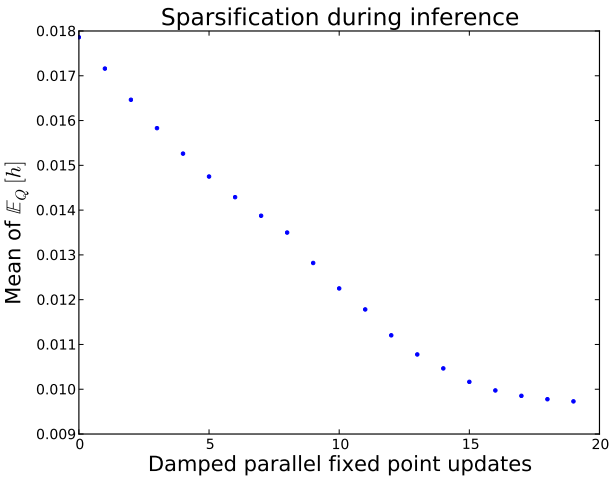

Fig. 3. The explaining-away effect makes the S3C representation become more sparse with each damped iteration of the variational inference fixed point equations.

where $Q(h,s)$ is drawn from a restricted family of distributions. This family can be chosen to ensure that learning and inference with Q is tractable.

Variational inference can be seen as analogous to the encoding step of the tradiational sparse coding algorithm. The key difference is that while sparse coding approximates the true posterior with a MAP point estimate of the latent variables, variational inference approximates the true posterior everywhere with the distribution Q.

4.1 Variational inference for S3C

When working with S3C, we constrain Q to be drawn from the family $Q(h,s) = \Pi_i Q(h_i, s_i)$. This is a richer approximation than the fully factorized family used in

the mean field approximation. It allows us to capture the tight correlation between each spike variable and its corresponding slab variable while still allowing simple and efficient inference in the approximating distribution. It also avoids a pathological condition in the mean field distribution where $Q(s_i)$ can never be updated if $Q(h_i) = 0$.

Observing that eq. (2) is an instance of the Euler-Lagrange equation, we find that the solution must take the form

$$Q(h_i = 1) = \hat{h}_i,$$
$$Q(s_i \mid h_i) = \mathcal{N}(s_i \mid h_i \hat{s}_i,\ (\alpha_i + h_i W_i^T \beta W_i)^{-1}) \quad (3)$$

where $\hat{h}_i$ and $\hat{s}_i$ must be found by an iterative process. In a typical application of variational inference, the iterative process consists of sequentially applying fixed point equations that give the optimal value of the parameters $\hat{h}_i$ and $\hat{s}_i$ for one factor $Q(h_i, s_i)$ given the value all of the other factors' parameters. This is for example the approach taken by Titsias and Lázaro-Gredilla (2011) who independently developed a variational inference procedure for the same problem. This process is only guaranteed to decrease the KL divergence if applied to each factor sequentially, i.e. first updating $\hat{h}_1$ and $\hat{s}_1$ to optimize $Q(h_1, s_1)$, then updating $\hat{h}_2$ and $\hat{s}_2$ to optimize $Q(h_2, s_2)$, and so on. In a typical application of variational inference, the optimal values for each update are simply given by the solutions to the Euler-Lagrange equations. For S3C, we make three deviations from this standard approach.

Because we apply S3C to very large-scale problems, we need an algorithm that can fully exploit the benefits of parallel hardware such as GPUs. Sequential updates across all N factors require far too much run-time to be competitive in this regime.

We have considered two different methods that enable parallel updates to all units. In the first method, we start each iteration by partially minimizing the KL divergence with respect to $\hat{s}$. The terms of the KL divergence that depend on $\hat{s}$ make up a quadratic function so this can be minimized via conjugate gradient descent. We implement conjugate gradient descent efficiently by using the R-operator (Pearlmutter, 1994) to perform Hessian-vector products rather than computing the entire Hessian explicitly (Schraudolph, 2002). This step is guaranteed to improve the KL divergence on each iteration. We next update $\hat{h}$ in parallel, shrinking the update by a damping coefficient. This approach is not guaranteed to decrease the KL divergence on each iteration but it is a widely applied approach that works well in practice (Koller and Friedman, 2009).

With the second method (Algorithm 1), we find in practice that we obtain faster convergence, reaching equally good solutions by replacing the conjugate gradient update to $\hat{s}$ with a more heuristic approach. We use a parallel damped update on $\hat{s}$ much like what we do for $\hat{h}$. In this case we make an additional heuristic modification to the update rule which is made necessary

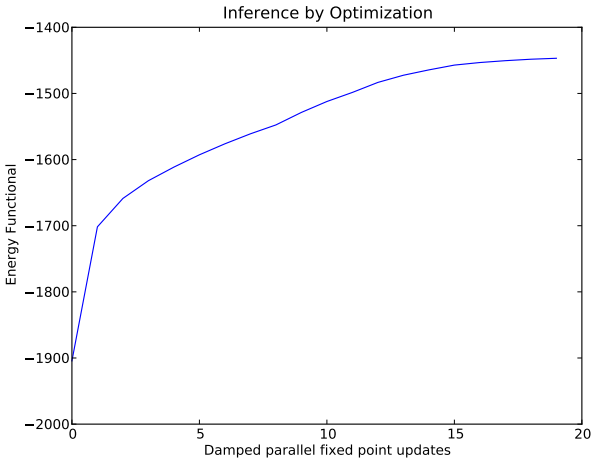

Fig. 4. The negative variational free energy of a batch of 5000 image patches increases during the course of variational inference.

by the unbounded nature of $\hat{s}$. We clip the update to $\hat{s}$ so that if $\hat{s}_{\text{new}}$ has the opposite sign from $\hat{s}$, its magnitude is at most $\rho|\hat{s}|$. In all of our experiments we used $\rho = 0.5$ but any value in $[0, 1]$ is sensible. This prevents a case where multiple mutually inhibitory s units inhibit each other so strongly that rather than being driven to 0 they change sign and actually increase in magnitude. This case is a failure mode of the parallel updates that can result in $\hat{s}$ amplifying without bound if clipping is not used.

Note that Algorithm 1 does not specify a convergence criterion. Many convergence criteria are possible–the convergence criterion could be based on the norm of the gradient of the KL divergence with respect to the variational parameters, the amount that the KL divergence has decreased in the last iteration, or the amount that the variational parameters have changed in the final iteration. Salakhutdinov and Hinton (2009) use the third approach when training deep Boltzmann machines and we find that it works well for S3C and the PD-DBM as well.

We include some visualizations that demonstrate the effect of our inference procedure. Figure 2 shows that it produces a sparse representation. Figure 3 shows that the explaining-away effect incrementally makes the representation more sparse. Figure 4 shows that the inference procedure increases the negative variational free energy.

4.2 Variational inference for the PD-DBM

Inference in the PD-DBM is very similar to inference in S3C. We use the variational family

$$Q(s, h) = \Pi_{i=1}^{N_0} Q(s_i, h_i^{(0)}) \Pi_{l=1}^{L} \Pi_{i=1}^{N_l} Q(h_i^{(l)})$$

whose solutions take the form

$$Q(h_i^{(l)} = 1) = \hat{h}_i^{(l)},$$
$$Q(s_i \mid h_i^{(0)}) = \mathcal{N}(s_i \mid h_i^{(0)} \hat{s}_i,\ (\alpha_i + h_i W_i^T \beta W_i)^{-1}).$$

Algorithm 1 `Fixed-Point Inference`

Initialize $\hat{h}(0) = \sigma(b)$, $\hat{s}(0) = \mu$, and $k = 0$.
while not converged **do**
Compute the individually optimal value $\hat{s}_i^*$ for each i simultaneously:

$$\hat{s}_i^* = \frac{\mu_i \alpha_{ii} + v^T \beta W_i - W_i \beta \left[\sum_{j \neq i} W_j \hat{h}_j \hat{s}_j(k)\right]}{\alpha_{ii} + W_i^T \beta W_i}$$

Clip reflections by assigning

$$c_i = \rho \text{sign}(\hat{s}_i^*) |\hat{s}_i(k)|$$

for all i such that $\text{sign}(\hat{s}_i^*) \neq \text{sign}(\hat{s}_i(k))$ and $|\hat{s}_i^*| > \rho |\hat{s}_i(k)|$, and assigning $c_i = \hat{s}_i^*$ for all other i.
Damp the updates by assigning

$$\hat{s}(k+1)_i = \eta_s c + (1 - \eta_s)\hat{s}(k)$$

where $\eta_s \in (0, 1]$.
Compute the individually optimal values for $\hat{h}$:

$$z_i = \left(v - \sum_{j \neq i} W_j \hat{s}_j(k+1) \hat{h}_j(k) - \frac{1}{2} W_i \hat{s}_i(k+1) \right)^T \beta W_i \hat{s}_i(k+1) + b_i - \frac{1}{2} \alpha_{ii} (\hat{s}_i(k+1) - \mu_i)^2 - \frac{1}{2} \log(\alpha_{ii} + W_i^T \beta W_i) + \frac{1}{2} \log(\alpha_{ii})$$

$$\hat{h}_i^* = \sigma(z)$$

Damp the update to $\hat{h}$:

$$\hat{h}(k+1) = \eta_h \hat{h}^* + (1 - \eta_h)\hat{h}(k)$$

$k \leftarrow k + 1$
end while

We apply more or less the same inference procedure as in S3C. On each update step we update either $\hat{s}$ or $\hat{h}^{(l)}$ for some value of l. The update to $\hat{s}$ is exactly the same as in S3C. The update to $\hat{h}^{(0)}$ changes slightly to incorporate top-down influence from $\hat{h}^{(1)}$. When computing the individually optimal values of the elements of $\hat{h}^{(0)}$ we use the following fixed-point formula:

$$\hat{h}_i^{(0)*} = \sigma(z_i + W^{(1)} \hat{h}^{(1)})$$

The update to $\hat{h}^{(l)}$ for $l > 0$ is simple; it is the same as the mean field update in the DBM. No damping is necessary for this update. The conditional independence properties of the DBM guarantee that the optimal values of the elements of $\hat{h}^{(l)}$ do not depend on each other, so the individually optimal values are globally optimal (for a given $\hat{h}^{(l-1)}$ and $\hat{h}^{(l+1)}$). The update is given by

$$\hat{h}^{(l)*} = \sigma\left(b^{(l)} + \hat{h}^{(l-1)T} W^{(l)} + W^{(l+1)} \hat{h}^{(l+1)}\right)$$

where the term for layer $l + 1$ is dropped if $l + 1 > L$.

5 Comparison to other feature encoding methods

Here we compare S3C as a feature discovery algorithm to other popular approaches. We describing how S3C occupies a middle ground between two of these methods, sparse coding and the ssRBM, while avoiding many of the respective disadvantages when applied as feature discovery algorithms.

5.1 Comparison to sparse coding

Sparse coding (Olshausen and Field, 1997) has been widely used to discover features for classification (Raina *et al.*, 2007). Recently Coates and Ng (2011) showed that this approach achieves excellent performance on the CIFAR-10 object recognition dataset. Sparse coding refers to a class of generative models where the observed data v is normally distributed given a set of continuous latent variables s and a dictionary matrix W: $v \sim \mathcal{N}(Ws, \sigma \mathbb{I})$. Sparse coding places a factorial and heavy tailed prior distribution over s (e.g. a Cauchy or Laplace distribution) chosen to encourage the mode of the posterior $p(s \mid v)$ to be sparse. One can derive the S3C model from sparse coding by replacing the factorial Cauchy or Laplace prior with a spike-and-slab prior.

One drawback of sparse coding is that the latent variables are not merely encouraged to be sparse; they are encouraged to remain close to 0, even when they are active. This kind of regularization is not necessarily undesirable, but in the case of simple but popular priors such as the Laplace prior (corresponding to an L_1 penalty on the latent variables s), the degree of regularization on active units is confounded with the degree of sparsity. There is little reason to believe that in realistic settings, these two types of complexity control should be so tightly bound together. The S3C model avoids this issue by controlling the sparsity of units via the b parameter that determines how likely each spike unit is to be active, while separately controlling the

magnitude of active units via the μ and α parameters that govern the distribution over s. Sparse coding has no parameter analogous to μ and cannot control these aspects of the posterior independently.

Another drawback of sparse coding is that the factors are not actually sparse in the generative distribution. Indeed, each factor is zero with probability zero. The features extracted by sparse coding are only sparse because they are obtained via MAP inference. In the S3C model, the spike variables ensure that each factor is zero with non-zero probability in the generative distribution. Since this places a greater restriction on the code variables, we hypothesize that S3C features provide more of a regularizing effect when solving classification problems.

Sparse coding is also difficult to integrate into a deep generative model of data such as natural images. While Yu *et al.* (2011) and Zeiler *et al.* (2011) have recently shown some success at learning hierarchical sparse coding, our goal is to integrate the feature extraction scheme into a proven generative model framework such as the deep Boltzmann machine (Salakhutdinov and Hinton, 2009). Existing inference schemes known to work well in the DBM-type (deep Boltzmann machine) setting are all either sample-based or are based on variational approximations to the model posteriors, while sparse coding schemes typically employ MAP inference. Our use of variational inference makes the S3C framework well suited to integrate into the known successful strategies for learning and inference in DBM models. In fact, the compatibility of the S3C and DBM inference procedures is confirmed by the success of the PD-DBM inference procedure. It is not obvious how one can employ a variational inference strategy to standard sparse coding with the goal of achieving sparse feature encoding.

Sparse coding models can be learned efficiently by alternately running MAP inference for several examples and then making a large, closed-form updates to the parameters. The same approach is also possible with S3C, and is in fact more principled since it is based on maximizing a variational lower bound rather than the MAP approximation. We do not explore this learning method for S3C in this paper.

5.2 Comparison to restricted Boltzmann machines

The S3C model also resembles another class of models commonly used for feature discovery: the RBM. An RBM (Smolensky, 1986) is a model defined through an energy function that describes the interactions between the observed data variables and a set of latent variables. It is possible to interpret the S3C as an energy-based model, by rearranging $p(v, s, h)$ to take the form $\exp\{-E(v, s, h)\}/Z$, with the following energy function:

$$E(v,s,h) = \frac{1}{2}\left(v - \sum_i W_i s_i h_i\right)^T \beta \left(v - \sum_i W_i s_i h_i\right) + \frac{1}{2}\sum_{i=1}^{N} \alpha_i (s_i - \mu_i h_i)^2 - \sum_{i=1}^{N} b_i h_i, \quad (4)$$

The ssRBM model family is a good starting point for S3C because it has demonstrated both reasonable performance as a feature discovery scheme and remarkable performance as a generative model (Courville *et al.*, 2011b). Within the ssRBM family, S3C's closest relative is a variant of the μ-ssRBM, defined by the following energy function:

$$E(v,s,h) = -\sum_{i=1}^{N} v^T \beta W_i s_i h_i + \frac{1}{2} v^T \beta v + \frac{1}{2}\sum_{i=1}^{N} \alpha_i (s_i - \mu_i h_i)^2 - \sum_{i=1}^{N} b_i h_i, \quad (5)$$

where the variables and parameters are defined identically to those in S3C. Comparison of equations 4 and 5 reveals that the simple addition of a latent factor interaction term $\frac{1}{2}(h \circ s)^T W^T \beta W (h \circ s)$ to the ssRBM energy function turns the ssRBM into the S3C model. With the inclusion of this term S3C moves from an undirected ssRBM model to the directed graphical model described in equation (1). This change from undirected modeling to directed modeling has three important effects, that we describe in the following paragraphs:

The effect on the partition function: The most immediate consequence of the transition to directed modeling is that the partition function becomes tractable. Because the RBM parition function is intractable, most training algorithms for the RBM require making stochastic approximations to the partition function, the same as our learning procedure for the PD-DBM does. Since the S3C partition function is tractable, we can follow its true gradient, which provides one advantage over the RBM. The partition function of S3C is also guaranteed to exist for all possible settings of the model parameters, which is not true of the ssRBM. In the ssRBM, for some parameter values, it is possible for $p(s, v \mid h)$ to take the form of a normal distribution whose covariance matrix is not positive definite. Courville *et al.* (2011b) have explored resolving this issue by constraining the parameters, but this was found to hurt classification performance.

The effect on the posterior: RBMs have a factorial posterior, but S3C and sparse coding have a complicated posterior due to the "explaining away" effect. For this reason, RBMs can use exact inference and maximum likelihood estimation. Models with an intractable posterior such as S3C and DBMs must use approximate inference and are often trained with a variational lower bound on the likelihood.

The RBMs factorial posterior means that features defined by similar basis functions will have similar activations, while in directed models, similar features will compete so that only the most relevant features will remain significantly active. As shown by Coates and Ng (2011), the sparse Gaussian RBM is not a very good feature extractor – the set of basis functions W learned by the RBM actually work better for supervised learning when these parameters are plugged into a sparse coding model than when the RBM itself is used for feature

extraction. We think this is due to the factorial posterior. In the vastly overcomplete setting, being able to selectively activate a small set of features that cooperate to explain the input likely provides S3C a major advantage in discriminative capability.

Considerations of biological plausibility also motivate the use of a model with a complicated posterior. As described in (Hyvärinen *et al.*, 2009), a phenomenon called "end stopping" similar to explaining away has been observed in V1 simple cells. End-stopping occurs when an edge detector is inhibited when retinal cells near the ends of the edge it detects are stimulated. The inhibition occurs due to lateral interactions with other simple cells, and is a major motivation for the lateral interactions present in the sparse coding posterior.

The effect on the prior: The addition of the interaction term causes S3C to have a factorial prior. This probably makes it a poor generative model, but this is not a problem for the purpose of feature discovery. Moreover, the quality of the generative model can be improved by incorporating S3C into a deeper architecture, as we will show.

RBMs were designed with a nonfactorial prior because factor models with factorial priors are generally known to result in poor generative models. However, in the case of real-valued data, typical RBM priors are not especially useful. For example, the ssRBM variant described in eq. (5) has the following prior:

$$p(s,h) \propto \exp\left\{ \frac{1}{2}\left(\sum_{i=1}^{N} W_i s_i h_i\right)^T \beta \left(\sum_{i=1}^{N} W_i s_i h_i\right) - \frac{1}{2}\sum_{i=1}^{N} \alpha_i \left(s_i - \mu_i h_i\right)^2 + \sum_{i=1}^{N} b_i h_i \right\}.$$

It is readily apparent from the first term (all other terms factorize across hidden units) that this prior acts to correlate units that have similar basis vectors, which is almost certainly not a desirable property for feature extraction tasks. Indeed it is this nature of the RBM prior that causes both the desirable (easy computation) and undesirable (no explaining away) properties of the posterior.

5.3 Other related work

The notion of a spike-and-slab prior was established in statistics by Mitchell and Beauchamp (1988). Outside the context of unsupervised feature discovery for supervised learning, the basic form of the S3C model (i.e. a spike-and-slab latent factor model) has appeared a number of times in different domains (Lücke and Sheikh, 2011; Garrigues and Olshausen, 2008; Mohamed *et al.*, 2012; Zhou *et al.*, 2009; Titsias and Lázaro-Gredilla, 2011). To this literature, we contribute an inference scheme that scales to the kinds of object classifications tasks that we consider. We outline this inference scheme next.

6 Runtime results

Our inference scheme achieves very good computational performance, both in terms of memory consumption and in terms of run-time. The computational bottleneck in our classification pipeline is SVM training, not feature learning or feature extraction.

Comparing the computational cost of our inference scheme to others is a difficult task because it could be confounded by differences in implementation and because it is not clear exactly what sparse coding problem is equivalent to an equivalent spike-and-slab sparse coding problem. However, we observed informally during our supervised learning experiments that feature extraction using S3C took roughly the same amount of time as feature extraction using sparse coding.

In Fig. 5, we show that our improvements to spike-and-slab inference performance allow us to scale spike-and-slab modeling to the problem sizes needed for object recognition tasks. Previous work on spike-and-slab modeling was not able to use similar amounts of hidden units or training examples.

As a large-scale test of our inference scheme's ability, we trained over 8,000 densely-connected filters on full 32×32 color images. A visualization of the learned filters is shown in Fig. 6. This test demonstrated that our approach scales well to large (over 3,000 dimensional) inputs, though it is not yet known how to use features for classification as effectively as patch-based features which can be incorporated into a convolutional architecture with pooling. For comparison, to our knowledge the largest image patches used in previous spike-and-slab models with lateral interactions were 16×16 (Garrigues and Olshausen, 2008).

Finally, we performed a series of experiments to compare our heuristic method of updating $\hat{s}$ with the conjugate gradient method of updating $\hat{s}$. The conjugate gradient method is guaranteed to reduce the KL divergence on each update to $\hat{s}$. The heuristic method has no such guarantee. These experiments provide an empirical justification for the use of the heuristic method.

We considered three different models, each on a different dataset. We used MNIST (LeCun *et al.*, 1998), CIFAR-100 (Krizhevsky and Hinton, 2009) , and whitened 6×6 patches drawn from CIFAR-100 as the three datasets.

Because we wish to compare different inference algorithms and inference affects learning, we did not want to compare the algorithms on models whose parameters were the result of learning. Instead we obtained the value of W by drawing randomly selected patches ranging in size from 6×6 to the full image size for each dataset. This provides a data-driven version of W with some of the same properties like local support that learned filters tend to have. None of the examples used to initialize W were used in the later timing experiments. We initialized b, μ, α, and β randomly. We used 400 hidden units for some experiments and 1600 units for others, to investigate the effect of overcompleteness on runtime.

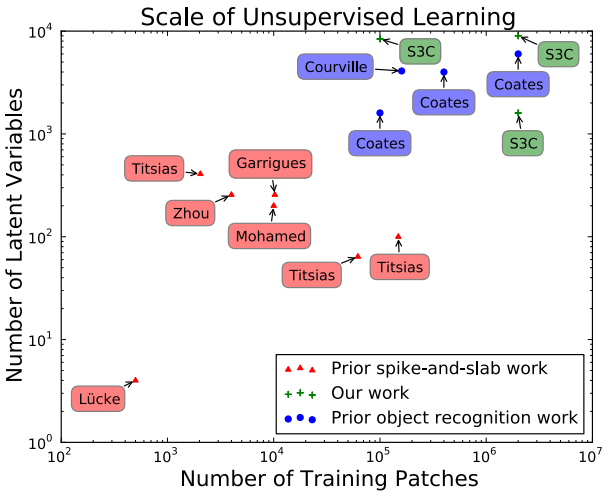

Fig. 5. Our inference scheme enables us to extend spike-and-slab modeling from small problems to the scale needed for object recognition. Previous object recognition work is from (Coates and Ng, 2011; Courville *et al.*, 2011b). Previous spike-and-slab work is from (Mohamed *et al.*, 2012; Zhou *et al.*, 2009; Garrigues and Olshausen, 2008; Lücke and Sheikh, 2011; Titsias and Lázaro-Gredilla, 2011).

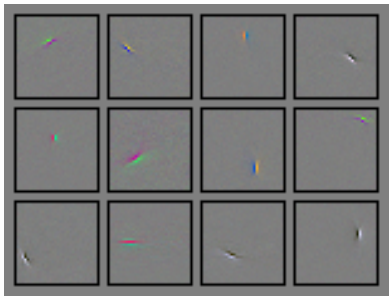

Fig. 6. Example filters from a dictionary of over 8,000 learned on full 32x32 images.

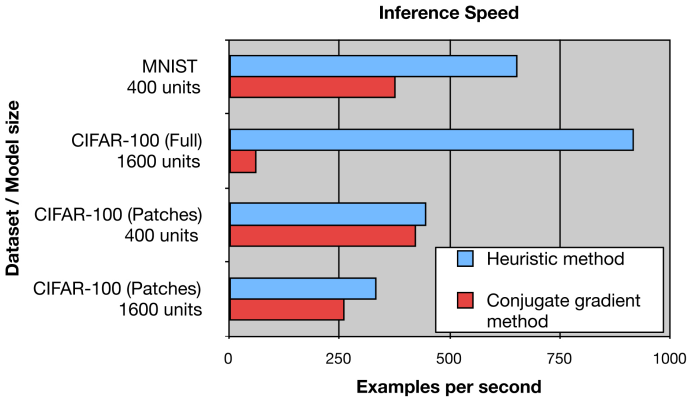

Fig. 7. The inference speed for each method was computed based on the inference time for the same set 100 examples from each dataset. The heuristic method is consistently faster than the conjugate gradient method. The conjugate gradient method is slowed more by problem size than the heuristic method is, as shown by the conjugate gradient method's low speed on the CIFAR-100 full image task. The heuristic method has a very low cost per iteration but is strongly affected by the strength of explaining-away interactions–moving from CIFAR-100 full images to CIFAR-100 patches actually slows it down because the degree of overcompleteness increases.

For each inference scheme considered, we found the fastest possible variant obtainable via a two-dimensional grid search over η_h and either η_s in the case of the heuristic method or the number of conjugate gradient steps to apply per $\hat{s}$ update in the case of the conjugate gradient method. We used the same value of these parameters on every pair of update steps. It may be possible to obtain faster results by varying the parameters throughout the course of inference.

For these timing experiments, it is necessary to make sure that each algorithm is not able to appear faster by converging early to an incorrect solution. We thus replace the standard convergence criterion based on the size of the change in the variational parameters with a requirement that the KL divergence reach within 0.05 on average of our best estimate of the true minimum value of the KL divergence found by batch gradient descent.

All experiments were performed on an Nvidia GeForce GTX-580.

The results are summarized in Fig. 7.

7 Classification Results

Because S3C forms the basis of all further model development in this line of research, we concentrate on validating its value as a feature discovery algorithm. We conducted experiments to evaluate the usefulness of S3C features for supervised learning on the CIFAR-10 and CIFAR-100 (Krizhevsky and Hinton, 2009) datasets. Both datasets consist of color images of objects such as animals and vehicles. Each contains 50,000 train and 10,000 test examples. CIFAR-10 contains 10 classes while CIFAR-100 contains 100 classes, so there are fewer labeled examples per class in the case of CIFAR-100.

For all experiments, we used the same overall procedure as Coates and Ng (2011) except for feature learning. CIFAR-10 consists of 32×32 images. We train our feature extractor on 6×6 contrast-normalized and ZCA-whitened patches from the training set (this preprocessing step is not necessary to obtain good performance with S3C; we included it primarily to facilitate comparison with other work). At test time, we extract features from all 6×6 patches on an image, then average-pool them. The average-pooling regions are arranged on a non-overlapping grid. Finally, we train an L2-SVM with a linear kernel on the pooled features.

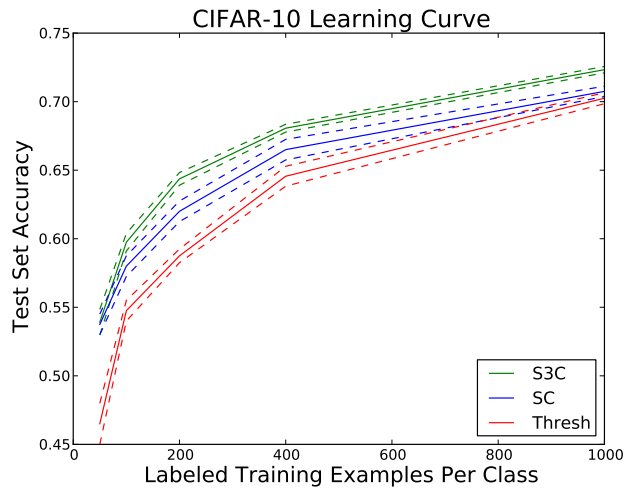

Fig. 8. Semi-supervised classification accuracy on subsets of CIFAR-10. Thresholding, the best feature extractor on the full dataset, performs worse than sparse coding when few labels are available. S3C improves upon sparse coding's advantage.

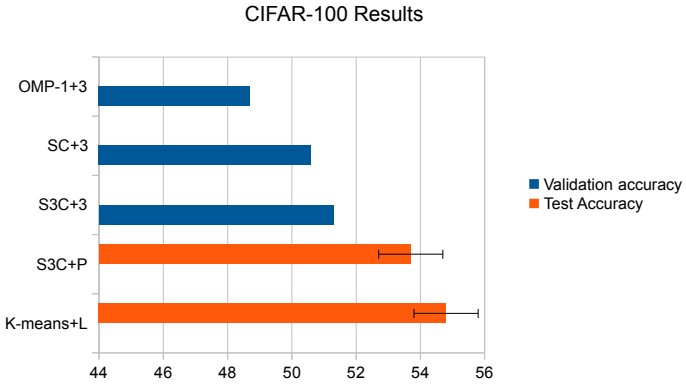

Fig. 9. CIFAR-100 classification accuracy for various models. As expected, S3C outperforms SC (sparse coding) and and OMP-1. S3C with spatial pyramid pooling is near the state-of-the-art method, which uses a learned pooling structure.

7.1 CIFAR-10

We use CIFAR-10 to evaluate our hypothesis that S3C is similar to a more regularized version of sparse coding.

Coates and Ng (2011) used 1600 basis vectors in all of their sparse coding experiments. They post-processed the sparse coding feature vectors by splitting them into the positive and negative part for a total of 3200 features per average-pooling region. They average-pool on a 2×2 grid for a total of 12,800 features per image (i.e. each element of the 2×2 grid averages over a block with sides $\lceil (32 - 6 + 1)/2 \rceil$ or $\lfloor (32 - 6 + 1)/2 \rfloor$). We used $\mathbb{E}_Q[h]$ as our feature vector. Unlike the output of sparse coding, this does not have a negative part, so using a 2×2 grid we would have only 6,400 features. In order to compare with similar sizes of feature vectors we used a 3×3 pooling grid for a total of 14,400 features (i.e. each element of the 3×3 grid averages over 9×9 locations) when evaluating S3C. To ensure this is a fair means of comparison, we confirmed that running sparse coding with a 3×3 grid and absolute value rectification performs worse than sparse coding with a 2×2 grid and sign splitting (76.8% versus 77.9% on the validation set).

We tested the regularizing effect of S3C by training the SVM on small subsets of the CIFAR-10 training set, but using features that were learned on patches drawn from the entire CIFAR-10 train set. The results, summarized in Figure 8, show that S3C has the advantage over both thresholding and sparse coding for a wide range of amounts of labeled data. (In the extreme low-data limit, the confidence interval becomes too large to distinguish sparse coding from S3C).

On the full dataset, S3C achieves a test set accuracy of 78.3 ± 0.9 % with 95% confidence. Coates and Ng (2011) do not report test set accuracy for sparse coding with "natural encoding" (i.e., extracting features in a model whose parameters are all the same as in the model used for training) but sparse coding with different parameters for feature extraction than training achieves an accuracy of $78.8 \pm 0.9\%$ (Coates and Ng, 2011). Since we have not enhanced our performance by modifying parameters at feature extraction time these results seem to indicate that S3C is roughly equivalent to sparse coding for this classification task. S3C also outperforms ssRBMs, which require 4,096 basis vectors per patch and a 3×3 pooling grid to achieve $76.7 \pm 0.9\%$ accuracy. All of these approaches are close to the best result, using the pipeline from Coates and Ng (2011), of 81.5% achieved using thresholding of linear features learned with OMP-1. These results show that S3C is a useful feature extractor that performs comparably to the best approaches when large amounts of labeled data are available.

7.2 CIFAR-100

Having verified that S3C features help to regularize a classifier, we proceed to use them to improve performance on the CIFAR-100 dataset, which has ten times as many classes and ten times fewer labeled examples per class. We compare S3C to two other feature extraction methods: OMP-1 with thresholding, which Coates and Ng (2011) found to be the best feature extractor on CIFAR-10, and sparse coding, which is known to perform well when less labeled data is available. We evaluated only a single set of hyperparameters for S3C. For sparse coding and OMP-1 we searched over the same set of hyperparameters as Coates and Ng (2011) did: $\{0.5, 0.75, 1.0, 1.25, 1.25\}$ for the sparse coding penalty and $\{0.1, 0.25, 0.5, 1.0\}$ for the thresholding value. In order to use a comparable amount of computational resources in all cases, we used at most 1600 hidden units and a 3×3 pooling grid for all three methods. For S3C, this was the only feature encoding we evaluated. For SC (sparse coding) and OMP-1, which double their number

of features via sign splitting, we also evaluated 2×2 pooling with 1600 latent variables and 3×3 pooling with 800 latent variables to be sure the models do not suffer from overfitting caused by the larger feature set. These results are summarized in Fig. 9.

The best result to our knowledge on CIFAR-100 is $54.8 \pm 1\%$ (Jia and Huang, 2011), achieved using a learned pooling structure on top of "triangle code" features from a dictionary learned using k-means. This feature extractor is very similar to thresholded OMP-1 features and is known to perform slightly worse on CIFAR-10. The validation set results, which all use the same control pooling layer, in Fig. 9 show that S3C is the best known detector layer on CIFAR-100. Using a pooling strategy of concatenating 1×1, 2×2 and 3×3 pooled features we achieve a test set accuracy of $53.7 \pm 1\%$.

7.3 Transfer learning challenge

For the NIPS 2011 Workshop on Challenges in Learning Hierarchical Models (Le *et al.*, 2011b), the organizers proposed a transfer learning competition. This competition used a dataset consisting of 32×32 color images, including 100,000 unlabeled examples, 50,000 labeled examples of 100 object classes not present in the test set, and 120 labeled examples of 10 object classes present in the test set. The test set was not made public until after the competition. We recognized this contest as a chance to demonstrate S3C's ability to perform well with extremely small amounts of labeled data. We chose to disregard the 50,000 labels and treat this as a semi-supervised learning task.

We applied the same approach as on the CIFAR datasets, albeit with a small modification to the SVM training procedure. Due to the small labeled dataset size, we used leave-one-out cross validation rather than five fold cross validation.

We won the competition, with a test set accuracy of 48.6 %. We do not have any information about the competing entries, other than that we outperformed them. Our test set accuracy was tied with a method run by the contest organizers, based on a combination of methods (Coates *et al.*, 2011; Le *et al.*, 2011a). Since these methods do not use transfer learning either, this suggests that the contest primarily provides evidence that S3C is a powerful semi-supervised learning tool.

7.4 Ablative analysis

In order to better understand which aspects of our S3C object classification method are most important to obtaining good performance, we conducted a series of ablative analysis experiments. For these experiments we trained on 5,000 labels of the STL-10 dataset (Coates *et al.*, 2011). Previous work on the STL-10 dataset is based on training on 1,000 label subsets of the training set, so the performance numbers in this section should only be compared to each other, not to previous work. The results are presented in Fig. 10.

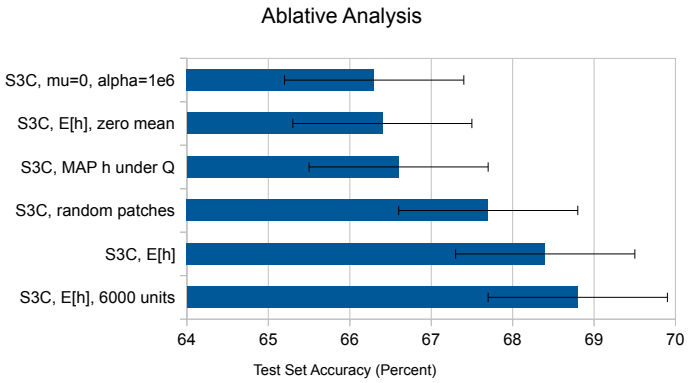

Fig. 10. Performance of several limited variants of S3C

Our best-performing method uses $\mathbb{E}_Q[h]$ as features. This allows us to abstract out the s variables, so that they achieve a form of per-component brightness invariance. Our experiments show that including the s variables or using MAP inference in Q rather than an expectation hurts classification performance. We experimented with fixing μ to 0 so that s is regularized to be small as well as sparse, as in sparse coding. We found that this hurts performance even more. Lastly we experimented with replacing S3C learning with simply assigning W to be a set of randomly selected patches from the training set. We call this approach S3C-RP. We found that this does not impair performance much, so learning is not very important compared to our inference algorithm. This is consistent with Coates and Ng (2011)'s observations that the feature extractor matters more than the learning algorithm and that learning matters less for large numbers of hidden units.

8 Sampling Results

In order to demonstrate the improvements in the generative modeling capability conferred by adding a DBM prior on h, we trained an S3C model and a PD-DBM model on the MNIST dataset. We chose to use MNIST for this portion of the experiments because it is easy for a human observer to qualitatively judge whether samples come from the same distribution as this dataset.

For the PD-DBM, we used $L = 1$, for a total of two hidden layers. We did not use greedy, layerwise pretraining– the entire model was learned jointly. Such joint learning without greedy pretraining has never been accomplished with similar deep models such as DBMs or DBNs.

The S3C samples and basis vectors are shown in Fig. 11. The samples do not resemble digits, suggesting that S3C has failed to model the data. However, inspection of the S3C filters shows that S3C has learned a good basis set for representing MNIST digits using digit templates, pen strokes, etc. It simply does not have the correct prior on these bases and as a result activates subsets of them that do not correspond to MNIST digits. The PD-DBM samples clearly resemble digits, as shown in Fig. 12. For

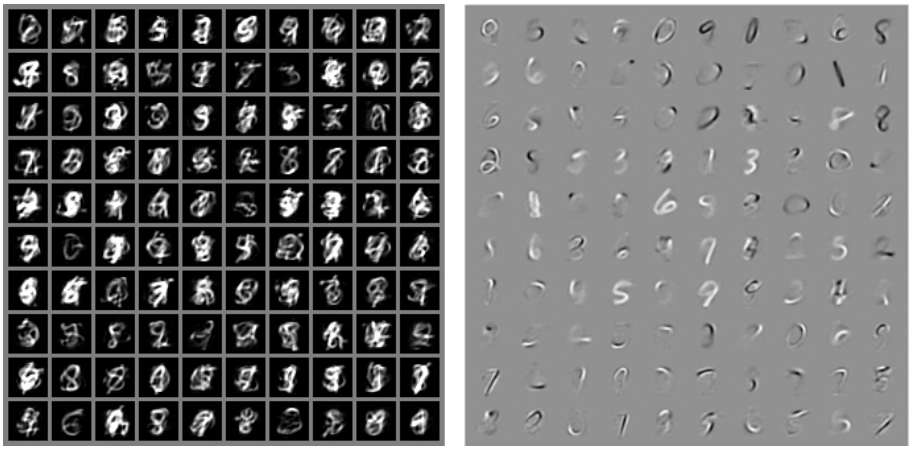

Fig. 11. *Left:* Samples drawn from an S3C model trained on MNIST. *Right:* The filters used by this S3C model.

Fig. 12. *Left:* Samples drawn from a PD-DBM model trained on MNIST using joint training only. *Center:* Samples drawn from a DBM model of the same size, trained using greedy layerwise pretraining followed by joint training. *Right:* Samples drawn from a DBM trained using joint training only

comparison, Fig. 12 also shows samples from two DBMs. In all cases we display the expected value of the visible units given the hidden units.

The first DBM was trained by running the demo code that accompanies (Salakhutdinov and Hinton, 2009). We used the same number of *units* in each layer in order to make these models comparable (500 in the first layer and 1,000 in the second). This means that the PD-DBM has a slightly greater number of *parameters* than the DBM, since the first layer units of the PD-DBM have both mean and precision parameters while the first layer units of the DBM have only a bias parameter. Note that the DBM operates on a binarized version of MNIST while S3C and the PD-DBM regard MNIST as real-valued. Additionally, the DBM demo code uses the MNIST labels during generative training while the PD-DBM and S3C were not trained with the benefit of the labels. The DBM demo code is hardcoded to pretrain the first layer for 100 epochs, the second layer for 200 epochs, and then jointly train the DBM for 300 epochs. We trained the PD-DBM starting from a random initialization for 350 epochs.

The second DBM was trained using two modifications from the demo code in order to train it in as similar a fashion to our PD-DBM model as possible: first, it was trained without access to labels, and second, it did not receive any pretraining. This model was trained for only 230 epochs because it had already converged to a bad local optimum by this time. This DBM is included to provide an example of how DBM training fails when greedy layerwise pretraining is not used. DBM training can fail in a variety of ways and no example should be considered representative of all of them.

To analyze the differences between these models, we display a visualization of the weights of the models that shows how the layers interact in Fig. 13.

9 Conclusion

We have motivated the use of the S3C model for unsupervised feature discovery. We have described a variational approximation scheme that makes it feasible to perform learning and inference in large-scale S3C and

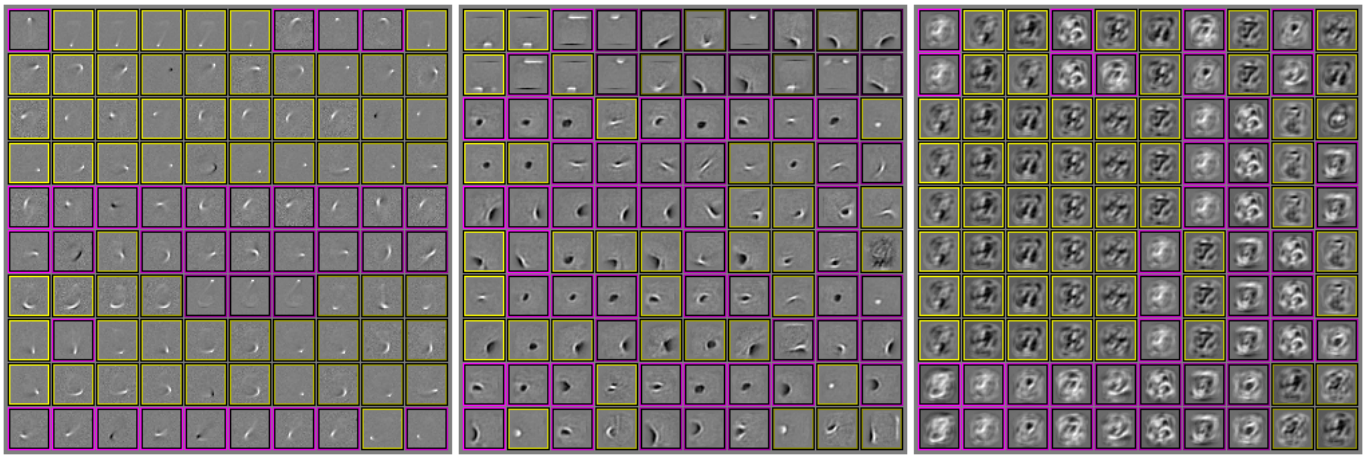

Fig. 13. Each panel shows a visualization of the weights for a different model. Each row represents a different second layer hidden unit. We show ten units for each model corresponding to those with the largest weight vector norm. Within each row, we plot the weight vectors for the ten most strongly connected first layer units. Black corresponds to inhibition, white to excitation, and gray to zero weight. This figure is best viewed in color–units plotted with a yellow border have excitatory second layer weights while units plotted with a magenta border have inhibitory second layer weights. *Left:* PD-DBM model trained jointly. Note that each row contains many similar filters. This is how the second layer weights achieve invariance to some transformations such as image translation. This is one way that deep architectures are able to disentangle factors of variation. One can also see how the second layer helps implement the correct prior for the generative task. For example, the unit plotted in the first row excites filters used to draw 7s and inhibits filters used to draw 1s. Also, observe that the first layer filters are much more localized and contained fewer templates than those in Fig. 11 right. This suggests that joint training has a significant effect on the quality of the first layer weights; greedy pretraining would have attempted to solve the generative task with more templates due to S3C's independent prior. *Center:* DBM model with greedy pretraining followed by joint training. These weights show the same disentangling and invariance properties as those of the PD-DBM. Note that the filters have more black areas. This is because the RBM must use inhibitory weights to limit hidden unit activities, while S3C accomplishes the same purpose via the explaining-away effect. *Right:* DBM with joint training only. Note that many of the second layer weight vectors are duplicates of each other. This is because the second layer has a pathological tendency to focus on modeling a handful of first-layer units that learn interesting responses earliest in learning.

PD-DBM models. We have demonstrated that S3C is an effective feature discovery algorithm for both supervised and semi-supervised learning with small amounts of labeled data. This work addresses two scaling problems: the computation problem of scaling spike-and-slab sparse coding to the problem sizes used in object recognition, and the problem of scaling object recognition techniques to work with more classes. We demonstrate that this work can be extended to a deep architecture using a similar inference procedure, and show that the deeper architecture is better able to model the input distribution. Remarkably, this deep architecture does not require greedy training, unlike its DBM predecessor.

Acknowledgments

This work was funded by DARPA and NSERC. The authors would like to thank Pascal Vincent for helpful discussions. The computation done for this work was conducted in part on computers of RESMIQ, Clumeq and SharcNet. We would like to thank the developers of Theano (Bergstra *et al.*, 2010) and pylearn2 (Warde-Farley *et al.*, 2011).

References

Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, **2**(1), 1–127. Also published as a book. Now Publishers, 2009.

Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. (2007). Greedy layer-wise training of deep networks. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19 (NIPS'06)*, pages 153–160. MIT Press.

Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., and Bengio, Y. (2010). Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*. Oral Presentation.

Coates, A. and Ng, A. Y. (2011). The importance of encoding versus training with sparse coding and vector quantization. In *ICML'2011*.

Coates, A., Lee, H., and Ng, A. Y. (2011). An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2011)*.

Courville, A., Bergstra, J., and Bengio, Y. (2011a). A Spike and Slab Restricted Boltzmann Machine. In *AISTATS'2011*.

Courville, A., Bergstra, J., and Bengio, Y. (2011b). Unsupervised models of images by spike-and-slab RBMs. In *Proceedings of the Twenty-eight International Conference on Machine Learning (ICML'11)*.

Deng, L., Seltzer, M., Yu, D., Acero, A., Mohamed, A., and Hinton, G. (2010). Binary coding of speech spectrograms using a deep auto-encoder. In *Interspeech 2010*, Makuhari, Chiba, Japan.

Desjardins, G., Courville, A. C., and Bengio, Y. (2012). On training deep Boltzmann machines. *CoRR*, **abs/1203.4416**.

Douglas, S., Amari, S.-I., and Kung, S.-Y. (1999). On gradient adaptation with unit-norm constraints.

Garrigues, P. and Olshausen, B. (2008). Learning horizontal connections in a sparse coding model of natural images. In *NIPS'07*, pages 505–512. MIT Press, Cambridge, MA.

Hinton, G. E. (2000). Training products of experts by minimizing contrastive divergence. Technical Report GCNU TR 2000-004, Gatsby Unit, University College London.

Hinton, G. E. (2010). A practical guide to training restricted Boltzmann machines. Technical Report UTML TR 2010-003, Department of Computer Science, University of Toronto.

Hinton, G. E., Osindero, S., and Teh, Y. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, **18**, 1527–1554.

Hyvärinen, A., Hurri, J., and Hoyer, P. O. (2009). *Natural Image Statistics: A probabilistic approach to early computational vision*. Springer-Verlag.

Jia, Y. and Huang, C. (2011). Beyond spatial pyramids: Receptive field learning for pooled image features. *NIPS*2011 Workshop on Deep Learning and Unsupervised Feature Learning*.

Kavukcuoglu, K., Sermanet, P., Boureau, Y.-L., Gregor, K., Mathieu, M., and LeCun, Y. (2010). Learning convolutional feature hierarchies for visual recognition. In *NIPS'10*.

Koller, D. and Friedman, N. (2009). *Probabilistic Graphical Models: Principles and Techniques*. MIT Press.

Krizhevsky, A. and Hinton, G. (2009). Learning multiple layers of features from tiny images. Technical report, University of Toronto.

Le, Q. V., Karpenko, A., Ngiam, J., and Ng, A. Y. (2011a). Ica with reconstruction cost for efficient overcomplete feature learning. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 1017–1025.

Le, Q. V., Ranzato, M., Salakhutdinov, R., Ng, A., and Tenenbaum, J. (2011b). *NIPS Workshop on Challenges in Learning Hierarchical Models: Transfer Learning and Optimization*. https://sites.google.com/site/nips2011workshop.

Le Roux, N. and Bengio, Y. (2008). Representational power of restricted Boltzmann machines and deep belief networks. *Neural Computation*, **20**(6), 1631–1649.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, **86**(11), 2278–2324.

Lücke, J. and Sheikh, A.-S. (2011). A closed-form EM algorithm for sparse coding. arXiv:1105.2493.

Mitchell, T. J. and Beauchamp, J. J. (1988). Bayesian variable selection in linear regression. *J. Amer. Statistical Assoc.*, **83**(404), 1023–1032.

Mohamed, S., Heller, K., and Ghahramani, Z. (2012). Bayesian and l1 approaches to sparse unsupervised learning. In *ICML'2012*.

Montavon, G. and Müller, K.-R. (2012). Learning feature hierarchies with cented deep Boltzmann machines. *CoRR*, **abs/1203.4416**.

Olshausen, B. A. and Field, D. J. (1997). Sparse coding with an overcomplete basis set: a strategy employed by V1? *Vision Research*, **37**, 3311–3325.

Pearlmutter, B. (1994). Fast exact multiplication by the Hessian. *Neural Computation*, **6**(1), 147–160.

Raina, R., Battle, A., Lee, H., Packer, B., and Ng, A. Y. (2007). Self-taught learning: transfer learning from unlabeled data. In Z. Ghahramani, editor, *ICML 2007*, pages 759–766. ACM.

Salakhutdinov, R. and Hinton, G. (2009). Deep Boltzmann machines. In *AISTATS'2009*.

Saul, L. K. and Jordan, M. I. (1996). Exploiting tractable substructures in intractable networks. In *NIPS'95*. MIT Press, Cambridge, MA.

Schraudolph, N. N. (2002). Fast curvature matrix-vector products for second-order gradient descent. *Neural Computation*, **14**(7), 1723–1738.

Smolensky, P. (1986). Information processing in dynamical systems: Foundations of harmony theory. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing*, volume 1, chapter 6, pages 194–281. MIT Press, Cambridge.

Tieleman, T. (2008). Training restricted Boltzmann machines using approximations to the likelihood gradient. In W. W. Cohen, A. McCallum, and S. T. Roweis, editors, *ICML 2008*, pages 1064–1071. ACM.

Titsias, M. K. and Lázaro-Gredilla, M. (2011). Spike and slab variational inference for multi-task and multiple kernel learning. In *NIPS'2011*.

Titterington, D., Smith, A., and Makov, U. (1985). *Statistical Analysis of Finite Mixture Distributions*. Wiley, New York.

Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. In *ICML 2008*.

Warde-Farley, D., Goodfellow, I., Lamblin, P., Desjardins, G., Bastien, F., and Bengio, Y. (2011). pylearn2. http://deeplearning.net/software/pylearn2.

Younes, L. (1998). On the convergence of markovian stochastic algorithms with rapidly decreasing ergodicity rates. In *Stochastics and Stochastics Models*, pages 177–228.

Yu, K., Lin, Y., and Lafferty, J. (2011). Learning image representations from the pixel level via hierarchical sparse coding. In *CVPR*.

Zeiler, M., Taylor, G., and Fergus, R. (2011). Adaptive deconvolutional networks for mid and high level feature learning. In *ICML*.

Zhou, M., Chen, H., Paisley, J. W., Ren, L., Sapiro, G., and Carin, L. (2009). Non-parametric Bayesian dictionary learning for sparse image representations. In *NIPS'09*, pages 2295–2303.

Ian J. Goodfellow received the BS and MS degrees in computer science from Stanford University in 2009. He is currently working toward the PhD degree at Université de Montréal. His research interests include machine learning and computer vision, with specific interests in probabilistic modeling and deep learning.

Aaron Courville is an Assistant Professor of the Department of Computer Science and Operations Research at the University of Montreal. His recent research interests have focused on the development of deep learning models and methods. He is particularly interested in developing probabilistic models and novel inference methods.

Yoshua Bengio is Full Professor of the Department of Computer Science and Operations Research and head of the Machine Learning Laboratory (LISA) at the University of Montreal, CIFAR Fellow in the Neural Computation and Adaptive Perception program, Canada Research Chair in Statistical Learning Algorithms, and he also holds the NSERC-Ubisoft industrial chair. His main research ambition is to understand principles of learning that yield intelligence.