Composite denoising autoencoders

Krzysztof J. Geras and Charles Sutton

Institute of Adaptive Neural Computation, School of Informatics,
The University of Edinburgh, Edinburgh, EH8 9AB, UK
k.j.geras@sms.ed.ac.uk, csutton@inf.ed.ac.uk

Abstract. In representation learning, it is often desirable to learn features at different levels of scale. For example, in image data, some edges will span only a few pixels, whereas others will span a large portion of the image. We introduce an unsupervised representation learning method called a composite denoising autoencoder (CDA) to address this. We exploit the observation from previous work that in a denoising autoencoder, training with lower levels of noise results in more specific, fine-grained features. In a CDA, different parts of the network are trained with different versions of the same input, corrupted at different noise levels. We introduce a novel cascaded training procedure which is designed to avoid types of bad solutions that are specific to CDAs. We show that CDAs learn effective representations on two different image data sets.

Keywords: denoising autoencoders, unsupervised learning, neural networks.

1 Introduction

In most applications of representation learning, we wish to learn features at different levels of scale. For example, in image data, some edges will span only a few pixels, whereas others, such as a boundary between foreground and background, will span a large portion of the image. Similarly, in speech data, different phonemes and different words vary a lot in their duration. In text data, some features in the representation might model specialized topics that use only a few words. For example a topic about electronics would often use words such as "big", "screen" and "tv". Other features model more general topics that use many different words. Good representations should model both of these phenomena, containing features at different levels of granularity.

Denoising autoencoders [28, 29, 12] provide a particularly natural framework to formalise this intuition. In a denoising autoencoder, the network is trained to be able to reconstruct each data point from a corrupted version. The noise process used to perform the corruption is chosen by the modeller, and is an important aspect of the learning process that affects the final representation. On a digit recognition task, Vincent et al. [29] noticed that using a low level of noise leads to learning blob detectors, while increasing it results in obtaining detectors of strokes or parts of digits. They also recognise that either too low or

too high level of noise harms the representation learnt. The relationship between the level of noise and spatial extent of the filters was also noticed by Karklin and Simoncelli [18] for a different feature learning model. Despite impressive practical results with denoising autoencoders (e.g. [13, 23]), how to choose the noise distribution is not fully understood.

In this paper, we introduce *composite denoising autoencoders* (CDA), in which different parts of the network receive versions of the input that are corrupted with different levels of noise. This encourages different hidden units of the network to learn features at different scales. A key challenge is that finding good parameters in a CDA requires some care, because naive training methods will cause the network to rely mostly on the low-noise corruptions, without fully training the features for the high-noise corruptions, because after all the low noise corruptions provide more information about the original input. We introduce a training method specifically for CDA that sidesteps this problem.

On two different data sets of images, we show that CDAs learn significantly better representations that standard DAs. In particular, we achieve to our knowledge the best accuracy on the CIFAR-10 data set with a permutation invariant model, outperforming scheduled denoising autoencoders [10].

2 Background

The core idea of learning a representation by learning to reconstruct artificially corrupted training data dates back at least to the work of Seung [24], who suggested using a recurrent neural network for this purpose. Using unsupervised layer-wise learning of representations for classification purposes appeared later in the work of Bengio et al. [3] and Hinton et al. [16].

The denoising autoencoder (DA) [28] is based on the same intuition as the work of Seung [24] that a good representation should contain enough information to reconstruct corrupted versions of an original input. In its simplest form, it is a single-layer feed-forward neural network. Let $\mathbf{x} \in \mathbb{R}^d$ be the input to the network. The output of the network is a hidden representation $\mathbf{y} \in \mathbb{R}^{d'}$, which is simply computed as $f_\theta(\mathbf{x}) = h(\mathbf{W}\mathbf{x} + \mathbf{b})$, where the matrix $\mathbf{W} \in \mathbb{R}^{d' \times d}$ and the vector $\mathbf{b} \in \mathbb{R}^{d'}$ are the parameters of the network, and h is a typically nonlinear transfer function, such as a sigmoid. We write $\theta = (\mathbf{W}, \mathbf{b})$. The function f is called an *encoder* because it maps the input to a hidden representation. In an autoencoder, we have also a *decoder* that "reconstructs" the input vector from the hidden representation. The decoder has a similar form to the encoder, namely, $g_{\theta'}(\mathbf{y}) = h'(\mathbf{W}'\mathbf{y} + \mathbf{b}')$, except that here $\mathbf{W}' \in \mathbb{R}^{d \times d'}$ and $\mathbf{b}' \in \mathbb{R}^d$. It can be useful to allow the transfer function h' for the decoder to be different from that for the encoder. Typically, $\mathbf{W}$ and $\mathbf{W}'$ are constrained by $\mathbf{W}' = \mathbf{W}^T$, which has been justified theoretically by Vincent [27].

During training, our objective is to learn the encoder parameters $\mathbf{W}$ and $\mathbf{b}$. As a byproduct, we will need to learn the decoder parameters $\mathbf{b}'$ as well. We do this by defining a *noise distribution* $p(\tilde{\mathbf{x}}|\mathbf{x}, \nu)$. The amount of corruption is controlled by a parameter ν. We train the autoencoder weights to be able to

reconstruct a random input from the training distribution $\mathbf{x}$ from its corrupted version $\tilde{\mathbf{x}}$ by running the encoder and the decoder in sequence. Formally, this process is described by minimising the autoencoder reconstruction error with respect to the parameters θ^* and θ'^*, i.e.,

$$\theta^*, \theta'^* = \arg\min_{\theta,\theta'} \mathbb{E}_{(X,\tilde{X})} \left[L\left(X, g_{\theta'}(f_\theta(\tilde{X}))\right) \right], \tag{1}$$

where L is a loss function over the input space, such as squared error. Typically we minimize this objective function using SGD with mini-batches, where at each iteration we sample new values for both the uncorrupted and corrupted inputs.

In the absence of noise, this model is known simply as an autoencoder or autoassociator. A classic result [2] states that when $d' < d$, then under certain conditions, an autoencoder learns the same subspace as PCA. If the dimensionality of the hidden representation is too large, i.e., if $d' > d$, then the autoencoder can obtain zero reconstruction error simply by learning the identity map. In a denoising autoencoder, in contrast, the noise forces the model to learn interesting structure even when there are a large number of hidden units. Indeed, in practical denoising autoencoders, the best results are found with *overcomplete* representations for which $d' > d$.

There are several choices to be made here, including the noise distribution, the transformations h and h' and the loss function L. For the loss function L, for continuous $\mathbf{x}$, squared error can be used. For binary $\mathbf{x}$ or $\mathbf{x} \in [0, 1]$, as we consider in this paper, it is common to use the *cross entropy* loss,

$$L(\mathbf{x}, \mathbf{z}) = -\sum_{i=1}^{D} \left(x_i \log z_i + (1 - x_i) \log (1 - z_i) \right).$$

For the transfer functions, common choices include the sigmoid $h(v) = \frac{1}{1+e^{-v}}$ for both the encoder and decoder, or to use a rectifier $h(v) = \max(0, v)$ in the encoder paired with sigmoid decoder.

One of the most important parameters in a denoising autoencoder is the noise distribution p. For continuous $\mathbf{x}$, Gaussian noise $p(\tilde{\mathbf{x}}|\mathbf{x}, \nu) = N(\tilde{\mathbf{x}}; \mathbf{x}, \nu)$ can be used. For binary $\mathbf{x}$ or $\mathbf{x} \in [0, 1]$, it is most common to use *masking noise*, that is, for each $i \in 1, 2, \ldots d$, we sample $\tilde{x}_i$ independently as

$$p(\tilde{x}_i|x_i, \nu) = \begin{cases} 0 & \text{with probability } \nu, \\ x_i & \text{otherwise.} \end{cases} \tag{2}$$

In either case, the level of noise ν affects the degree of corruption of the input. If ν is high, the inputs are more heavily corrupted during training. The noise level has a significant effect on the representations learnt. For example, if the input data are images, masking only a few pixels will bias the process of learning the representation to deal well with local corruptions. On the other hand, masking many pixels will push the algorithm to use information from more distant regions.

It is possible to train multiple layers of representations with denoising autoencoders by training a denoising autoencoder with data mapped to a representation

learnt by the encoder of another denoising autoencoder. This model is known as the stacked denoising autoencoder [28, 29]. As an alternative to stacking, constructing deep autoencoders with denoising autoencoders was explored by Xie et al. [30].

Although the standard denoising autoencoders are not, by construction, generative models, Bengio et al. [5] proved that, under mild regularity conditions, denoising autoencoders can be used to sample from a distribution which consistently estimates the data generating distribution. This method, which consists of alternately adding noise to a sample and denoising it, yields competitive performance in terms of estimated log-likelihood of the samples. An important connection was also made by Vincent [27], who showed that optimising the training objective of a denoising autoencoder is equivalent to performing score matching [17] between the Parzen density estimator of the training data and a particular energy-based model.

3 Composite denoising autoencoders

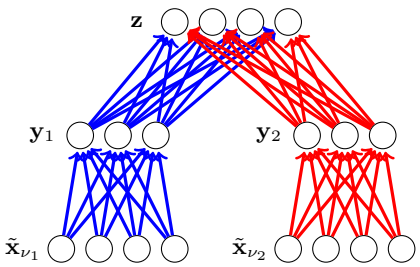

Fig. 1. A composite denoising autoencoder using two levels of noise.

Composite denoising autoencoders learn a diverse representation by leveraging the observation that the types of features learnt by the standard denoising autoencoders differ depending on the level of noise. Instead of training all of the hidden units to extract features from data corrupted with the same level of noise, we can partition the hidden units, training each subset of model's parameters with a different noise level.

More formally, let $\boldsymbol{\nu} = (\nu_1, \nu_2, \ldots, \nu_S)$ denote the set of noise levels that is to be used in the model. For each noise level ν_s the network includes a vector $\mathbf{y}_s \in \mathbb{R}^{D_s}$ of hidden units and a weight matrix $\mathbf{W}_s \in \mathbb{R}^{D_s \times d}$. Note that different noise levels may have different numbers of hidden units. We use $\mathbf{D} = (D_1, D_2, \ldots D_S)$ to denote a vector containing the number of hidden units for each noise level.

When assigning a representation to a new input $\mathbf{x}$, the CDA is very similar to the DA. In particular, the hidden representation is computed as

$$\mathbf{y}_s = h(\mathbf{W}_s \mathbf{x} + \mathbf{b}_s) \qquad \forall s \in 1, \ldots, S, \tag{3}$$

where as before h is a nonlinear transfer function such as the sigmoid. The full representation $\mathbf{y}$ for $\mathbf{x}$ is constructed by concatenating the individual representations as $\mathbf{y} = (\mathbf{y}_1, \ldots, \mathbf{y}_S)$.

Where the CDA differs from the DA is in the training procedure. Given a training input $\mathbf{x}$, we corrupt it S times, once for each level of noise, yielding corrupted vectors

$$\tilde{\mathbf{x}}_s \sim p(\tilde{\mathbf{x}}_s|\mathbf{x}, \nu_s) \qquad \forall s. \tag{4}$$

Then each of the corrupted vectors are fed into the corresponding encoders, yielding the representation

$$\mathbf{y}_s = h(\mathbf{W}_s \tilde{\mathbf{x}}_s + \mathbf{b}_s) \qquad \forall s \in 1, \ldots, S. \tag{5}$$

The reconstruction $\mathbf{z}$ is computed by taking all of the hidden layers as input

$$\mathbf{z} = h'\left(\sum_{s=1}^{S} \mathbf{W}_s^\top \mathbf{y}_s + \mathbf{b}'\right), \tag{6}$$

where as before h' is a nonlinear transfer function, potentially different from h. Finally given a loss function L, such as squared error, an update to the parameters can be made by taking a gradient step on $L(\mathbf{z}, \mathbf{x})$.

This procedure can be seen as a stochastic gradient on an objective function that takes the expectation over the corruptions:

$$\mathbb{E}_{(X, \tilde{X}_{\nu_1}, \ldots, \tilde{X}_{\nu_S})}\left[L\left(X, h'\left(\sum_{s=1}^{S} \mathbf{W}_s^\top h\left(\mathbf{W}_s \tilde{\mathbf{x}}_{\nu_s} + \mathbf{b}_s\right) + \mathbf{b}'\right)\right)\right], \tag{7}$$

This architecture is illustrated in Figure 1 for two levels of noise, where we use the different colours to indicate the weights in the network that are specific to a single noise level.

3.1 Learning

A CDA could be trained by standard optimization methods, such as stochastic gradient descent on the objective (7). As we will show, however, it is difficult to achieve good performance with these methods (4.1). Instead, we propose a new cascaded training procedure for CDAs, which we describe in this section.

Cascaded training is based on two ideas. First, previous work [10] found that pretraining at high noise levels helps learning the parameters for the low noise levels. Second, and more interesting, the problem with taking a joint gradient step on (7) is that low noise levels provide more information about the original input $\mathbf{x}$ than high noise levels, which can cause a training procedure to get stuck in a local optimum in which it relies on the low noise features without using the high noise features. Cascaded training first trains the weights that correspond to high noise levels, and then freezes them before moving on to low noise levels. This way the hidden units trained with lower levels of noise are trained to correct what the hidden units associated with higher noise levels missed.

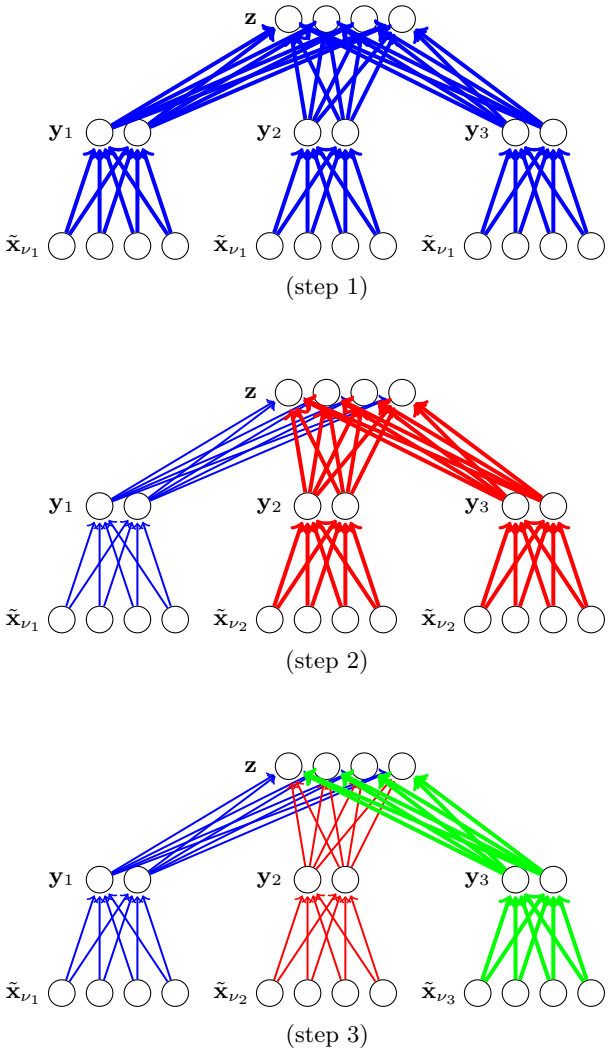

Fig. 2. The cascaded training procedure for a composite denoising autoencoder with three noise levels. We use the notation $\mathbf{y}_{1:3} = (\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3)$. First all parameters are trained using the level of noise ν_1. In the second step, the blue parameters remain frozen and the red parameters are trained using the noise ν_2. Finally, in the third step, only the green parameters are trained, using the noise ν_3. This is more formally described in Algorithm 1.

Algorithm 1 Training the composite denoising autoencoder

for R in $1, \ldots, S$ **do**
  for K_R steps **do**
    Randomly choose a training input $\mathbf{x}$
    Sample $\tilde{\mathbf{x}}_s \sim p(\cdot|\mathbf{x}, \nu_s)$ for $s \in \{1, 2, \ldots, R-1\}$
    Sample $\tilde{\mathbf{x}}_s \sim p(\cdot|\mathbf{x}, \nu_R)$ for $s \in \{R, R+1, \ldots, S\}$
    Compute $\mathbf{y}_s$ for all s as in (5)
    Compute reconstruction $\mathbf{z}$ as in (6)
    Take a gradient step

$$\mathbf{W}_s \leftarrow \mathbf{W}_s - \alpha \nabla_{\mathbf{W}_s} L(\mathbf{z}, \mathbf{x})$$
$$\mathbf{b}_s \leftarrow \mathbf{b}_s - \alpha \nabla_{\mathbf{b}_s} L(\mathbf{z}, \mathbf{x})$$
$$\mathbf{b}' \leftarrow \mathbf{b}' - \alpha \nabla_{\mathbf{b}'} L(\mathbf{z}, \mathbf{x})$$

    for $s \in \{R, R+1, \ldots S\}$
  end for
end for

Putting these ideas together, cascaded training works as follows. We assume that the noise levels are ordered so that $\nu_1 > \nu_2 > \cdots > \nu_S$. Then the first step is that we train *all* of the parameters $\mathbf{W}_1 \ldots \mathbf{W}_S, \mathbf{b}_1, \ldots \mathbf{b}_S, \mathbf{b}'$, but using only the noise level ν_1 to corrupt all S copies $\tilde{\mathbf{x}}_1 \ldots \tilde{\mathbf{x}}_S$ of the input. Once this is done, we freeze the weights $\mathbf{W}_1, \mathbf{b}_1$ and we do not alter them again during training. Then we train the weights $\mathbf{W}_2 \ldots \mathbf{W}_S, \mathbf{b}_2 \ldots \mathbf{b}_S, \mathbf{b}'$, where the corrupted input $\tilde{\mathbf{x}}_1$ is as before corrupted with noise ν_1, and the $S-1$ corrupted copies $\tilde{\mathbf{x}}_2 \ldots \tilde{\mathbf{x}}_S$ are all corrupted with noise ν_2. We repeat this process until at the end we are training the weights $\mathbf{W}_S, \mathbf{b}_S, \mathbf{b}'$ using the noise level ν_S. This process is illustrated graphically in Figure 2 and in pseudocode in Algorithm 1. To keep the exposition simple, this algorithm assumes that we employ SGD with only one training example per update, although in practice we use mini-batches.

The composite denoising autoencoder builds on several ideas and intuitions. Firstly, our training procedure can be considered an application of the idea of curriculum learning [4, 14]. That is, we start by training all units with high noise level, which serves as a form of *unsupervised pretraining* for the units that will be trained later with lower levels of noise, giving them a good starting point for further optimisation. We experimentally show that the training of a denoising autoencoder learning with data corrupted with high noise levels needs less training epochs to converge, therefore, it can be considered an *easier* problem. This is shown in Figure 3. Secondly, we are inspired by multi-column neural networks (e.g. Ciresan et al. [7]), which achieve excellent performance for supervised problems. Finally, our work is similar in motivation to scheduled denoising autoencoders [10], which learn a diverse set of features thanks to the training procedure which involves using a sequence of levels of noise. Composite denoising autoencoders achieve this goal more explicitly thanks to their training objective.

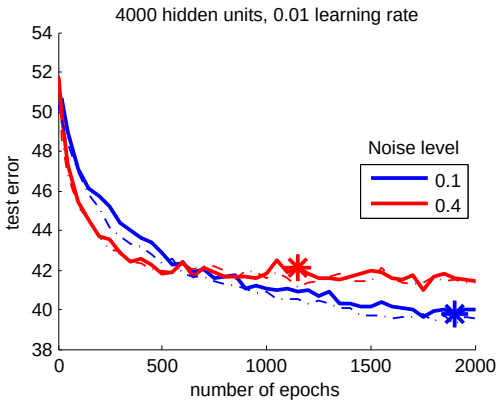

Fig. 3. Classification results with the CIFAR-10 data set yielded by representations learnt with standard denoising autoencoders and data corrupted with two different noise levels. Dashed lines indicate the errors on the validation set. The stars indicate the test errors for the epochs at which the validation errors had its lowest value. The DA trained with high noise level learns faster at the beginning but stops to improve earlier. See section 4 for the details of the experimental setup.

3.2 Recovering the standard denoising autoencoder

If, for every training example, the corrupted inputs $\tilde{\mathbf{x}}_{\nu_i}$ were always identical, $[\mathbf{W}_1, \ldots, \mathbf{W}_S]$ were initialised randomly from the same distribution as $\mathbf{W}$ in the standard denoising autoencoder, $\mathbf{b}_i$ and $\mathbf{b}'$ were initalised to $\mathbf{0}$ and $\mathbf{V}_i$ were constrained to be $\mathbf{V}_i = \mathbf{W}_i^T$, then this model is exactly equivalent to the standard denoising autoencoder described in section 2. Therefore, it is natural to incrementally corrupt the training examples shown to the composite denoising autoencoders in such a way that when all the noise levels are the same, this equivalency holds. For example, when working with masking noise, consider two noise levels ν_i and ν_j such that $\nu_i > \nu_j$. Denote the random variables indicating the presence of corruption of a pixel in a training datum by C_{ν_i} and C_{ν_j}. Assuming $\mathrm{C}_{\nu_j} \sim \mathrm{Bernoulli}(\nu_j)$, we want $\mathrm{C}_{\nu_i} \sim \mathrm{Bernoulli}(\nu_i)$, such that when $\mathrm{C}_{\nu_j} = 1$ then also $\mathrm{C}_{\nu_i} = 1$. It can be easily shown that this is satisfied when $\mathrm{C}_{\nu_i} = \max(\mathrm{C}_{\nu_j} + \mathrm{C}_{\nu_j \to \nu_i}, 1)$, where $\mathrm{C}_{\nu_j \to \nu_i} \sim \mathrm{Bernoulli}(\frac{\nu_i - \nu_j}{1 - \nu_j})$. We use this incremental noising procedure in all our experiments.

4 Experiments

We used two image recognition data sets to evaluate the CDA, the CIFAR-10 data set [19] and a variant of the NORB data set [21]. To evaluate the quality of the learnt representations, we employ a procedure similar to that used by Coates et al. [8] and by many other works[1]. That is, we first learn the representation in

[1] We do not use any form of pooling, keeping our setup invariant to the permutation of the features.

an unsupervised fashion and then use the learnt representation within a linear classifier as a measure of its quality. For both data sets, in the unsupervised feature learning stage, we use masking noise as the corruption process, a sigmoid encoder and decoder and cross entropy loss (Equation 2) following Vincent et al. (2008, 2010). To do optimisation, we use stochastic gradient descent with mini-batches. For the classification step, we use $L2$-regularised logistic regression with the regularisation parameter chosen to minimise the validation error. Additionally, with the CIFAR-10 data set, we also trained a single-layer supervised neural network using the parameters of the encoder we learnt in the unsupervised stage as an initialisation. When conducting our experiments, we first find the best hyperparameters using the validation set, then merge it with the training set, retrain the model with the hyperparameters found in the previous step and report the error achieved with this model.

We implemented all neural network models using Theano [6] and we used logistic regression implemented by Fan et al. [9]. We followed the advice of Glorot and Bengio [11] on random initialisation of the parameters of our networks.

4.1 CIFAR-10

This data set consists of 60000 colour images spread evenly between ten classes. There are 50000 training and validation images and 10000 test images. Each image has a size of 32×32 pixels and each pixel has three colour channels, which are represented with a number in $\{0, \ldots, 255\}$. We divide the training and validation set into 40000 training instances and 10000 validation instances. The only preprocessing step we use is dividing the intensity of every pixel by 255 to get numbers in $[0, 1]$.

In our experiments with this data set we trained autoencoders with the total number of 2000 hidden units (undercomplete representation) and 4000 hidden units (overcomplete representation).

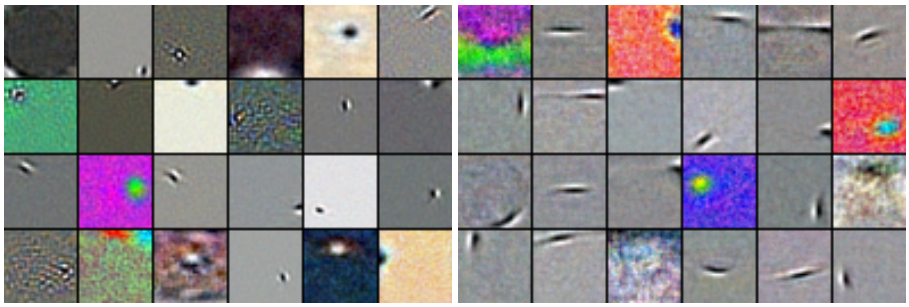

Fig. 4. Example filters (columns of the matrix $\mathbf{W}$) learnt by standard denoising autoencoders with $\nu = 0.1$ (left) and $\nu = 0.5$ (right).

Training the baselines The simplest possible baseline, logistic regression trained with raw pixel values, achieved 59.4% test error. To get the best possible baseline denoising autoencoder we explored combinations of different learning rates, noise levels and numbers of training epochs. For 2000 hidden units we considered $\nu \in \{0.05, 0.1, 0.2, 0.3, 0.4, 0.5\}$ and for 4000 hidden units we also additionally considered $\nu = 0.15$. For both sizes of the hidden layers we tried learning rates $\in \{0.01, 0.02, 0.04\}$. Each model was trained for up to 2000 training epochs and we measured the validation error every 50 epochs. The best baselines we got achieved the test errors of 40.71% (2000 hidden units) and 38.35% (4000 hidden units).

Concatenating representations learnt independently To demonstrate that diversity in noise levels improves the representation, we evaluate representations yielded by concatenating the representations from two different DAs, trained independently. We will combine DAs trained with noise levels $\nu \in \{0.1, 0.2, \ldots, 0.5\}$ for each noise level training three DAs with different random seeds. Denote parameters learnt by a DA with the noise level ν and using the random seed R by $\left(\mathbf{W}^{(R,\nu)}, \mathbf{b}^{(R,\nu)}, \mathbf{b}'^{(R,\nu)}\right)$ and denote by E_{ij}^{kl} the classification error on the test set yielded by the concatenating the representations of two independently trained DAs, the first trained with random seed R_k and noise level ν_i, and the second trained by random seed R_l and noise level ν_j. For each pair of noise levels (ν_i, ν_j), we measure the average error across random seeds, that is, $\bar{\mathrm{E}}_{ij} = \frac{1}{2\binom{K}{2}} \left(\sum_{k \neq l} \mathrm{E}_{ij}^{kl} + \mathrm{E}_{ji}^{kl}\right)$. The results of this experiment are shown in Figure 5. For every ν we used, it was optimal to concatenate the representation learnt with ν with a representation learnt with a different noise level. To understand this intuition, we visually examine features from DAs with different noise levels (Figure 4). From this figure it can be seen that features at higher noise levels depend on larger regions of the image. This demonstrates the benefit of using a more diverse representation for classification.

Comparison of CDA to DA The CDA offers freedom to choose the number of noise levels, the value ν_s for each noise level, and the number D_s of hidden units at each noise level.

For computational reasons, we limit the space of possible combinations of hyperparameters in the following manner (of course, expanding the search space would only make our results better). We considered models containing up to four different noise levels. We first consider only the models with two noise levels and hidden units divided equally between them. For 2000 total hidden units, we consider all possible pairs of noise levels drawn from the set $\{0.5, 0.4, 0.3, 0.2, 0.1, 0.05\}$. Once we have found the value of $\boldsymbol{\nu}$ that minimizes that validation error for $D_1 = D_2$, we try splitting hidden units such that the ratio $D_1 : D_2 = 1 : 3$ or $D_1 : D_2 = 3 : 1$. Similarly, for four noise levels, we consider the following sets of noise levels $\boldsymbol{\nu} \in \{(0.5, 0.4, 0.3, 0.2), (0.4, 0.3, 0.2, 0.1), (0.3, 0.2, 0.1, 0.05)\}$. We select the value of $\boldsymbol{\nu}$ that has lowest validation error for an equal split

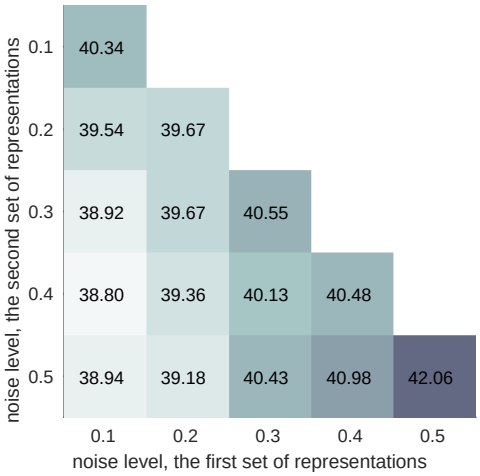

Fig. 5. Classification errors for representations constructed by concatenating representations learnt independently.

$D_1 = \cdots = D_4$, and then try splitting the hidden units with different ratios: $D_1 : D_2 : D_3 : D_4 = 3 : 1 : 1 : 1$, $D_1 : D_2 : D_3 : D_4 = 9 : 1 : 1 : 1$ and the permutations of these ratios. As for the learning rate, we train each of the cascaded DAs with the learning rate that had the best validation error for the first noise level ν_1. The models were trained for up to 500 epochs at each consecutive noise level and we computed the validation error every 50 training epochs. Note that when training with four noise levels, it is possible that the lowest validation error occurs before the training procedure has moved on to the final noise level. In this circumstance, it is possible that the final model will have only two or three noise levels instead of four.

We trained the models with 4000 hidden units the same way, except that we used different sets of noise levels for this higher number of hidden units. This is because our experience with the baseline DAs was that units with 4000 hidden units do better with lower noise levels. For the CDAs with four noise levels, we compared three difference choices for $\boldsymbol{\nu}$: $(0.4, 0.3, 0.2, 0.1)$, $(0.3, 0.2, 0.1, 0.05)$, and $(0.2, 0.15, 0.1, 0.05)$. For the models with two noise levels the values were drawn from $\{0.4, 0.3, 0.2, 0.15, 0.1, 0.05\}$.

For either number of hidden units, we find that CDAs perform better than simple DAs. The best models with 2000 hidden units and 4000 hidden units we found achieved the test errors of 38.86% and 37.53% respectively, thus yielding a significant improvement over the representations trained with a standard DA. These results are compared to the baselines in Table 1. It is also noteworthy that a CDA performs better than concatenating two indepedently trained DAs with different noise levels (cf. Figure 5).

Table 1. Classification errors of standard denoising autoencoders and composite denoising autoencoders.

hidden units	best DA	test error	best CDA	test error
2000	$\nu = 0.2$	40.71%	$\boldsymbol{\nu} = (0.3, 0.2, 0.1)$, $\mathbf{D} = (500, 500, 1000)$	38.86%
4000	$\nu = 0.1$	38.35%	$\boldsymbol{\nu} = (0.3, 0.05)$, $\mathbf{D} = (1000, 3000)$	37.53%

Comparison of Optimization Methods One could consider several simpler alternatives to the cascaded training procedure from Section 3.1. The simplest alternative, which we call *joint SGD*, is to train all of the model parameters jointly, at every iteration sampling each corrupted input $\tilde{\mathbf{x}}_s$ using its corresponding noise level ν_s. This is simply SGD on the objective (7). A second alternative, which we call *alternating SGD*, is block coordinate descent on (7), where we assign each weight matrix $\mathbf{W}_s$ to a separate block. In other words, at each iteration we choose a different parameter block $\mathbf{W}_s$, and take a gradient update only on $\mathbf{W}_s$ (note that this requires computing a corrupted input $\tilde{\mathbf{x}}_s$ for all noise levels ν_s). Neither of these simpler methods try to prevent undertraining of the parameters for the high noise levels in the way that cascaded training does.

Figure 6 shows a comparison of joint SGD, alternating SGD, and our cascaded SGD methods on a CDA with four noise levels $\boldsymbol{\nu} = (0.4, 0.3, 0.2, 0.1)$ and $\mathbf{D} = (500, 500, 500, 500)$. We ran both joint SGD and cascaded SGD until they converged in validation error, and then we ran alternating SGD until it had made the same number of parameter updates as joint SGD. This means that alternating SGD was run for four times as many iterations as joint SGD, because alternating SGD only updates one-quarter of the parameters at each iteration. Cascaded SGD was stopped early when it converged according to validation error. The vertical dashed lines in the figure indicate the epochs at which alternating SGD switched between parameter blocks.

From these results, it is clear that the cascaded training procedure is significantly more effective than either joint or alternating SGD. Joint SGD seems to have converged to much worse parameters than cascaded SGD. We hypothesize that this is because the parameters corresponding to the high noise levels are undertrained. To verify this, in Figure 7 we show the features learned by a composite CDA with joint training at two different noise levels. Note that at the higher noise level (at right) there are many filters that are mostly noise; this is not observed at the lower noise or to the same extent in an standard DA. Alternating SGD seems to converge fairly slowly. It is possible that its error would continue to decrease, but even after 8000 iterations its solution is still much worse than that found by cascading SGD after only 3500 iterations.

We have made similar comparisons for other choices of $\boldsymbol{\nu}$ and found a similar difference in performance between joint, alternating, and cascaded SGD. One exception to this is that alternating SGD seems to work much better on models with only two noise levels ($S = 2$) than those with four noise levels. In those situations, the performance of alternating SGD often equals, but usually does not exceed, that of cascaded SGD.

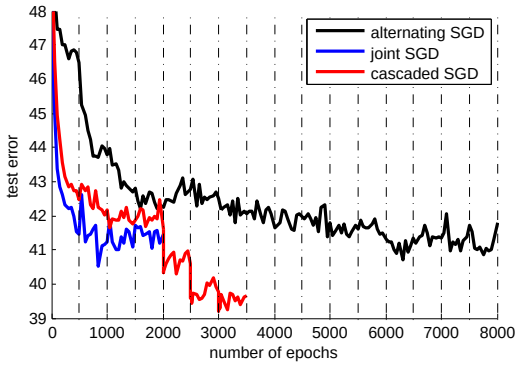

Fig. 6. Classification errors achieved by three different methods of optimising the objective in (7).

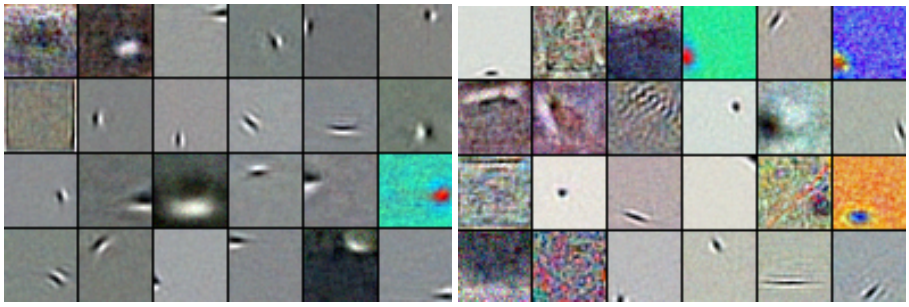

Fig. 7. Example filters (columns of the matrix $\mathbf{W}$) learnt by composite denoising autoencoders with $\nu = 0.1$ (left) and $\nu = 0.4$ (right) when all the parameters were optimise using joint SGD. While the filters associated with $\nu_2 = 0.1$ have managed to learn interesting features, many of these associated with $\nu_1 = 0.4$ remained undertrained. These hard to interpret filters are much more rare with cascaded SGD.

Fine-tuning We also trained a supervised single-layer neural network using parameters of the encoder as the initialisation of the parameters of the hidden layer of the network. This procedure is known as fine-tuning. We did that for the best standard DAs and CDAs with 4000 hidden units. The learning rate, the same for all parameters, was chosen from the set $\{0.00125, 0.00125 \cdot 2^{-1}, \ldots, 0.00125 \cdot 2^{-4}\}$ and the maximum number of training epochs was 2000 (we computed the validation error after each epoch). We report the test error for the combination of the learning rate and the number of epochs yielding the lowest validation error. The results are shown in Table 3. Fine-tuning makes the performance of DA and CDA much more similar, which is to be expected since the fine-tuning procedure is identical for both models. However, note that the result achieved with a standard denoising autoencoder and supervised fine-tuning we present here is an extremely well tuned one. In fact, its error is lower than any previous result achieved by a permutation-invariant method on the CIFAR-10 data

set. Our best model, yielding the error of 35.06% is, by a considerable margin, more accurate than any previously considered permutation-invariant model for this task, outperforming a variety of methods. A summary of the best results reported in the literature is shown in Table 2.

Table 2. Summary of the results on CIFAR-10 among permutation-invariant methods.

Model	Test error
Composite Denoising Autoencoder	**35.06%**
Scheduled Denoising Autoencoder [10]	35.7%
Zero-bias Autoencoder [22]	35.9%
Fastfood FFT [20]	36.9%
Nonparametrically Guided Autoencoder [25]	43.25%
Deep Sparse Rectifier Neural Network [12]	49.52%

Table 3. Test errors on CIFAR-10 data set for the best DA and CDA models trained without supervised fine-tuning and their fine-tuned versions.

DA		**CDA**	
no fine-tuning	**fine-tuning**	**no fine-tuning**	**fine-tuning**
38.35%	35.30%	37.53%	35.06%

4.2 NORB

To show that the advantage of our model is consistent across data sets, we did the same experiment use a variant of the small NORB normalized-uniform data set [21], which contains 24300 examples for training and validation and 24300 test examples. It contains images of 50 toys belonging to five generic categories: animals, human figures, airplanes, trucks, and cars. The 50 toys are evenly divided between the training and validation set and the test set. The objects were photographed by two cameras under different lighting conditions, elevations and azimuths. Every example consists of a stereo pair of grayscale images, each of size 96×96 pixels whose intensities are represented as a number $\in \{0, \ldots, 255\}$. We transform the data set by taking the middle 64×64 pixels from both images in a pair and dividing the intensity of every pixel by 255 to get numbers in $[0, 1]$. The simplest baseline, logistic regression using raw pixels, achieved the test error of 42.32%.

In the experiments with learning the representations with this data set we used the hidden layer with 1000 hidden units and adapted the set up we used for CIFAR-10. To find the best possible standard DA we considered all combinations of the noise levels $\in \{0.1, 0.2, 0.3, 0.4\}$ and the learning rates $\in \{0.005, 0.01, 0.02\}$.

The representation learnt by the best denoising autoencoder yielded 18.75% test error when used with logistic regression. By contrast, a composite denoising autoencoder with $\boldsymbol{\nu} = (0.4, 0.3, 0.2, 0.1)$ and $\mathbf{D} = (250, 250, 250, 250)$ results in a representation that yields a test error of 17.03%.

5 Discussion

We introduced a new unsupervised representation learning method, called a composite denoising autoencoder, by modifying the standard DA so that different parts of the network were exposed to corruptions of the input at different noise levels. Naive training procedures for the CDA can get stuck in bad local optima, so we designed a cascaded training procedure to avoid this. We showed that CDAs learned more effective representations than DAs on two different image data sets.

A few pieces of prior work have considered related techniques. In the context of RBMs, the benefits of learning a diverse representation was also noticed by Tang and Mohamed [26], achieving diversity by manipulating the resolution of the image. Also, ensembles of denoising autoencoders, where each member of the ensemble is trained with a different level or different type of noise, have been considered by Agostinelli et al. [1]. This work differs from ours because in their method all DAs in the ensemble are trained independently, whereas we show that training the different representations together is better than independent training. The cascaded training procedure has some similarities in spirit to the incremental training procedure of Zhou et al. [31], but that work considered only DAs with one level of noise. Usefulness of varying the level of noise during training of neural nets was also noticed by Gulcehre et al. [15], who add noise to the activation functions. Our training procedure also resembles the walkback training suggested by Bengio et al. [5], however, we do not require our training loss to be interpretable as negative log-likelihood. Understanding the relative merits of walkback training, scheduled denoising autoencoders and composite denoising autoencoders would be an interesting future challenge.

References

[1] Agostinelli, F., Anderson, M.R., Lee, H.: Robust image denoising with multi-column deep neural networks. In: NIPS (2013)
[2] Baldi, P., Hornik, K.: Neural networks and principal component analysis: Learning from examples without local minima. Neural Networks 2 (1989)
[3] Bengio, Y., Lamblin, P., Popovici, D., Larochelle, H.: Greedy layer-wise training of deep networks. In: NIPS (2007)
[4] Bengio, Y., Louradour, J., Collobert, R., Weston, J.: Curriculum learning. In: ICML (2009)
[5] Bengio, Y., Yao, L., Alain, G., Vincent, P.: Generalized denoising auto-encoders as generative models. In: NIPS (2013)
[6] Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., Bengio, Y.: Theano: a CPU and GPU math expression compiler. SciPy (2010)

[7] Ciresan, D., Meier, U., Schmidhuber, J.: Multi-column deep neural networks for image classification. In: CVPR (2012)
[8] Coates, A., Ng, A.Y., Lee, H.: An analysis of single-layer networks in unsupervised feature learning. In: AISTATS (2011)
[9] Fan, R.E., Chang, K.W., Hsieh, C.J., Wang, X.R., Lin, C.J.: LIBLINEAR: A library for large linear classification. JMLR 9 (2008)
[10] Geras, K.J., Sutton, C.: Scheduled denoising autoencoder. In: ICLR (2015)
[11] Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: AISTATS (2010)
[12] Glorot, X., Bordes, A., Bengio, Y.: Deep sparse rectifier networks. In: AISTATS (2011)
[13] Glorot, X., Bordes, A., Bengio, Y.: Domain adaptation for large-scale sentiment classification: A deep learning approach. In: ICML (2011)
[14] Gulcehre, C., Bengio, Y.: Knowledge matters: Importance of prior information for optimization. JMLR 17 (2016)
[15] Gulcehre, C., Moczulski, M., Denil, M., Bengio, Y.: Noisy activation functions. arXiv:1603.00391 (2016)
[16] Hinton, G.E., Osindero, S., Teh, Y.W.: A fast learning algorithm for deep belief nets. Neural Computation 18(7) (2006)
[17] Hyvärinen, A., Dayan, P.: Estimation of non-normalized statistical models by score matching. Journal of Machine Learning Research 6 (2005)
[18] Karklin, Y., Simoncelli, E.P.: Efficient coding of natural images with a population of noisy linear-nonlinear neurons. In: NIPS (2011)
[19] Krizhevsky, A.: Learning multiple layers of features from tiny images. Tech. rep., University of Toronto (2009)
[20] Le, Q., Sarlós, T., Smola, A.: Fastfood-computing Hilbert space expansions in loglinear time. In: ICML (2013)
[21] LeCun, Y., Huang, F.J., Bottou, L.: Learning methods for generic object recognition with invariance to pose and lighting. In: CVPR (2004)
[22] Memisevic, R., Konda, K., Krueger, D.: Zero-bias autoencoders and the benefits of co-adapting features. In: ICLR (2015)
[23] Mesnil, G., Dauphin, Y., Glorot, X., Rifai, S., Bengio, Y., Goodfellow, I.J., Lavoie, E., Muller, X., Desjardins, G., Warde-Farley, D., Vincent, P., Courville, A.C., Bergstra, J.: Unsupervised and transfer learning challenge: a deep learning approach. In: ICML Unsupervised and Transfer Learning Workshop (2012)
[24] Seung, H.S.: Learning continuous attractors in recurrent networks. In: NIPS (1998)
[25] Snoek, J., Adams, R.P., Larochelle, H.: Nonparametric guidance of autoencoder representations using label information. JMLR 13 (2012)
[26] Tang, Y., Mohamed, A.r.: Multiresolution deep belief networks. In: AISTATS (2012)
[27] Vincent, P.: A connection between score matching and denoising autoencoders. Neural Computation 23 (2011)
[28] Vincent, P., Larochelle, H., Bengio, Y., Manzagol, P.A.: Extracting and composing robust features with denoising autoencoders. In: ICML (2008)
[29] Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., Manzagol, P.A.: Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. JMLR (2010)
[30] Xie, J., Xu, L., Chen, E.: Image denoising and inpainting with deep neural networks. In: NIPS (2012)
[31] Zhou, G., Sohn, K., Lee, H.: Online incremental feature learning with denoising autoencoders. In: AISTATS (2012)