Semi-Supervised Learning with Ladder Networks

Antti Rasmus
The Curious AI Company, Finland

Harri Valpola
The Curious AI Company, Finland

Mikko Honkala
Nokia Labs, Finland

Mathias Berglund
Aalto University & The Curious AI Company, Finland

Tapani Raiko
Aalto University & The Curious AI Company, Finland

Abstract

We combine supervised learning with unsupervised learning in deep neural networks. The proposed model is trained to simultaneously minimize the sum of supervised and unsupervised cost functions by backpropagation, avoiding the need for layer-wise pre-training. Our work builds on the Ladder network proposed by Valpola (2015), which we extend by combining the model with supervision. We show that the resulting model reaches state-of-the-art performance in semi-supervised MNIST and CIFAR-10 classification, in addition to permutation-invariant MNIST classification with all labels.

1 Introduction

In this paper, we introduce an unsupervised learning method that fits well with supervised learning. The idea of using unsupervised learning to complement supervision is not new. Combining an auxiliary task to help train a neural network was proposed by Suddarth and Kergosien (1990). By sharing the hidden representations among more than one task, the network generalizes better. There are multiple choices for the unsupervised task, for example, reconstruction of the inputs at every level of the model (e.g., Ranzato and Szummer, 2008) or classification of each input sample into its own class (Dosovitskiy *et al.*, 2014).

Although some methods have been able to simultaneously apply both supervised and unsupervised learning (Ranzato and Szummer, 2008; Goodfellow *et al.*, 2013a), often these unsupervised auxiliary tasks are only applied as pre-training, followed by normal supervised learning (e.g., Hinton and Salakhutdinov, 2006). In complex tasks there is often much more structure in the inputs than can be represented, and unsupervised learning cannot, by definition, know what will be useful for the task at hand. Consider, for instance, the autoencoder approach applied to natural images: an auxiliary decoder network tries to reconstruct the original input from the internal representation. The autoencoder will try to preserve all the details needed for reconstructing the image at pixel level, even though classification is typically invariant to all kinds of transformations which do not preserve pixel values. Most of the information required for pixel-level reconstruction is irrelevant and takes space from the more relevant invariant features which, almost by definition, cannot alone be used for reconstruction.

Our approach follows Valpola (2015), who proposed a Ladder network where the auxiliary task is to denoise representations at every level of the model. The model structure is an autoencoder with skip connections from the encoder to decoder and the learning task is similar to that in denoising autoencoders but applied to every layer, not just the inputs. The skip connections relieve the pressure to represent details in the higher layers of the model because, through the skip connections, the

decoder can recover any details discarded by the encoder. Previously, the Ladder network has only been demonstrated in unsupervised learning (Valpola, 2015; Rasmus *et al.*, 2015a) but we now combine it with supervised learning.

The key aspects of the approach are as follows:

Compatibility with supervised methods. The unsupervised part focuses on relevant details found by supervised learning. Furthermore, it can be added to existing feedforward neural networks, for example multi-layer perceptrons (MLPs) or convolutional neural networks (CNNs) (Section 3). We show that we can take a state-of-the-art supervised learning method as a starting point and improve the network further by adding simultaneous unsupervised learning (Section 4).

Scalability resulting from local learning. In addition to a supervised learning target on the top layer, the model has local unsupervised learning targets on every layer, making it suitable for very deep neural networks. We demonstrate this with two deep supervised network architectures.

Computational efficiency. The encoder part of the model corresponds to normal supervised learning. Adding a decoder, as proposed in this paper, approximately triples the computation during training but not necessarily the training time since the same result can be achieved faster through the better utilization of the available information. Overall, computation per update scales similarly to whichever supervised learning approach is used, with a small multiplicative factor.

As explained in Section 2, the skip connections and layer-wise unsupervised targets effectively turn autoencoders into hierarchical latent variable models which are known to be well suited for semi-supervised learning. Indeed, we obtain state-of-the-art results in semi-supervised learning in the MNIST, permutation invariant MNIST and CIFAR-10 classification tasks (Section 4). However, the improvements are not limited to semi-supervised settings: for the permutation invariant MNIST task, we also achieve a new record with the normal full-labeled setting.[1]

2 Derivation and justification

Latent variable models are an attractive approach to semi-supervised learning because they can combine supervised and unsupervised learning in a principled way. The only difference is whether the class labels are observed or not. This approach was taken, for instance, by Goodfellow *et al.* (2013a) with their multi-prediction deep Boltzmann machine. A particularly attractive property of hierarchical latent variable models is that they can, in general, leave the details for the lower levels to represent, allowing higher levels to focus on more invariant, abstract features that turn out to be relevant for the task at hand.

The training process of latent variable models can typically be split into inference and learning, that is, finding the posterior probability of the unobserved latent variables and then updating the underlying probability model to fit the observations better. For instance, in the expectation-maximization (EM) algorithm, the E-step corresponds to finding the expectation of the latent variables over the posterior distribution assuming the model fixed and the M-step then maximizes the underlying probability model assuming the expectation fixed.

The main problem with latent variable models is how to make inference and learning efficient. Suppose there are layers l of latent variables $\mathbf{z}^{(l)}$. Latent variable models often represent the probability distribution of all the variables explicitly as a product of terms, such as $p(\mathbf{z}^{(l)} \mid \mathbf{z}^{(l+1)})$ in directed graphical models. The inference process and model updates are then derived from Bayes' rule, typically as some kind of approximation. The inference is often iterative as it is generally impossible to solve the resulting equations in a closed form as a function of the observed variables.

There is a close connection between denoising and probabilistic modeling. On the one hand, given a probabilistic model, you can compute the optimal denoising. Say you want to reconstruct a latent z using a prior $p(z)$ and an observation $\tilde{z} = z + \text{noise}$. We first compute the posterior distribution $p(z \mid \tilde{z})$, and use its center of gravity as the reconstruction $\hat{z}$. One can show that this minimizes the expected denoising cost $(\hat{z} - z)^2$. On the other hand, given a denoising function, one can draw

---

[1]Preliminary results on the full-labeled setting on a permutation invariant MNIST task were reported in a short early version of this paper (Rasmus *et al.*, 2015b). Compared to that, we have added noise to all layers of the model and further simplified the denoising function g. This further improved the results.

samples from the corresponding distribution by creating a Markov chain that alternates between corruption and denoising (Bengio *et al.*, 2013).

Valpola (2015) proposed the Ladder network, where the inference process itself can be learned by using the principle of denoising, which has been used in supervised learning (Sietsma and Dow, 1991), denoising autoencoders (dAE) (Vincent *et al.*, 2010), and denoising source separation (DSS) (Särelä and Valpola, 2005) for complementary tasks. In dAE, an autoencoder is trained to reconstruct the original observation $\mathbf{x}$ from a corrupted version $\tilde{\mathbf{x}}$. Learning is based simply on minimizing the norm of the difference of the original $\mathbf{x}$ and its reconstruction $\hat{\mathbf{x}}$ from the corrupted $\tilde{\mathbf{x}}$; that is the cost is $\|\hat{\mathbf{x}} - \mathbf{x}\|^2$.

While dAEs are normally only trained to denoise the observations, the DSS framework is based on the idea of using denoising functions $\hat{\mathbf{z}} = g(\mathbf{z})$ of the latent variables $\mathbf{z}$ to train a mapping $\mathbf{z} = f(\mathbf{x})$ which models the likelihood of the latent variables as a function of the observations. The cost function is identical to that used in a dAE except that the latent variables $\mathbf{z}$ replace the observations $\mathbf{x}$; that is, the cost is $\|\hat{\mathbf{z}} - \mathbf{z}\|^2$. The only thing to keep in mind is that $\mathbf{z}$ needs to be normalized somehow as otherwise the model has a trivial solution at $\mathbf{z} = \hat{\mathbf{z}} = $ constant. In a dAE, this cannot happen as the model cannot change the input $\mathbf{x}$.

Figure 1 depicts the optimal denoising function $\hat{z} = g(\tilde{z})$ for a one-dimensional bimodal distribution, which could be the distribution of a latent variable inside a larger model. The shape of the denoising function depends on the distribution of z and the properties of the corruption noise. With no noise at all, the optimal denoising function would be the identity function. In general, the denoising function pushes the values towards higher probabilities, as shown by the green arrows.

Figure 2 shows the structure of the Ladder network. Every layer contributes to the cost function a term $C_d^{(l)} = \|\mathbf{z}^{(l)} - \hat{\mathbf{z}}^{(l)}\|^2$ which trains the layers above (both encoder and decoder) to learn the denoising function $\hat{\mathbf{z}}^{(l)} = g^{(l)}(\tilde{\mathbf{z}}^{(l)}, \hat{\mathbf{z}}^{(l+1)})$ which maps the corrupted $\tilde{\mathbf{z}}^{(l)}$ onto the denoised estimate $\hat{\mathbf{z}}^{(l)}$. As the estimate $\hat{\mathbf{z}}^{(l)}$ incorporates all prior knowledge about $\mathbf{z}$, the same cost function term also trains the encoder layers below to find cleaner features which better match the prior expectation.

Since the cost function needs both the clean $\mathbf{z}^{(l)}$ and corrupted $\tilde{\mathbf{z}}^{(l)}$, during training the encoder is run twice: a clean pass for $\mathbf{z}^{(l)}$ and a corrupted pass for $\tilde{\mathbf{z}}^{(l)}$. Another feature which differentiates the Ladder network from regular dAEs is that each layer has a skip connection between the encoder and decoder. This feature mimics the inference structure of latent variable models and makes it possible for the higher levels of the network to leave some of the details for lower levels to represent. Rasmus *et al.* (2015a) showed that such skip connections allow dAEs to focus on abstract invariant features on the higher levels, making the Ladder network a good fit with supervised learning that can select which information is relevant for the task at hand.

One way to picture the Ladder network is to consider it as a collection of nested denoising autoencoders which share parts of the denoising machinery with each other. From the viewpoint of the autoencoder on layer l, the representations on the higher layers can be treated as hidden neurons. In other words, there is no particular reason why $\hat{\mathbf{z}}^{(l+i)}$ as produced by the decoder should resemble the corresponding representations $\mathbf{z}^{(l+i)}$ as produced by the encoder. It is only the cost function $C_d^{(l+i)}$ that ties these together and forces the inference to proceed in reverse order in the decoder. This sharing helps a deep denoising autoencoder to learn the denoising process as it splits the task into meaningful sub-tasks of denoising intermediate representations.

3 Implementation of the Model

The steps involved in implementing the Ladder network (Section 3.1) are typically as follows: 1) take a feedforward model which serves supervised learning as the encoder (Section 3.2); 2) add a decoder which can invert the mappings on each layer of the encoder and supports unsupervised learning (Section 3.3); and 3) train the whole Ladder network by minimizing the sum of all the cost function terms.

In this section, we will go through these steps in detail for a fully connected MLP network and briefly outline the modifications required for convolutional networks, both of which are used in our experiments (Section 4).
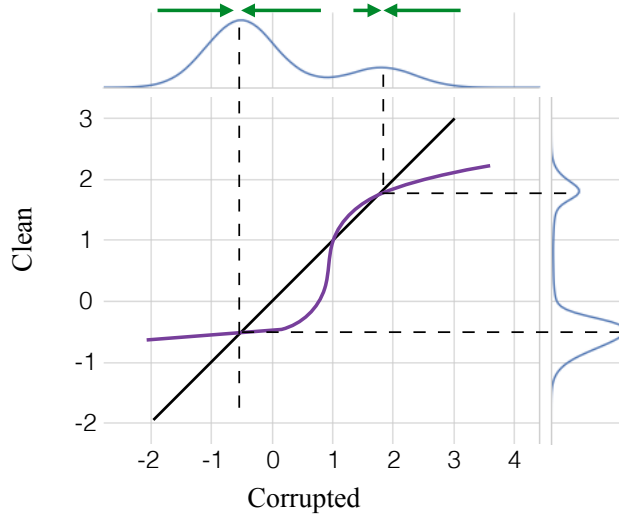
Figure 1: A depiction of an optimal denoising function for a bimodal distribution. The input for the function is the corrupted value (x axis) and the target is the clean value (y axis). The denoising function moves values towards higher probabilities as show by the green arrows.
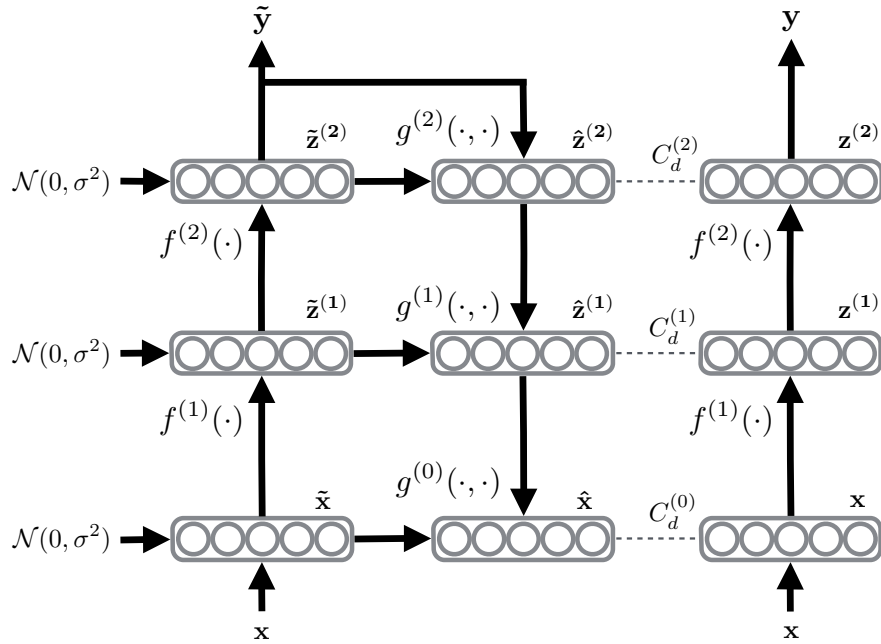

Figure 2: A conceptual illustration of the Ladder network when $L = 2$. The feedforward path $(\mathbf{x} \to \mathbf{z}^{(1)} \to \mathbf{z}^{(2)} \to \mathbf{y})$ shares the mappings $f^{(l)}$ with the corrupted feedforward path, or encoder $(\mathbf{x} \to \tilde{\mathbf{z}}^{(1)} \to \tilde{\mathbf{z}}^{(2)} \to \tilde{\mathbf{y}})$. The decoder $(\tilde{\mathbf{z}}^{(l)} \to \hat{\mathbf{z}}^{(l)} \to \hat{\mathbf{x}})$ consists of the denoising functions $g^{(l)}$ and has cost functions $C_d^{(l)}$ on each layer trying to minimize the difference between $\hat{\mathbf{z}}^{(l)}$ and $\mathbf{z}^{(l)}$. The output $\tilde{\mathbf{y}}$ of the encoder can also be trained to match available labels $t(n)$.

Algorithm 1 Calculation of the output $\mathbf{y}$ and cost function C of the Ladder network

Require: $\mathbf{x}(n)$
  # Corrupted encoder and classifier
  $\tilde{\mathbf{h}}^{(0)} \leftarrow \tilde{\mathbf{z}}^{(0)} \leftarrow \mathbf{x}(n) + \texttt{noise}$
  for l = 1 **to** L **do**
    $\tilde{\mathbf{z}}^{(l)} \leftarrow \texttt{batchnorm}(\mathbf{W}^{(l)}\tilde{\mathbf{h}}^{(l-1)}) + \texttt{noise}$
    $\tilde{\mathbf{h}}^{(l)} \leftarrow \texttt{activation}(\boldsymbol{\gamma}^{(l)} \odot (\tilde{\mathbf{z}}^{(l)} + \boldsymbol{\beta}^{(l)}))$
  end for
  $P(\tilde{\mathbf{y}} \mid \mathbf{x}) \leftarrow \tilde{\mathbf{h}}^{(L)}$
  # Clean encoder (for denoising targets)
  $\mathbf{h}^{(0)} \leftarrow \mathbf{z}^{(0)} \leftarrow \mathbf{x}(n)$
  for l = 1 **to** L **do**
    $\mathbf{z}_{\text{pre}}^{(l)} \leftarrow \mathbf{W}^{(l)}\mathbf{h}^{(l-1)}$
    $\boldsymbol{\mu}^{(l)} \leftarrow \texttt{batchmean}(\mathbf{z}_{\text{pre}}^{(l)})$
    $\boldsymbol{\sigma}^{(l)} \leftarrow \texttt{batchstd}(\mathbf{z}_{\text{pre}}^{(l)})$
    $\mathbf{z}^{(l)} \leftarrow \texttt{batchnorm}(\mathbf{z}_{\text{pre}}^{(l)})$
    $\mathbf{h}^{(l)} \leftarrow \texttt{activation}(\boldsymbol{\gamma}^{(l)} \odot (\mathbf{z}^{(l)} + \boldsymbol{\beta}^{(l)}))$
  end for

  # Final classification:
  $P(\mathbf{y} \mid \mathbf{x}) \leftarrow \mathbf{h}^{(L)}$
  # Decoder and denoising
  for l = L **to** 0 **do**
    if l = L **then**
      $\mathbf{u}^{(L)} \leftarrow \texttt{batchnorm}(\tilde{\mathbf{h}}^{(L)})$
    else
      $\mathbf{u}^{(l)} \leftarrow \texttt{batchnorm}(\mathbf{V}^{(l+1)}\hat{\mathbf{z}}^{(l+1)})$
    end if
    $\forall i : \hat{z}_i^{(l)} \leftarrow g(\tilde{z}_i^{(l)}, u_i^{(l)})$ # Eq. (2)
    $\forall i : \hat{z}_{i,\text{BN}}^{(l)} \leftarrow \frac{\hat{z}_i^{(l)} - \mu_i^{(l)}}{\sigma_i^{(l)}}$
  end for
  # Cost function C for training:
  $C \leftarrow 0$
  if $t(n)$ **then**
    $C \leftarrow -\log P(\tilde{\mathbf{y}} = t(n) \mid \mathbf{x}(n))$
  end if
  $C \leftarrow C + \sum_{l=0}^{L} \lambda_l \left\| \mathbf{z}^{(l)} - \hat{\mathbf{z}}_{\text{BN}}^{(l)} \right\|^2$ # Eq. (3)

3.1 General Steps for Implementing the Ladder Network

Consider training a classifier,[2] or a mapping from input $\mathbf{x}$ to output y with targets t, from a training set of pairs $\{\mathbf{x}(n), t(n) \mid 1 \leq n \leq N\}$. Semi-supervised learning (Chapelle *et al.*, 2006) studies how auxiliary unlabeled data $\{\mathbf{x}(n) \mid N + 1 \leq n \leq M\}$ can help in training a classifier. It is often the case that labeled data are scarce whereas unlabeled data are plentiful, that is $N \ll M$.

The Ladder network can improve results even without auxiliary unlabeled data but the original motivation was to make it possible to take well-performing feedforward classifiers and augment them with an auxiliary decoder as follows:

1. Train any standard feedforward neural network. The network type is not limited to standard MLPs, but the approach can be applied, for example, to convolutional or recurrent networks. This will be the encoder part of the Ladder network.

2. For each layer, analyze the conditional distribution of representations given the layer above, $p(\mathbf{z}^{(l)} \mid \mathbf{z}^{(l+1)})$. The observed distributions could resemble for example Gaussian distributions where the mean and variance depend on the values $\mathbf{z}^{(l+1)}$, bimodal distributions where the relative probability masses of the modes depend on the values $\mathbf{z}^{(l+1)}$, and so on.

3. Define a function $\hat{\mathbf{z}}^{(l)} = g(\tilde{\mathbf{z}}^{(l)}, \hat{\mathbf{z}}^{(l+1)})$ which can approximate the optimal denoising function for the family of observed distributions. The function g is therefore expected to form a reconstruction $\hat{\mathbf{z}}^{(l)}$ that resembles the clean $\mathbf{z}^{(l)}$ given the corrupted $\tilde{\mathbf{z}}^{(l)}$ and the higher-level reconstruction $\hat{\mathbf{z}}^{(l+1)}$.

4. Train the whole network in a fully-labeled or semi-supervised setting using standard optimization techniques such as stochastic gradient descent.

3.2 Fully Connected MLP as Encoder

As a starting point we use a fully connected MLP network with rectified linear units. We follow Ioffe and Szegedy (2015) and apply batch normalization to each preactivation including the topmost layer in the L-layer network. This serves two purposes. First, it improves convergence as a result of reduced covariate shift as originally proposed by Ioffe and Szegedy (2015). Second, as explained

---

[2]Here we only consider the case where the output $t(n)$ is a class label but it is trivial to apply the same approach to other regression tasks.

in Section 2, DSS-type cost functions for all but the input layer require some type of normalization to prevent the denoising cost from encouraging the trivial solution where the encoder outputs just constant values as these are the easiest to denoise. Batch normalization conveniently serves this purpose, too.

Formally, batch normalization for the layers $l = 1 \ldots L$ is implemented as

$$\mathbf{z}^{(l)} = \mathrm{N_B}(\mathbf{W}^{(l)}\mathbf{h}^{(l-1)})$$

$$\mathbf{h}^{(l)} = \phi\big(\boldsymbol{\gamma}^{(l)}(\mathbf{z}^{(l)} + \boldsymbol{\beta}^{(l)})\big),$$

where $\mathbf{h}^{(0)} = \mathbf{x}$, $\mathrm{N_B}$ is a component-wise batch normalization $\mathrm{N_B}(x_i) = (x_i - \hat{\mu}_{x_i})/\hat{\sigma}_{x_i}$, where $\hat{\mu}_{x_i}$ and $\hat{\sigma}_{x_i}$ are estimates calculated from the minibatch, $\boldsymbol{\gamma}^{(l)}$ and $\boldsymbol{\beta}^{(l)}$ are trainable parameters, and $\phi(\cdot)$ is the activation function such as the rectified linear unit (ReLU) for which $\phi(\cdot) = \max(0, \cdot)$. For outputs $\mathbf{y} = \mathbf{h}^{(L)}$ we always use the softmax activation. For some activation functions the scaling parameter $\boldsymbol{\beta}^{(l)}$ or the bias $\boldsymbol{\gamma}^{(l)}$ are redundant and we only apply them in non-redundant cases. For example, the rectified linear unit does not need scaling, the linear activation function needs neither scaling nor bias, but softmax requires both.

As explained in Section 2 and shown in Figure 2, the Ladder network requires two forward passes, one clean and one corrupted, which produce clean $\mathbf{z}^{(l)}$ and $\mathbf{h}^{(l)}$ and corrupted $\tilde{\mathbf{z}}^{(l)}$ and $\tilde{\mathbf{h}}^{(l)}$, respectively. We implemented corruption by adding isotropic Gaussian noise $\mathbf{n}$ to inputs and after each batch normalization:

$$
\begin{aligned}
\tilde{\mathbf{x}} &= \tilde{\mathbf{h}}^{(0)} = \mathbf{x} + \mathbf{n}^{(0)} \\
\tilde{\mathbf{z}}^{(l)}_{\mathrm{pre}} &= \mathbf{W}^{(l)}\tilde{\mathbf{h}}^{(l-1)} \\
\tilde{\mathbf{z}}^{(l)} &= \mathrm{N_B}(\tilde{\mathbf{z}}^{(l)}_{\mathrm{pre}}) + \mathbf{n}^{(l)} \\
\tilde{\mathbf{h}}^{(l)} &= \phi\big(\boldsymbol{\gamma}^{(l)}(\tilde{\mathbf{z}}^{(l)} + \boldsymbol{\beta}^{(l)})\big).
\end{aligned}
$$

Note that we collect the value $\tilde{\mathbf{z}}^{(l)}_{\mathrm{pre}}$ here because it will be needed in the decoder cost function in Section 3.3.

The supervised cost C_{c} is the average negative log probability of the noisy output $\tilde{\mathbf{y}}$ matching the target $t(n)$ given the inputs $\mathbf{x}(n)$

$$C_{\mathrm{c}} = -\frac{1}{N}\sum_{n=1}^{N} \log P(\tilde{\mathbf{y}} = t(n) \mid \mathbf{x}(n)).$$

In other words, we also use the noise to regularize supervised learning.

We saw networks with this structure reach close to state-of-the-art results in purely supervised learning (see e.g. Table 1), which makes them good starting points for improvement via semi-supervised learning by adding an auxiliary unsupervised task.

3.3 Decoder for Unsupervised Learning

When designing a suitable decoder to support unsupervised learning, we had to make a choice as to what kinds of distributions of the latent variables the decoder would optimally be able to denoise. We ultimately ended up choosing a parametrization that supports the optimal denoising of Gaussian latent variables. We also experimented with alternative denoising functions, more details of which can be found in Appendix B. Further analysis of different denoising functions was recently published by Pezeshki *et al.* (2015).

In order to derive the chosen parametrization and justify why it supports Gaussian latent variables, let us begin with the assumption that the noisy value of one latent variable $\tilde{z}$ that we want to denoise has the form $\tilde{z} = z + n$, where z is the clean latent variable value that has a Gaussian distribution with variance σ_z^2, and n is the Gaussian noise with variance σ_n^2.

We now want to estimate $\hat{z}$, a denoised version of $\tilde{z}$, so that the estimate minimizes the squared error of the difference to the clean latent variable values z. It can be shown that the functional form of $\hat{z} = g(\tilde{z})$ has to be linear in order to minimize the denoising cost, with the assumption being

6

that both the noise and the latent variable have a Gaussian distribution (Valpola, 2015, Section 4.1). Specifically, the result will be a weighted sum of the corrupted $\tilde{z}$ and a prior μ. The weight υ of the corrupted $\tilde{z}$ will be a function of the variance of z and n according to:

$$\upsilon = \frac{\sigma_z^2}{\sigma_z^2 + \sigma_n^2}$$

The denoising function will therefore have the form:

$$\hat{z} = g(\hat{z}) = \upsilon * \tilde{z} + (1 - \upsilon) * \mu = (\tilde{z} - \mu) * \upsilon + \mu \tag{1}$$

We could let υ and μ be trainable parameters of the model, where the model would learn some estimate of the optimal weighting υ and prior μ. The problem with this formulation is that it only supports the optimal denoising of latent variables with a Gaussian distribution, as the function g is linear wrt. $\tilde{z}$.

We relax this assumption by making the model only require the distribution of z of a layer to be Gaussian conditional on the values of the latent variables of the layer above. In a similar vein, in a layer of multiple latent variables we can assume that the latent variables are independent conditional on the latent variables of the layer above. The distribution of the latent variables $\mathbf{z}^{(l)}$ is therefore assumed to follow the distribution

$$p(\mathbf{z}^{(l)} \mid \mathbf{z}^{(l+1)}) = \prod_i p(z_i^{(l)} \mid \mathbf{z}^{(l+1)})$$

where $p(z_i^{(l)} \mid \mathbf{z}^{(l+1)})$ are conditionally independent Gaussian distributions.

One interpretation of this formulation is that we are modeling the distribution of $\mathbf{z}^{(l)}$ as a mixture of Gaussians with diagonal covariance matrices, where the value of the above layer $\mathbf{z}^{(l+1)}$ modulates the form of the Gaussian that $\mathbf{z}^{(l)}$ is distributed as. In practice, we will implement the dependence of $\mathbf{v}$ and $\boldsymbol{\mu}$ on $\hat{\mathbf{z}}^{(l+1)}$ with a batch normalized projection from $\hat{\mathbf{z}}^{(l+1)}$ followed by an expressive nonlinearity with trainable parameters. The final formulation of the denoising function is therefore

$$\hat{z}_i^{(l)} = g_i(\tilde{z}_i^{(l)}, u_i^{(l)}) = \left(\tilde{z}_i^{(l)} - \mu_i(u_i^{(l)}) \right) \upsilon_i(u_i^{(l)}) + \mu_i(u_i^{(l)}) \tag{2}$$

where $u_i^{(l)}$ propagates information from $\hat{\mathbf{z}}^{(l+1)}$ by a batch normalized projection:

$$\mathbf{u}^{(l)} = \mathrm{N}_{\mathrm{B}}(\mathbf{V}^{(l+1)}\hat{\mathbf{z}}^{(l+1)}),$$

where the matrix $\mathbf{V}^{(l)}$ has the same dimension as the transpose of $\mathbf{W}^{(l)}$ on the encoder side. The projection vector $\mathbf{u}^{(l)}$ therefore has the same dimensionality as $\mathbf{z}^{(l)}$. Furthermore, the functions $\mu_i(u_i^{(l)})$ and $\upsilon_i(u_i^{(l)})$ are modeled as expressive nonlinearities:

$$\mu_i(u_i^{(l)}) = a_{1,i}^{(l)}\mathtt{sigmoid}(a_{2,i}^{(l)}u_i^{(l)} + a_{3,i}^{(l)}) + a_{4,i}^{(l)}u_i^{(l)} + a_{5,i}^{(l)}$$

$$\upsilon_i(u_i^{(l)}) = a_{6,i}^{(l)}\mathtt{sigmoid}(a_{7,i}^{(l)}u_i^{(l)} + a_{8,i}^{(l)}) + a_{9,i}^{(l)}u_i^{(l)} + a_{10,i}^{(l)},$$

where $a_{1,i}^{(l)} \ldots a_{10,i}^{(l)}$ are the trainable parameters of the nonlinearity for each neuron i in each layer l. It is worth noting that in this parametrization, each denoised value $\hat{z}_i^{(l)}$ only depends on $\tilde{z}_i^{(l)}$ and not the full $\tilde{\mathbf{z}}^{(l)}$. This means that the model can only optimally denoise conditionally independent distributions. While this nonlinearity makes the number of parameters in the decoder slightly higher than in the encoder, the difference is insignificant as most of the parameters are in the vertical projection mappings $\mathbf{W}^{(l)}$ and $\mathbf{V}^{(l)}$, which have the same dimensions (apart from transposition). Note the slight abuse of the notation here since $g_i^{(l)}$ is now a function of the scalars $\tilde{z}_i^{(l)}$ and $u_i^{(l)}$ rather than the full vectors $\tilde{\mathbf{z}}^{(l)}$ and $\hat{\mathbf{z}}^{(l+1)}$. Given $\mathbf{u}^{(l)}$, this parametrization is linear with respect to $\tilde{\mathbf{z}}^{(l)}$, and both the slope and the bias depend nonlinearly on $\mathbf{u}^{(l)}$, as we hoped.

For the lowest layer, $\hat{\mathbf{x}} = \hat{\mathbf{z}}^{(0)}$ and $\tilde{\mathbf{x}} = \tilde{\mathbf{z}}^{(0)}$ by definition, and for the highest layer we chose $\mathbf{u}^{(L)} = \tilde{\mathbf{y}}$. This allows the highest-layer denoising function to utilize prior information about the

classes being mutually exclusive, which seems to improve convergence in cases where there are very few labeled samples.

As a side note, if the values of $\mathbf{z}^{(l)}$ are truly independently distributed Gaussians, there is nothing left for the layer above, $\hat{\mathbf{z}}^{(l+1)}$, to model. In that case, a mixture of Gaussians is not needed to model $\mathbf{z}^{(l)}$, but a diagonal Gaussian which can be modeled with a linear denoising function with constant values for v and μ as in Equation 1, would suffice. In this parametrization all correlations, non-linearities, and non-Gaussianities in the latent variables $\mathbf{z}^{(l)}$ have to be represented by modulations from the layers above for optimal denoising. As the parametrization allows the distribution of $\mathbf{z}^{(l)}$ to be modulated by $\mathbf{z}^{(l+1)}$ through $\mathbf{u}^{(l)}$, it encourages the decoder to find representations $\mathbf{z}^{(l)}$ that have high mutual information with $\mathbf{z}^{(l+1)}$. This is crucial as it allows supervised learning to have an indirect influence on the representations learned by the unsupervised decoder: any abstractions selected by supervised learning will bias the lower levels to find more representations which carry information about the same abstractions.

The cost function for the unsupervised path is the mean squared reconstruction error per neuron, but there is a slight twist which we found to be important. Batch normalization has useful properties, as noted in Section 3.2, but it also introduces noise which affects both the clean and corrupted encoder pass. This noise is highly correlated between $\mathbf{z}^{(l)}$ and $\tilde{\mathbf{z}}^{(l)}$ because the noise derives from the statistics of the samples that happen to be in the same minibatch. This highly correlated noise in $\mathbf{z}^{(l)}$ and $\tilde{\mathbf{z}}^{(l)}$ biases the denoising functions to be simple copies[3] $\hat{\mathbf{z}}^{(l)} \approx \tilde{\mathbf{z}}^{(l)}$.

The solution we found was to implicitly use the projections $\mathbf{z}_{\text{pre}}^{(l)}$ as the target for denoising and scale the cost function in such a way that the term appearing in the error term is the batch normalized $\mathbf{z}^{(l)}$ instead. For the moment, let us see how that works for a scalar case:

$$
\begin{aligned}
\frac{1}{\sigma^2}\left\| z_{\text{pre}} - \hat{z} \right\|^2 &= \left\| \frac{z_{\text{pre}} - \mu}{\sigma} - \frac{\hat{z} - \mu}{\sigma} \right\|^2 = \left\| z - \hat{z}_{\text{BN}} \right\|^2 \\
z &= \mathrm{N_B}(z_{\text{pre}}) = \frac{z_{\text{pre}} - \mu}{\sigma} \\
\hat{z}_{\text{BN}} &= \frac{\hat{z} - \mu}{\sigma},
\end{aligned}
$$

where μ and σ are the batch mean and batch std of z_{pre}, respectively, that were used in batch normalizing z_{pre} into z. The unsupervised denoising cost function C_{d} is thus

$$
C_{\text{d}} = \sum_{l=0}^{L} \lambda_l C_{\text{d}}^{(l)} = \sum_{l=0}^{L} \frac{\lambda_l}{N m_l} \sum_{n=1}^{N} \left\| \mathbf{z}^{(l)}(n) - \hat{\mathbf{z}}_{\text{BN}}^{(l)}(n) \right\|^2, \tag{3}
$$

where m_l is the layer's width, N the number of training samples, and the hyperparameter λ_l a layer-wise multiplier determining the importance of the denoising cost.

The model parameters $\mathbf{W}^{(l)}, \boldsymbol{\gamma}^{(l)}, \boldsymbol{\beta}^{(l)}, \mathbf{V}^{(l)}, \mathbf{a}_i^{(l)}, b_i^{(l)}, and c_i^{(l)}$ can be trained simply by using the backpropagation algorithm to optimize the total cost $C = C_{\text{c}} + C_{\text{d}}$. The feedforward pass of the full Ladder network is listed in Algorithm 1. Classification results are read from the $\mathbf{y}$ in the clean feedforward path.

3.4 Variations

Section 3.3 detailed how to build a decoder for the Ladder network to match the fully connected encoder described in Section 3.2. It is easy to extend the same approach to other encoders, for instance, convolutional neural networks (CNN). For the decoder of fully connected networks we used vertical mappings whose shape is a transpose of the encoder mapping. The same treatment works for the convolution operations: in the networks we have tested in this paper, the decoder has convolutions whose parametrization mirrors the encoder and effectively just reverses the flow of

---

[3]The whole point of using *denoising* autoencoders rather than regular autoencoders is to prevent skip connections from short-circuiting the decoder and force the decoder to learn meaningful abstractions which help in denoising.

information. As the idea of convolution is to reduce the number of parameters by weight sharing, we applied this to the parameters of the denoising function g, too.

Many convolutional networks use pooling operations with stride; that is, they downsample the spatial feature maps. The decoder needs to compensate for this with a corresponding upsampling. There are several alternative ways to implement this and in this paper we chose the following options: 1) on the encoder side, pooling operations are treated as separate layers with their own batch normalization and linear activation function, and 2) the downsampling of the pooling on the encoder side is compensated for by upsampling with copying on the decoder side. This provides multiple targets for the decoder to match, helping the decoder to recover the information lost on the encoder side.

It is worth noting that a simple special case of the decoder is a model where $\lambda_l = 0$ when $l < L$. This corresponds to a denoising cost only on the top layer and means that most of the decoder can be omitted. This model, which we call the Γ-model because of the shape of the graph, is useful as it can easily be plugged into any feedforward network without decoder implementation. In addition, the Γ-model is the same for MLPs and convolutional neural networks. The encoder in the Γ-model still includes both the clean and the corrupted paths as in the full ladder.

4   Experiments

With the experiments with the MNIST and CIFAR-10 datasets, we wanted to compare our method to other semi-supervised methods but also show that we can attach the decoder both to a fully connected MLP network and to a convolutional neural network, both of which were described in Section 3. We also wanted to compare the performance of the simpler Γ-model (Sec. 3.4) to the full Ladder network and experimented with only having a cost function on the input layer. With CIFAR-10, we only tested the Γ-model.

We also measured the performance of the supervised baseline models which only included the encoder and the supervised cost function. In all cases where we compared these directly with Ladder networks, we did our best to optimize the hyperparameters and regularization of the baseline supervised learning models so that any improvements could not be explained, for example, by the lack of suitable regularization which would then have been provided by the denoising costs.

With convolutional networks, our focus was exclusively on semi-supervised learning. The supervised baselines for all labels only intend to show that the performance of the selected network architectures is in line with the ones reported in the literature. We make claims neither about the optimality nor the statistical significance of these baseline results.

We used the Adam optimization algorithm (Kingma and Ba, 2015) for the weight updates. The learning rate was 0.002 for the first part of the learning, followed by an annealing phase during which the learning rate was linearly reduced to zero. The minibatch size was 100. The source code for all the experiments is available at `https://github.com/arasmus/ladder` unless explicitly noted in the text.

4.1   MNIST dataset

For evaluating semi-supervised learning, we used the standard 10,000 test samples as a held-out test set and randomly split the standard 60,000 training samples into a 10,000-sample validation set and used $M = 50,000$ samples as the training set. From the training set, we randomly chose $N = 100$, 1000, or all labels for the supervised cost.[4] All the samples were used for the decoder, which does not need the labels. The validation set was used for evaluating the model structure and hyperparameters. We also balanced the classes to ensure that no particular class was over-represented. We repeated each training 10 times, varying the random seed that was used for the splits.

---

[4]In all the experiments, we were careful not to optimize any parameters, hyperparameters, or model choices on the basis of the results on the held-out test samples. As is customary, we used 10,000 labeled validation samples even for those settings where we only used 100 labeled samples for training. Obviously, this is not something that could be done in a real case with just 100 labeled samples. However, MNIST classification is such an easy task, even in the permutation invariant case, that 100 labeled samples there correspond to a far greater number of labeled samples in many other datasets.

Table 1: A collection of previously reported MNIST test errors in the permutation invariant setting followed by the results with the Ladder network. * = SVM. Standard deviation in parentheses.

Test error % with # of used labels	100	1000	All
Semi-sup. Embedding (Weston *et al.*, 2012)	16.86	5.73	1.5
Transductive SVM (from Weston *et al.*, 2012)	16.81	5.38	1.40*
MTC (Rifai *et al.*, 2011)	12.03	3.64	0.81
Pseudo-label (Lee, 2013)	10.49	3.46	
AtlasRBF (Pitelis *et al.*, 2014)	8.10 ($\pm$ 0.95)	3.68 ($\pm$ 0.12)	1.31
DGN (Kingma *et al.*, 2014)	3.33 ($\pm$ 0.14)	2.40 ($\pm$ 0.02)	0.96
DBM, Dropout (Srivastava *et al.*, 2014)			0.79
Adversarial (Goodfellow *et al.*, 2015)			0.78
Virtual Adversarial (Miyato *et al.*, 2015)	2.12	1.32	0.64 ($\pm$ 0.03)
Baseline: MLP, BN, Gaussian noise	21.74 ($\pm$ 1.77)	5.70 ($\pm$ 0.20)	0.80 ($\pm$ 0.03)
Γ-model (Ladder with only top-level cost)	3.06 ($\pm$ 1.44)	1.53 ($\pm$ 0.10)	0.78 ($\pm$ 0.03)
Ladder, only bottom-level cost	1.09 ($\pm$0.32)	0.90 ($\pm$ 0.05)	0.59 ($\pm$ 0.03)
Ladder, full	**1.06** ($\pm$ 0.37)	**0.84** ($\pm$ 0.08)	**0.57** ($\pm$ 0.02)

After optimizing the hyperparameters, we performed the final test runs using all the $M = 60,000$ training samples with 10 different random initializations of the weight matrices and data splits. We trained all the models for 100 epochs followed by 50 epochs of annealing. With minibatch size of 100, this amounts to 75,000 weight updates for the validation runs and 90,000 for the final test runs.

4.1.1 Fully connected MLP

A useful test for general learning algorithms is the permutation invariant MNIST classification task. Permutation invariance means that the results need to be invariant with respect to permutation of the elements of the input vector. In other words, one is not allowed to use prior information about the spatial arrangement of the input pixels. This excludes, among others, convolutional networks and geometric distortions of the input images.

We chose the layer sizes of the baseline model to be 784-1000-500-250-250-250-10. The network is deep enough to demonstrate the scalability of the method but does not yet represent overkill for MNIST.

The hyperparameters we tuned for each model are the noise level that is added to the inputs and to each layer, and the denoising cost multipliers $\lambda^{(l)}$. We also ran the supervised baseline model with various noise levels. For models with just one cost multiplier, we optimized them with a search grid $\{\ldots, 0.1, 0.2, 0.5, 1, 2, 5, 10, \ldots\}$. Ladder networks with a cost function on all their layers have a much larger search space and we explored it much more sparsely. For instance, the optimal model we found for $N = 100$ labels had $\lambda^{(0)} = 1000$, $\lambda^{(1)} = 10$, and $\lambda^{(\geq 2)} = 0.1$. A good value for the std of the Gaussian corruption noise $\mathbf{n}^{(l)}$ was mostly 0.3 but with $N = 1000$ labels, 0.2 was a better value. For the complete set of selected denoising cost multipliers and other hyperparameters, please refer to the code.

The results presented in Table 1 show that the proposed method outperforms all the previously reported results. Encouraged by the good results, we also tested with $N = 50$ labels and got a test error of 1.62 % ($\pm$ 0.65 %).

The simple Γ-model also performed surprisingly well, particularly for $N = 1000$ labels. With $N = 100$ labels, all the models sometimes failed to converge properly. With bottom level or full costs in Ladder, around 5 % of runs result in a test error of over 2 %. In order to be able to estimate the average test error reliably in the presence of such random outliers, we ran 40 instead of 10 test runs with random initializations.

10

Table 2: CNN results for MNIST

Test error without data augmentation % with # of used labels	100	all
EmbedCNN (Weston *et al.*, 2012)	7.75	
SWWAE (Zhao *et al.*, 2015)	9.17	0.71
Baseline: Conv-Small, supervised only	6.43 ($\pm$ 0.84)	0.36
Conv-FC	0.99 ($\pm$ 0.15)	
Conv-Small, Γ-model	**0.89** ($\pm$ 0.50)	

4.1.2 Convolutional networks

We tested two convolutional networks for the general MNIST classification task but omitted data augmentation such as geometric distortions. We focused on the 100-label case since with more labels the results were already so good even in the more difficult permutation invariant task.

The first network was a straightforward extension of the fully connected network tested in the permutation invariant case. We turned the first fully connected layer into a convolution with 26-by-26 filters, resulting in a 3-by-3 spatial map of 1000 features. Each of the nine spatial locations was processed independently by a network with the same structure as in the previous section, finally resulting in a 3-by-3 spatial map of 10 features. These were pooled with a global mean-pooling layer. Essentially we thus convolved the image with the complete fully connected network. Depooling on the topmost layer and deconvolutions on the layers below were implemented as described in Section 3.4. Since the internal structure of each of the nine almost independent processing paths was the same as in the permutation invariant task, we used the same hyperparameters that were optimal for the permutation invariant task. In Table 2, this model is referred to as Conv-FC.

With the second network, which was inspired by ConvPool-CNN-C from Springenberg *et al.* (2014), we only tested the Γ-model. The MNIST classification task can typically be solved with a smaller number of parameters than CIFAR-10, for which this topology was originally developed, so we modified the network by removing layers and reducing the number of parameters in the remaining layers. In addition, we observed that adding a small fully connected layer with 10 neurons on top of the global mean pooling layer improved the results in the semi-supervised task. We did not tune other parameters than the noise level, which was chosen from $\{0.3, 0.45, 0.6\}$ using the validation set. The exact architecture of this network is detailed in Table 4 in Appendix A. It is referred to as Conv-Small since it is a smaller version of the network used for the CIFAR-10 dataset.

The results in Table 2 confirm that even the single convolution on the bottom level improves the results over the fully connected network. More convolutions improve the Γ-model significantly, although the high variance of the results suggests that the model still suffers from confirmation bias. The Ladder network with denoising targets on every level converges much more reliably. Taken together, these results suggest that combining the generalization ability of convolutional networks[5] and efficient unsupervised learning of the full Ladder network would have resulted in even better performance but this was left for future work.

4.2 Convolutional networks on CIFAR-10

The CIFAR-10 dataset consists of small 32-by-32 RGB images from 10 classes. There are 50,000 labeled samples for training and 10,000 for testing. Like the MNIST dataset, it has been used for testing semi-supervised learning so we decided to test the simple Γ-model with a convolutional network that has been reported to perform well in the standard supervised setting with all labels. We tested a few model architectures and selected ConvPool-CNN-C by Springenberg *et al.* (2014). We also evaluated the strided convolutional version by Springenberg *et al.* (2014), and while it performed well with all labels, we found that the max-pooling version overfitted less with fewer labels, and thus used it.

---

[5]In general, convolutional networks excel in the MNIST classification task. The performance of the fully supervised Conv-Small with all labels is in line with the literature and is provided as a rough reference only (only one run, no attempts to optimize, not available in the code package).

Table 3: Test results for CNN on CIFAR-10 dataset without data augmentation

Test error % with # of used labels	4 000	All
All-Convolutional ConvPool-CNN-C (Springenberg *et al.*, 2014)		9.31
Spike-and-Slab Sparse Coding (Goodfellow *et al.*, 2012)	31.9	
Baseline: Conv-Large, supervised only	23.33 ($\pm$ 0.61)	9.27
Conv-Large, Γ-model	**20.40** ($\pm$ 0.47)	

The main differences to ConvPool-CNN-C are the use of Gaussian noise instead of dropout and the convolutional per-channel batch normalization following Ioffe and Szegedy (2015). While dropout was useful with all labels, it did not seem to offer any advantage over additive Gaussian noise with fewer labels. For a more detailed description of the model, please refer to model Conv-Large in Table 4.

While testing the purely supervised model performance with a limited number of labeled samples ($N = 4000$), we found out that the model overfitted quite severely: the training error for most samples decreased so much that the network effectively learned nothing from them as the network was already very confident about their classification. The network was equally confident about validation samples even when they were misclassified. We noticed that we could regularize the network by stripping away the scaling parameter $\beta^{(L)}$ from the last layer. This means that the variance of the input to the softmax is restricted to unity. We also used this setting with the corresponding Γ-model although the denoising target already regularizes the network significantly and the improvement was not as pronounced.

The hyperparameters (noise level, denoising cost multipliers, and number of epochs) for all models were optimized using $M = 40,000$ samples for training and the remaining $10,000$ samples for validation. After the best hyperparameters were selected, the final model was trained with these settings on all the $M = 50,000$ samples. All experiments were run with with four different random initializations of the weight matrices and data splits. We applied global contrast normalization and whitening following Goodfellow *et al.* (2013b), but no data augmentation was used.

The results are shown in Table 3. The supervised reference was obtained with a model closer to the original ConvPool-CNN-C in the sense that dropout rather than additive Gaussian noise was used for regularization.[6] We spent some time tuning the regularization of our fully supervised baseline model for $N = 4000$ labels and indeed, its results exceed the previous state of the art. This tuning was important to make sure that the improvement offered by the denoising target of the Γ-model is not a sign of a poorly regularized baseline model. Although the improvement is not as dramatic as with the MNIST experiments, it came with a very simple addition to standard supervised training.

5   Related Work

Early works on semi-supervised learning (McLachlan, 1975; Titterington *et al.*, 1985) proposed an approach where inputs $\mathbf{x}$ are first assigned to clusters, and each cluster has its class label. Unlabeled data would affect the shapes and sizes of the clusters, and thus alter the classification result. This approach can be reinterpreted as input vectors being corrupted copies $\tilde{\mathbf{x}}$ of the ideal input vectors $\mathbf{x}$ (the cluster centers), and the classification mapping being split into two parts: first denoising $\tilde{\mathbf{x}}$ into $\mathbf{x}$ (possibly probabilistically), and then labeling $\mathbf{x}$.

It is well known (see, e.g., Zhang and Oles, 2000) that when a probabilistic model that directly estimates $P(y \mid \mathbf{x})$ is being trained, unlabeled data cannot help. One way to study this is to assign probabilistic labels $q(y(n)) = P(y(n) \mid \mathbf{x}(n))$ to unlabeled inputs $\mathbf{x}(n)$ and try to train $P(y \mid \mathbf{x})$ using those labels: it can be shown (see, e.g., Raiko *et al.*, 2015, Eq. (31)) that the gradient will vanish. There are different ways of circumventing this phenomenon by adjusting the assigned labels $q(y(n))$. These are all related to the Γ-model.

---

[6]Same caveats hold for this fully supervised reference result for all labels as with MNIST: only one run, no attempts to optimize, not available in the code package.

Label propagation methods (Szummer and Jaakkola, 2003) estimate $P(y \mid \mathbf{x})$, but adjust probabilistic labels $q(y(n))$ on the basis of the assumption that the nearest neighbors are likely to have the same label. The labels start to propagate through regions with high-density $P(\mathbf{x})$. The Γ-model implicitly assumes that the labels are uniform in the vicinity of a clean input since corrupted inputs need to produce the same label. This produces a similar effect: the labels start to propagate through regions with high density $P(\mathbf{x})$. Weston *et al.* (2012) explored deep versions of label propagation.

Co-training (Blum and Mitchell, 1998) assumes we have multiple views on $\mathbf{x}$, say $\mathbf{x} = (\mathbf{x}^{(1)}, \mathbf{x}^{(2)})$. When we train classifiers for the different views, we know that even for the unlabeled data, the true label is the same for each view. Each view produces its own probabilistic labeling $q^{(j)}(y(n)) = P(y(n) \mid \mathbf{x}(n)^{(j)})$ and their combination $q(y(n))$ can be fed to train the individual classifiers. If we interpret having several corrupted copies of an input as different views on it, we see the relationship to the proposed method.

Lee (2013) adjusts the assigned labels $q(y(n))$ by rounding the probability of the most likely class to one and others to zero. The training starts by trusting only the true labels and then gradually increasing the weight of the so-called *pseudo-labels*. Similar scheduling could be tested with our Γ-model as it seems to suffer from confirmation bias. It may well be that the denoising cost which is optimal at the beginning of the learning is smaller than the optimal one at later stages of learning.

Dosovitskiy *et al.* (2014) pre-train a convolutional network with unlabeled data by treating each clean image as its own class. During training, the image is corrupted by transforming its location, scaling, rotation, contrast, and color. This helps to find features that are invariant to the transformations that are used. Discarding the last classification layer and replacing it with a new classifier trained on real labeled data leads to surprisingly good experimental results.

There is an interesting connection between our Γ-model and the contractive cost used by Rifai *et al.* (2011): a linear denoising function $\hat{z}_i^{(L)} = a_i \tilde{z}_i^{(L)} + b_i$, where a_i and b_i are parameters, turns the denoising cost into a stochastic estimate of the contractive cost.

Recently Miyato *et al.* (2015) achieved impressive results with a regularization method that is similar to the idea of contractive cost. They required the output of the network to change as little as possible close to the input samples. As this requires no labels, they were able to use unlabeled samples for regularization. While their semi-supervised results were not as good as ours with a denoising target on the input layer, their results with full labels come very close. Their cost function is on the last layer which suggests that the approaches are complementary and could be combined, potentially improving the results further.

So far we have reviewed semi-supervised methods which have an unsupervised cost function on the output layer only and therefore are related to our Γ-model. We will now move to other semi-supervised methods that concentrate on modeling the joint distribution of the inputs and the labels.

The Multi-prediction deep Boltzmann machine (MP-DBM) (Goodfellow *et al.*, 2013a) is a way to train a DBM with backpropagation through variational inference. The targets of the inference include both supervised targets (classification) and unsupervised targets (reconstruction of missing inputs) that are used in training simultaneously. The connections through the inference network are somewhat analogous to our lateral connections. Specifically, there are inference paths from observed inputs to reconstructed inputs that do not go all the way up to the highest layers. Compared to our approach, MP-DBM requires an iterative inference with some initialization for the hidden activations, whereas in our case, the inference is a simple single-pass feedforward procedure.

The Deep AutoRegressive Network (Gregor *et al.*, 2014) is an unsupervised method for learning representations that also uses lateral connections in the hidden representations. The connectivity within the layer is rather different from ours, though: each unit h_i receives input from the preceding units $h_1 \ldots h_{i-1}$, whereas in our case each unit $\hat{z}_i$ receives input only from z_i. Their learning algorithm is based on approximating a gradient of a description length measure, whereas we use a gradient of a simple loss function.

Kingma *et al.* (2014) proposed deep generative models for semi-supervised learning, based on variational autoencoders. Their models can be trained with the variational EM algorithm, stochastic gradient variational Bayes, or stochastic backpropagation. They also experimented on a stacked version (called M1+M2) where the bottom autoencoder M1 reconstructs the input data, and the top

autoencoder M2 can concentrate on classification and on reconstructing only the hidden representation of M1. The stacked version performed the best, hinting that it might be important not to carry all the information up to the highest layers. Compared with the Ladder network, an interesting point is that the variational autoencoder computes the posterior estimate of the latent variables with the encoder alone while the Ladder network uses the decoder too to compute an implicit posterior approximate (the encoder provides the likelihood part, which gets combined with the prior). It will be interesting to see whether the approaches can be combined. A Ladder-style decoder might provide the posterior and another decoder could then act as the generative model of variational autoencoders.

Zeiler *et al.* (2011) train deep convolutional autoencoders in a manner comparable to ours. They define max-pooling operations in the encoder to feed the max function upwards to the next layer, while the argmax function is fed laterally to the decoder. The network is trained one layer at a time using a cost function that includes a pixel-level reconstruction error, and a regularization term to promote sparsity. Zhao *et al.* (2015) use a similar structure and call it the stacked what-where autoencoder (SWWAE). Their network is trained simultaneously to minimize a combination of the supervised cost and reconstruction errors on each level, just like ours.

Recently Bengio (2014) proposed target propagation as an alternative to backpropagation. The idea is to base learning not on errors and gradients but on expectations. This is very similar to the idea of denoising source separation and therefore resembles the propagation of expectations in the decoder of the Ladder network. In the Ladder network, the additional lateral connections between the encoder and the decoder play an important role and it remains to be seen whether the lateral connections are compatible with target propagation. Nevertheless, it is an interesting possibility that while the Ladder network includes two mechanisms for propagating information, backpropagation of gradients and forward propagation of expectations in the decoder, it may be possible to rely solely on the latter, thus avoiding problems related to the propagation of gradients through many layers, such as exploding gradients.

6 Discussion

We showed how a simultaneous unsupervised learning task improves CNN and MLP networks reaching the state of the art in various semi-supervised learning tasks. In particular, the performance obtained with very small numbers of labels is much better than previous published results, which shows that the method is capable of making good use of unsupervised learning. However, the same model also achieves state-of-the-art results and a significant improvement over the baseline model with full labels in permutation invariant MNIST classification, which suggests that the unsupervised task does not disturb supervised learning.

The proposed model is simple and easy to implement with many existing feedforward architectures, as the training is based on backpropagation from a simple cost function. It is quick to train and the convergence is fast, thanks to batch normalization.

Not surprisingly, the largest improvements in performance were observed in models which have a large number of parameters relative to the number of available labeled samples. With CIFAR-10, we started with a model which was originally developed for a fully supervised task. This has the benefit of building on existing experience but it may well be that the best results will be obtained with models which have far more parameters than fully supervised approaches could handle.

An obvious future line of research will therefore be to study what kind of encoders and decoders are best suited to the Ladder network. In this work, we made very small modifications to the encoders, whose structure has been optimized for supervised learning, and we designed the parametrization of the vertical mappings of the decoder to mirror the encoder: the flow of information is just reversed. There is nothing preventing the decoder from having a different structure than the encoder.

An interesting future line of research will be the extension of the Ladder networks to the temporal domain. While datasets with millions of labeled samples for still images exist, it is prohibitively costly to label thousands of hours of video streams. The Ladder networks can be scaled up easily and therefore offer an attractive approach for semi-supervised learning in such large-scale problems.

Acknowledgements

References

Bastien, F., Lamblin, P., Pascanu, R., Bergstra, J., Goodfellow, I. J., Bergeron, A., Bouchard, N., and Bengio, Y. (2012). Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop.

Bengio, Y. (2014). How auto-encoders could provide credit assignment in deep networks via target propagation. *arXiv:1407.7906*.

Bengio, Y., Yao, L., Alain, G., and Vincent, P. (2013). Generalized denoising auto-encoders as generative models. In *Advances in Neural Information Processing Systems 26 (NIPS 2013)*, pages 899–907.

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 2010)*. Oral Presentation.

Blum, A. and Mitchell, T. (1998). Combining labeled and unlabeled data with co-training. In *Proc. of the Eleventh Annual Conference on Computational Learning Theory (COLT '98)*, pages 92–100.

Chapelle, O., Schölkopf, B., Zien, A., *et al.* (2006). *Semi-supervised learning*. MIT Press.

Dosovitskiy, A., Springenberg, J. T., Riedmiller, M., and Brox, T. (2014). Discriminative unsupervised feature learning with convolutional neural networks. In *Advances in Neural Information Processing Systems 27 (NIPS 2014)*, pages 766–774.

Goodfellow, I., Bengio, Y., and Courville, A. C. (2012). Large-scale feature learning with spike-and-slab sparse coding. In *Proc. of ICML 2012*, pages 1439–1446.

Goodfellow, I., Mirza, M., Courville, A., and Bengio, Y. (2013a). Multi-prediction deep Boltzmann machines. In *Advances in Neural Information Processing Systems 26 (NIPS 2013)*, pages 548–556.

Goodfellow, I., Shlens, J., and Szegedy, C. (2015). Explaining and harnessing adversarial examples. In *the International Conference on Learning Representations (ICLR 2015)*. arXiv:1412.6572.

Goodfellow, I. J., Warde-Farley, D., Mirza, M., Courville, A., and Bengio, Y. (2013b). Maxout networks. In *Proc. of ICML 2013*.

Gregor, K., Danihelka, I., Mnih, A., Blundell, C., and Wierstra, D. (2014). Deep autoregressive networks. In *Proc. of ICML 2014*, Beijing, China.

Hinton, G. E. and Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, **313**(5786), 504–507.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning (ICML)*, pages 448–456.

Kingma, D. and Ba, J. (2015). Adam: A method for stochastic optimization. In *the International Conference on Learning Representations (ICLR 2015)*, San Diego. arXiv:1412.6980.

Kingma, D. P., Mohamed, S., Rezende, D. J., and Welling, M. (2014). Semi-supervised learning with deep generative models. In *Advances in Neural Information Processing Systems 27 (NIPS 2014)*, pages 3581–3589.

Lee, D.-H. (2013). Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In *Workshop on Challenges in Representation Learning, ICML 2013*.

McLachlan, G. (1975). Iterative reclassification procedure for constructing an asymptotically optimal rule of allocation in discriminant analysis. *J. American Statistical Association*, **70**, 365–369.

Miyato, T., Maeda, S., Koyama, M., Nakae, K., and Ishii, S. (2015). Distributional smoothing by virtual adversarial examples. *arXiv:1507.00677*.

Pezeshki, M., Fan, L., Brakel, P., Courville, A., and Bengio, Y. (2015). Deconstructing the ladder network architecture. *arXiv:1511.06430*.

Pitelis, N., Russell, C., and Agapito, L. (2014). Semi-supervised learning using an unsupervised atlas. In *Machine Learning and Knowledge Discovery in Databases (ECML PKDD 2014)*, pages 565–580. Springer.

Raiko, T., Berglund, M., Alain, G., and Dinh, L. (2015). Techniques for learning binary stochastic feedforward neural networks. In *ICLR 2015*, San Diego.

Ranzato, M. A. and Szummer, M. (2008). Semi-supervised learning of compact document representations with deep networks. In *Proc. of ICML 2008*, pages 792–799. ACM.

Rasmus, A., Raiko, T., and Valpola, H. (2015a). Denoising autoencoder with modulated lateral connections learns invariant representations of natural images. *arXiv:1412.7210*.

Rasmus, A., Valpola, H., and Raiko, T. (2015b). Lateral connections in denoising autoencoders support supervised learning. *arXiv:1504.08215*.

Rifai, S., Dauphin, Y. N., Vincent, P., Bengio, Y., and Muller, X. (2011). The manifold tangent classifier. In *Advances in Neural Information Processing Systems 24 (NIPS 2011)*, pages 2294–2302.

Särelä, J. and Valpola, H. (2005). Denoising source separation. *JMLR*, **6**, 233–272.

Sietsma, J. and Dow, R. J. (1991). Creating artificial neural networks that generalize. *Neural networks*, **4**(1), 67–79.

Springenberg, J. T., Dosovitskiy, A., Brox, T., and Riedmiller, M. A. (2014). Striving for simplicity: The all convolutional net. *arxiv:1412.6806*.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *JMLR*, **15**(1), 1929–1958.

Suddarth, S. C. and Kergosien, Y. (1990). Rule-injection hints as a means of improving network performance and learning time. In *Proceedings of the EURASIP Workshop 1990 on Neural Networks*, pages 120–129. Springer.

Szummer, M. and Jaakkola, T. (2003). Partially labeled classification with Markov random walks. *Advances in Neural Information Processing Systems 15 (NIPS 2002)*, **14**, 945–952.

Titterington, D., Smith, A., and Makov, U. (1985). Statistical analysis of finite mixture distributions. In *Wiley Series in Probability and Mathematical Statistics*. Wiley.

Valpola, H. (2015). From neural PCA to deep unsupervised learning. In *Adv. in Independent Component Analysis and Learning Machines*, pages 143–171. Elsevier. arXiv:1411.7783.

van Merriënboer, B., Bahdanau, D., Dumoulin, V., Serdyuk, D., Warde-Farley, D., Chorowski, J., and Bengio, Y. (2015). Blocks and fuel: Frameworks for deep learning. *CoRR*, **abs/1506.00619**.

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., and Manzagol, P.-A. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *JMLR*, **11**, 3371–3408.

Weston, J., Ratle, F., Mobahi, H., and Collobert, R. (2012). Deep learning via semi-supervised embedding. In *Neural Networks: Tricks of the Trade*, pages 639–655. Springer.

Zeiler, M. D., Taylor, G. W., and Fergus, R. (2011). Adaptive deconvolutional networks for mid and high level feature learning. In *ICCV 2011*, pages 2018–2025. IEEE.

Zhang, T. and Oles, F. (2000). The value of unlabeled data for classification problems. In *Proc. of ICML 2000*, pages 1191–1198.

Zhao, J., Mathieu, M., Goroshin, R., and Lecun, Y. (2015). Stacked what-where auto-encoders. arXiv:1506.02351.

A Specification of the convolutional models

Table 4: ConvPool-CNN-C by Springenberg *et al.* (2014) and our networks based on it.

Model		
ConvPool-CNN-C	Conv-Large (for CIFAR-10)	Conv-Small (for MNIST)
Input 32×32 or 28×28 RGB or monochrome image		
3×3 conv. 96 ReLU	3×3 conv. 96 BN LeakyReLU	5×5 conv. 32 ReLU
3×3 conv. 96 ReLU	3×3 conv. 96 BN LeakyReLU	
3×3 conv. 96 ReLU	3×3 conv. 96 BN LeakyReLU	
3×3 max-pooling stride 2	2×2 max-pooling stride 2 BN	2×2 max-pooling stride 2 BN
3×3 conv. 192 ReLU	3×3 conv. 192 BN LeakyReLU	3×3 conv. 64 BN ReLU
3×3 conv. 192 ReLU	3×3 conv. 192 BN LeakyReLU	3×3 conv. 64 BN ReLU
3×3 conv. 192 ReLU	3×3 conv. 192 BN LeakyReLU	
3×3 max-pooling stride 2	2×2 max-pooling stride 2 BN	2×2 max-pooling stride 2 BN
3×3 conv. 192 ReLU	3×3 conv. 192 BN LeakyReLU	3×3 conv. 128 BN ReLU
1×1 conv. 192 ReLU	1×1 conv. 192 BN LeakyReLU	
1×1 conv. 10 ReLU	1×1 conv. 10 BN LeakyReLU	1×1 conv. 10 BN ReLU
global meanpool	global meanpool BN	global meanpool BN
		fully connected 10 BN
10-way softmax		

Here we describe two model structures, Conv-Small and Conv-Large, that were used for the MNIST and CIFAR-10 datasets, respectively. They were both inspired by ConvPool-CNN-C by Springenberg *et al.* (2014). Table 4 details the model architectures and differences between the models in this work and ConvPool-CNN-C. It is noteworthy that this architecture does not use any fully connected layers, but replaces them with a global mean pooling layer just before the softmax function. The main differences between our models and ConvPool-CNN-C are the use of Gaussian noise instead of dropout and the convolutional per-channel batch normalization following Ioffe and Szegedy (2015). We also used 2x2 stride 2 max-pooling instead of 3x3 stride 2 max-pooling. LeakyReLU was used to speed up training, as mentioned by Springenberg *et al.* (2014). We utilized batch normalization to all layers, including the pooling layers. Gaussian noise was also added to all layers, instead of applying dropout in only some of the layers as with ConvPool-CNN-C.

B Formulation of the Denoising Function

The denoising function g tries to map the clean $\mathbf{z}^{(l)}$ to the reconstructed $\hat{\mathbf{z}}^{(l)}$, where $\hat{\mathbf{z}}^{(l)} = g(\tilde{\mathbf{z}}^{(l)}, \hat{\mathbf{z}}^{(l+1)})$. The reconstruction is therefore based on the corrupted value and the reconstruction of the layer above.

An optimal functional form of g depends on the conditional distribution $p(\mathbf{z}^{(l)} \mid \mathbf{z}^{(l+1)})$ that we want the model to be able to denoise. For example, if the distribution $p(\mathbf{z}^{(l)} \mid \mathbf{z}^{(l+1)})$ is Gaussian, the optimal function g, that is, the function that achieves the lowest reconstruction error, is going to be linear with respect to $\tilde{\mathbf{z}}^{(l)}$ (Valpola, 2015, Section 4.1). This is the parametrization that we chose on the basis of preliminary comparisons of different denoising function parametrizations.

The proposed parametrization of the denoising function was therefore:

$$g(\tilde{z}, u) = (\tilde{z} - \mu(u))\, \upsilon(u) + \mu(u)\,. \tag{4}$$

We modeled both $\mu(u)$ and $\upsilon(u)$ with an expressive nonlinearity [7]: $\mu(u) = a_1 \texttt{sigmoid}(a_2 u + a_3) + a_4 u + a_5$ and $\upsilon(u) = a_6 \texttt{sigmoid}(a_7 u + a_8) + a_9 u + a_{10}$. We have left out the superscript (l) and subscript i in order not to clutter the equations. Given u, this parametrization is linear with respect to $\tilde{z}$, and both the slope and the bias depended nonlinearly on u.

In order to test whether the elements of the proposed function g were necessary, we systematically removed components from g or replaced g altogether and compared the resulting performance to

---

[7]The parametrization can also be interpreted as a miniature MLP network

the results obtained with the original parametrization. We tuned the hyperparameters of each comparison model separately using a grid search over some of the relevant hyperparameters. However, the standard deviation of additive Gaussian corruption noise was set to 0.3. This means that the comparison does not include the best-performing models reported in Table 1 that achieved the best validation errors after more careful hyperparameter tuning.

As in the proposed function g, all comparison denoising functions mapped neuron-wise the corrupted hidden layer pre-activation $\tilde{\mathbf{z}}^{(l)}$ to the reconstructed hidden layer activation given one projection from the reconstruction of the layer above: $\hat{z}_i^{(l)} = g(\tilde{z}_i^{(l)}, u_i^{(l)})$.

Test error % with # of used labels	100	1000
Proposed g: Gaussian z	**1.06** ($\pm$ 0.07)	**1.03** ($\pm$ 0.06)
Comparison g_1: miniature MLP with $\tilde{z}u$	1.11 ($\pm$ 0.07)	1.11 ($\pm$ 0.06)
Comparison g_2: No augmented term $\tilde{z}u$	2.03 ($\pm$ 0.09)	1.70 ($\pm$ 0.08)
Comparison g_3: Linear g but with $\tilde{z}u$	1.49 ($\pm$ 0.10)	1.30 ($\pm$ 0.08)
Comparison g_4: Only the mean depends on u	2.90 ($\pm$ 1.19)	2.11 ($\pm$ 0.45)

Table 5: Semi-supervised results from the MNIST dataset. The proposed function g is compared to alternative parametrizations. Note that the hyperparameter search was not as exhaustive as in the final results, which means that the results of the proposed model deviate slightly from the final results presented in Table 1.

The comparison functions $g_{1...4}$ are parametrized as follows:

Comparison g_1: Miniature MLP with $\tilde{z}u$

$$\hat{z} = g(\tilde{z}, u) = \mathbf{a}\boldsymbol{\xi} + b\,\texttt{sigmoid}(\mathbf{c}\boldsymbol{\xi}) \tag{5}$$

where $\boldsymbol{\xi} = [1, \tilde{z}, u, \tilde{z}u]^T$ is an augmented input, $\mathbf{a}$ and $\mathbf{c}$ are trainable weight vectors, b is a trainable scalar weight. This parametrization is capable of learning denoising of several different distributions including sub- and super-Gaussian and bimodal distributions.

Comparison g_2: No augmented term

$$g_2(\tilde{z}, u) = \mathbf{a}\boldsymbol{\xi}' + b\,\texttt{sigmoid}(\mathbf{c}\boldsymbol{\xi}') \tag{6}$$

where $\boldsymbol{\xi}' = [1, \tilde{z}, u]^T$. g_2 therefore differs from g_1 in that the input lacks the augmented term $\tilde{z}u$.

Comparison g_3: Linear g

$$g_3(\tilde{z}, u) = \mathbf{a}\boldsymbol{\xi}. \tag{7}$$

g_3 differs from g in that it is linear and does not have a sigmoid term. As this formulation is linear, it only supports Gaussian distributions. Although the parametrization has the augmented term that lets u modulate the slope and shift of the distribution, the scope of possible denoising functions is still fairly limited.

Comparison g_4: u affects only the mean of $p(z \mid u)$

$$g_4(\tilde{z}, u) = a_1 u + a_2 \texttt{sigmoid}(a_3 u + a_4) + a_5 \tilde{z} + a_6 \texttt{sigmoid}(a_7 \tilde{z} + a_8) + a_9 \tag{8}$$

g_4 differs from g_1 in that the inputs from u are not allowed to modulate the terms that depend on $\tilde{z}$, but that the effect is additive. This means that the parametrization only supports optimal denoising functions for a conditional distribution $p(z \mid u)$ where u only shifts the mean of the distribution of z but otherwise leaves the shape of the distribution intact.

Results All models were tested in a similar setting as the semi-supervised fully connected MNIST task using $N = 1000$ labeled samples. We also reran the best comparison model on $N = 100$ labels. The results of the analyses are presented in Table 5.

As can be seen from the table, the alternative parametrizations of g are inferior to the proposed parametrization, at least in the model structure we use.

These results support the finding by Rasmus *et al.* (2015a) that modulation of the lateral connection from $\tilde{z}$ to $\hat{z}$ by u is critical for encouraging the development of invariant representations at the higher layers of the model. Comparison function g_4 lacked this modulation and it clearly performed worse than any other denoising function listed in Table 5. Even the linear g_3 performed very well as long it had the term $\tilde{z}u$. Leaving the nonlinearity but removing $\tilde{z}u$ in g_2 hurt the performance much more.

In addition to the alternative parametrizations for the g-function, we ran experiments using a more standard autoencoder structure. In that structure, we attached an additional decoder to the standard MLP by using one hidden layer as the input to the decoder and the reconstruction of the clean input as the target. The structure of the decoder was set to be the same as the encoder: that is, the number and size of the layers from the input to the hidden layer where the decoder was attached were the same as the number and size of the layers in the decoder. The final activation function in the decoder was set to be the sigmoid nonlinearity. During training, the target was the weighted sum of the reconstruction cost and the classification cost.

We tested the autoencoder structure with 100 and 1000 labeled samples. We ran experiments for all possible decoder lengths: that is, we tried attaching the decoder to all the hidden layers. However, we did not manage to get a significantly better performance than the standard supervised model without any decoder in any of the experiments.