Deconstructing the Ladder Network Architecture

Mohammad Pezeshki
Departement d'informatique et de recherche opérationnelle
Université de Montréal
Montréal, QC H3C 3J7
mohammad.pezeshki@umontreal.ca

Linxi Fan
Department of Computer Science
Columbia University
New York, New York 10027
linxi.fan@columbia.edu

Philémon Brakel, Aaron Courville, Yoshua Bengio[*]
Departement d'informatique et de recherche opérationnelle
Université de Montréal
Montréal, QC H3C 3J7
pbpop3@gmail.com, {courvila, <findme>}@iro.umontreal.ca

Abstract

The Ladder Network is a recent new approach to semi-supervised learning that turned out to be very successful. While showing impressive performance, the Ladder Network has many components intertwined, whose contributions are not obvious in such a complex architecture. This paper presents an extensive experimental investigation of variants of the Ladder Network in which we replaced or removed individual components to learn about their relative importance. For semi-supervised tasks, we conclude that the most important contribution is made by the lateral connections, followed by the application of noise, and the choice of what we refer to as the 'combinator function'. As the number of labeled training examples increases, the lateral connections and the reconstruction criterion become less important, with most of the generalization improvement coming from the injection of noise in each layer. Finally, we introduce a combinator function that reduces test error rates on Permutation-Invariant MNIST to 0.57% for the supervised setting, and to 0.97% and 1.0% for semi-supervised settings with 1000 and 100 labeled examples, respectively.

1 Introduction

Labeling data sets is typically a costly task and in many settings there are far more unlabeled examples than labeled ones. Semi-supervised learning aims to improve the performance on some supervised learning problem by using information obtained from both labeled and unlabeled examples. Since the recent success of deep learning methods has mainly relied on supervised learning based on very large labeled datasets, it is interesting to explore semi-supervised deep learning approaches to extend the reach of deep learning to these settings.

Since unsupervised methods for pre-training layers or neural networks were an essential part of the first wave of deep learning methods (Hinton et al., 2006; Vincent et al., 2008; Bengio, 2009), a natural next step is to investigate how ideas inspired by Restricted Boltzmann Machine training and regularized autoencoders can be used for semi-supervised learning. Examples of approaches based on such ideas are the discriminative RBM (Larochelle & Bengio, 2008) and a deep architecture based on semi-supervised autoencoders that was used for document classification (Ranzato & Szummer, 2008). More recent examples of approaches for semi-supervised deep learning are the semi-supervised Variational Autoencoder (Kingma et al., 2014) and the Ladder Network (Rasmus et al., 2015) which obtained very impressive, state of the art results (1.13% error) on the MNIST handwritten digits classification benchmark using just 100 labeled training examples.

---

[*]Yoshua Bengio is a CIFAR Senior Fellow

The Ladder Network adds an unsupervised component to the supervised learning objective of a deep feedforward network by treating this network as part of a deep stack of denoising autoencoders or DAEs (Vincent et al., 2010) that learns to reconstruct each layer (including the input) based on a corrupted version of it, using feedback from upper levels. The term 'ladder' refers to how this architecture extends the stacked DAE in the way the feedback paths are formed.

This paper is focusing on the design choices that lead to the Ladder Network's superior performance and tries to disentangle them empirically. We identify some general properties of the model that make it different from standard feedforward networks and compare various architectures to identify those properties and design choices that are the most essential for obtaining good performance with the Ladder Network. While the original authors of the Ladder Network paper explored some variants of their model already, we provide a thorough comparison of a large number of architectures controlling for both hyperparameter settings and data set selection. Finally, we also introduce a variant of the Ladder Network that yields new state-of-the-art results for the Permutation-Invariant MNIST classification task in both semi- and fully- supervised settings.

2 THE LADDER NETWORK ARCHITECTURE

In this section, we describe the Ladder Network Architecture[1]. Consider a dataset with N labeled examples $(x(1), y^*(1)), (x(2), y^*(2)), ..., (x(N), y^*(N))$ and M unlabeled examples $x(N+1), x(N+2), ..., x(N+M)$ where $M \gg N$. The objective is to learn a function that models $P(y|x)$ by using both the labeled examples and the large quantity of unlabeled examples. In the case of the Ladder Network, this function is a deep Denoising Auto Encoder (DAE) in which noise is injected into all hidden layers and the objective function is a weighted sum of the supervised Cross Entropy cost on the top of the encoder and the unsupervised denoising Square Error costs at each layer of the decoder. Since all layers are corrupted by noise, another encoder path with shared parameters is responsible for providing the clean reconstruction targets, i.e. the noiseless hidden activations (See Figure 1).

Through lateral skip connections, each layer of the noisy encoder is connected to its corresponding layer in the decoder. This enables the higher layer features to focus on more abstract and task-specific features. Hence, at each layer of the decoder, two signals, one from the layer above and the other from the corresponding layer in the encoder are combined.

Formally, the Ladder Network is defined as follows:

$$\tilde{x}, \tilde{z}^{(1)}, ..., \tilde{z}^{(L)}, \tilde{y} = \text{Encoder}_{noisy}(x), \tag{1}$$

$$x, z^{(1)}, ..., z^{(L)}, y = \text{Encoder}_{clean}(x), \tag{2}$$

$$\hat{x}, \hat{z}^{(1)}, ..., \hat{z}^{(L)} = \text{Decoder}(\tilde{z}^{(1)}, ..., \tilde{z}^{(L)}), \tag{3}$$

where Encoder and Decoder can be replaced by any multi-layer architecture such as a multi-layer perceptron in this case. The variables x, y, and $\tilde{y}$ are the input, the noiseless output, and the noisy output respectively. The variables $z^{(l)}$, $\tilde{z}^{(l)}$, and $\hat{z}^{(l)}$ are the hidden representation, its noisy version, and its reconstructed version at layer l. The objective function is a weighted sum of supervised (Cross Entropy) and unsupervised costs (Reconstruction costs).

$$Cost = -\Sigma_{n=1}^{N} \log P(\tilde{y}(n) = y^*(n)|x(n)) + \Sigma_{n=N+1}^{M} \Sigma_{l=1}^{L} \lambda_l \text{ ReconstructionCost}(z^{(l)}(n), \hat{z}^{(l)}(n)). \tag{4}$$

Note that while the noisy output $\tilde{y}$ is used in the Cross Entropy term, the classification task is performed by the noiseless output y at test time.

In the forward path, individual layers of the encoder are formalized as a linear transformation followed by Batch Normalization (Ioffe & Szegedy, 2015) and then application of a nonlinear activation function:

---

[1]Please refer to (Rasmus et al., 2015; Valpola, 2014) for more detailed explanation of the Ladder Network architecture.

$$\tilde{z}_{pre}^{(l)} = W^{(l)} \cdot \tilde{h}^{(l-1)}, \tag{5}$$

$$\tilde{\mu}^{(l)} = \text{mean}(\tilde{z}_{pre}^{(l)}), \tag{6}$$

$$\tilde{\sigma}^{(l)} = \text{stdv}(\tilde{z}_{pre}^{(l)}), \tag{7}$$

$$\tilde{z}^{(l)} = \frac{\tilde{z}_{pre}^{(l)} - \tilde{\mu}^{(l)}}{\tilde{\sigma}^{(l)}} + \mathcal{N}(0, \sigma^2), \tag{8}$$

$$\tilde{h}^{(l)} = \phi(\gamma^{(l)}(\tilde{z}^{(l)} + \beta^{(l)})) \tag{9}$$

where $\tilde{h}^{(l-1)}$ is the post-activation at layer $l - 1$ and $W^{(l)}$ is the weight matrix from layer $l - 1$ to layer l. Batch Normalization is applied to the pre-normalization $\tilde{z}_{pre}^{(l)}$ using the mini-batch mean $\mu^{(l)}$ and standard deviation $\sigma^{(l)}$. The next step is to add Gaussian noise with mean 0 and variance σ^2 to compute pre-activation $\tilde{z}^{(l)}$. The parameters $\beta^{(l)}$ and $\gamma^{(l)}$ are responsible for shifting and scaling before applying the nonlinearity $\phi(\cdot)$. Note that the above equations describe the *noisy* encoder. If we remove noise ($\mathcal{N}(0, \sigma^2)$) and replace $\tilde{h}$ and $\tilde{z}$ with h and z respectively, we will obtain the noiseless version of the encoder.

At each layer of the decoder in the backward path, the signal from the layer $\hat{z}^{(l+1)}$ and the noisy signal $\tilde{z}^{(l)}$ are combined into the reconstruction $\hat{z}^{(l)}$ by the following equations:

$$u_{pre}^{(l+1)} = V^{(l)} \cdot \hat{z}^{(l+1)}, \tag{10}$$

$$\mu^{(l+1)} = \text{mean}(u_{pre}^{(l+1)}), \tag{11}$$

$$\sigma^{(l+1)} = \text{stdv}(u_{pre}^{(l+1)}), \tag{12}$$

$$u^{(l+1)} = \frac{u_{pre}^{(l+1)} - \mu^{(l+1)}}{\sigma^{(l+1)}}, \tag{13}$$

$$\hat{z}^{(l)} = g(\tilde{z}^{(l)}, u^{(l+1)}) \tag{14}$$

where $V^{(l)}$ is a weight matrix from layer $l + 1$ to layer l. We call the function $g(\cdot, \cdot)$ the *combinator function* as it combines the vertical $u^{(l+1)}$ and the lateral $\tilde{z}^{(l)}$ connections in an element-wise fashion. The original Ladder Network proposes the following design for $g(\cdot, \cdot)$, which we call the *vanilla combinator*:

$$
\begin{aligned}
g(\tilde{z}^{(l)}, u^{(l+1)}) = {} & b_0 + w_{0z} \odot \tilde{z}^{(l)} + w_{0u} \odot u^{(l+1)} + w_{0zu} \odot \tilde{z}^{(l)} \odot u^{(l+1)} \\
& + w_\sigma \odot \text{Sigmoid}(b_1 + w_{1z} \odot \tilde{z}^{(l)} + w_{1u} \odot u^{(l+1)} + w_{1zu} \odot \tilde{z}^{(l)} \odot u^{(l+1)}),
\end{aligned} \tag{15}
$$

where $\odot$ is an element-wise multiplication operator and each per-element weight is initialized as:

$$
\begin{cases}
w_{\{0,1\}z} & \leftarrow 1 \\
w_{\{0,1\}u} & \leftarrow 0 \\
w_{\{0,1\}zu}, b_{\{0,1\}} & \leftarrow 0 \\
w_\sigma & \leftarrow 1
\end{cases} \tag{16}
$$

In later sections, we will explore alternative initialization schemes on the vanilla combinator.

Finally, the ReconstructionCost($z^{(l)}, \hat{z}^{(l)}$) in equation (4) is defined as the following:

$$\text{ReconstructionCost}(z^{(l)}, \hat{z}^{(l)}) = ||\frac{\hat{z}^{(l)} - \mu^{(l)}}{\sigma^{(l)}} - z^{(l)}||^2. \tag{17}$$

where $\hat{z}^{(l)}$ is normalized using $\mu^{(l)}$ and $\sigma^{(l)}$ which are the *encoder*'s sample mean and standard deviation statistics of the current mini batch, respectively. The reason for this second normalization is to cancel the effect of unwanted noise introduced by the limited batch size of Batch Normalization.
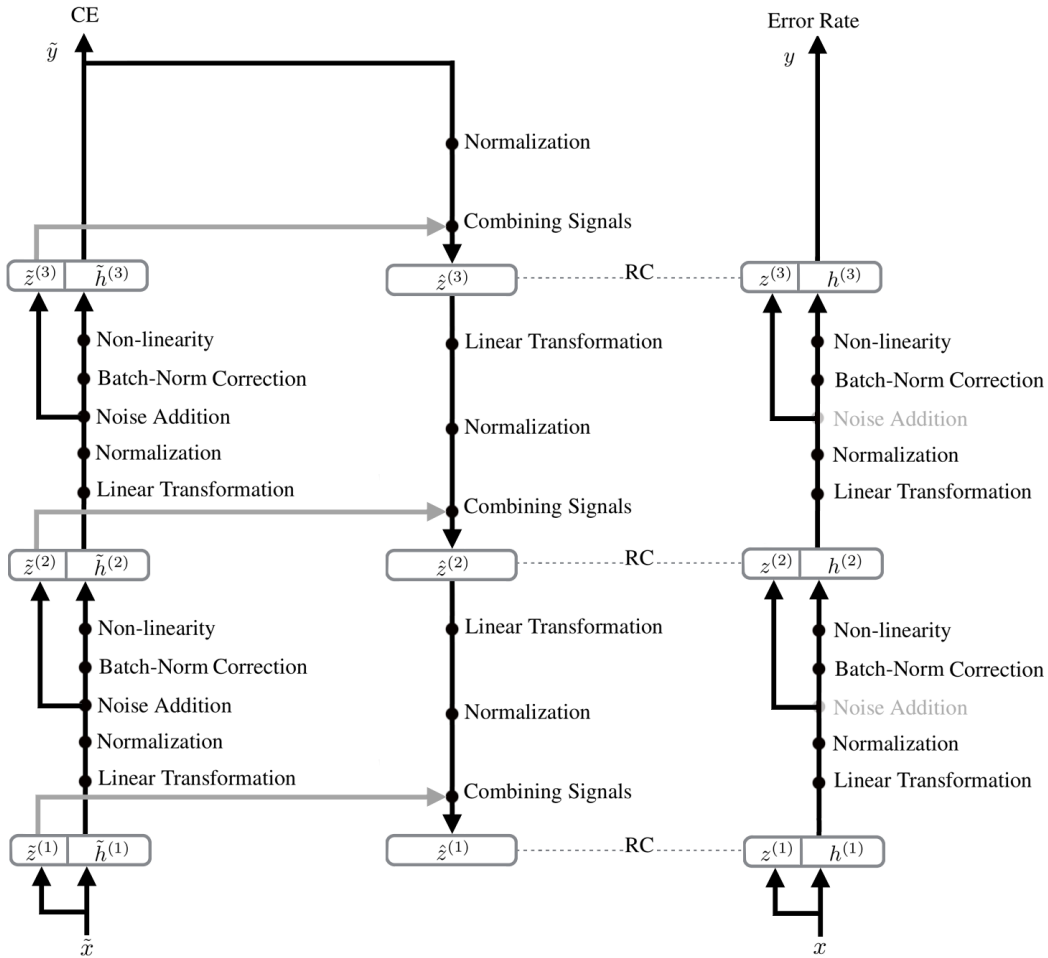
Figure 1: The Ladder Network consists of two encoders (on each side of the figure) and one decoder (in the middle). At each layer of both encoders (equations 5 to 9), $z^{(l)}$ and $\tilde{z}^{(l)}$ are computed by applying a linear transformation and normalization on $h^{(l-1)}$ and $\tilde{h}^{(l-1)}$, respectively. The noisy version of the encoder (left) has an extra Gaussian noise injection term. Batch normalization correction (γ^l, β^l) and non-linearity are then applied to obtain $h^{(l)}$ and $\tilde{h}^{(l)}$. At each layer of the decoder, two streams of information, the lateral connection $\tilde{z}^{(l)}$ (gray lines) and the vertical connection $u^{(l+1)}$, are required to reconstruct $\hat{z}^{(l)}$ (equations 10 to 14). Acronyms CE and RC stand for Cross Entropy and Reconstruction Cost respectively. The final objective function is a weighted sum of all Reconstruction costs and the Cross Entropy cost.

3 COMPONENTS OF THE LADDER NETWORK

Now that the precise architecture of the Ladder Network has been described in details, we can identify a couple of important additions to the standard feed-forward neural network architecture that may have a pronounced impact on the performance. A distinction can also be made between those design choices that follow naturally from the motivation of the ladder network as a deep autoencoder and those that are more ad-hoc and task specific.

The most obvious addition is the extra reconstruction cost for every hidden layer and the input layer. While it is clear that the reconstruction cost provides an unsupervised objective to harness the unlabeled examples, it is not clear how important the penalization is for each layer and what role it plays for fully-supervised tasks.

A second important change is the addition of Gaussian noise to the input and the hidden representations. While adding noise to the first layer is a part of denoising autoencoder training, it is again not clear whether it is necessary to add this noise at every layer or not. We would also like to know if the noise helps by making the reconstruction task nontrivial and useful or just by regularizing the feed-forward network in a similar way that noise-based regularizers like dropout (Srivastava et al., 2014) and adaptive weight noise (Graves, 2011) do.

Finally, the lateral skip connections are the most notable deviation from the standard denoising autoencoder architecture. The way the vanilla Ladder Network combines the lateral stream of information $\tilde{z}^{(l)}$ and the downward stream of information $u^{(l+1)}$ is somewhat unorthodox. For this reason, we have conducted extensive experiments on both the importance of the lateral connections and the precise choice for the function that combines the lateral and downward information streams (which we refer to as the *combinator function*).

4 EXPERIMENTAL SETUP

In this section, we introduce different variants of the Ladder Architecture and describe our experiment methodology. Each variant differs from either the vanilla model or a previous variant in only one component. This enables us to isolate each component and observe its effects while other components remain unchanged. Table 1 depicts the different variants we have investigated and their abbreviations.

Table 1: Schematic ordering of the different architectures and variants we investigated.

	Description	
Vanilla	The original Ladder Network.	
FirstNoise	Noise only to the 1st layer.	
FirstRecons	Reconstruction only at the 1st layer.	
FirstN&R	Noise only to the 1st layer and also reconstruction only at the 1st layer.	
NoLateral	FirstN&R with no lateral connection (resembles a DAE with added supervised cost).	
RandInit	Random initialization of the vanilla combinator weights.	
RevInit	Reverse initialization ($w_z \leftarrow 0$ and $w_u \leftarrow 1$)	
NoSig	No non-linearity in the vanilla combinator.	
NoMult	No multiplication in the vanilla combinator.	
Linear	Simple linear combination of $\tilde{z}^l$ and $u^{(l+1)}$ (No sigmoid and no multiplication).	
Gaussian	Equations 19 to 21.	
GatedGauss	Gaussian combinator with gating.	
MLP	MLP combinator function.	
AMLP	Augmented MLP with multiplicative unit.	

Tree structure (left side):
- Vanilla Ladder
  - Noise
    - FirstNoise
    - FirstRecons
    - FirstN&R
    - NoLateral
  - Combinator
    - RandInit
    - RevInit
    - NoSig
    - NoMult
    - Linear
    - Gaussian
    - GatedGauss
    - MLP
    - AMLP

4.1 NOISE VARIANTS

Different configurations of noise injection, penalizing reconstruction errors, and the lateral connection removal suggest four different variants:

- Add noise only to the first layer (FIRSTNOISE).

- Only penalize the reconstruction at the first layer (FIRSTRECONS), i.e. $\lambda^{(l \geq 1)}$ are set to 0.

- Apply both of the above changes: add noise and penalize the reconstruction only at the first layer (FIRSTN&R).

- Remove all lateral connections from FIRSTN&R. Therefore, equivalent to to a denoising autoencoder with an additional supervised cost at the top, the encoder and the decoder are connected only through the topmost connection. We call this variant NOLATERAL.

4.2 VANILLA COMBINATOR VARIANTS

We try different variants of the vanilla combinator function that combines the two streams of information from the lateral and the vertical connections in an unusual way. As defined in equation 15, the output of the vanilla combinator depends on u, $\tilde{z}$, and $u \odot \tilde{z}$ [2], which are connected to the output via two paths, one linear and the other through a sigmoid non-linearity unit. (See Figure 2(a))

Note that the vanilla combinator is initialized in a very specific way (equation 16), which sets the initial weights for lateral connection $\tilde{z}$ to 1 and vertical connection u to 0. This particular scheme encourages the Ladder decoder path to learn more from the lateral information stream $\tilde{z}$ than the vertical u at the beginning of training.

We explore two variants of the initialization scheme:

- Random initialization (RANDINIT): all per-element weights w_* and biases b_* are randomly initialized to $\mathcal{N}(0, 0.2)$.

- Reverse initialization (REVINIT) $\begin{cases} w_{\{0,1\}z} & \leftarrow 0 \\ w_{\{0,1\}u} & \leftarrow 1 \\ w_{\{0,1\}zu}, b_{\{0,1\}} & \leftarrow 0 \\ w_\sigma & \leftarrow 1 \end{cases}$

The simplest way of combining lateral connections with vertical connections is to simply add them in an element-wise fashion, similar to the nature of skip-connections in a recently published work on Residual Learning (He et al., 2015). We call this variant LINEAR combinator function. We also derive two more variants NOSIG and NOMULT in the way of stripping down the vanilla combinator function to the simple LINEAR one:

- Remove sigmoid non-linearity (NOSIG). The corresponding per-element weights are initialized in the same way as the vanilla combinator.

- Remove the multiplicative term $\tilde{z} \odot u$ (NOMULT).

- Simple linear combination (LINEAR)

$$g(\tilde{z}, u) = b + w_u \odot u + w_z \odot \tilde{z} \tag{18}$$

where the initialization scheme resembles the vanilla one: $\begin{cases} w_z & \leftarrow 1 \\ w_u, b & \leftarrow 0 \end{cases}$

4.3 GAUSSIAN COMBINATOR VARIANTS

Another choice for the combinator function with a probabilistic interpretation is the GAUSSIAN combinator proposed in the original paper about the Ladder Architecture (Rasmus et al., 2015). Based on the theory in the Section 4.1 of (Valpola, 2014), assuming that both additive noise and the conditional distribution $P(z^{(l)}|u^{(l+1)})$ are Gaussian distributions, the denoising function is linear with respect to $\tilde{z}^{(l)}$. Hence, the denoising function could be a weighted sum over $\tilde{z}^{(l)}$ and a prior on $z^{(l)}$. The weights and the prior are modeled as a function of the vertical signal:

---

[2] For simplicity, subscript i and superscript l are implicit from now on.

$$g(\tilde{z}, u) = \nu(u) \odot \tilde{z} + (1 - \nu(u)) \odot \mu(u), \tag{19}$$

$$\mu(u) = w_1 \odot \text{Sigmoid}(w_2 \odot u + w_3) + w_4 \odot u + w_5, \tag{20}$$

$$\nu(u) = w_6 \odot \text{Sigmoid}(w_7 \odot u + w_8) + w_9 \odot u + w_{10}. \tag{21}$$

Strictly speaking, $\nu(u)$ is not a proper weight, because it is not guaranteed to be positive all the time.

To make the Gaussian interpretation rigorous, we explore a variant that we call GATEDGAUSS, where equations 19 and 20 stay the same but 21 is replaced by:

$$\nu(u) = \text{Sigmoid}(w_6 \odot u + w_7). \tag{22}$$

GATEDGAUSS guarantees that $0 < \nu(u) < 1$. We expect that $\nu(u)_i$ will be close to 1 if the information from the lateral connection for unit i is more helpful to reconstruction, and close to 0 if the vertical connection becomes more useful. The GATEDGAUSS combinator is similar in nature to the gating mechanisms in other models such as Gated Recurrent Unit (Cho et al., 2014) and highway networks (Srivastava et al., 2015).

4.4 MLP (MULTILAYER PERCEPTRON) COMBINATOR VARIANTS

We also propose another type of element-wise combinator functions based on fully-connected MLPs. We have explored two classes in this family. The first one, denoted simply as MLP, maps two scalars $[u, \tilde{z}]$ to a single output $g(\tilde{z}, u)$ (figure 2(b)). We empirically determine the choice of activation function for the hidden layers. Preliminary experiments show that the Leaky Rectifier Linear Unit (LReLU) (Maas et al., 2013) performs better than either the conventional ReLU or the sigmoid unit. Our LReLU function is formulated as

$$\text{LReLU}(x) = \begin{cases} x, & \text{if } x \geq 0, \\ 0.1\,x, & \text{otherwise} \end{cases}. \tag{23}$$

We experiment with different numbers of layers and hidden units per layer in the MLP. We present results for three specific configurations: $[4]$ for a single hidden layer of 4 units, $[2, 2]$ for 2 hidden layers each with 2 units, and $[2, 2, 2]$ for 3 hidden layers. For example, in the $[2, 2, 2]$ configuration, the MLP combinator function is defined as:

$$g(\tilde{z}, u) = W_3 \cdot \text{LReLU}\Big(W_2 \cdot \text{LReLU}(W_1 \cdot [u, \tilde{z}] + b_1) + b_2\Big) + b_3 \tag{24}$$

where W_1, W_2, and W_3 are 2×2 weight matrices; b_1, b_2, and b_3 are 2×1 bias vectors.

The second class, which we denote as AMLP (Augmented MLP), has a multiplicative term as an augmented input unit (figure 2(c)). We expect that this multiplication term allows the vertical signal $(u^{(l+1)})$ to override the lateral signal $(\tilde{z})$, and also allows the lateral signal to select where the vertical signal is to be instantiated. Since the approximation of multiplication is not easy for a single-layer MLP, we explicitly add the multiplication term as an extra input to the combinator function. AMLP maps three scalars $[u, \tilde{z}, u \odot \tilde{z}]$ to a single output. We use the same LReLU unit for AMLP. We do similar experiments as in the MLP case and include results for $[4]$, $[2, 2]$ and $[2, 2, 2]$ hidden layer configurations.

Both MLP and AMLP weight parameters are randomly initialized to $\mathcal{N}(0, \eta)$. η is considered to be a hyperparameter and tuned on the validation set. Precise values for the best η values are listed in Appendix A.

4.5 METHODOLOGY

The experimental setup includes two semi-supervised classification tasks with 100 and 1000 labeled examples and a fully-supervised classification task with 60000 labeled examples for Permutation-Invariant MNIST handwritten digit classification. Labeled examples are chosen randomly but the
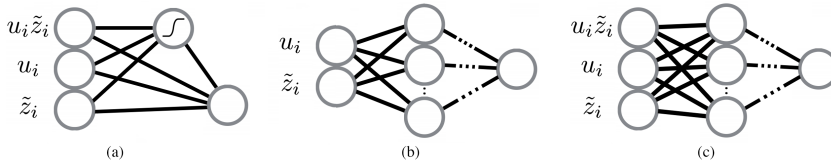
Figure 2: **(a)** Vanilla combinator **(b)** MLP combinator **(c)** Augmented MLP combinator

number of examples for different classes is balanced. The test set is not used during all the hyperparameter search and tuning. Each experiment is repeated 10 times with 10 different but fixed random seeds to measure the standard error of the results for different parameter initializations and different selections of labeled examples.

All variants and the vanilla Ladder Network itself are trained using the ADAM optimization algorithm (Kingma & Ba, 2014) with a learning rate of 0.002 for 100 iterations followed by 50 iterations with a learning rate decaying linearly to 0. Hyperparameters including the standard deviation of the noise injection and the denoising weights at each layer are tuned separately for each variant and each experiment setting (100-, 1000-, and fully-labeled). Hyperparmeters are optimized by either a random search (Bergstra & Bengio, 2012), or a grid search, depending on the number of hyperparameters involved (see Appendix A for precise configurations).

5 RESULTS & DISCUSSION

Table 2 collects all results for the variants and the baselines. The results are organized into two main categories for all of the three tasks. Boxplots of four interesting variants are also shown in Figure 3. The BASELINE model is a simple feed-forward neural network with no reconstruction penalty and BASELINE+NOISE is the same network but with additive noise at each layer. The best results in terms of average error rate on the test set are achieved by the proposed AMLP combinator function: in the fully-supervised setting, the best average error rate is 0.569 ± 0.010, while in the semi-supervised settings with 100 and 1000 labeled examples, the averages are 1.002 ± 0.037 and 0.974 ± 0.021 respectively.

5.1 NOISE VARIANTS

The results in the first part of the table indicate that in the fully-supervised setting, adding noise either to the first layer only or to all layers leads to a lower error rate with respect to the baselines. Our intuition is that the effect of additive noise to layers is very similar to the weight noise regularization method (Graves, 2011) and dropout (Hinton et al., 2012).

In addition, the error rates in the first part of the table tell us that removing the lateral connections hurts much more than the absence of noise injection or reconstruction penalty in the intermediate layers. It is also worth mentioning that hyperparameter tuning yields zero weights for penalizing the reconstruction errors in all layers except the input layer in the fully-supervised task for the vanilla model. Something similar happens for NOLATERAL as well, where hyperparameter tuning yields zero reconstruction weights for all layers including the input layer. In other words, NOLATERAL and BASELINE+NOISE become the same models for the fully-supervised task. Moreover, the weights for the reconstruction penalty of the hidden layers are relatively small in the semi-supervised task. This is in line with similar observations (relatively small weights for the unsupervised part of the objective) for the hybrid discriminant RBM (Larochelle & Bengio, 2008).

5.2 COMBINATOR FUNCTION VARIANTS

The second part of Table 2 shows the relative performance of different combinator functions. Unsurprisingly, the performance deteriorates considerably if we remove the sigmoid non-linearity (NOSIG) or the multiplicative term (NOMULT) or both (LINEAR) from the vanilla combinator. Judging from the size of the increase in average error rates, the multiplicative term is more important than the sigmoid unit.

Table 2: PI MNIST classification results for the vanilla Ladder Network and its variants trained on 100, 1000, and 60000 (full) labeled examples. AER and SE stand for Average Error Rate and its Standard Error of each variant over 10 different runs. BASELINE is a multi-layer feed-forward neural network with no reconstruction penalty.

	100		1000		60000	
Variant	**AER (%)**	**SE**	**AER (%)**	**SE**	**AER (%)**	**SE**
Baseline	25.804	± 0.40	8.734	± 0.058	1.182	± 0.010
Baseline+noise	23.034	± 0.48	6.113	± 0.105	0.820	± 0.009
Vanilla	1.086	± 0.023	1.017	± 0.017	0.608	± 0.013
FirstNoise	1.856	± 0.193	1.381	± 0.029	0.732	± 0.015
FirstRecons	1.691	± 0.175	1.053	± 0.021	0.608	± 0.013
FirstN&R	1.856	± 0.193	1.058	± 0.175	0.732	± 0.016
NoLateral	16.390	± 0.583	5.251	± 0.099	0.820	± 0.009
RandInit	1.232	± 0.033	1.011	± 0.025	0.614	± 0.015
RevInit	1.305	± 0.129	1.031	± 0.017	0.631	± 0.018
NoSig	1.608	± 0.124	1.223	± 0.014	0.633	± 0.010
NoMult	3.041	± 0.914	1.735	± 0.030	0.674	± 0.018
Linear	5.027	± 0.923	2.769	± 0.024	0.849	± 0.014
Gaussian	1.064	± 0.021	0.983	± 0.019	0.604	± 0.010
GatedGauss	1.308	± 0.038	1.094	± 0.016	0.632	± 0.011
MLP $[4]$	1.374	± 0.186	0.996	± 0.028	0.605	± 0.012
MLP $[2, 2]$	1.209	± 0.116	1.059	± 0.023	0.573	± 0.016
MLP $[2, 2, 2]$	1.274	± 0.067	1.095	± 0.053	0.602	± 0.010
AMLP $[4]$	1.072	± 0.015	**0.974**	± 0.021	0.598	± 0.014
AMLP $[2, 2]$	1.193	± 0.039	1.029	± 0.023	**0.569**	± 0.010
AMLP $[2, 2, 2]$	**1.002**	± 0.038	0.979	± 0.025	0.578	± 0.013

As described in Section 4.2 and Equation 16, the per-element weights of the lateral connections are initialized to ones while those of the vertical are initialized to zeros. Interestingly, the results are slightly worse for the RANDINIT variant, in which these weights are initialized randomly. The REVINIT variant is even worse than the random initialization scheme. We suspect that the reason is that the optimization algorithm finds it easier to reconstruct a representation z starting from its noisy version $\tilde{z}$, rather than starting from an initially arbitrary reconstruction from the untrained upper layers. Another justification is that the initialization scheme in Equation 16 corresponds to optimizing the Ladder Network as if it behaves like a stack of decoupled DAEs initially, therefore during early training it is like that the Auto-Encoders are trained more independently.

The GAUSSIAN combinator performs better than the vanilla combinator. GATEDGAUSS, the other variant with strict $0 < \sigma(u) < 1$, does not perform as well as the one with unconstrained $\sigma(u)$. In the GAUSSIAN formulation, $\tilde{z}$ is regulated by two functions of u: $\mu(u)$ and $\sigma(u)$. This combinator interpolates between the noisy activations and the predicted reconstruction (Equation **??**), and the scaling parameter can be interpreted as a measure of the certainty of the network.

Finally, the AMLP model yields state-of-the-art results in all of 100-, 1000- and 60000-labeled experiments for PI MNIST. It outperforms both the MLP and the vanilla model. The additional multiplicative input unit $u \odot \tilde{z}$ helps the learning process significantly.

5.3 PROBABILISTIC INTERPRETATIONS OF THE LADDER NETWORK

Since many of the motivations behind regularized autoencoder architectures are based on observations about generative models, we briefly discuss how the Ladder Network can be related to some other models with varying degrees of probabilistic interpretability. Considering that the components that are most defining of the Ladder Network seem to be the most important ones for semi-supervised learning in particular, comparisons with generative models are at least intuitively appealing to get more insight about how the model learns about unlabeled examples.
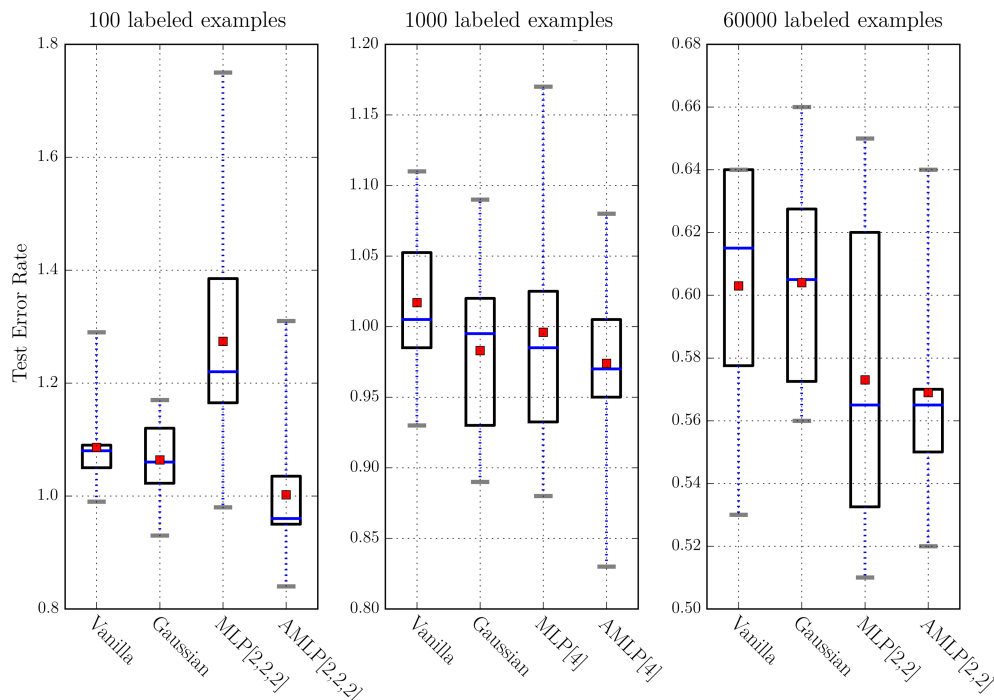
Figure 3: Boxplots summarizing all individual experiments of four variants for 10 different seeds. Box boundaries represent the 25th and 75th percentiles. The blue line and the red square represent the median and the mean, respectively. The gray caps also show the minimum and maximum values. The variants that perform much worse than the vanilla Ladder Network are not plotted.

By training the individual denoising autoencoders that make up the Ladder Network with a single objective function, this coupling goes as far as encouraging the lower levels to produce representations that are going to be easy to reconstruct by the upper levels. We find a similar term (-log of the top-level prior evaluated at the output of the encoder) in hierarchical extensions of the variational autoencoder (Rezende et al., 2014; Bengio, 2014). While the Ladder Network differs too much from an actual variational autoencoder to be treated as such, the similarities can still give one intuitions about the role of the noise and the interactions between the layers. Conversely, one also may wonder how a variational autoencoder might benefit from some of the components of Ladder Networks like Batch Normalization and multiplicative connections.

When one simply views the Ladder Network as a peculiar type of denoising autoencoder, one could extend the recent work on the generative interpretation of denoising autoencoders (Alain & Bengio, 2013; Bengio et al., 2013) to interpret the Ladder Network as a generative model as well. It would be interesting to see if the Ladder Network architecture can be used to generate samples and if the architecture's success at semi-supervised learning translates to this profoundly different use of the model.

6 CONCLUSION

The paper systematically compares different variants of the recent Ladder Network architecture (Rasmus et al., 2015; Valpola, 2014) with two feedforward neural networks as the baselines and the standard architecture (proposed in the original paper). Comparisons are done in a deconstructive way, starting from the standard architecture. Based on the comparisons of different variants we conclude that:

- Unsurprisingly, the reconstruction cost is crucial to obtain the desired regularization from unlabeled data.

- Applying additive noise to each layer and especially the first layer has a regularization effect which helps generalization. This seems to be one of the most important contributors to the performance on the fully supervised task.

- The lateral connection is a vital component in the Ladder architecture to the extent that removing it considerably deteriorates the performance for all of the semi-supervised tasks.

- The precise choice of the combinator function has a less dramatic impact, although the vanilla combinator can be replaced by the Augmented MLP to yield better performance, in fact allowing us to improve the record error rates on Permutation-Invariant MNIST for semi- and fully-supervised settings.

We hope that these comparisons between different architectural choices will help to improve understanding of semi-supervised learning's success for the Ladder Network and like architectures, and perhaps even deep architectures in general.

REFERENCES

Guillaume Alain and Yoshua Bengio. What regularized auto-encoders learn from the data generating distribution. In *ICLR'2013*. also arXiv report 1211.4246, 2013.

Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian Goodfellow, Arnaud Bergeron, Nicolas Bouchard, David Warde-Farley, and Yoshua Bengio. Theano: new features and speed improvements. *arXiv preprint arXiv:1211.5590*, 2012.

Yoshua Bengio. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

Yoshua Bengio. How auto-encoders could provide credit assignment in deep networks via target propagation. Technical report, arXiv:1407.7906, 2014.

Yoshua Bengio, Li Yao, Guillaume Alain, and Pascal Vincent. Generalized denoising auto-encoders as generative models. In *NIPS'2013*, 2013.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *The Journal of Machine Learning Research*, 13(1):281–305, 2012.

James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: a cpu and gpu math expression compiler. In *Proceedings of the Python for scientific computing conference (SciPy)*, volume 4, pp. 3. Austin, TX, 2010.

Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

Alex Graves. Practical variational inference for neural networks. In *Advances in Neural Information Processing Systems*, pp. 2348–2356, 2011.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. Technical report, arXiv:1207.0580, 2012.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Diederik P Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. Semi-supervised learning with deep generative models. In *Advances in Neural Information Processing Systems*, pp. 3581–3589, 2014.

Hugo Larochelle and Yoshua Bengio. Classification using discriminative restricted boltzmann machines. In *Proceedings of the 25th international conference on Machine learning*, pp. 536–543. ACM, 2008.

Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML*, volume 30, 2013.

Marc'Aurelio Ranzato and Martin Szummer. Semi-supervised learning of compact document representations with deep networks. In *Proceedings of the 25th international conference on Machine learning*, pp. 792–799. ACM, 2008.

Antti Rasmus, Harri Valpola, Mikko Honkala, Mathias Berglund, and Tapani Raiko. Semi-supervised learning with ladder network. *arXiv preprint arXiv:1507.02672*, 2015.

Danilo J. Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *ICML'2014*, 2014.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Rupesh K Srivastava, Klaus Greff, and Jürgen Schmidhuber. Training very deep networks. In *Advances in Neural Information Processing Systems*, pp. 2368–2376, 2015.

Harri Valpola. From neural pca to deep unsupervised learning. *arXiv preprint arXiv:1411.7783*, 2014.

Bart van Merriënboer, Dzmitry Bahdanau, Vincent Dumoulin, Dmitriy Serdyuk, David Warde-Farley, Jan Chorowski, and Yoshua Bengio. Blocks and fuel: Frameworks for deep learning. *arXiv preprint arXiv:1506.00619*, 2015.

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pp. 1096–1103. ACM, 2008.

Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Machine Learning Res.*, 11, 2010.

A
HYPERPARAMETERS FOR DIFFERENT VARIANTS

Here we provide the best hyperparameter combinations we have found for different variants in different settings. We consider the standard deviation of additive Gaussian noise and the reconstruction penalty weights in the decoder as the hyperparameters. For each variant, we fix the best hyperparameters tuned on the validation set and run the variant 10 times with 10 different but fixed data seeds (used to choose 100 or 1000 labeled examples).

Depending on each variant and its hyperparameter space, we used either random search or grid search. Table 3 specifies the search space for hyperparameters and tables 4, 5, and 6 collect the best hyperparameter combinations for each experiment setting. In the case of MLP and AMLP combinator functions, standard deviation of the Gaussian initialization η is chosen from a grid of $(0.0001, 0.006, 0.0125, 0.025, 0.05)$. The best η values are listed in Table 7.

Table 3: Two different hyperparameter search methods. For random search, we run 20 random hyperparameter combinations for each variant and in each task.

Search method	Noise stddev search space ($\times 10^{-1}$)	Reconstruction weights search space
Random search	(0, 0, 0, 0, 0, 0, 0) (1, 1, 1, 1, 1, 1, 1) (2, 2, 2, 2, 2, 2, 2) (3, 3, 3, 3, 3, 3, 3) (4, 4, 4, 4, 4, 4, 4) (5, 5, 5, 5, 5, 5, 5) (6, 6, 6, 6, 6, 6, 6) (7, 7, 7, 7, 7, 7, 7) (8, 8, 8, 8, 8, 8, 8)	(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0) (10.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0) (50.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0) (100.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0) (500.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0) (800.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0) (1000.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0) (2000.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0) (4000.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0) (6000.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0) (500, 10.0, 0.1, 0.1, 0.1, 0.1, 0.1) (1000, 10.0, 0.1, 0.1, 0.1, 0.1, 0.1) (2000, 20.0, 0.2, 0.2, 0.2, 0.2, 0.2)
Grid 100 & 1000	(2, 2, 2, 2, 2, 2, 2) (3, 3, 3, 3, 3, 3, 3) (4, 4, 4, 4, 4, 4, 4)	(1000, 10.0, 0.1, 0.1, 0.1, 0.1, 0.1) (2000, 20.0, 0.2, 0.2, 0.2, 0.2, 0.2) (5000, 50.0, 0.5, 0.5, 0.5, 0.5, 0.5) (10000, 100.0, 1.0, 1.0, 1.0, 1.0, 1.0)
Grid 60000	(2, 2, 2, 2, 2, 2, 2) (3, 3, 3, 3, 3, 3, 3) (4, 4, 4, 4, 4, 4, 4)	(500, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0) (1000, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0) (2500, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0) (5000, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0)

Table 4: Best hyperparameters for the semi-supervised task with 100 labeled examples.

Variant	Search method	Best noise stddev ($\times 10^{-1}$)	Best reconstruction weights
Baseline+noise	Random	(3, 3, 3, 3, 3, 3, 3)	(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0)
Vanilla	Grid	(3, 3, 3, 3, 3, 3, 3)	(1000, 10.0, 0.1, 0.1, 0.1, 0.1, 0.1)
FirstNoise	Random	(6, 0, 0, 0, 0, 0, 0)	(1000, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0)
FirstRecons	Random	(3, 3, 3, 3, 3, 3, 3)	(1000, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0)
FirstN&R	Random	(6, 0, 0, 0, 0, 0, 0)	(1000, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0)
NoLateral	Random	(7, 0, 0, 0, 0, 0, 0)	(100.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0)
RandInit	Grid	(3, 3, 3, 3, 3, 3, 3)	(1000, 10.0, 0.1, 0.1, 0.1, 0.1, 0.1)
RevInit	Grid	(3, 3, 3, 3, 3, 3, 3)	(1000, 10.0, 0.1, 0.1, 0.1, 0.1, 0.1)
NoSig	Grid	(3, 3, 3, 3, 3, 3, 3)	(1000, 10.0, 0.1, 0.1, 0.1, 0.1, 0.1)
NoMult	Grid	(3, 3, 3, 3, 3, 3, 3)	(1000, 10.0, 0.1, 0.1, 0.1, 0.1, 0.1)
Linear	Grid	(3, 3, 3, 3, 3, 3, 3)	(2000, 20.0, 0.2, 0.2, 0.2, 0.2, 0.2)
Gaussian	Grid	(3, 3, 3, 3, 3, 3, 3)	(1000, 10.0, 0.1, 0.1, 0.1, 0.1, 0.1)
GatedGauss	Grid	(2, 2, 2, 2, 2, 2, 2)	(2000, 20.0, 0.2, 0.2, 0.2, 0.2, 0.2)
MLP[4]	Grid	(2, 2, 2, 2, 2, 2, 2)	(5000, 50.0, 0.5, 0.5, 0.5, 0.5, 0.5)
MLP[2,2]	Grid	(2, 2, 2, 2, 2, 2, 2)	(2000, 20.0, 0.2, 0.2, 0.2, 0.2, 0.2)
MLP[2,2,2]	Grid	(2, 2, 2, 2, 2, 2, 2)	(10000, 100.0, 1.0, 1.0, 1.0, 1.0, 1.0)
AMLP[4]	Grid	(3, 3, 3, 3, 3, 3, 3)	(2000, 20.0, 0.2, 0.2, 0.2, 0.2, 0.2)
AMLP[2,2]	Grid	(3, 3, 3, 3, 3, 3, 3)	(2000, 20.0, 0.2, 0.2, 0.2, 0.2, 0.2)
AMLP[2,2,2]	Grid	(3, 3, 3, 3, 3, 3, 3)	(1000, 10.0, 0.2, 0.2, 0.2, 0.2, 0.2)

Table 5: Best hyperparameters for the semi-supervised task with 1000 labeled examples.

Variant	Search method	Best noise stddev ($\times 10^{-1}$)	Best reconstruction weights
Baseline+noise	Random	(2, 2, 2, 2, 2, 2, 2)	(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0)
Vanilla	Grid	(2, 2, 2, 2, 2, 2, 2)	(1000, 10.0, 0.1, 0.1, 0.1, 0.1, 0.1)
FirstNoise	Random	(6, 0, 0, 0, 0, 0, 0)	(1000, 10.0, 0.1, 0.1, 0.1, 0.1, 0.1)
FirstRecons	Random	(3, 3, 3, 3, 3, 3, 3)	(4000, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0)
FirstN&R	Random	(6, 0, 0, 0, 0, 0, 0)	(1000, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0)
NoLateral	Random	(6, 0, 0, 0, 0, 0, 0)	(100.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0)
RandInit	Grid	(2, 2, 2, 2, 2, 2, 2)	(1000, 10.0, 0.1, 0.1, 0.1, 0.1, 0.1)
RevInit	Grid	(2, 2, 2, 2, 2, 2, 2)	(1000, 10.0, 0.1, 0.1, 0.1, 0.1, 0.1)
NoSig	Grid	(2, 2, 2, 2, 2, 2, 2)	(1000, 10.0, 0.1, 0.1, 0.1, 0.1, 0.1)
NoMult	Grid	(3, 3, 3, 3, 3, 3, 3)	(1000, 10.0, 0.1, 0.1, 0.1, 0.1, 0.1)
Linear	Grid	(3, 3, 3, 3, 3, 3, 3)	(2000, 20.0, 0.2, 0.2, 0.2, 0.2, 0.2)
Gaussian	Grid	(3, 3, 3, 3, 3, 3, 3)	(1000, 10.0, 0.1, 0.1, 0.1, 0.1, 0.1)
GatedGauss	Grid	(2, 2, 2, 2, 2, 2, 2)	(1000, 10.0, 0.1, 0.1, 0.1, 0.1, 0.1)
MLP[4]	Grid	(3, 3, 3, 3, 3, 3, 3)	(10000, 100.0, 1.0, 1.0, 1.0, 1.0, 1.0)
MLP[2,2]	Grid	(3, 3, 3, 3, 3, 3, 3)	(5000, 50.0, 0.5, 0.5, 0.5, 0.5, 0.5)
MLP[2,2,2]	Grid	(3, 3, 3, 3, 3, 3, 3)	(2000, 20.0, 0.2, 0.2, 0.2, 0.2, 0.2)
AMLP[4]	Grid	(3, 3, 3, 3, 3, 3, 3)	(5000, 50.0, 0.5, 0.5, 0.5, 0.5, 0.5)
AMLP[2,2]	Grid	(3, 3, 3, 3, 3, 3, 3)	(2000, 20.0, 0.2, 0.2, 0.2, 0.2, 0.2)
AMLP[2,2,2]	Grid	(3, 3, 3, 3, 3, 3, 3)	(1000, 10.0, 0.2, 0.2, 0.2, 0.2, 0.2)

Table 6: Best found hyperparameters for the task of semi-supervised with 60000 labeled examples.

Variant	Search method	Best noise stddev ($\times 10^{-1}$)	Best reconstruction weights
Baseline+noise	Random	(3, 3, 3, 3, 3, 3)	(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0)
Vanilla	Grid	(3, 3, 3, 3, 3, 3)	(500, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0)
FirstNoise	Random	(6, 0, 0, 0, 0, 0, 0)	(500, 10.0, 0.1, 0.1, 0.1, 0.1, 0.1)
FirstRecons	Random	(3, 3, 3, 3, 3, 3)	(500, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0)
FirstN&R	Random	(6, 0, 0, 0, 0, 0, 0)	(500, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0)
NoLateral	Random	(6, 0, 0, 0, 0, 0, 0)	(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0)
RandInit	Grid	(3, 3, 3, 3, 3, 3)	(500, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0)
RevInit	Grid	(3, 3, 3, 3, 3, 3)	(500, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0)
NoSig	Grid	(3, 3, 3, 3, 3, 3)	(500, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0)
NoMult	Grid	(3, 3, 3, 3, 3, 3)	(500, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0)
Linear	Grid	(3, 3, 3, 3, 3, 3)	(500, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0)
Gaussian	Grid	(3, 3, 3, 3, 3, 3)	(1000, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0)
GatedGauss	Grid	(3, 3, 3, 3, 3, 3)	(2000, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0)
MLP[4]	Grid	(3, 3, 3, 3, 3, 3)	(2000, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0)
MLP[2,2]	Grid	(3, 3, 3, 3, 3, 3)	(2000, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0)
MLP[2,2,2]	Grid	(3, 3, 3, 3, 3, 3)	(1000, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0)
AMLP[4]	Grid	(3, 3, 3, 3, 3, 3)	(2000, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0)
AMLP[2,2]	Grid	(3, 3, 3, 3, 3, 3)	(2000, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0)
AMLP[2,2,2]	Grid	(3, 3, 3, 3, 3, 3)	(2000, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0)

Table 7: Best MLP initialization η for all settings.

MLP variant	100 labels	1000 labels	fully-labeled
MLP[4]	0.006	0.006	0.0125
MLP[2,2]	0.05	0.0125	0.05
MLP[2,2,2]	0.025	0.025	0.05
AMLP[4]	0.006	0.025	0.0125
AMLP[2,2]	0.0125	0.0125	0.025
AMLP[2,2,2]	0.006	0.006	0.006