Deep Semi-Supervised Learning with Ladder Networks and Virtual Adversarial Training

Student: Saki Shinoda
Supervisor: Gabriel J. Brostow
MSc Computational Statistics & Machine Learning

September 2017

This report is submitted as part requirement for the MSc Degree in Computational Statistics and Machine Learning at University College London. It is substantially the result of my own work except where explicitly indicated in the text.

The report may be freely copied and distributed provided the source is explicitly acknowledged.

Department of Computer Science
University College London

Abstract

Even as data acquisition becomes increasingly inexpensive and deep learning becomes more powerful, there is a bottleneck in the supervised learning pipeline of obtaining high quality labels. With few labelled training data, the flexibility and power of neural networks make them prone to overfitting. Deep semi-supervised learning attempts to extend the power of neural networks to datasets with few labelled examples by extracting information from much more cheaply acquired unlabelled examples.

We propose a new class of models for semi-supervised learning based on the ladder network and virtual adversarial training. We trained these models with 5, 10, or 100 labelled examples per class from the MNIST dataset, and evaluated performance on both the standard test set and adversarial examples. We found that our models achieve state-of-the-art performance and are additionally very stable in the 5- or 10-per-class setting. Our *ladder with layer-wise virtual adversarial noise* (LVAN-LW) model in particular outperforms the ladder network on the MNIST test set and VAT on adversarial examples generated with L_1 and L_2 norms.

Acknowledgements

I would like to thank my supervisor, Gabriel Brostow, and the Prism group for their advice during the process of working on this project. Many special thanks to Daniel Worrall for his guidance, direction, and encouragement throughout the vicissitudes of working on a deep learning thesis.

Code

Code is available in the public repository at `http://github.com/sakishinoda/tf-ssl/`

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Semi-supervised learning is an active area of research within machine learning that seeks to solve classification and regression problems using a small labelled dataset supplemented by a much larger unlabelled dataset drawn from the same distribution. Equivalently the dataset can be seen as many examples drawn from the distribution of interest, but where only some of the examples are labelled. It seeks to reconcile the traditional dichotomy of supervised learning on the labelled examples and unsupervised learning on unlabelled examples. Here labelling refers to assigning a target value y, discrete in the case of classification and continuous in the case of regression, to an input data example $\mathbf{x}$.

The semi-supervised learning paradigm is relevant to many real-world applications, where it may be easy or inexpensive to collect large amounts of data but expensive or difficult to label all data. One example is medical imaging, where images themselves can be readily acquired but features of interest, such as tumours, require a medical expert to manually annotate the images; for rare conditions, there may be very few people capable of providing high-quality labels. Another example is labelling videos, where it makes very little sense to annotate every frame no matter how easy labelling a single frame is, given the similarity between neighbouring frames and the sheer quantity of frames associated with a single video. The ease of acquiring data differs for each of these examples, but the common thread is that labelling data is significantly more expensive than collecting it. In both of these cases, the aim of semi-supervised learning would be to make use of the unlabelled images in learning the supervised task on the labelled images.

To successfully carry out semi-supervised learning, one must solve two problems:

how to extract information from unlabelled examples, and how to assist a supervised learner using the extracted information. Naively one might solve these two separately, the former with unsupervised learning, and the latter with regularisation. This indeed is an approach that has been taken both using classical methods and in deep learning. However, the most successful methods usually rely on a simultaneous solution to both of these problems, neither of which are trivial on their own.

1.1 Research goals

This thesis compares state-of-the-art deep learning methods for semi-supervised learning in the image classification domain. Our goal was to extend a theoretical understanding of deep methods in semi-supervised learning as regularisation through a comparative analysis of two end-to-end deep semi-supervised systems.

Based on our analysis we propose a class of new models combining elements from the existing state-of-the-art ladder network and virtual adversarial training (VAT) methods, which have differing theoretical approaches to the semi-supervised learning problem. Combining these approaches allows us to exceed state-of-the-art performance in certain tasks; by further exploring the circumstances under which each of these approaches is successful we are able to expand on our understanding of their regularisation mechanisms.

1.2 Background and related work

Semi-supervised learning requires that the unlabelled data provide information that is relevant to the classification (or regression) task for which the labelled examples are labelled, *i.e.,* for data x with labels y, the knowledge we can obtain about $\Pr(x)$ from unlabelled data must provide some additional information about $\Pr(y|x)$. Semi-supervised learning thus depends upon a number of different assumptions about the proximity of data points in input and output space which are discussed in Section 2.1. Many early methods in semi-supervised learning are clearly motivated by one or a combination of these assumptions, and are briefly reviewed in Section 2.2.

Recently, with the rapid development of deep learning, the field of semi-supervised learning has become more empirical, favouring adaptation of successful techniques from supervised and unsupervised deep learning. Deep learning refers to the use of artificial neural networks, typically fitted to data ('trained') using the backpropagation algorithm and a variant of stochastic gradient descent. Regularisation is an important aspect of any machine learning method, and a number of techniques have recently been developed specifically for deep learning. Section 2.3 provides an overview of the construction, training and regularisation of neural networks.

Some of the approaches in deep learning that have proven successful in semi-supervised learning are embeddings, generative models, adversarial training, and regularisation through random or local perturbations. These are reviewed in Section 2.4.

This thesis in particular builds directly on ladder networks [30] and virtual adversarial training [23]. Ladder networks are a form of specialised generative model that simultaneously carries out supervised learning on labelled examples using a classification loss and unsupervised learning on unlabelled examples using a reconstruction loss. Virtual adversarial training (VAT) regularises neural networks with perturbations that most sharply shift the output probabilities of the model. These methods are further described and analysed in Chapter 3.

1.3 Our contribution

Based on our analysis in Chapter 3, in Chapter 4 we propose a class of models that build on the ladder network and virtual adversarial training (VAT):

Ladder with virtual adversarial cost (LVAC): applies virtual adversarial cost of ladder network output with respect to inputs.

Ladder with virtual adversarial cost, layer-wise (LVAC-LW): applies virtual adversarial cost of ladder network output with respect to activations of every layer in encoder.

Ladder with virtual adversarial noise (LVAN): adds a virtual adversarial perturbation to the input images of the ladder network.

Ladder with virtual adversarial noise, layer-wise (LVAN-LW): adds a virtual adversarial perturbation to activations of each layer in the encoder.

1.4 Experiments and findings

We tested the performance of the proposed models on the standard benchmarks of MNIST image classification with 100 and 1000 labelled classes to allow direct comparison with the ladder network and VAT method on which the models are based. We then further investigated the performance of the ladder network, VAT, and our proposed models with only 50 labelled examples.

In addition to the standard measure of average error rate on a held-out test set, we also considered the average error rate on adversarial examples generated using the fast gradient method [12].

We confirmed the strength on the standard benchmarks of the ladder network, which as published outperforms VAT. We found that our models outperform the ladder network in accuracy and in stability for the cases with very few examples (5 per class, 10 per class). Additionally, we found that our models are highly robust to adversarial attacks.

1.5 Further work

Many avenues of improvement and further work suggest themselves on the basis of the results of this investigation and the hypotheses which could not be pursued due to a lack of time. Given the opportunity, we would extend the work in this thesis by carrying out more thorough hyperparameter optimisations to obtain more accurate and precise estimates of our model performance.

We attempted, but did not have time to complete, evaluating our models on more challenging datasets than MNIST, specifically SVHN and CIFAR-10, and extending our models to work with convolutional networks. Another extension that was considered was to combine the ladder network with virtual adversarial training for unlabelled examples *and* adversarial training for labelled examples.

Possible future investigations could examine the performance of not only our proposed models but the base models of the ladder network and VAT on a dataset with more than 10 classes, such as CIFAR-100. We would also like to see how the recently proposed *universal adversarial perturbations* could be applied to semi-supervised learning.

Chapter 2

Background and related work

2.1 Foundations of semi-supervised learning

This section provides a brief summary of relevant concepts in semi-supervised learning (SSL). For a fuller exposition, the reader is referred to [6].

To understand semi-supervised learning, we first remind the reader of the formalisations of the two traditional paradigms of machine learning, supervised and unsupervised learning.

In *unsupervised* learning, we have a dataset of n points $X = (x_1, ..., x_n)$, where we assume that the points are drawn from the same underlying data distribution $\Pr(x)$. If we also assume that the points are drawn independently of each other we have the common i.i.d. (independent and identically distributed) setting. Unsupervised learning seeks structure in the data X. Tasks within unsupervised learning include estimating the density $\Pr(x)$, dimensionality reduction, clustering, and anomaly detection.

The dataset in *supervised* learning comprises pairs of examples x_i and the corresponding labels y_i and the goal is to learn the mapping from x to y. Again, it is commonly assumed that examples x_i are i.i.d.. The performance of supervised learning algorithms is typically evaluated using predictions on unseen examples, whether using accuracy, precision, recall, or other metrics of performance dependent on the task.

The prediction problem in supervised learning can be formulated as estimating the density $\Pr(y|x)$. By Bayes' theorem, this can be written:

$$\Pr(y|x) = \frac{\Pr(x|y)\Pr(y)}{\int_y \Pr(x|y)\Pr(y)}.$$

This gives rise to two approaches in supervised learning: generative, which seeks to model the class-conditional density $\Pr(x|y)$; and discriminative, which directly estimates $\Pr(y|x)$ without seeking to model how the data pairs are generated.

In the *semi-supervised* setting, the learning algorithm has access to unlabelled data and some supervised information. Usually this is labels for a subset of the examples. This is the standard setting for semi-supervised learning, though there are other forms of partially supervised learning where the supervised information is not the labels themselves but some constraints that can be applied to unsupervised learning.

Within the standard semi-supervised learning setting, there is a further distinction between *transductive* and *inductive* learning. Given a labelled training set and an unlabelled 'test' set, the aim of transduction is to perform predictions on the test set. This differs from the standard *supervised* case in that the algorithm can also learn from the test set, though it does not have access to labels for those examples. By contrast, the goal of induction is to model a general function to perform prediction on any example drawn from the input space. Most classical methods of semi-supervised learning, summarised in the next section, are transductive. Recent deep learning approaches, which are reviewed in Section 2.4, are inductive.

Another paradigm that falls outside the realm of traditional supervised or unsupervised learning is active learning, which also seeks to address the high cost of labelling data relative to acquiring data. In active learning, also called query selection or experiment design in the statistical literature, the algorithm has access to a pool of unlabelled data, from which it selects examples to be labelled [38]. This is typically done by modelling the uncertainty, risk, or cost the model associates with different data examples and querying labels for examples so as to decrease the uncertainty, risk, or cost most. Active learning thus aims to make the labelling process more efficient by optimising a priori over the examples in the labelled dataset. In semi-supervised learning, the split of the labelled and unlabelled examples must be considered fixed.

2.1.1 Assumptions of semi-supervised learning

The core assumptions of semi-supervised learning as stated in [6] can be paraphrased as:

Smoothness assumption: if two points x_1, x_2 in a high-density region of input space are close, then the corresponding outputs y_1, y_2 should also be close.

Cluster assumption: if points are in the same cluster, they are likely to be of the same class. Clusters can be considered regions of high-density, thus there is an equivalent formulation of this assumption that decision boundaries between classes should lie in low-density regions (principle of *low density separation*).

Manifold assumption: the (high-dimensional) data lie roughly on a low-dimensional manifold. These manifolds can approximate high-density regions, and geodesic distances on these manifolds can define the closeness of points as used in the smoothness assumption.

2.2 Classical semi-supervised learning

This section very briefly outlines the methods of classical semi-supervised learning based on each of the assumptions defined in the previous section, primarily to establish a context against which we can compare the deep learning methods discussed in Section 2.4.

2.2.1 Generative models

As defined in Section 2.1, generative models estimate the conditional density $\Pr(x|y)$, in which case any additional information about the marginal distribution $\Pr(x)$ provided by unlabelled data can be useful. Generative modelling can be thought to apply the clustering assumption as the labels y are associated with different clusters. This can be illustrated by example: if one assumes that $\Pr(x|y)$ is Gaussian, the expectation-maximisation (EM) algorithm for mixture-of-Gaussians models can be applied with y replacing the unknown hidden or latent variable for labelled examples [6].

2.2.2 Low density separation

Methods in this group are generally extensions of existing classifiers that use the *cluster* assumption and push decision boundaries produced by the underlying classifier to low-density regions.

The *transductive support vector machine* (TSVM) is an extension of the support vector machine, a classic supervised learning algorithm, to the semi-supervised setting by maximising the distance to decision boundaries for unlabelled as well as labelled points [6, Ch. 6]. *Gaussian processes*, another classic and powerful supervised learning algorithm, can be extended to binary semi-supervised classification by introducing a null class to label the space between regular classes [6, Ch. 8]. *Entropy minimisation* encourages class-conditional probabilities $\Pr(y|x)$ to be close to either 1 or 0 for labelled and unlabelled points; this results in high-density areas having probabilities close to 1 or 0 throughout the region, and lower probability regions correspond to decision boundaries [6, Ch. 9]. *Data-dependent regularisation* based on entropy multiplies the L_2 norm regulariser with a factor corresponding to data density at decision boundaries, directly pushing boundaries to low density areas.[6, Ch. 10].

2.2.3 Graph-based methods

In graph-based semi-supervised learning, data are represented as a graph with examples as nodes and the pairwise distances between examples as weights on edges. This can be considered to form an approximation of the manifold on which the data lies, with the weighted shortest path between examples corresponding to a geodesic on that manifold. Because of the discrete and finite representation of data as a graph, graph-based methods are, without significant extension, necessarily transductive. Algorithms such as label propagation and label spreading iteratively spread labels from the labelled examples to unlabelled [3].

Graph regularisation methods use the graph to impose a smoothness penalty as a term in the optimisation of the model. This is often based on the graph Laplacian, a characteristic matrix of the graph which can be seen to represent the smoothness of the graph. Penalising the Laplacian penalises changes in labels over short distances on the graph, which corresponds to smoothing the manifold the graph approximates, as is consistent with the assumptions stated in Section 2.1.1.

2.2.4 Change of representation

Another group of methods in classical semi-supervised learning involve two steps. The first is an unsupervised step to map the examples to a new space with desirable properties. In the second step, a supervised learning carries out classification in the new feature space. The change of representation, which may also be a change of metric or change of kernel, should ideally emphasise the smoothness and clustering properties of the data distribution [6].

2.3 Neural networks and deep learning

Neural networks are essentially functions with many parameters created through the composition of many smaller functions, largely affine transformations and non-linear functions. This section is based on the content presented in [27].

2.3.1 Basic components

The simplest but possibly most commonly used building block is the linear layer, also called *fully connected* or *dense* layers. The linear layer is an affine transformation that can be described by the matrix-vector operation

$$\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

for vectors $\mathbf{x}, \mathbf{y}, \mathbf{b}$ and matrix $\mathbf{W}$. $\mathbf{W}$ contains the *weights* of the layer and the $\mathbf{b}$ the *biases*. Each element of the output vector $\mathbf{y}$ is a weighted sum of the elements of the input, shifted by the bias, *i.e.,* $y_i = \sum_j W_{ij}x_j + b_i$. From this form we can see that a linear layer is simply equivalent to *linear regression*.

Linear layers are typically followed by an activation function, also called a non-linearity. Common activation functions include the sigmoid function

$$\sigma(x) = \frac{1}{(1 + e^{-x})},$$

the Rectified Linear Unit (ReLU)

$$y(x) = \text{maximum}(0, x),$$

and leaky ReLU (LReLU)

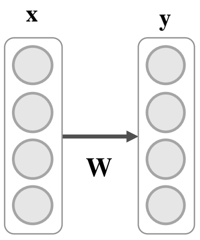
$$y(x) = \text{maximum}(\alpha x, x),\ \alpha \leq 1.$$

The sigmoid function is also called a rescaling function as it maps all inputs to $[0, 1]$; this is a useful property as it can be used to model probabilities, for example the probability of example x belonging to a class y. In fact, a linear layer followed by a sigmoid can be written

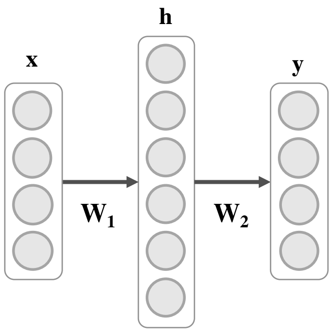
$$y = \sigma\left(\sum_i w_i x_i + b\right)$$

for a scalar output y, which is equivalent to *logistic regression*.

Figure 2.1: Conceptual illustrations of basic neural networks.



(a) Representation of a single layer network $\mathbf{y} = \mathbf{W}\mathbf{x}$ with inputs $\mathbf{x}$, outputs $\mathbf{y}$, weights $\mathbf{W}$.



(b) Representation of a network with one hidden layer, $\mathbf{y} = \mathbf{W_2}\left(\mathbf{W_1}\mathbf{x}\right)$ with inputs $\mathbf{x}$, outputs $\mathbf{y}$, weights $\mathbf{W}_1, \mathbf{W}_2$. $\mathbf{h}$ is the hidden layer.

Thus a linear layer composed with a sigmoid function can be used for *binary classification*. A generalisation of the sigmoid function that can allow *multi-class classification* is the softmax function, which has vector-valued output $\mathbf{y}$ with elements

$$y_i = \frac{e^{x_i}}{\sum_{k=1}^{K} e^{x_j}}.$$

Elements of $\mathbf{y}$ are non-negative and sum to one, so the vector $\mathbf{y}$ can be interpreted as a categorical probability distribution over the K indices of $\mathbf{y}$.

A softmax function is often applied at the 'end' of a multi-class classification model and represents the predictive distribution of the model. For an input $\mathbf{x}$, a vector $\mathbf{y}$ is output by the model where y_i denotes the probability under the model of the example $\mathbf{x}$ being labelled as class i, $\Pr(y = i|\mathbf{x})$. We can compute the negative log-likelihood (NLL) of the true label under the model using a 'one-hot' *target* vector $\mathbf{t}$ with $t_i = 1$ for the true class i and $t_i = 0$ otherwise:

$$\mathrm{NLL}(\mathbf{t}, \mathbf{y}) = -\sum_{i=1}^{C} t_i \log y_i.$$

This is also called the (softmax) cross-entropy.

Hidden layers extend neural networks beyond linear and logistic regression. Linear and logistic regression can be seen as single-layer networks, consisting of one linear/affine transformation followed by a non-linearity. Inputs are mapped by one transformation to outputs. Networks with hidden layers map inputs to intermediate representations; the term *hidden* refers to any layer that does not take as its input the inputs to the network $\mathbf{x}$ or produce as output the outputs of the network y. Figure 2.1a and 2.1b illustrate a single-layer network and network with one hidden layer respectively. A network with a single hidden layer can be written as:

$$\mathbf{h} = \phi_1(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$$
$$\mathbf{y} = \phi_2(\mathbf{W}_2\mathbf{h} + \mathbf{b}_2)$$

where ϕ_i are activation functions. $\mathbf{W}_1, \mathbf{b}_1, \phi_1$ correspond to the first layer of the network and $\mathbf{W}_2, \mathbf{b}_2, \phi_2$ to the second.

In the example above, the dimensionality of the weight matrix $\mathbf{W}_1$ determines the dimensionality of the hidden representation $\mathbf{h}$. Multiplication of a $m \times n$ matrix by a $p \times q$ matrix requires $n = p$ and produces a product with dimension $m \times q$. Thus if the weight matrix $\mathbf{W}_1$ has dimensions $m \times n$, m must match the dimensionality of the inputs, and n is the dimensionality of the hidden layer. n is the number of *neurons* or *units* in the hidden layer. Networks consisting of only fully-connected layers and activations are called *feedforward networks* or *multilayer perceptrons* (MLP).

By transforming inputs to an intermediate representation rather than directly to the outputs, a neural network with hidden layers has immense flexibility; it has been shown that neural networks with only one hidden layer are 'universal function approximators' [26]. In practice this means that neural networks can model highly complicated, non-linear functions.

2.3.2 Learning in neural networks

As applied to neural networks, the process of fitting model parameters to data is typically called *training* or *learning*. To train a neural network, we define a function of the data and the model parameters called the *loss*, *cost*, or *objective* function. The model is then trained by minimising the loss with respect to the model parameters. This minimisation is typically carried out using iterative optimisation methods, particularly variants of gradient descent. This section is based on [27] and [33].

2.3.2.1 Backpropagation

Gradient descent is used for neural network optimisation because the gradients of the loss function of a neural network with respect to its parameters can be computed efficiently using the *backpropagation* algorithm. As a neural network can be written as nested function compositions, the chain rule of derivatives

$$y = f(g(x)) \implies \frac{\mathrm{d}y}{\mathrm{d}x} = \frac{\mathrm{d}f}{\mathrm{d}g}\frac{\mathrm{d}g}{\mathrm{d}x}$$

or for the multivariate case

$$y = f\left(g^{(1)}(x), \ldots, g^{(m)}(x)\right) \implies \frac{\partial y}{\partial x} = \sum_{i=1}^{i=m} \frac{\partial f}{\partial g^{(i)}} \frac{\partial g^{(i)}}{\partial x}$$

can be used to efficiently combine modular derivatives. To do this, a *compute graph* is constructed with each module as a node; message passing algorithms on the compute graph can be used to calculate the full derivative efficiently. Traversing the graph from inputs to outputs is called *forward mode automatic differentiation*. Traversing backwards from outputs to inputs is called *reverse mode automatic differentiation*, or more commonly, *backpropagation* or *backprop*.

The modular nature of the backpropagation algorithm means that a neural network can be constructed of any module $f(\mathbf{x})$ with three functions: firstly, return output $\mathbf{y}$ given input $\mathbf{x}$ (forward propagation),

$$\mathbf{y} = f(\mathbf{x});$$

secondly, compute gradient of loss with respect to inputs $\partial L/\partial x_i$, given gradient of loss with respect to outputs $\partial L/\partial y_i$ (backpropagation),

$$\frac{\partial L}{\partial x_i} = \sum_{j=1}^{J} \frac{\partial L}{\partial y_j} \frac{\partial y_j}{\partial x_i};$$

thirdly, compute gradient of loss with respect to parameters $\partial L/\partial \theta_i$ given gradient with respect to outputs $\partial L/\partial y_i$,

$$\frac{\partial L}{\partial \theta_i} = \sum_{j=1}^{J} \frac{\partial L}{\partial y_j} \frac{\partial y_j}{\partial \theta_i}.$$

2.3.2.2 Gradient descent

The basic principle behind optimisation by gradient descent is to iteratively follow the direction of steepest downward gradient, since the derivative, *i.e.,* gradient, of a function is zero at a minimum.

The simplest form of gradient descent is batch gradient descent, where each gradient step is computed on all data. For loss function J on a model with parameters θ, the update rule is given by

$$\theta \leftarrow \theta - \eta \nabla_\theta J(\theta),$$

where η is the *learning rate* or *step size*, a training hyperparameter that controls the magnitude of the updates applied and hence how quickly optimisation proceeds.

In online gradient descent, updates are computed on individual examples x_i, y_i:

$$\theta \leftarrow \theta - \eta \nabla_\theta J(\theta; x^{(i)}; y^{(i)}).$$

Batch gradient descent is guaranteed to converge to a minimum but is slow and can carry out redundant computations; online gradient descent is fast but has high variance, causing significant fluctuations. To overcome these shortcomings, *minibatch* gradient descent is most commonly used. In minibatch gradient descent, n samples from the dataset are used to compute the update:

$$\theta \leftarrow \theta - \eta \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)}).$$

We use *stochastic gradient descent* (SGD) to refer to the minibatch setting.

2.3.2.3 Momentum

SGD can often overshoot and oscillate where the curvature of the objective function varies significantly in different directions; this often occurs around local minima, which are the target of optimisation by gradient descent. *Momentum* dampens these oscillations by increasing updates in dimensions with consistent gradient directions over updates. The parameter update with momentum is defined as

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta \nabla_\theta J(\theta) \\ \theta &\leftarrow \theta - v_t, \end{aligned}$$

where $\gamma \in [0, 1)$ is the momentum coefficient, usually set to 0.9.

Nesterov momentum is a popular version of momentum with stronger convergence guarantees for convex optimisation problems and which works well in practice for neural network training. It uses an approximation of the next position of parameters in computing the update. The parameter update is given by

$$\begin{aligned} v_t &= \gamma\, v_{t-1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1}) \\ \theta &\leftarrow \theta - v_t \end{aligned}$$

2.3.2.4 Adaptive gradient methods

Most recent optimisation methods used for training neural networks use momentum and additionally *adaptive* learning rates for different dimensions. Such methods include AdaGrad, RMSProp, and Adam, which we use in this work. Adam, introduced by Kingma and Ba in [14], estimates the first and second moments of the gradients to adapt learning rates for each parameter. The algorithm for Adam is given in 2.1.

Algorithm 2.1 Adam algorithm for stochastic optimisation. Source: [14]

```
**Require:** $\eta$: Stepsize
**Require:** $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
**Require:** $f(\theta)$: Stochastic objective function with parameters
**Require:** $\theta_0$: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialise 2nd moment vector)
  $t \leftarrow 0$ (Initialise timestep)
  **while** $\theta_t$ not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_2) \cdot g$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_1 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\hat{m}_t \leftarrow m_t/(1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\hat{v}_t \leftarrow v_t/(1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \eta \cdot \hat{m}_t/(\sqrt{\hat{v}_t} + \epsilon)$ (Update parameters)
  **end while**
  **return** $\theta_t$ (Resulting parameters)
```

2.3.2.5 Initialisation

Stochastic gradient descent and the training of neural networks is highly sensitive to the initialisation of the weights. Given the number of parameters in a neural network, which can easily be in the tens of thousands, they are most easily initialised by sampling randomly; the researcher controls the initialisation by specifying the parameters of the distribution to sample the initialisations from. In particular architectures there are specific initialisations which are theoretically or empirically motivated, but in general a normal or uniform distribution close to zero is used. One initialisation scheme which is commonly used is the Xavier or Glorot initialisation scheme, which aims to keep gradients from shrinking or growing too quickly as they propagate through layers, to maintain a steady training signal [7]. The Glorot initialisation for a layer with n_{in} connections into the layer and n_{out} connections out is either a zero-mean Gaussian with variance

$$\mathrm{Var}(W) = \frac{2}{n_{in} + n_{out}}$$

or a uniform distribution bounded by $\pm\sqrt{\mathrm{Var}(W)}$.

2.3.2.6 Hyperparameter optimisation

An important and yet under-documented part of the deep learning pipeline is hyperparameter optimisation and model selection. Hyperparameters which are commonly tuned include the distribution for random weight initialisation; learning rate, momentum and other optimiser parameters; schedule of decay for learning rates; strength of regularisation (see Section 2.3.5); and other, often model-specific, parameters. It is common practice to carry out grid searches, with the range of the grid determined by researchers' prior beliefs, or random searches for high–dimensional hyperparameter spaces as suggested by Bergstra and Bengio in [4].

2.3.3 Batch normalisation

Batch normalisation, introduced by Ioffe and Szegedy in [13], significantly accelerates and stabilises training of neural networks. The nonlinearities used in neural networks have saturation regimes, usually for very large or very small inputs, where gradients tend to zero. This is the so-called *vanishing gradient* problem, which is particularly pronounced in very deep networks due to *internal covariate shift* whereby the effect of stacked layers is to amplify shifts of activations into saturation regimes. Ioffe and Szegedy hypothesised that stabilising the distribution of inputs to each layer would reduce this shift and accelerate training.

The principle of batch normalisation is inspired by *whitening* which is usually a preprocessing step on images. Each dimension of the input $\mathbf{x}$ is centred to zero and normalised to unit standard deviation to give $\widehat{\mathbf{x}}$:

$$\widehat{x}_k = \frac{x_k - \mathrm{E}\left[x_k\right]}{\sqrt{\mathrm{Var}\left[x_k\right]}},$$

where the expectation and variance are computed over the whole training data.

The key to batch normalisation is to approximate this whitening operation for each layer using the mean and variance statistics of the minibatch used for minibatch gradient descent. The batch normalisation operation is further parametrised by trainable parameters γ and β, which rescale and shift the batch-whitened inputs. The procedure at training time is as follows:

$$\begin{aligned}
\mu_B &\leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \\
\sigma_B^2 &\leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2 \\
\widehat{x}_i &\leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \\
y_i &\leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN} x_i \gamma, \beta
\end{aligned}$$

At testing time, the outputs should depend deterministically on the inputs and not be affected by the selection of examples in the testing batch; indeed, testing should be possible with any number, including only one, of examples at a time. Thus minibatches are batch-normalised using not the batch statistics but estimates of the population statistics accumulated during training. These estimates are typically weighted moving averages of the batch statistics during training.

2.3.4 Architectures

Whereas historically machine learning relied heavily on feature engineering, neural networks learn the most salient features directly from the data. It could be argued that feature engineering is replaced in deep learning by architecture engineering, in which individual layers and overall structure of networks are designed using priors about the data and the task.

Convolutional layers are a simple example at the layer level, which use priors on the structure of image data: that images are localised and invariant to spatial translations [9, Ch. 9]. Another commonly used component with architecture strongly informed by priors about data are recurrent neural networks for sequence data [9, Ch. 10]. At the scale of whole networks, architectures can be specialised to varying degrees. Autoencoders, which are discussed in greater detail in Section 3.1.1, rely on their structure to carry out unsupervised learning.

2.3.5 Regularisation for neural networks

Generalisation refers to the ability of a model fitted to some training set to perform well, according to some metric, on previously unseen data, *e.g.,* a held-out test set. Regularisation techniques aim to reduce this test error, often at the expense of increased training set error.

Many traditional regularisation techniques limit the capacity of models by adding a penalty term proportional to a norm of the parameters to the optimisation objective. Such penalty terms easily extend from classic machine learning techniques such as linear regression to neural networks. When the penalty is with an L_2 norm, this corresponds to *Tikhonov regularisation*, *ridge regression*, or in deep learning, *weight decay*.

Bishop showed in [5] that perturbing inputs with Gaussian noise during training of neural networks is equivalent to regularisation with an L_2 penalty. Many methods of regularising neural networks have since been developed that take advantage of stochastic perturbations in other ways. Adding Gaussian noise to inputs or hidden layers has been to some extent replaced by dropout [40], where only a randomly selected subset of units in the neural network are used for each parameter update. Input units are typically included with probability 0.8, hidden units with 0.5 [9]. Dropout can be considered an extension of bagging, whereby many models are trained and used for prediction on each test example, to neural networks, where the time and resources needed to train a single model make traditional bagging prohibitively expensive. Dropout effectively ensembles many sub-networks of the base model.

Another form of perturbations that are used to regularise neural networks, particularly in computer vision, is data augmentation. Images in the training data are transformed in ways under which their labels are invariant, *e.g.,* cropping or rotating, and the transformed images are used as additional training examples. This can be seen as incorporating a prior about the structure of the data and invariances in the underlying distribution [9]. The augmentations can also be randomised, which adds stochasticity to the perturbation process.

Adversarial perturbations have also been successfully used for regularisation. Adversarial training (AT) originates in the observation that many deep neural network classifiers, though often made robust to small random perturbations by the methods above, remain highly sensitive to certain directions, *i.e.,* a small perturbation of the input in a so-called *adversarial* direction can cause a model to misclassify an example [41]. Such adversarial *attacks* have been studied as a vulnerability of neural networks, but their study has yielded some insights into methods of regularisation by increasing robustness to these examples, as explored in [12].

There is a close connection between regularisation and semi-supervised learning, which becomes apparent when examining the deep semi-supervised learning techniques described in the next section.

2.4 Deep semi-supervised learning

Results obtained by many of the papers mentioned in this section are shown in Table 6.1 in Chapter 6.

Supervised learning with neural networks has been very successful on image classification and other computer vision tasks, but deep learning generally requires very large amounts of data to learn. The challenge of extending the successes in deep supervised learning to semi-supervised is thus to regularise the models by somehow using unlabelled data.

One approach to deep semi-supervised learning (SSL) is training feedforward classifiers in the supervised paradigm, but with an *auxiliary* loss, a penalty from an unsupervised embedding of the data [44]. An embedding, a term common in deep learning literature for the change of representation of data to useful features spaces previously described in Section 2.2, can be learnt using unsupervised architectures, *e.g.,* autoencoder, or specialised loss functions based on similarities, *e.g.,* triplet loss [36].

A specialised architecture for semi-supervised learning based on the autoencoder and which uses a form of auxiliary loss is the ladder network [42, 30]. Autoencoders are a class of unsupervised learning architectures that aim to learn a useful mapping to a representation (also called a latent code) in an encoder network by optimising a reconstruction loss on the outputs from that representation in a decoder, *i.e.,* the whole encoder-decoder structure aims to minimise the loss in copying the input to the output by learning a useful intermediate representation. Autoencoders are discussed in further detail in Section 3.1.1. The ladder network combines a feedforward classifier to compute supervised loss on labelled examples and an autoencoder architecture to compute a reconstruction cost for unlabelled examples at every layer of the encoder. The ladder network is one of two methods that forms the basis of this work, and is discussed in detail in Section 3. After Rasmus et al. introduced the ladder network for semi-supervised learning in [30], Pezeshki et al. analysed it in detail in [29] and were able to achieve slightly better performance using a small multilayer perceptron as the denoising function.

Recent advances in generative modelling using neural networks has fuelled a resurgence of generative models for semi-supervised learning. One category of generative models used in SSL are the deep generative networks introduced by Kingma et al. in [15] (DGM) and further developed as the Auxiliary Deep Generative Model (ADGM) and Skip Deep Generative Model (SDGM) by Maaløe et al. in [19], which approach semi-supervised learning as a label imputation problem on unlabelled examples using variational inference on latent variable models. In deep generative networks, probability distributions on latent and visible variables are parametrised by several different neural networks; this can make the models difficult to train. Other than the SDGM, these models cannot be trained end-to-end using backpropagation. While they achieve very good results, their instability and the difficulty of training with approximate inference currently limit their scalability and wider adoption within semi-supervised learning.

Another category of generative models are Generative Adversarial Networks (GANs) introduced by Goodfellow et al. in [10]. GANs simultaneously train a generator network $G(\mathbf{z}; \boldsymbol{\theta}^{(G)})$ and a discriminator network $D(\mathbf{x})$. The generator is trained to transform noise vectors $\mathbf{z}$ to samples from the data distribution $\mathrm{Pr}_{data}(\mathbf{x})$. The discriminator is trained to distinguish between samples from the true data distribution and the distribution of the generator samples $\mathrm{Pr}_{model}(\mathbf{x})$. The training signal for the generator comes from the discriminator; the generator learns to try to 'fool' the discriminator into classifying a generated sample as being drawn from $\mathrm{Pr}_{data}(\mathbf{x})$. Training of an adversarial net can be characterised as a two-player game in the game theoretic sense, and there is a natural convergence point at the *Nash equilibrium* of the game. The entire system can be trained by backpropagation without using approximate inference as in the generative models previously discussed, but the optimisation problem is non-convex and high-dimensional.

The extension of GANs to semi-supervised learning by Salimans et al. in [35] is to add to the supervised cross-entropy loss of a standard classifier an additional unsupervised loss function for the discriminator to distinguish between 'true' unlabelled examples from $\Pr_{data}(\mathbf{x})$ and samples generated according to $\Pr_{model}(\mathbf{x})$. [35] achieved state-of-the-art results in SSL using adversarial nets trained with feature matching, where the training target for the generator is not to maximise the discriminator output (the probability that the input is real rather than generated), but to match the statistics of an intermediate layer of the discriminator. This approach is similar to the layer-wise denoising carried out by the aforementioned ladder network.

Another successful approach with GANs is the Categorical Generative Adversarial Network (CatGAN) by Springenberg [39], which generalises the discriminator in the GAN from the binary generated/real classification regime to an unsupervised clustering into a predetermined number of classes. The CatGAN is applied to SSL by combining this unsupervised cost with the standard cross-entropy loss on labelled examples.

A related technique that also generalises GANs to unsupervised learning is the adversarial autoencoder (AAE), developed by Makhzani et al. in [21]. AAEs use the machinery of the GAN to perform variational inference by matching the distribution of the latent code of an autoencoder, which acts as the generator network, to a prior distributional form. AAEs, though originally designed as an unsupervised method, can be used for semi-supervised learning by passing a one-hot class vector into the discriminator where the classes include the original label classes and an additional class for unlabelled examples.

While adversarial nets can generate very high quality samples indistinguishable by humans as generated examples [35] and have had excellent results on semi-supervised learning, training of GANs is challenging as they often fail to converge [11].

A similar vein of work to adversarial nets that extends the idea of adversarial training from the two-player game context is that of *adversarial training* (AT), as briefly mentioned in Section 2.3.5. Adversarial training as originally formulated requires the ground truth labels of the examples to be perturbed. In [23], Miyato et al. introduced *virtual adversarial training* (VAT), a technique based on similar principles but which does not require true labels. In place of the adversarial direction which causes misclassification, the *virtual* adversarial direction gives the perturbation in input space which most shifts the prediction probabilities of a model. VAT is the second of the two methods (following the ladder) on which this work is based, and both AT and VAT are discussed in greater detail in Chapter 3.

VAT belongs to a class of semi-supervised learning methods which can also be seen as a regularisation method, as mentioned in Section 2.3.5. Another such method is randomised data augmentation [34]. The *temporal ensembling* method of Laine and Aila in [17], which uses an ensemble of models from different points during training, extends the regularising effect of ensembles that is also exploited by methods such as dropout.

Many of the perturbation-based regularisation and semi-supervised learning methods such as VAT and data augmentation can be thought to exploit the smoothness and manifold assumptions described in Section 2.1.1. In general and for these methods in particular, we assume that inputs occur only along a collection of manifolds containing a small subset of points, with variations of interest in the output only occurring along directions that lie in the manifold, or when moving from one manifold to another [9, page 156]. This is supported by the distribution of real-world data, which are highly concentrated in salient feature spaces, *e.g.,* images, text strings, sounds. If data lie on a manifold, then there exists a set of transformation that allow traversal of the manifold by moving from an example to a highly similar one. In images, these correspond to the familiar transformations used in data augmentation such as rotation, lighting, and scaling. A class of older methods including tangent propagation, double backpropagation, contractive autoencoders [32], and the manifold tangent classifier [31], directly aim to smooth this manifold either isotropically or in particular directions relative to the manifold. We will see in Chapter 3 that virtual adversarial training has a similar effect.

Chapter 3

Comparative analysis of ladder networks and virtual adversarial training

In this chapter we discuss two state-of-the-art methods in deep semi-supervised learning: the ladder network and virtual adversarial training. The former represents the approach common in deep learning in general, of designing architectures suited to the problem, applied to semi-supervised learning. In its theoretical motivations the latter resembles an adaptation to deep learning of the approaches taken historically in classical semi-supervised learning. We have chosen to study and build on these models in particular as they both perform very well while still being relatively simple, fast, and stable to train, especially compared to the generative models in the field.

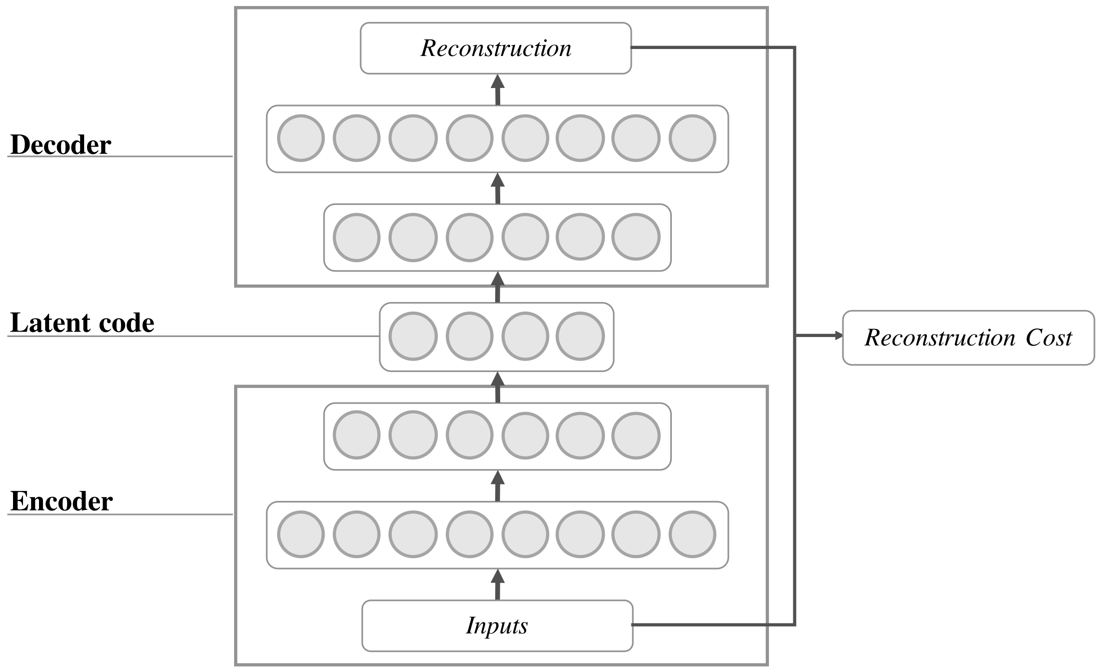
We first describe each method in detail, then compare the two approaches on the basis of their theoretical foundations and practical utility.

3.1 Ladder network

3.1.1 Autoencoders

The ladder network is modelled on the autoencoder (AE) architecture, which is conceptually illustrated in Figure 3.1. An autoencoder is an architecture originating in unsupervised learning which aims to generate a representation $\mathbf{c}(\mathbf{x})$ from which the input $\mathbf{x}$ can be reconstructed [2]. The *encoder* takes as input $\mathbf{x}$ and generates the representation $\mathbf{c}(\mathbf{x})$ (also called the encoding), and the *decoder* takes as input the representation

Figure 3.1: Conceptual illustration of an autoencoder architecture.



$\mathbf{c}(\mathbf{x})$ and outputs $\tilde{\mathbf{x}}$, a reconstruction of $\mathbf{x}$. The autoencoder is trained by minimising the reconstruction cost RC, which is most commonly the negative log-likelihood of the input given the encoding:

$$RC \propto -\log \Pr(\mathbf{x}|\mathbf{c}(\mathbf{x})).$$

For Gaussian $\mathbf{x}|\mathbf{c}(\mathbf{x})$ the reconstruction cost becomes the squared error between input $\mathbf{x}$ and reconstruction $\tilde{\mathbf{x}}$:

$$RC \propto (\mathbf{x} - \tilde{\mathbf{x}})^\top (\mathbf{x} - \tilde{\mathbf{x}})$$

.

As no label is required, the reconstruction cost can be used for purely unsupervised learning. Although the autoencoder is trained to optimise the quality of the reconstruction, the learnt representations $\mathbf{c}(\mathbf{x})$ are usually of much greater interest than the reconstructions.

The standard AE architecture as illustrated in Figure 3.1 has an informational bottleneck at the latent code, which constrains $\mathbf{c}(\mathbf{x})$ to have lower dimensionality than the input space. This case, where the code dimension is lower than the input dimension, is called *undercomplete*. The motivation for an undercomplete AE is to force the network to learn only the most salient features in the inputs [9]. Where the code dimension is higher than the input dimension, the autoencoder is *overcomplete*. An overcomplete encoder has high model capacity, and this can yield highly expressive representations useful for tasks other than the pure reconstruction such as classification.

With non-linear undercomplete and all overcomplete autoencoders, the network can potentially learn a trivial mapping from inputs to latent code to outputs, *e.g.,* map each training example to a number i for a one-dimensional latent code, which the decoder can equivalently map back to the training example. Though this does not usually happen in practice, due to minibatch gradient descent acting as a regulariser [2, 45], a variety of regularisation techniques exist for AE's.

These adaptations can also induce latent codes with particular useful properties. Examples include the sparse autoencoder, which induces sparse codes by adding a sparsity penalty to the code layer; the denoising autoencoder (DAE), which aims to reconstruct $\mathbf{x}$ given a corrupted, noisy version $\tilde{\mathbf{x}}$ as input; and the contractive autoencoder (CAE), which regularises the code with a norm of the Jacobian of the code with respect to the inputs, inducing invariance of the code to small changes in the inputs [9, 32, 31].

3.1.2 Ladder network architecture

The ladder network, first introduced by Valpola in [42] and extended to semi-supervised learning by Rasmus et al. in [30], builds on the autoencoder to create a powerful architecture for both supervised and semi-supervised learning. The ladder network is trained using a sum of supervised and unsupervised terms in the loss to be minimised.

The ladder network has at its core the standard encoder-decoder structure. Like the denoising autoencoder, noise is applied to the inputs, but unlike in the DAE, noise is also applied to every layer in the encoder before each non-linear activation. In addition to this *corrupted encoder*, there is also a *clean encoder*, which is a copy of the corrupted encoder but without the added noise of the corrupted path. The decoder has a symmetrical structure to the encoder path, and not only has to reconstruct the input image, but the activations in each layer of the encoder. Each layer of the decoder has an input from the corrupted encoder path, giving the corrupted version of the activations at that layer, and an input from the previous layer in the decoder. The reconstruction target for each layer is provided by the corresponding layer in the clean encoder.

This structure is illustrated in Figure 3.2 for a ladder network with $L = 2$ layers. The encoder layers are numbered $l = 0, 1, ..., L$ with $l = 0$ corresponding to the inputs,

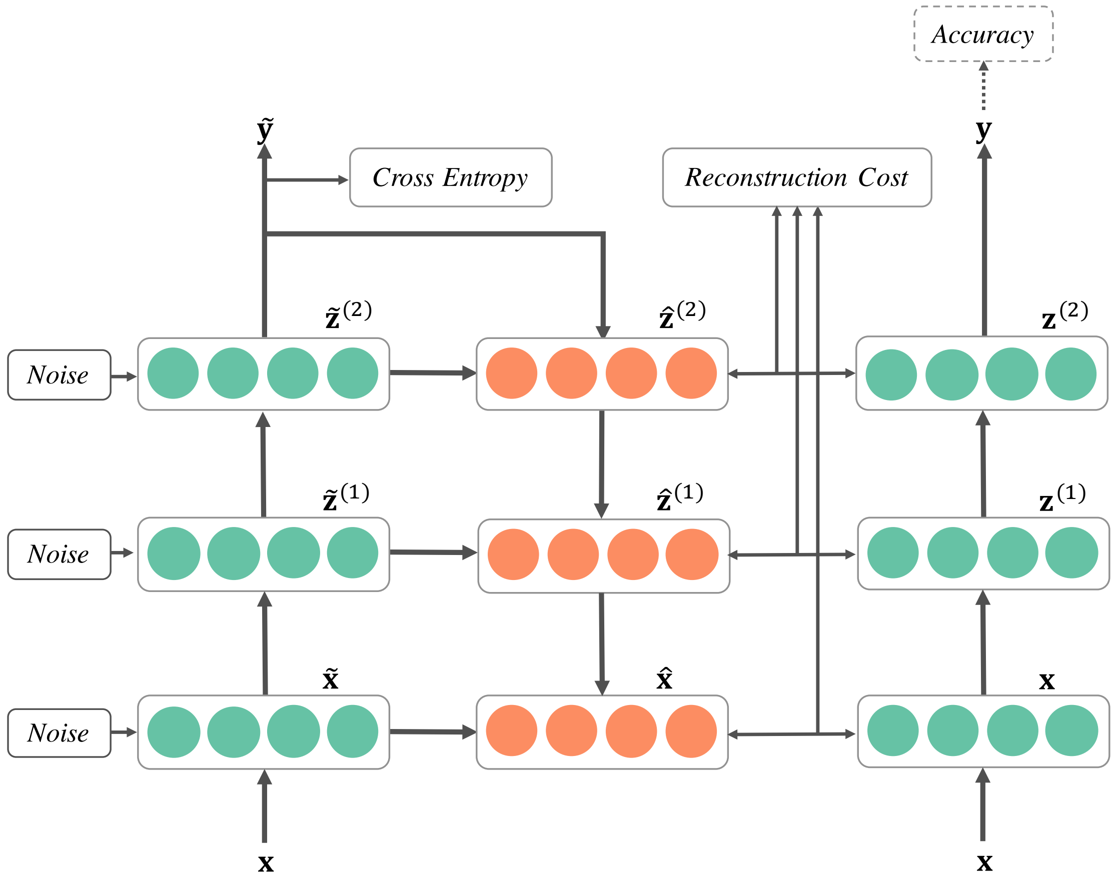


Figure 3.2: Illustration of a ladder network architecture.

Left: Corrupted encoder with activations $\tilde{\mathbf{z}}^{(l)}$, Gaussian noise $\mathcal{N}(0, \sigma^2)$ injected at each layer, and outputs $\tilde{\mathbf{y}}$, on which the cross-entropy loss is computed. **Centre:** Decoder path, where input from the layer above and from the corresponding layer in the corrupted encoder are combined with a denoising (also combinator) function $g^{(l)}(\cdot, \cdot)$ to form the reconstruction of that layer $\hat{\mathbf{z}}^{(l)}$. **Right:** Clean encoder path, the weights of which are shared with the corrupted encoder; the activations $\mathbf{z}^{(l)}$ are the targets against which the reconstruction cost at each layer is computed, and the output probabilities $\mathbf{y}$ are used for predictions.

and the decoder layers are numbered $l = L, L-1, ..., 0$ starting with $l = L$ the final output layer of the encoder. Activations at layer l in the clean encoder are denoted $\mathbf{z}^{(l)}$. The corresponding noisy activations in the corrupted encoder are denoted $\tilde{\mathbf{z}}^{(l)}$. The corresponding reconstructions, which are the activations of the decoder at l, are denoted $\hat{\mathbf{z}}^{(l)}$.

A weighted sum of the reconstruction costs at each layer comprises the unsupervised term of the training loss. The supervised loss term is computed as the cross-entropy loss between the output probabilities of the corrupted path and the true labels. The cross-entropy loss, Equation 3.2, is computed only on the labelled examples in each batch, and the reconstruction loss is computed only on the unlabelled examples. The labelled dataset is usually included without labels in the larger unlabelled pool. The total loss L which is minimised during training is thus given by:

$$L = C_c + C_d \tag{3.1}$$

$$C_c = \frac{-1}{N} \sum_{n=1}^{N} \log \Pr(\tilde{y} = t(n) | \mathbf{x}(n)) \tag{3.2}$$

$$C_d = \sum_{l=0}^{L} \frac{\lambda^{(l)}}{N m_l} \sum_{n=1}^{N} \left\| \mathbf{z}^{(l)}(n) - \hat{\mathbf{z}}_{BN}^{(l)}(n) \right\|^2, \tag{3.3}$$

where the sum over n is over unlabelled training examples, N is the total number of unlabelled examples, m_l is the number of units in layer l, and $\lambda^{(l)}$ is a hyperparameter for the weighting given to each layer's reconstruction cost.

The algorithmic procedure for computing the outputs and cost function of the ladder network, which further elucidates the architecture, is given in Algorithm 3.1, using the same notation as in Figure 3.2. Additionally examples and activations corresponding to labelled examples are subscripted with S and unlabelled examples with U.

When computing the cross-entropy loss, corrupted encoder probabilities are used to take advantage of the regularisation properties of additive Gaussian noise Section 2.3.5. Another source of stochasticity which regularises the autoencoder structure from learning a trivial code is batch normalisation. In the ladder network, batch normalisation is applied at each layer of the encoder and decoder. Batch statistics are computed on the clean encoder path and applied to corrupted encoder path. Statistics are computed separately for labelled and unlabelled data at training time on the clean encoder

Algorithm 3.1 Calculation of the output $\mathbf{y}$ and cost function C of the Ladder network

```
Require: $\mathbf{x}(n)$
  # Clean encoder (for denoising targets)
  $\mathbf{h}^{(0)} \leftarrow \mathbf{z}^{(0)} \leftarrow \mathbf{x}(n)$
  for l = 1 to L do
    $\mathbf{z}_{pre}^{(l)} \leftarrow \mathbf{W}^{(l)}\mathbf{h}^{(l-1)}$
    # BN labelled examples, save stats
    $\boldsymbol{\mu}_S^{(l)} \leftarrow \texttt{batchmean}(\mathbf{z}_{pre,S}^{(l)})$
    $\boldsymbol{\sigma}_S^{(l)} \leftarrow \texttt{batchstd}(\mathbf{z}_{pre,S}^{(l)})$
    $\mathbf{z}_{pre,S}^{(l)} \leftarrow (\mathbf{z}_{pre,S}^{(l)} - \boldsymbol{\mu}_S^{(l)})/\boldsymbol{\sigma}_S^{(l)}$
    # BN unlabelled examples
    $\boldsymbol{\mu}_U^{(l)} \leftarrow \texttt{batchmean}(\mathbf{z}_{pre,U}^{(l)})$
    $\boldsymbol{\sigma}_U^{(l)} \leftarrow \texttt{batchstd}(\mathbf{z}_{pre,U}^{(l)})$
    $\mathbf{z}_{pre,U}^{(l)} \leftarrow (\mathbf{z}_{pre,U}^{(l)} - \boldsymbol{\mu}_U^{(l)})/\boldsymbol{\sigma}_U^{(l)}$
    # Concatenate and activate together
    $\mathbf{z}^{(l)} = \texttt{concat}(\mathbf{z}_S^{(l)}, \mathbf{z}_U^{(l)})$
    $\mathbf{h}^{(l)} \leftarrow \phi(\boldsymbol{\gamma}^{(l)} \odot (\mathbf{z}^{(l)} + \boldsymbol{\beta}^{(l)}))$
  end for
  # 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
    $\tilde{\mathbf{z}}_{pre}^{(l)} \leftarrow \mathbf{W}^{(l)}\tilde{\mathbf{h}}^{(l-1)}$
    # BN labelled with clean stats
    $\tilde{\mathbf{z}}_{pre,S}^{(l)} \leftarrow (\mathbf{z}_{pre,S}^{(l)} - \boldsymbol{\mu}_S^{(l)})/\boldsymbol{\sigma}_S^{(l)}$
    # BN unlabelled with batch stats
    $\boldsymbol{\mu} \leftarrow \texttt{batchmean}(\tilde{\mathbf{z}}_{pre,U}^{(l)})$
    $\boldsymbol{\sigma} \leftarrow \texttt{batchstd}(\tilde{\mathbf{z}}_{pre,U}^{(l)})$
    $\tilde{\mathbf{z}}_{pre,U}^{(l)} \leftarrow (\mathbf{z}_{pre,U}^{(l)} - \boldsymbol{\mu})/\boldsymbol{\sigma}$
    # Concatenate and activate together
    $\tilde{\mathbf{z}}^{(l)} \leftarrow \texttt{concat}(\tilde{\mathbf{z}}_{pre,S}^{(l)}, \tilde{\mathbf{z}}_{pre,U}^{(l)}) + \texttt{noise}$
    $\tilde{\mathbf{h}}^{(l)} \leftarrow \phi(\boldsymbol{\gamma}^{(l)} \odot (\tilde{\mathbf{z}}^{(l)} + \boldsymbol{\beta}^{(l)}))$
  end for
  # Outputs of corrupted/clean encoders
  $P(\tilde{\mathbf{y}} \mid \mathbf{x}) \leftarrow \tilde{\mathbf{h}}^{(L)}$
  $P(\mathbf{y} \mid \mathbf{x}) \leftarrow \mathbf{h}^{(L)}$
  # Decoder and denoising
  for l = L to 0 do
    if l = L then
      $\mathbf{u}_{pre}^{(L)} \leftarrow \tilde{\mathbf{h}}^{(L)}$
    else
      $\mathbf{u}_{pre}^{(l)} \leftarrow \mathbf{V}^{(l+1)}\hat{\mathbf{z}}^{(l+1)}$
    end if
    $\boldsymbol{\mu} \leftarrow \texttt{batchmean}(\mathbf{u}_{pre}^{(l)})$
    $\boldsymbol{\sigma} \leftarrow \texttt{batchstd}(\mathbf{u}_{pre}^{(l)})$
    $\mathbf{u}^{(l)} \leftarrow (\mathbf{u}_{pre}^{(l)} - \boldsymbol{\mu})/\boldsymbol{\sigma}$
    # Apply combinator element-wise
    $\forall i : \hat{z}_i^{(l)} \leftarrow g(\tilde{z}_i^{(l)}, u_i^{(l)})$
    # BN element-wise with clean batch stats
    $\forall i : \hat{z}_{i,\text{BN}}^{(l)} \leftarrow (\hat{z}_i^{(l)} - \mu_{U,i}^{(l)})/\sigma_{U,i}^{(l)}$
  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 $+ \sum_{l=0}^{L} \lambda_l \left\| \mathbf{z}^{(l)} - \hat{\mathbf{z}}_{\text{BN}}^{(l)} \right\|^2$
```

path and saved for each layer. The batch statistics from the clean encoder for unlabelled data is used to normalise the reconstructions in the decoder to give the $\hat{\mathbf{z}}_{BN}^{(l)}$ terms in the denoising cost of Equation 3.3.

An important architectural choice in the ladder network is the denoising function $g^{(l)}(\cdot,\cdot)$ which for layer l takes as inputs the activations of the previous layer in the decoder, $\hat{\mathbf{z}}^{(l+1)}$, and the noisy version of the activations, $\tilde{\mathbf{z}}^{(l)}$. This is then passed through an activation function $\phi(\cdot)$ to give $\hat{\mathbf{z}}^{(l)} = \phi(\gamma^{(l)})$. The denoising function originally proposed in [30] is an approximation to the optimal denoising function for a Gaussian distribution of activations in the encoder given the activations in the layer above, *i.e.,* $\mathbf{z}^{(l)}|\mathbf{z}^{(l+1)}$ is Gaussian-distributed. It is formulated as

$$\hat{z}_i^{(l)} = g_i(\tilde{z}_i^{(l)}, u_i^{(l)}) = \left(\tilde{z}_i^{(l)} - \mu_i(u_i^{(l)})\right) \upsilon_i(u_i^{(l)}) + \mu_i(u_i^{(l)}) \tag{3.4}$$

where $u_i^{(l)}$ is the appropriate element from the batch-normalised projection from the previous layer

$$\mathbf{u}^{(l)} = \mathrm{N_B}(\mathbf{V}^{(l+1)}\hat{\mathbf{z}}^{(l+1)})$$

and the functions $\mu_i(u_i^{(l)})$ and $\upsilon_i(u_i^{(l)})$ are defined as

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

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

with $a_{1,i}^{(l)}, ..., a_{10,i}^{(l)}$ are trainable parameters initialised to zero except for a_2 and a_7, which are initialised to unity.

3.1.3 Extensions of the ladder architecture

The general principle of the ladder network is to augment a supervised neural network classifier with an auxiliary decoder to allow it to take advantage of unlabelled data [30]. The classifier, which becomes the encoder of the ladder network, is not limited to fully-connected feed-forward architectures; it has also been implemented on convolutional feed-forward classifiers [30] and is theoretically possible with recurrent neural networks.

In the classic ladder architecture, the decoder is symmetric to the encoder. For a fully-connected feed-forward encoder, the decoder has fully-connected layers with weights that are the transpose shape of the encoder weights. For a convolutional or recurrent neural network encoder, the decoder may have a more complicated architecture to implement and train. A possible solution to this is the Γ architecture[30], which uses a reconstructive cost only at the top layer ($l = L$) of the encoder.

The choice of denoising function $g^{(l)}(\tilde{\mathbf{z}}^{(l)}, \mathbf{u}^{(l)})$ need not be the rule given in Equation 3.4, which might in fact seem arbitrary though theoretically motivated (details in [30, 42]). In a deconstructive analysis of the ladder network architecture, [29] tested the effect of replacing the denoising function (which they call a *combinator* function) and found that replacing the original denoising function (the 'vanilla' combinator) with a multi-layer perceptron (MLP), *i.e.,* feed-forward network, with one or two small hidden layers (such that the total number of trainable parameters remained similar to the original combinator) outperformed the original. Their best performance was reported on the 'augmented MLP' with three inputs, $\tilde{\mathbf{z}}^{(l)}, \mathbf{u}^{(l)}$, and their element-wise product $\tilde{\mathbf{z}}^{(l)} \odot \mathbf{u}^{(l)}$, mapped to a single output for every pixel.

3.2 Virtual adversarial training

3.2.1 Adversarial perturbations

As described in Section 2.3.5, the smoothness assumption introduced in Section 2.1 and utilised by some of the methods described in Section 2.2 has been observed to be advantageous in deep supervised and semi-supervised learning: training models to be robust to random, local perturbations can improve classifier generalisation. However, a recent vein of research has revealed that models trained to be robust to random perturbations are highly vulnerable to even very small perturbations in the so-called *adversarial* direction, *i.e.,* the direction in input space to which the classifier's output probabilities are most sensitive [12].

These adversarial perturbations on inputs $\mathbf{x}$ with labels y are mathematically defined as follows [12]:

$$\mathbf{r}_{\text{adv}} := \arg\max_{\mathbf{r};||\mathbf{r}||\leq\epsilon} D\left[h(y), \Pr(y|\mathbf{x}+\mathbf{r}, \boldsymbol{\theta})\right], \tag{3.5}$$

where ϵ is a parameter dictating the size of the perturbation; $h(y)$ is the target distribution, *i.e.,* a one-hot vector of the true labels; $\Pr(y|\mathbf{x}+\mathbf{r}, \boldsymbol{\theta})$ is the output probabilities of the model with parameters $\boldsymbol{\theta}$; and $D[p, q]$ is some non-negative function that measures the 'distance' between two distributions P and Q, such as the Kullback–Leibler (KL) divergence [20]

$$D_{\mathrm{KL}}[P,Q] := \sum_x P(x)\log\frac{P(x)}{Q(x)}. \tag{3.6}$$

Adversarial perturbations for the L_q norm can be approximated as

$$r_{\mathrm{adv}} \approx \epsilon\frac{g}{\|g\|_q} \text{ where } g = \nabla_x D\left[h(y), \Pr(y|\mathbf{x}, \boldsymbol{\theta})\right]. \tag{3.7}$$

For L_∞ norm, the approximation further simplifies to the *fast gradient sign method* [12],

$$r_{\mathrm{adv}} \approx \epsilon \mathrm{sign}(g). \tag{3.8}$$

In classification problems the supervised loss is typically the cross entropy of the predictions with respect to true labels, which is related to KL–divergence by a constant additive term. In this case, by choosing $D[p,q]$ to be the cross entropy, the gradient used to compute the adversarial example g can be efficiently computed using back-propagation on the compute graph of the model.

Adversarial training refers to training a model using the adversarial loss

$$L_{\mathrm{adv}}(\mathbf{x}, y, \boldsymbol{\theta}) := D\left[h(y), \Pr(y|\mathbf{x}+\mathbf{r}_{\mathrm{adv}}, \boldsymbol{\theta})\right].$$

[12] found that adversarial training not only increased the robustness to adversarial perturbations but also increased generalisation performance.

3.2.2 Virtual adversarial perturbations

Virtual adversarial perturbations were first presented by Miyato et al. in [23] and extend adversarial perturbations to the case where there are no labels by approximating $h(y)$ with the current estimate $\Pr(y|\mathbf{x}, \hat{\boldsymbol{\theta}})$. The perturbation is defined as

$$\mathbf{r}_{\mathrm{vadv}} := \underset{\mathbf{r};\|\mathbf{r}\|_2\leq\epsilon}{\arg\max} D\left[\Pr(y|\mathbf{x}, \hat{\boldsymbol{\theta}}), \Pr(y|\mathbf{x}+\mathbf{r}.\boldsymbol{\theta})\right], \tag{3.9}$$

The virtual adversarial training loss is then defined as the average over all input data points of

$$L_{\mathrm{vadv}}(\mathbf{x}, \boldsymbol{\theta}) := D\left[\Pr(y|\mathbf{x}, \hat{\boldsymbol{\theta}}), \Pr(y|\mathbf{x}+\mathbf{r}_{\mathrm{adv}}, \boldsymbol{\theta})\right].$$

3.2.3 Approximating virtual adversarial perturbations

The computational difficulties of VAT arise not from the computation of $D[p,q]$, which is straightforward for p,q that can be approximated with exponential family distributions, but from the efficient computation of $\mathbf{r}_{\mathrm{vadv}}$. [22] proceed as follows.

For simplicity, denote $D\left[\Pr(y|\mathbf{x},\hat{\boldsymbol{\theta}}),\Pr(y|\mathbf{x}+\mathbf{r},\boldsymbol{\theta})\right]$ as $D(\mathbf{r},\mathbf{x},\boldsymbol{\theta})$ and assume $\Pr(y|\mathbf{x},\boldsymbol{\theta})$ is twice differentiable with respect to $\boldsymbol{\theta}$ and $\mathbf{x}$ almost everywhere. Since D is a metric between $\mathbf{x}$ and $\mathbf{x}+\mathbf{r}$, it is minimised for $r=0$, *i.e.,* has zero first derivative $\nabla_{\mathbf{r}}D(\mathbf{r},\mathbf{x},\boldsymbol{\theta})|_{\mathbf{r}=\mathbf{0}}$. Hence by second-order Taylor expansion:

$$D(\mathbf{r},\mathbf{x},\hat{\boldsymbol{\theta}}) \approx \frac{1}{2}\mathbf{r}^\top\mathbf{H}(\mathbf{x},\hat{\boldsymbol{\theta}})\mathbf{r},\ \text{where } \mathbf{H}(\mathbf{x},\hat{\boldsymbol{\theta}}) := \nabla\nabla_{\mathbf{r}}D(\mathbf{r},\mathbf{x},\hat{\boldsymbol{\theta}})|_{\mathbf{r}=\mathbf{0}} \tag{3.10}$$

$\mathbf{H}$ is the Hessian matrix. Under this approximation

$$\mathbf{r}_{\text{vadv}} \approx \arg\max_{\mathbf{r}} \left\{\mathbf{r}^\top\mathbf{H}\mathbf{r}; \|\mathbf{r}\|_2 \le \epsilon\right\}, \tag{3.11}$$

$\mathbf{r}^\top\mathbf{H}\mathbf{r}$ is maximised for $\mathbf{r} \parallel (\mathbf{H}\mathbf{r})$, which requires $\mathbf{r}$ to be an eigenvector of $\mathbf{H}$. Thus $\mathbf{r}_{\text{vadv}}$ lies in the direction of an eigenvector of $\mathbf{H}$, which implies $\mathbf{r}^\top\mathbf{H}\mathbf{r} = \lambda\mathbf{r}^\top\mathbf{r} = \lambda\|\mathbf{r}\|_2^2$ where λ is the eigenvalue of $\mathbf{H}$ corresponding to the eigenvector $\mathbf{r}$. Hence the approximate $\mathbf{r}_{\text{vadv}}$ is chosen to be the eigenvector with largest eigenvalue, *i.e.,* the dominant eigenvector, of $\mathbf{H}$, with magnitude ϵ.

Algorithm 3.2 Power method for approximating dominant eigenvector of a matrix $\mathbf{A}$.

```
Generate random vector $\mathbf{q}^{(0)} \in \mathbb{R}^n$.
**for** k=1 **to** K **do**
    $\mathbf{z}^{(k)} \leftarrow \mathbf{A}\mathbf{q}^{(k-1)}$
    $\mathbf{q}^{(k)} \leftarrow \mathbf{z}^{(k)}/\|\mathbf{z}^{(k)}\|$
**end for**
**return** $\mathbf{q}^{(K)}$
```

Eigenvector computation is $O(n^3)$ in the dimension of the matrix, thus in VAT the dominant eigenvector is approximated using the power iteration method given in Algorithm 3.2 [8], which relies on iterative computations of

$$\mathbf{q} \leftarrow \frac{\mathbf{H}\mathbf{q}}{\|\mathbf{H}\mathbf{q}\|_2}.$$

A further finite difference approximation is applied for efficient computation of the Hessian-vector product as:

$$\mathbf{H}\mathbf{q} \approx \frac{\nabla_{\mathbf{r}}D(\mathbf{r},\mathbf{x},\hat{\boldsymbol{\theta}})|_{\mathbf{r}=\xi\mathbf{q}} - \nabla_{\mathbf{r}}D(\mathbf{r},\mathbf{x},\hat{\boldsymbol{\theta}})|_{\mathbf{r}=\mathbf{0}}}{\xi} \tag{3.12}$$

$$= \frac{\nabla_{\mathbf{r}}D(\mathbf{r},\mathbf{x},\hat{\boldsymbol{\theta}})|_{\mathbf{r}=\xi\mathbf{q}}}{\xi} \tag{3.13}$$

[23] and [22] found by cross-validation that performance of VAT did not substantially increase when the number of iterations was increased above $K = 1$, which we also confirmed in our experiments (see Section 6.3). When the number of power iterations is $K = 1$, this gives an expression for VAT similar to Equation 3.7 for AT:

$$\mathbf{r}_{\text{vadv}} = \epsilon \frac{\mathbf{g}}{\|\mathbf{g}\|_2} \text{ where } \mathbf{g} = \nabla_{\mathbf{r}}\, D\left[\Pr(y|\mathbf{x}, \hat{\boldsymbol{\theta}}), \Pr(y|\mathbf{x}+\mathbf{r}, \boldsymbol{\theta})\right]\Big|_{\mathbf{r}=\xi\mathbf{q}} \tag{3.14}$$

3.3 Ladder, VAT, and the manifold hypothesis

Both the ladder network and virtual adversarial training tackle semi-supervised learning by minimising a penalty term computed on unsupervised examples which induces robustness to small perturbations. In the ladder, these perturbations are Gaussian and applied at every layer of the encoder, increasing its performance above the denoising autoencoder, and the penalty is a squared loss term. In VAT, the perturbations are virtual adversarial (and therefore anisotropic) and applied only at the input level.

The derivation in Section 3.2 for the approximation to $\mathbf{r}_{\text{vadv}}$ illustrates that finding the direction $\mathbf{q}$ to move from the current data point to maximise the KL-divergence of the distributions $\Pr(y|\mathbf{x}+\mathbf{q})$ and $\Pr(y|\mathbf{x})$ is equivalent to finding the principal eigenvector of the Hessian of the network outputs (predictive probabilities for each of the classes) with respect to the input images. This suggests possible extensions of VAT that smooth the curvature of latent space in other ways, *e.g.,* by regularisation more than just the dominant eigenvector of the Hessian. Such approaches would however be limited by the computational cost of calculating the Hessian, which has been avoided here by use of the finite difference method for the Hessian-vector product. However, the anisotropic, directed smoothing carried out by VAT could be considered a strength which can be applied in conjunction with more isotropic smoothing methods.

Autoencoders rely on the manifold assumption and aim to learn the structure of the manifold on which the data lie. A regularised AE, in learning the salient features for the reconstruction task, learns the directions of variation necessary to reconstruct examples drawn from the inputs' distribution [9].

The encoder in the ladder network combines learning a latent code with outputting a probability distribution over labels. Its weights must be trained in such a way as to learn features relevant to both the classification and the reconstruction task. We hypothesise that the ladder network must firstly learn the directions necessary for reconstruction, *i.e.,* directions of variation in natural images, essentially finding a manifold representing the subspace of natural images within the higher dimensional latent spaces of its layers; and secondly, it must learn within that subspace the manifolds in which variations do not affect the classification outcome. The strength of the ladder network may come from the end-to-end learning of both of these tasks at once. Where embedding-based methods in deep SSL and change of representation methods in classical SSL use a two step process of first projecting to a feature space and then classifying in that space, the ladder network optimises for both objectives with every parameter update. In addition to this, the training signal for the ladder network includes the denoising cost at every intermediate activation, which pushes the two requirements stated above onto every layer.

The reconstructive architecture of the ladder network maps out a manifold for natural images in the space of each of its layers. Adversarial perturbations, by identifying directions to which class probabilities are most sensitive, can effectively push or pull apart classes in latent space; virtual adversarial perturbations can approximate this behaviour and make use of unlabelled data. This leads us to believe that applying virtual adversarial training to ladder networks could improve performance on supervised tasks after training in a semi-supervised manner, and motivates the models we propose in the next chapter.

3.4 Practical considerations

The ladder network and virtual adversarial training are appealing as relatively straightforward but powerful methods of applying semi-supervised learning to pre-existing classifier neural networks.

In [30], Rasmus et al. used a decoder for the full ladder network that mirrors the encoder structure while noting that there is no explicit need for the decoder to match the encoder in structure or for every activation layer to be reconstructed. However, it was observed that reconstruction only at the top level (Γ model) or at the bottom and top layers only did not perform as well as the full ladder where the decoder mirrors every layer in the encoder. Thus to implement the full ladder on a new encoder, one must implement a corresponding decoder. This can be quite complicated for models such as convolutional neural networks, which have layers like max- or average-pooling, the ‘reverse’ of which can be hard to choose unambiguously.

Implementing a symmetric decoder structure also involves doubling the number of trainable weights in the network. While a pre-trained classifier can be used as an encoder, the decoder weights must be fully trained from initialisation. Using the reconstruction cost also requires hyperparameter optimisation over the weights applied to each layer’s denoising cost, as well as a number of other hyperparameters associated with the ladder such as the width of the additive Gaussian noise in the corrupted encoder.

By contrast virtual adversarial training does not increase the number of trainable parameters. It further only has two hyperparameters α and ϵ, and [22] showed that for small ϵ, α has very little effect, recommending that only ϵ be tuned. VAT thus is very straightforward to implement on new models.

Chapter 4

Virtual Adversarial Ladder Networks

4.1 Motivation

We believe that virtual adversarial training can improve the semi-supervised classification performance of the ladder network by enhancing the semi-supervised separation of classes in the ladder network's learned latent spaces. Thus we propose four models that test the effect of adding either a virtual adversarial cost or a virtual adversarial perturbation, either with respect to the input layer or with respect to every activation layer in the encoder.

The four models we propose in this section have ladder network architectures with virtual adversarial training (VAT) incorporated into the corrupted encoder path. Two models, ladder with virtual adversarial cost (LVAC) and ladder with layer-wise virtual adversarial cost (LVAC-LW), apply VAT as originally proposed in [23] and described in Section 3.2. The other two models, ladder with virtual adversarial noise (LVAN) and ladder with layer-wise virtual adversarial noise (LVAN-LW), attempt to boost the performance of the ladder by making the denoising task more informative through the incorporation of virtual adversarial perturbations (VAP). Using virtual adversarial noise instead of virtual adversarial cost reduces the number of pairs of forward and backward passes required at training time from three to two as the explicit VAT cost does not have to be computed.

Other than the additions to corrupted encoder to incorporate VAT, we have used the original configurations of the ladder network in [30]. However, the combinator used (Equation 3.4) assumes Gaussianity of noise and of activations in the encoder path conditioned on the layer above. Ideally we would have used a more generic combinator function such as the AMLP tested by [29], but tests of this on our implementation of the ladder found it to be less stable than the original combinator function.

All models retain the same decoder and clean encoder structure as the ladder network (Figure 3.2), so only the corrupted encoder path are shown in the conceptual illustrations of the models in Figures 4.1a–4.1d. Incorporating virtual adversarial training into the ladder does not increase the number of parameters, so all of our models have the same memory footprint as the base ladder network.

All models were trained with losses including the supervised term C_c and the reconstruction/denoising term C_d previously stated in Equations 3.2 and 3.3 and repeated below:

$$C_c = \frac{-1}{N} \sum_{n=1}^{N} \log \Pr(\tilde{y} = t(n) | \mathbf{x}(n))$$

$$C_d = \sum_{l=0}^{L} \frac{\lambda_l}{N m_l} \sum_{n=1}^{N} \left\| \mathbf{z}^{(l)}(n) - \hat{\mathbf{z}}^{(l)}_{BN}(n) \right\|^2 .$$

All virtual adversarial perturbations are computed with L_2 norm and KL–divergence $D_{KL}[P, Q] := \sum_x P(x) \log P(x) - \sum_x P(x) \log Q(x)$ as the distance metric $D[P, Q]$.

4.2 Ladder with virtual adversarial cost (LVAC)

In addition to the supervised cross-entropy cost and unsupervised activation reconstruction cost, a virtual adversarial cost of the predictions with respect to the input images is added to the training loss of a ladder network. The model is conceptually illustrated in Figure 4.1a.

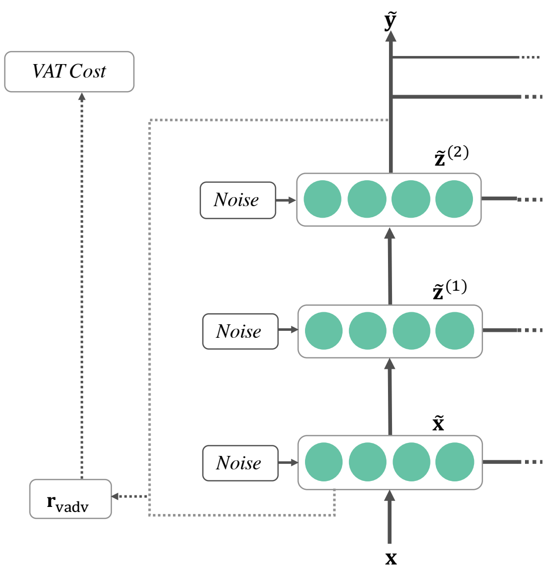
Using the notation of Section 3.2 and $\tilde{\mathrm{x}}, \tilde{y}, \boldsymbol{\theta}$ to denote the input, output, and parameters of the corrupted encoder of the ladder, the virtual adversarial cost for LVAC can be expressed as

$$C_{\text{vadv}} = \alpha D_{KL} \left[\Pr(\tilde{y} | \tilde{\mathbf{x}}, \boldsymbol{\theta}), \Pr(\tilde{y} | \tilde{\mathbf{x}} + \mathbf{r}_{\text{vadv}}, \boldsymbol{\theta}) \right] \tag{4.1}$$

$$\mathbf{r}_{\text{vadv}} = \arg\max_{\mathbf{r}} \left\{ D_{KL} \left[\Pr(\tilde{y} | \tilde{\mathbf{x}}, \boldsymbol{\theta}), \Pr(\tilde{y} | \tilde{\mathbf{x}} + \mathbf{r}, \boldsymbol{\theta}) \right] ; \; \|\mathbf{r}\|_2 \leq \epsilon \right\} . \tag{4.2}$$

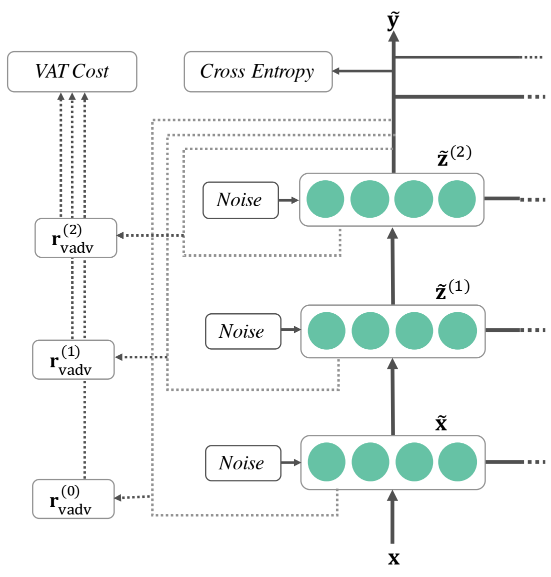
This most closely resembles vanilla virtual adversarial training applied to the ladder network. In addition to the reconstruction cost weight hyperparameters of the ladder network, LVAC has the α and ϵ parameters of VAT, where α is the weighting of the

Figure 4.1: Conceptual illustrations of our proposed models. All show the corrupted encoder path only; the decoder and clean encoder, which are not shown, are identical to that of the standard ladder shown in Figure 3.2.



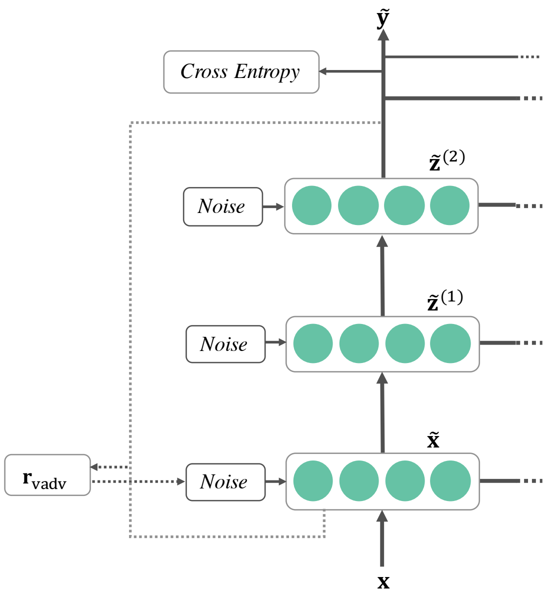
VAT Cost
$\tilde{\mathbf{y}}$
$\tilde{\mathbf{z}}^{(2)}$
Noise
$\tilde{\mathbf{z}}^{(1)}$
Noise
$\tilde{\mathbf{x}}$
Noise
$\mathbf{r}_{\text{vadv}}$
$\mathbf{x}$

(a) LVAC

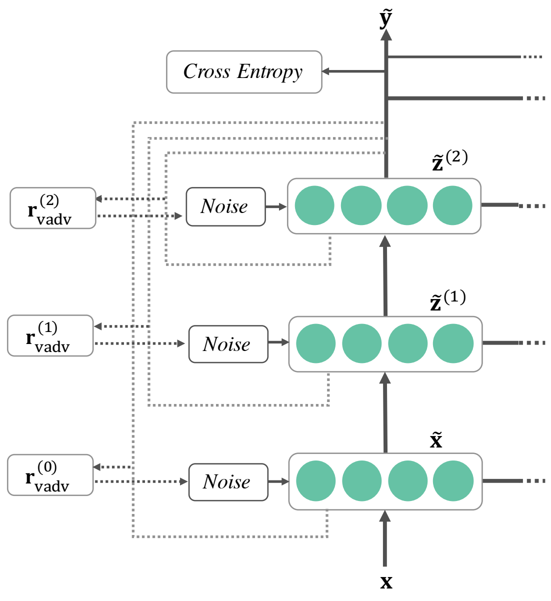


VAT Cost
Cross Entropy
$\tilde{\mathbf{y}}$
$\tilde{\mathbf{z}}^{(2)}$
Noise
$\mathbf{r}^{(2)}_{\text{vadv}}$
$\tilde{\mathbf{z}}^{(1)}$
Noise
$\mathbf{r}^{(1)}_{\text{vadv}}$
$\tilde{\mathbf{x}}$
Noise
$\mathbf{r}^{(0)}_{\text{vadv}}$
$\mathbf{x}$

(b) LVAC-LW



(c) LVAN



Cross Entropy
$\tilde{\mathbf{y}}$
$\tilde{\mathbf{z}}^{(2)}$
$\mathbf{r}^{(2)}_{\text{vadv}}$
Noise
$\tilde{\mathbf{z}}^{(1)}$
$\mathbf{r}^{(1)}_{\text{vadv}}$
Noise
$\tilde{\mathbf{x}}$
$\mathbf{r}^{(0)}_{\text{vadv}}$
Noise
$\mathbf{x}$

(d) LVAN-LW

VAT cost relative to the supervised cost and ϵ is the maximal magnitude of the virtual adversarial perturbation.

4.3 Ladder with virtual adversarial cost, layer-wise (LVAC-LW)

As with LVAC, a virtual adversarial cost is included in the training loss, but this is computed for the predictions with respect to each layer of activations in the encoder of the ladder network. The structure is illustrated in Figure 4.1b. The VAT loss can be written

$$C_{\text{vadv}} = \sum_l \alpha^{(l)} D_{KL}\left[\Pr(\tilde{y}|\tilde{\mathbf{z}}^{(l)}, \boldsymbol{\theta}), \Pr(\tilde{y}|\tilde{\mathbf{z}}^{(l)} + \mathbf{r}^{(l)}_{\text{vadv}}, \boldsymbol{\theta})\right], \tag{4.3}$$

$$\mathbf{r}^{(l)}_{\text{vadv}} = \arg\max_{\mathbf{r}} \left\{ D_{KL}\left[\Pr(\tilde{y}|\tilde{\mathbf{z}}^{(l)}, \boldsymbol{\theta}), \Pr(\tilde{y}|\tilde{\mathbf{z}}^{(l)} + \mathbf{r}, \boldsymbol{\theta})\right] ; \|\mathbf{r}\|_2 \leq \epsilon^{(l)} \right\} \tag{4.4}$$

where $\tilde{\mathbf{z}}^{(l)}$ is the activation in layer l of the corrupted encoder path, with $\tilde{\mathbf{z}}^{(0)} = \tilde{\mathbf{x}}$.

In LVAC-LW we compute a virtual adversarial perturbation for each layer, thus there is a tunable magnitude $\epsilon^{(l)}$ and weighting $\alpha^{(l)}$ at each layer l.

4.4 Ladder with virtual adversarial noise (LVAN)

In addition to the Gaussian noise added to the activations of the corrupted encoder at each layer, a virtual adversarial perturbation of the same form as used in LVAC (Equation 4.2) is computed for and added to each input image. The structure is shown in Figure 4.1c.

The reconstruction cost for the lowest level of the network is a squared loss corresponding to denoising an image perturbed with both isotropic Gaussian noise $\mathcal{N}(0, \sigma^2)$ *and* a virtual adversarial perturbation (VAP). There is no explicit VAT cost term in the optimisation objective. LVAN adds one additional hyperparameter to the ladder, ϵ determining the magnitude of the VAP.

4.5 Ladder with virtual adversarial noise, layer-wise (LVAN-LW)

As with LVAN, there is no explicit VAT cost term. The noise added on the corrupted encoder path at each layer is an addition of a Gaussian and a VAP of the predictions with respect to the activations at that layer. This structure is illustrated in Figure 4.1d. The VAP added at each layer l is of the same form as the perturbations for LVAC-LW, given in Equation 4.4.

As with LVAC-LW, there is a magnitude parameter $\epsilon^{(l)}$ for the virtual adversarial perturbations generated at each layer l, but as it has no explicit VAT cost term, like LVAN it has no associated α parameters.

Chapter 5

Methodology

5.1 Data

We carried out our experiments on the MNIST dataset of handwritten digits [18]. There are 60,000 examples in the training set and a test set of 10,000 examples. The digits are size-normalised and centred in the fixed-size image of 28×28 pixels with one greyscale channel. The MNIST dataset was used as almost all recent semi-supervised learning methods have been benchmarked on the dataset using 100 and 1000 labels. It is also lightweight and relatively quick to train on compared to larger datasets such as SVHN or CIFAR-10, thus it made sense to use this for the scope of this time- and resource-limited project. The images are passed to models as 784–dimensional vectors. Labels are one-hot encoded vectors with dimensionality equal to the number of classes (10).

As is standard for semi-supervised learning, our experiments are averaged over different selections of labelled examples. The labelled dataset in all investigations is class–balanced. The random seeds generating the partitionings are fixed across experiments to ensure that comparisons are fair.

5.2 Model implementation

Our models were constructed using the Tensorflow library and its Python API [1].

5.2.1 Ladder network implementation

We have made some small changes to the architecture in [30]: we have used leaky rectified linear units (LReLU) instead of rectified linear units (ReLU) and a slightly modififed averaging scheme for batch normalisation statistics. The combinator function used was the original denoising function used in [30]. We also implemented the MLP combinator introduced in [29], but the models using this combinator failed to converge.

Our TensorFlow implementation is modelled on the Theano source code[1] accompanying [30].

5.2.2 VAT implementation

Our implementation is modelled on that of [22], using as reference the accompanying Theano code[2] and reusing much of the publicly available TensorFlow code[3].

The underlying model which was trained using VAT in [22] was a fully-connected feedforward network with hidden layers of size 1200, 600, 300 and 150. The baseline model which we trained and for which we have reported results used the same encoder structure as was used for the ladder and all of our proposed models (see next).

5.2.3 Our models

The base architecture used for the encoders in our models consisted of five hidden layers with 1000, 500, 250, 250, 250 units respectively. The same code is used for these models as for our ladder network and VAT implementations, allowing direct comparison between the results obtained with our models and with these baseline models. Any discrepancy between the results seen in replicating benchmarks with our implementations of the VAT and ladder will also be reflected in our proposed models.

5.3 Model training

As time and resources for hyperparameter optimisation were limited, we chose a training schedule for all our models by compromising between the settings used in [30] for the ladder and [22] for VAT. Both papers used initial learning rates 0.002 with the Adam optimiser, but [22] used lower β_1, β_2 values than the defaults used by [30]. In [22], models were trained for 400 epochs with linear decay from 200 epochs; whereas [30] trained models for 150 epochs with linear decay after 100.

We thus used the Adam optimiser with 0.002 learning rate for 200 epochs, followed by 50 epochs with linearly decaying learning rate. $\beta_1 = 0.9, \beta_2 = 0.999$, the values used in [30], were used. All models were trained for 200 epochs with 0.002

[1]`http://github.com/CuriousAI/ladder/`

[2]`http://github.com/takerum/vat`

[3]`http://github.com/takerum/vat_tf`

learning rate, followed by 50 epochs where the learning rate was linearly decayed. All weights were initialised randomly using the Glorot initialisation scheme [7].

5.4 Hyperparameter optimisation

Given limited time and resources, the possible hyperparameter space had to be sparsely explored.

As detailed in Section 5.3, hyperparameters relating to the learning schedule, such as initial learning rate, learning rate decay schedule, and Adam β_1, β_2 coefficients, were fixed.

The hyperparameters which were optimised over for the ladder network in [30] and [29] were the standard deviation σ of the noise in the encoder, and the reconstruction weights $\lambda^{(l)}$ for each layer. Both papers used the same network structure as we did with seven total layers (one input, five hidden, one output) and hence seven reconstruction weights, but only tuned three values: $\lambda^{(0)}, \lambda^{(1)}$ and $\lambda^{(l\geq 2)}$.

For VAT, [22] showed that since ϵ gives the size of the perturbation, it was often sufficient to fix $\alpha = 1$ and tune only ϵ.

For our models, we carried out very approximate hyperparameter optimisation. We fixed $\sigma = 0.3$ and $\alpha = 1$ for all models. For LVAC and LVAN, we optimised $\lambda^{(0)}, \lambda^{(1)}, \lambda^{(\geq 2)},$ and ϵ. For the layer-wise models LVAC-LW and LVAN-LW, we optimised $\lambda^{(0)}, \lambda^{(1)}, \lambda^{(\geq 2)}, \epsilon^{(0)}, \epsilon^{(1)},$ and $\epsilon^{(\geq 2)}$.

As a grid search would have been prohibitively expensive time-wise, we used Bayesian optimisation as implemented in the scikit-optimize library [37]. This method uses Gaussian process regression of values of the objective function to estimate the next best point to evaluate the objective function, taking into account user–specified priors on the tunable parameters.

The objective function for our optimisation was the validation error on a fixed validation set of 1000 examples sampled from the training set, after 25 epochs of training (10% of the total training time used for the full models). We used only two iterations of the Gaussian process optimisation routine, which leaves significant capacity for improvement of the hyperparameter optimisation of our models.

The results of the optimisations which we used when training our full models for the experiments in Chapter 6 are given in Appendix A.

Chapter 6

Experimental results and discussion

6.1 Performance on MNIST

6.1.1 Benchmarks

Table 6.1 compares average error rate achieved on MNIST using 100, 1000 and all 60,000 labels for the most notable models mentioned above. We also include baseline results obtained by [30] using a feedforward neural network with batch normalisation and Gaussian noise regularisation, trained using supervised cost only. This baseline error rate for 100 labels is 21.74% $\pm$ 1.77, an order of magnitude larger than the error rates achieved by the methods that use unlabelled data. By comparison, the best performance in the fully supervised setting using all 60,000 labels is currently 0.21%, as achieved by Li et al. with DropConnect networks [43].

In this work we have applied our methods to the MNIST dataset only, but a number of recent papers, including [17] and [34], have not benchmarked performance on MNIST, instead reporting performance on the SVHN [25] and/or CIFAR-10 [16] datasets.

6.1.2 Our models

The results discussed in this section are summarised in Table 6.2.

We first investigated the performance of our proposed models on the standard MNIST benchmarks for semi-supervised learning. 100 and 1000 labels on MNIST are the standard benchmarks for semi-supervised learning methods.

For all experiments the unlabelled dataset was the entire training set of 60,000 examples. For each experiment we tested our proposed models on, we also benchmarked

Table 6.1: Benchmark average error rate (AER) on permutation-invariant MNIST.

	100 labels		1000 labels		All labels	
Model	%AER	SE	%AER	SE	%AER	SE
Feat match GAN [35]	0.93	± 0.065	–	–	–	–
ADGM [19]	0.96	± 0.02	–	–	–	–
Ladder$*$ [29]	1.002	± 0.038	0.979	± 0.025	0.578	± 0.013
Ladder [30]	1.06	± 0.37	0.84	± 0.08	0.57	± 0.02
SDGM [19]	1.32	± 0.07	–	–	–	–
VAT$**$ [22]	1.36	–	1.27	–	0.64	–
Adversarial AE [**?**]	1.90	± 0.10	1.60	± 0.08	0.85	± 0.02
CatGAN [39]	1.91	± 0.1	1.73	± 0.08	0.91	–
VAT [23]	2.33	–	1.36	–	0.637	± 0.046
DGM M1+M2 [15]	3.33	± 0.14	2.40	± 0.02	–	–
MLP baseline [30]	21.74	± 1.77	5.70	± 0.20	0.80	± 0.03

$*$ The deconstructive analysis of [29] replaced the combinator function in the original ladder network with an ‘augmented multilayer perceptron’ (MLP) and achieved slightly better performance.
$**$ VAT was introduced in [23] and extended by the same authors in [22], where better performance was achieved with stabilising additive Gaussian noise and a deeper network architecture.

our implementations of the ladder and VAT which our proposed models were built on. Our implementations of the ladder and VAT performed slightly worse than the published results, and we believe these are due to the differences in our implementation detailed in Section 5.2.

The performance metric used on MNIST is % average error rate (AER). We computed standard errors on each AER as the sample standard deviation over 5 training runs with different but fixed random seeds for the partitioning of the data and the initialisation of model weights.

For 1000 labelled examples, our implementation of the ladder model and VAT performed better than our proposed models, with 1.10 ± 0.05 AER for the ladder and 1.11 ± 0.05 for VAT. The best of our proposed models was LVAC-LW with 1.48 ±

Table 6.2: % Average Error Rate (AER) of our proposed models on MNIST with 50, 100 and 1000 labelled examples. Mean and standard deviation for 50 labels is computed across ten training runs with different seeds fixed between models; mean and standard deviations on 100 and 1000 labels are computed over five training runs.

	50 labels		**100 labels**		**1000 labels**	
Model	AER (%)	SE	AER (%)	SE	AER (%)	SE
VAT (ours)	5.38	± 2.92	2.14	± 0.64	1.11	± 0.05
Ladder (ours)	1.86	± 0.43	1.45	± 0.36	**1.10**	± **0.05**
LVAC	2.42	± 1.05	1.65	± 0.12	1.28	± 0.07
LVAC-LW	4.08	± 3.55	1.39	± 0.06	1.11	± 0.12
LVAN	1.52	± 0.20	1.30	± 0.09	1.48	± 0.03
LVAN-LW	**1.42**	± **0.12**	**1.25**	± **0.06**	1.51	± 0.06

0.12. Lower variance was found for LVAN, which had AER 1.51 ± 0.06.

For 100 labelled examples, the best-performing model was LVAN-LW with 1.25 ± 0.06. All of our models and our implementation of the ladder network outperformed VAT both in AER and in variance. LVAN-LW, LVAN, and LVAC-LW all outperformed our implementation of the ladder network (1.45 ± 0.36), though the magnitude of the variance on the ladder network means that we cannot claim with certainty that our models performed better. Remarkably, while the variance on the ladder AER increases significantly as number of labels is decreased from 1000 to 100, the variance on our models changes very little by comparison. This effect is particularly notable for LVAN-LW.

To investigate if stability with fewer labels was a systematic property of our proposed models, we decided to test performance after training on 50 labels only, *i.e.,* five examples for each class. In our review of the literature, we only found testing with 50 labelled examples in [30] on the ladder network, which achieved 1.62 ± 0.65 AER. Our implementation of the ladder achieved 1.86 ± 0.43, which was outperformed by both LVAN (1.52 ± 0.20) and LVAN-LW (1.42 ± 0.12). The variance on these models, while larger than for the 100 label case, are still very small compared to the ladder network or VAT, which is extremely unstable (5.38 ± 2.92) in the 50 label case. The LVAC

Table 6.3: Training time per epoch in seconds, ranked by median time taken.

	Time per Epoch (s)		
Model	Median	Mean	SE
VAT	10.25	28.81	± 23.00
Ladder	16.00	16.01	± 0.35
LVAN	36.75	38.66	± 8.00
LVAN-LW	42.50	46.36	± 8.80
LVAC	52.00	45.25	± 9.00
LVAC-LW	65.25	65.80	± 22.00

and LVAC-LW models also have large standard errors and high mean AER for these models. This suggests that the VAT cost term is very unstable for very few labelled examples, but the addition of VA perturbations still assists with training in these cases.

6.1.2.1 Comparisons of training time

We also compared the time taken per epoch for our models to train. Results are shown in Table 6.3. The ladder network is slightly slower than VAT despite VAT requiring three pairs of forward- and backpropagation whereas the ladder only requires one; we believe this is due to the ladder encoder-decoder structure having effectively double the depth of the network used for VAT, which corresponds just to the encoder. The ladder network also needs to make a second forward pass through the clean encoder to generate the reconstruction targets, though as these weights are shared with the corrupted encoder, backpropagation does not have to be carried out separately. We thus expect the ladder to take between two and three times longer than VAT at least. However we also find that epochs of VAT can vary substantially in their length, with the mean and median length differing significantly. This is not seen to such an extent in our proposed models although they incorporate elements of VAT.

As expected, the ladder variants with virtual adversarial noise are faster to train than with virtual adversarial cost as an additional forward pass is required in the LVAC models to compute the KL–divergence in the VAT cost. The LVAN models take at least double the time of the ladder, which again we expect as a second pair of forward- and backpropagation is needed for computation of the virtual adversarial perturbation.

6.2 Defending against adversarial attacks

One interpretation of adversarial attacks is taking advantage of a lack of smoothness in latent space, which allows small perturbations of the input to cause dramatic changes in final predictions [41]. The similarity of virtual adversarial training to adversarial training suggests that models trained using VAT could be more robust than otherwise to adversarial attacks. To test this hypothesis, we measured average error rates of each of our models on adversarial examples generated using the fast gradient method [12] with L_1, L_2, L_∞ norms and fixed $\epsilon = 0.3$. The examples were generated using the CleverHans library [28].

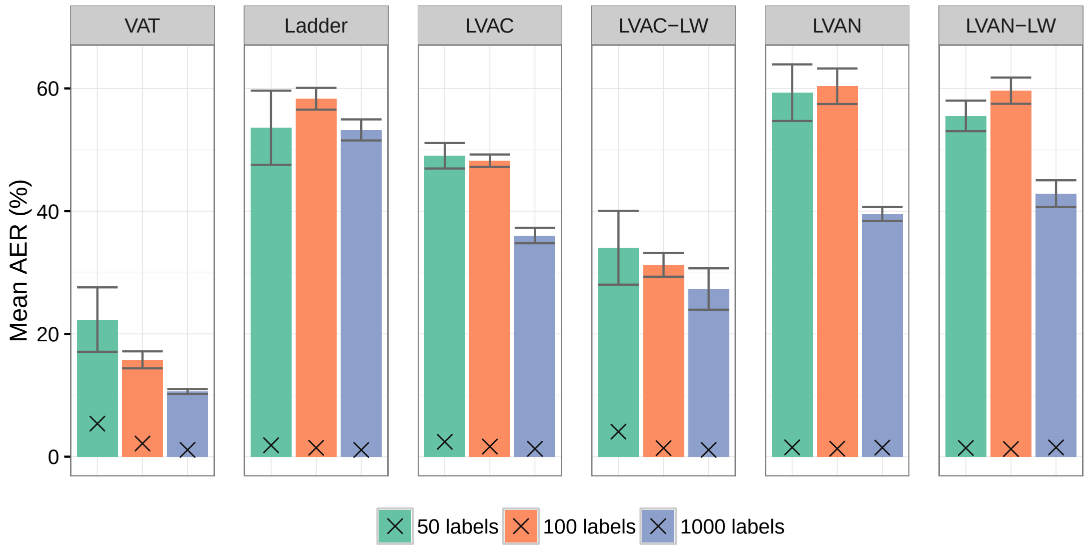
For each model (VAT, Ladder network, LVAC, LVAC-LW, LVAN, LVAN-LW), we had five fully trained models for each of the 50-, 100-, and 1000-label training cases. Each of these models was tested on adversarial example sets. The mean AERs on adversarial examples generated with the L_∞, L_1 and L_2 norms are reported in Tables 6.4, 6.5, and 6.6 respectively.

For the L_∞ norm, where the examples are generated using Equation 3.8, all models showed a significant difference between their performance on the regular MNIST test set and their poorer performance on the adversarial examples, as shown in Figure 6.1. VAT for all sizes of labelled training set outperformed all other models. LVAC and LVAC-LW, which are both trained with explicit VAT cost terms, did better than the ladder network, which has no virtual adversarial component, and the LVAN and LVAN-LW. Although LVAN and LVAN-LW incorporate virtual adversarial perturbations, there is no penalty for robustness of *classification* in their training losses.

For the L_1 and L_2 norm attacks, LVAN-LW had the lowest error rate in the 50- and 100-label cases, and LVAC-LW the lowest on the 1000-label case. It is surprising that both LVAN and LVAN-LW outperformed the LVAC and LVAC-LW models as they are not explicitly penalised with a virtual adversarial cost; it may be that the models themselves are simply more stable, as we can see in the marginally smaller standard errors associated with the normal test set errors for LVAN and LVAN-LW relative to LVAC and LVAC-LW (Table 6.2).

Our proposed models all significantly outperformed VAT. VAT performed poorly

Figure 6.1: Average Error Rate (%) on adversarial attacks with L_∞ norm. Bars show for each model (subplot) and each size of labelled dataset (colour) the average error rate (AER) for adversarial examples generated with L_∞ norm. Crosses indicate AER on the normal (non–adversarial) test set.



in the 50-label case for both norms, which could be due to the high variance and instability of the base VAT models trained on 50 labels, since even their errors on the regular test set are quite high. VAT also maintained the highest error rates for the 100- and 1000-label cases, though these were much closer to the corresponding error rates on the regular test set. The ladder network performed very poorly in the 50-label case, especially for L_2, but does similarly to our VAT-augmented models in the 100- and 1000-label cases.

6.3 Anisotropy of smoothing

[23] and [22] found that the magnitude of the virtual adversarial cost did not significantly change with the number of power iterations.

However, the number of power iterations determines the quality of the approximation of the virtual adversarial perturbation of the dominant eigenvector of the Hessian (as described in Section 3.2). We also know that isotropic noise, *e.g.,* from random perturbations, has a regularising effect, and the initialising vector in the power method is drawn from a unit random normal in each dimension. Thus we hypothesise that for

Table 6.4: Average error rate on adversarial examples with L_∞ norm.

	50 labels		**100 labels**		**1000 labels**	
Model	AER (%)	SE	AER (%)	SE	AER (%)	SE
VAT	**22.34**	± **5.25**	**15.78**	± **1.38**	**10.62**	± **0.40**
Ladder	53.59	± 6.04	58.31	± 1.77	53.24	± 1.71
LVAC	49.03	± 2.07	48.22	± 1.01	36.03	± 1.26
LVAC-LW	34.04	± 6.01	31.26	± 1.94	27.32	± 3.37
LVAN	59.30	± 4.62	60.35	± 2.90	39.53	± 1.13
LVAN-LW	55.52	± 2.49	59.63	± 2.14	42.85	± 2.18

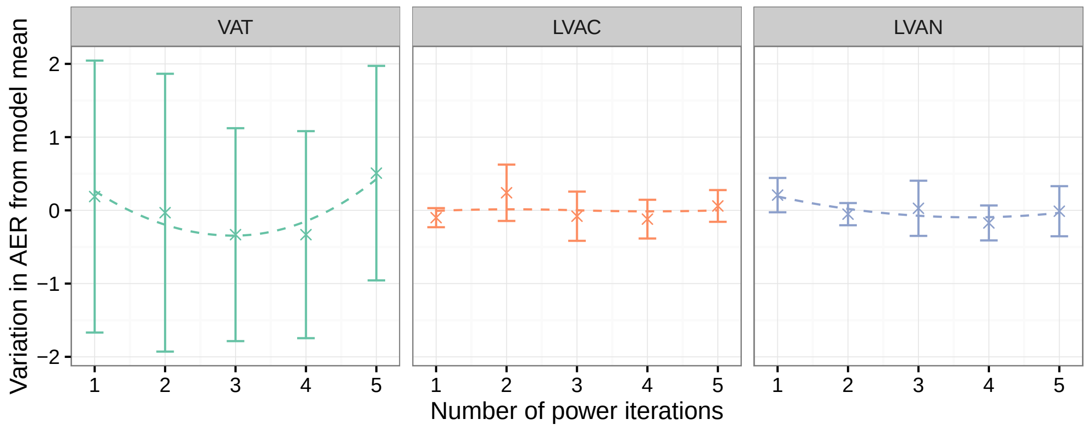
Table 6.5: Average error rate on adversarial examples with L_1 norm.

	50 labels		**100 labels**		**1000 labels**	
Model	AER (%)	SE	AER (%)	SE	AER (%)	SE
VAT	9.52	± 7.59	2.99	± 1.59	2.41	± 0.21
Ladder	25.79	± 5.39	1.55	± 0.44	1.52	± 0.10
LVAC	2.88	± 1.79	1.72	± 0.12	1.38	± 0.07
LVAC-LW	4.91	± 3.74	1.44	± 0.06	**1.18**	± **0.15**
LVAN	1.83	± 0.36	1.38	± 0.08	1.60	± 0.13
LVAN-LW	**1.60**	± **0.24**	**1.33**	± **0.09**	1.66	± 0.10

Table 6.6: Average error rate on adversarial examples with L_2 norm.

	50 labels		**100 labels**		**1000 labels**	
Model	AER (%)	SE	AER (%)	SE	AER (%)	SE
VAT	9.91	± 7.54	3.47	± 1.68	2.58	± 0.23
Ladder	68.60	± 6.51	2.31	± 0.41	2.31	± 0.22
LVAC	3.33	± 1.82	2.16	± 0.14	**1.78**	± **0.17**
LVAC-LW	4.05	± 3.70	1.93	± 0.11	1.58	± 0.09
LVAN	2.46	± 0.39	1.95	± 0.22	2.20	± 0.05
LVAN-LW	**2.41**	± **0.30**	**1.89**	± **0.16**	2.28	± 0.20

Figure 6.2: Effect of number of power iterations on average error rate. Average error rate after 25 training epochs for VAT, LVAC and LVAN models, as a function of number of power iterations, K, for computation of virtual adversarial perturbations. AER has been centred across K for each model. Regression lines using local polynomial regression are shown to illustrate overall trends.



models that already make use of isotropic noise as regularisation such as the ladder, and particularly the LVAN model which adds the VA perturbation to Gaussian noise, a better approximation to the VA direction as obtained by more power iterations will have a stronger regularising effect from anisotropic smoothing in the adversarial direction; *i.e.,* more of the performance gain will be due to smoothing in the adversarial direction as opposed to from isotropic smoothing as would be achieved by random perturbations.

To test this hypothesis we investigated the effect of the number of power iterations K on the average error rate of VAT, LVAC and LVAN models. For each K, the models were trained five times using five different seeds fixed across models and K. Due to the lengthy training time required and many runs ($5 \times 3 = 15$), each model was only trained for 25 epochs, which is before convergence but was judged to be a reasonable indicator of final performance.

As shown in Figure 6.2, the results of these experiments suggest that the number of power iterations does not have a significant effect on the average error rate. There was high variance between runs, particularly for VAT, which make overall trends difficult to discern. Contrary to the findings of [22], it appears that for VAT, there may be a small reduction in AER for $K = 3$ relative to the $K = 1$ which was used. For LVAN, it is possible that the AER decreases with increasing K, though this trend may lie within the random variation between runs. To further investigate the hypotheses stated above, it would be necessary to carry out these experiments again with longer training times for each run to ensure convergence, and more runs to minimise the variance on the mean.

It should also be noted that the implementation of VAT used for these experiments included the addition of Gaussian noise at each layer of the feedforward network to stabilise training; without this noise the effect of increasing K may be easier to discern, at the cost of even higher variance between runs.

Chapter 7

Conclusions and further work

7.1 Summary

In this work, we conducted an analysis of the ladder network from [30] and virtual adversarial training (VAT) from [23, 22] for semi-supervised learning and proposed four variants of a model applying virtual adversarial training to the ladder network: ladder with virtual adversarial cost (LVAC), ladder with layer-wise virtual adversarial cost (LVAC-LW), ladder with virtual adversarial noise (LVAN), and ladder with layer-wise virtual adversarial noise (LVAN-LW).

Based on the manifold and cluster assumptions of semi-supervised learning [6], we hypothesised that virtual adversarial training could improve the classification accuracy of ladder network trained in a semi-supervised context. This thesis tested this hypothesis on the MNIST dataset [18], by training models on training sets consisting of 50, 100 or 1000 labelled examples augmented by the full 60,000 images in the MNIST training set as unlabelled examples. We measured performance as error rate on the held-out test set of 10,000 examples.

We found that our models, most significantly LVAN-LW, improved on the performance of the ladder for 50 labels and 100 labels, achieving state-of-the-art error rates. For 1000 labels, both VAT and ladder baselines outperformed our models. This leads us to believe that the additional regularisation provided by virtual adversarial training to the ladder network are useful only when the task is sufficiently challenging, suggesting that we should our test models on more complex datasets.

Additionally we found that our models performed better than the ladder network on adversarial examples. VAT outperformed our models for L_∞ adversarial examples, but our models, again especially the LVAN-LW model, achieved best performance for the few-label cases (50 and 100 labels) on L_1 and L_2-normalised adversarial examples.

7.2 Evaluation

Due to limited time and computing resources, we only very approximately optimised a limited subset of the hyperparameters of our proposed models. Hyperparameter settings and learning schedules can alter performance significantly, and we cannot be certain of the results presented in this thesis without further optimisation considering the magnitude of the performance differences we draw our conclusions from.

We were able to fulfill the research goal of improving semi-supervised learning performance of the ladder network through augmentation with VAT. However, our experiments to further elucidate the mechanisms which make VAT and the ladder network using our models, such as the investigation of the effects of the number of power iterations on performance, were not thorough enough to give interpretable results.

7.3 Further work

While we have succeeded in achieving strong performance on semi-supervised learning benchmarks overall, with particular strength in both accuracy and stability with very few labelled training data, many potential avenues of further research remain open.

An immediate continuation of this work would be to carry out the experiments in this work more thoroughly, as we were limited by time and the availability of compute resources. More extensive hyperparameter optimisations should be carried out, and more training runs using different splits of labelled data are needed to more narrowly bound our performance metrics. Our investigation in this work into the role of anisotropic smoothing in semi-supervised learning performance was particularly limited, as we were unable to extend the experiments to the LVAC-LW and LVAN-LW models. Additionally, the experiments on adversarial attacks in this paper only used a fixed size of adversarial perturbation, which is a variable that could have significant influence on the relative performance of the models compared.

Another straightforward extension would be to combine adversarial and virtual adversarial training with the ladder, using virtual adversarial costs or perturbations on unlabelled examples as we have done in this work, and additionally using adversarial training or perturbations on labelled examples.

A further natural application would be to test our models on more challenging datasets such as SVHN and CIFAR-10, which are increasingly used as the benchmarks of choice for semi-supervised learning due to the saturation of performance on MNIST. This would allow comparisons with methods such as temporal ensembling and the Π model from [17] and the stochastic augmentation methods of [34], which like the ladder and VAT are relatively fast-but-powerful algorithms particularly adapted to semi-supervised learning. Testing on more complex images would almost certainly require implementing our models using a convolutional encoder/decoder structure.

The datasets SVHN and CIFAR-10 would be more challenging than MNIST as they have more colour channels and more varied images, but they are also 10-way classification problems like MNIST. An even more challenging task would be semi-supervised learning on the 100-class CIFAR-100 dataset [16], which has to date only been attempted by Laine and Aila with temporal ensembling [17]. As our models proved to be particularly effective in the most difficult of our tasks (fifty labelled examples, resisting adversarial attacks), we may see more strengths of our proposed models on these harder datasets.

In [24], Moosavi-Dezfooli et al. recently proposed an algorithm to generate *universal adversarial perturbations* (UAPs): perturbations engineered for a particular network architecture that have an adversarial effect on a majority of input images. Where adversarial perturbations are computed on input-output pairs, and virtual adversarial perturbations are computed on inputs, UAPs are computed for architectures. Augmenting the adversarial components of our proposed models with UAPs could improve performance; replacing the adversarial components with UAP may increase computational efficiency at a small accuracy cost. These hypotheses link back to the theory of smoothing the manifold for semi-supervised learning.

Appendix A

Hyperparameter settings

LVAC

No. labels	$\lambda^{(0)}$	$\lambda^{(1)}$	$\lambda^{(\geq 2)}$	ϵ
50	1504	16.15	0.0381	0.0733
100	1966	14.20	0.1563	0.0731
1000	3883	12.35	0.0539	2.5206

LVAC-LW

No. labels	$\lambda^{(0)}$	$\lambda^{(1)}$	$\lambda^{(\geq 2)}$	$\epsilon^{(0)}$	$\epsilon^{(1)}$	$\epsilon^{(\geq 2)}$
50	1000	10.00	0.1000	1.0000	0.1000	1.00×10^{-3}
100	1966	14.20	0.1563	0.0731	0.4822	1.402×10^{-3}
1000	3883	12.35	0.0539	2.5206	0.0143	6.002×10^{-4}

LVAN

No. labels	$\lambda^{(0)}$	$\lambda^{(1)}$	$\lambda^{(\geq 2)}$	ϵ
50	1504	16.15	0.0381	0.0733
100	1966	14.20	0.1563	0.0731
1000	3883	12.35	0.0539	2.5206

LVAN-LW

No. labels	$\lambda^{(0)}$	$\lambda^{(1)}$	$\lambda^{(\geq 2)}$	$\epsilon^{(0)}$	$\epsilon^{(1)}$	$\epsilon^{(\geq 2)}$
50	1504	16.15	0.0381	0.0733	0.3897	8.372×10^{-2}
100	1966	14.20	0.1563	0.0731	0.4822	1.402×10^{-3}
1000	3883	12.35	0.0539	2.5206	0.0143	6.002×10^{-4}

Ladder

No. labels	$\lambda^{(0)}$	$\lambda^{(1)}$	$\lambda^{(\geq 2)}$
50	1504	16.15	0.0381
100	1966	14.20	0.1563
1000	3883	12.35	0.0539

VAT

No. labels	ϵ
50	5.0
100	5.0
1000	2.5

Bibliography

[1] ABADI, M., AGARWAL, A., BARHAM, P., BREVDO, E., CHEN, Z., CITRO, C., CORRADO, G. S., DAVIS, A., DEAN, J., DEVIN, M., GHEMAWAT, S., GOODFELLOW, I., HARP, A., IRVING, G., ISARD, M., JIA, Y., JOZEFOWICZ, R., KAISER, L., KUDLUR, M., LEVENBERG, J., MANÉ, D., MONGA, R., MOORE, S., MURRAY, D., OLAH, C., SCHUSTER, M., SHLENS, J., STEINER, B., SUTSKEVER, I., TALWAR, K., TUCKER, P., VANHOUCKE, V., VASUDEVAN, V., VIÉGAS, F., VINYALS, O., WARDEN, P., WATTENBERG, M., WICKE, M., YU, Y., AND ZHENG, X. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

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

[3] BENGIO, Y., DELALLEAU, O., AND ROUX, N. L. *Label propagation and quadratic criterion.* MIT Press, Cambridge, Massachusetts, 2006, pp. 193–215.

[4] BERGSTRA, J., AND BENGIO, Y. Random search for hyper-parameter optimization. *J. Mach. Learn. Res. 13* (Feb. 2012), 281–305.

[5] BISHOP, C. M. Training with noise is equivalent to tikhonov regularization. *Neural Comput. 7*, 1 (Jan. 1995), 108–116.

[6] CHAPELLE, O., SCHÖLKOPF, B., AND ZIEN, A. *Semi-Supervised Learning.* MIT Press, Cambridge, Massachusetts, 2006.

[7] GLOROT, X., AND BENGIO, Y. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Con-*

ference on Artificial Intelligence and Statistics (Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010), Y. W. Teh and M. Titterington, Eds., vol. 9 of *Proceedings of Machine Learning Research*, PMLR, pp. 249–256.

[8] GOLUB, G. H., AND VAN DER VORST, H. A. Eigenvalue computation in the 20th century. *J. Comput. Appl. Math. 123*, 1-2 (Nov. 2000), 35–65.

[9] GOODFELLOW, I., BENGIO, Y., AND COURVILLE, A. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[10] GOODFELLOW, I., POUGET-ABADIE, J., MIRZA, M., XU, B., WARDE-FARLEY, D., OZAIR, S., COURVILLE, A., AND BENGIO, Y. Generative adversarial nets. In *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 2672–2680.

[11] GOODFELLOW, I. J. On distinguishability criteria for estimating generative models. *arXiv preprint arXiv:1412.6515* (2014). Presented as a workshop contribution at the 3rd International Conference on Learning Representations (San Diego, CA, USA, 7–9 May 2015).

[12] GOODFELLOW, I. J., SHLENS, J., AND SZEGEDY, C. Explaining and Harnessing Adversarial Examples. *arXiv preprint arXiv:1412.6572* (Dec. 2014). Presented at the 3rd International Conference on Learning Representations (San Diego, CA, USA, 7–9 May 2015).

[13] IOFFE, S., AND SZEGEDY, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning* (Lille, France, 07–09 Jul 2015), F. Bach and D. Blei, Eds., vol. 37 of *Proceedings of Machine Learning Research*, PMLR, pp. 448–456.

[14] KINGMA, D. P., AND BA, J. Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980* (Dec. 2014). Presented at the 3rd International Conference on Learning Representations (San Diego, CA, USA, 7–9 May 2015).

[15] KINGMA, D. P., MOHAMED, S., JIMENEZ REZENDE, D., AND WELLING, M. Semi-supervised learning with deep generative models. In *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 3581–3589.

[16] KRIZHEVSKY, A., AND HINTON, G. Learning multiple layers of features from tiny images.

[17] LAINE, S., AND AILA, T. Temporal Ensembling for Semi-Supervised Learning. *ArXiv e-prints* (Oct. 2016). Presented at the 5th International Conference on Learning Representations (Toulon, FR, 24–26 April 2017).

[18] LECUN, Y., BOTTOU, L., BENGIO, Y., AND HAFFNER, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE 86*, 11 (Nov 1998), 2278–2324.

[19] MAALØE, L., KAAE SØNDERBY, C., KAAE SØNDERBY, S., AND WINTHER, O. Auxiliary Deep Generative Models. *arXiv preprint arXiv:1602.05473* (Feb. 2016).

[20] MACKAY, D. J. C. *Information Theory, Inference & Learning Algorithms*. Cambridge University Press, New York, NY, USA, 2002.

[21] MAKHZANI, A., SHLENS, J., JAITLY, N., GOODFELLOW, I., AND FREY, B. Adversarial Autoencoders. *arXiv preprint arXiv:1511.05644* (Nov. 2015).

[22] MIYATO, T., MAEDA, S.-I., KOYAMA, M., AND ISHII, S. Virtual Adversarial Training: a Regularization Method for Supervised and Semi-supervised Learning. *arXiv preprint arXiv:1704.03976* (Apr. 2017).

[23] MIYATO, T., MAEDA, S.-I., KOYAMA, M., NAKAE, K., AND ISHII, S. Distributional Smoothing with Virtual Adversarial Training. *arXiv preprint arXiv:1507.00677* (July 2015). Presented at the 4th International Conference on Learning Representations (San Juan, PR, USA, 2–4 May 2016).

[24] MOOSAVI-DEZFOOLI, S.-M., FAWZI, A., FAWZI, O., AND FROSSARD, P. Universal adversarial perturbations. *arXiv preprint arXiv:1610.08401* (2016). Presented at the 2017 IEEE Conference on Computer Vision and Pattern Recognition (Honolulu, HI, USA, 21 Jul - 26 Jul 2017).

[25] NETZER, Y., WANG, T., COATES, A., BISSACCO, A., WU, B., AND NG, A. Y. Reading digits in natural images with unsupervised feature learning.

[26] NIELSEN, M. A. *Neural Networks and Deep Learning*. Determination Press, 2015.

[27] OSINDERO, S. Neural networks: Foundations (lecture slides). In *Advanced Topics in Machine Learning* (2017), DeepMind and University College London.

[28] PAPERNOT, N., GOODFELLOW, I., SHEATSLEY, R., FEINMAN, R., AND MCDANIEL, P. cleverhans v1.0.0: an adversarial machine learning library. *arXiv preprint arXiv:1610.00768* (2016).

[29] PEZESHKI, M., FAN, L., BRAKEL, P., COURVILLE, A., AND BENGIO, Y. Deconstructing the ladder network architecture. In *Proceedings of The 33rd International Conference on Machine Learning* (New York, New York, USA, 20–22 Jun 2016), M. F. Balcan and K. Q. Weinberger, Eds., vol. 48 of *Proceedings of Machine Learning Research*, PMLR, pp. 2368–2376.

[30] RASMUS, A., VALPOLA, H., HONKALA, M., BERGLUND, M., AND RAIKO, T. Semi-supervised learning with ladder networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems* (Cambridge, MA, USA, 2015), NIPS'15, MIT Press, pp. 3546–3554.

[31] RIFAI, S., DAUPHIN, Y. N., VINCENT, P., BENGIO, Y., AND MULLER, X. The manifold tangent classifier. In *Advances in Neural Information Processing Systems 24*, J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2011, pp. 2294–2302.

[32] RIFAI, S., VINCENT, P., MULLER, X., GLOROT, X., AND BENGIO, Y. Contractive auto-encoders: Explicit invariance during feature extraction. In *Proceed-*

ings of the 28th international conference on machine learning (ICML-11) (2011), pp. 833–840.

[33] RUDER, S. An overview of gradient descent optimization algorithms. *arXiv preprints arXiv:1609.04747* (Sept. 2016).

[34] SAJJADI, M., JAVANMARDI, M., AND TASDIZEN, T. Regularization with stochastic transformations and perturbations for deep semi-supervised learning. In *Advances in Neural Information Processing Systems* (2016), pp. 1163–1171.

[35] SALIMANS, T., GOODFELLOW, I., ZAREMBA, W., CHEUNG, V., RADFORD, A., AND CHEN, X. Improved Techniques for Training GANs. *arXiv preprint arXiv:1606.03498* (June 2016).

[36] SCHROFF, F., KALENICHENKO, D., AND PHILBIN, J. Facenet: A unified embedding for face recognition and clustering. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2015).

[37] SCIKIT-OPTIMIZE CONTRIBUTORS. Scikit-Optimize: Sequential model-based optimization with a 'scipy.optimize' interface, 2017. Software used under BSD license.

[38] SETTLES, B. Active learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning 6*, 1 (2012), 1–114.

[39] SPRINGENBERG, J. T. Unsupervised and Semi-supervised Learning with Categorical Generative Adversarial Networks. *arXiv preprint arXiv:1511.06390* (Nov. 2015).

[40] SRIVASTAVA, N., HINTON, G., KRIZHEVSKY, A., SUTSKEVER, I., AND SALAKHUTDINOV, R. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research 15* (2014), 1929–1958.

[41] SZEGEDY, C., ZAREMBA, W., SUTSKEVER, I., BRUNA, J., ERHAN, D., GOODFELLOW, I., AND FERGUS, R. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199* (2013).

[42] VALPOLA, H. From neural PCA to deep unsupervised learning. *arXiv preprint arXiv:1411.7783* (Nov. 2014).

[43] WAN, L., ZEILER, M., ZHANG, S., CUN, Y. L., AND FERGUS, R. Regularization of neural networks using dropconnect. In *Proceedings of the 30th International Conference on Machine Learning* (Atlanta, Georgia, USA, 17–19 Jun 2013), S. Dasgupta and D. McAllester, Eds., vol. 28 of *Proceedings of Machine Learning Research*, PMLR, pp. 1058–1066.

[44] WESTON, J., RATLE, F., AND COLLOBERT, R. Deep learning via semi-supervised embedding. In *Proceedings of the 25th International Conference on Machine Learning* (New York, NY, USA, 2008), ICML '08, ACM, pp. 1168–1175.

[45] ZHANG, C., BENGIO, S., HARDT, M., RECHT, B., AND VINYALS, O. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530* (2016). Presented at the 5th International Conference on Learning Representations (Toulon, FR, 24–26 April 2017).