Université de Montréal

Influencing the Properties of Latent Representations

par
Jeremie Zumer

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté à la Faculté des études supérieures
en vue de l'obtention du grade de Maître ès sciences (M.Sc.)
en informatique

Août, 2016

RÉSUMÉ

L'apprentissage automatique repose sur l'étude des méthodes de détermination de paramètres de modélisation de données afin d'accomplir une tâche, telle que la classification d'image ou la génération de phrases, pour un jeu de données. Ces paramètres forment des espaces latents qui peuvent démontrer diverses propriétés impactant la performance du modèle d'apprentissage sur la tâche visée.

Permettre à un modèle d'adapter son espace latent avec plus de liberté peut amener à l'amélioration de la performance de ce modèle. Un indicateur valide de la qualité de l'espace latent engendré par un modèle est la collection des propriétés exprimés par cet espace latent, par exemple en ce qui a trait aux aspects topologiques de la représentation apprise par le modèle.

Nous montrons des résultats appliqués dans des régimes supervisés et non-supervisés, et nous augmentons des modèles par plusieurs modes d'interactions inter-couches, comme des connections entre les couches d'un codeur au décodeur analogue dans un modèle autoencodeur, l'application de transformations affines à la suite de couches, et l'addition de réseaux de neurones auxiliaires connectés en parallèle. L'effet des méthodes proposés est évalué en mesurant soit la performance de classification ou la qualité des échantillons générés par ces modèles, ainsi qu'en comparant les courbes d'entrainement des algorithmes. Les modèles et méthodes sont évalués sur des jeux de données d'images populaires, comme MNIST et CIFAR10, et sont comparés aux méthodes à l'état de l'art pour les tâches accomplies.

Nous développons un modèle qui utilise la puissance générative des autoencodeurs variationnels, enrichi par des connections latérales dans le style des réseaux escaliers pour application en classification.

Nous proposons une méthode qui permet d'isoler les tâches réalisées par les couches convolutionnelles (soit l'apprentissage de filtres pour l'extraction de traits et l'agencement topologique nécessaire pour l'apprentissage par les couches sub-

séquentes) en utilisant des couches complètement connectées intercalées avec les couches convolutionnelles.

Enfin, nous appliquons la technique de normalisation par groupe aux couches du réseau récurrent à l'état de l'art (pixel-rnn) et nous démontrons que cela permet une augmentation prononcée du taux de convergence du modèle, ainsi qu'une amélioration notable de sa performance totale.

Mots-clefs: apprentissage automatique, apprentissage profond, vision par ordinateur, non-supervisé, autoencodeur, réseau de neurone

ABSTRACT

Machine learning models rely on learned parameters adapted to a given set of data to perform a task, such as classifying images or generating sentences. These learned parameters form latent spaces which can adapt various properties, to impact how well the model performs.

Enabling a model to better fit properties of the data in its latent space can improve the performance of the model. One criteria for quality is the set of properties expressed by the latent space - that is, for example, topological properties of the learned representation.

We develop a model which leverages a variational autoencoder's generative ability and augments it with the ladder network's lateral connections for discrimination.

We propose a method to decouple two tasks perfomed by convolutional layers (that of learning useful filters for feature extraction, and that of arranging the learned filters such that the next layer may train effectively) by using interspersed fully-connected layers.

Finally, we apply batch normalization to the recurrent state of the pixel-rnn layer and show that it significantly improves convergence speed as well as slightly improving overall performance.

We show results applied in unsupervised and supervised settings, and augment models with various inter-layer interractions, such as encoder-to-decoder connections, affine post-layer transformations, and side-network connections. The effects of the proposed methods are assessed by measuring supervised performance or the quality of samples produced by the model and training curves. Models and methods are tested on popular image datasets such as MNIST and CIFAR10 and are compared to the state-of-the-art on the task they are applied to.

Keywords: machine learning, deep learning, computer vision, unsupervised, autoencoder, neural network

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

BN	Batch Normalization
CNN	Convolutional Neural Network
Convnet	Convolutional (Neural) Network
DAE	Denoising Auto-Encoder
DBM	Deep Boltzman Machine
Dist	Distortion
GPU	Graphics Processing Unit
KL	Kullback-Leibler
LSTM	Long-Short Term Memory
MLP	Multi-Layer Perceptron
NLL	Negative Log-Likelihood
RBM	Restricted Boltzman Machine
Reprod	Reproduction
RNN	Recurrent Neural Network
Tanh	Hyperbolic Tangent
VAE	Variational Auto-Encoder
VRNN	Variational RNN

NOTATION

$D_{kl}(p\|\|q)$	Kullbach-Leibler Divergence between q and p
$\mathscr{L}$	Loss
$\mathscr{N}$	Gaussian distribution
$\mathscr{B}$	Bernouilli distribution
$\mathscr{C}\dashv\llcorner$	Categorical distribution
$\overrightarrow{x}$	Vector x
$\|\|\overrightarrow{x}\|\|$	2-norm of vector x
σ_x	Varriance of x
$\mathbb{E}_x(y)$	Expected value of y under x
$\overrightarrow{x}\cdot\overrightarrow{y}$	Element-wise multiplication
$\overrightarrow{x}*\overrightarrow{y}$	convolution
$W\overrightarrow{x}$	Matrix-vector inner product
$\overrightarrow{y}\overrightarrow{x}$	dot product
xy	x times y
$\sigma(x)$	sigmoid function
$\lfloor x \rfloor$	floor of x

CHAPTER 1

INTRODUCTION

Machine learning aims to develop methods to automatically capture patterns in data that can be used to solve tasks. Deep learning, which is concerned with models composed of many stacked layers, has made great strides in all areas of machine learning due to its tremendous ability to scale to complex problems where a lot of data is available, but no closed-form solution is known to solve a given task.

In deep learning, computational layers are stacked, forming a graph that links the space of inputs to the space of outputs for a task of interest, as illustrated in figure 1.1. An input is transformed through those computational stacks, and the quality of the model's prediction for that task is refined during the training process.

The layers between the input and output successively map the output of the previous layer into a new representation so as to ignore factors in the input which are not relevant to the task of interest, while retaining as much information relevant to the task as possible. At the same time, layers may project their input into a redundant format that may be easier to process by layers further in the pipeline.

If the representation chosen by a layer is not appropriate for the task, the model will perform the task poorly. If a layer is free to map its input into arbitrary representations, the space of possible solutions – few of which are useful to the task – will take a long time to search and the model might settle in a local minimum. On the other hand, if the layer is too restricted in the representation it can project its input unto, the representation useful to the task might not be present in the search space.

On a high level, supervised learning can be described as a paradigm where a function from input to label is learned, whereas unsupervised learning learns parameters that describe its input to generate new examples like it. More concrete

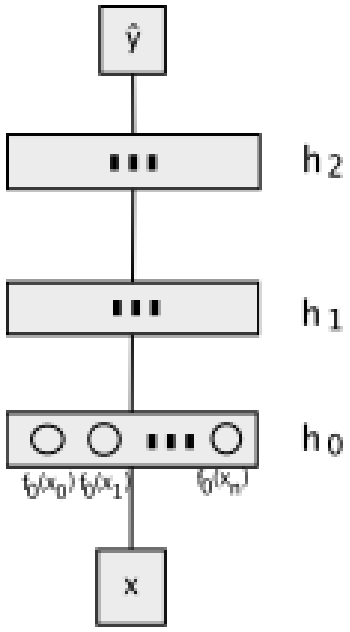


Figure 1.1: A high-level illustration of a directed machine learning model for classification. The inputs x are transformed successively through computational layers h_i. $\hat{y}$ is the prediction from the model for the current task.

examples of the learning paradigms can be found in the following chapters.

Unsupervised learning models the data distribution (e.g., a specific model is proprosed, and the parameters for that model are learned from the input data so as to best fit the data). On the other hand, supervised learning attempts to perform a task directly, based on the data (such as recognizing a digit from an image, or a phonem from a sound stream, or predicting the required settings to move a robot's arm for the next part of a motion). Finally, semi-supervised learning attempts to perform a task directly, except in the presence of data for which the expected outcome is not known in all cases.

For supervised learning, the quality of the internal representations in the model simply impacts model performance on a given task. In the case of unsupervised learning, the quality of the internal representations in the model directly impacts the model's ability to model the data, and thus the quality of samples that can be generated from the model. This setting allows many applications, such as data

compression, reconstructing partially missing data, generating unseen datapoints that match the distribution of the dataset, etc. Semi-supervised learning has to perform tasks similar to unsupervised models when the data is not labeled, and similar to supervised learning otherwise.

Thus, the impact of varying properties of the representations is much clearer on unsupervised and semi-supervised models, and so we focus our experiments on unsupervised and semi-supervised models for the most part. We explore a hybrid of the ladder network [36] – an autoencoder augmented by lateral connections – and variational autoencoder [17] – an autoencoder which models latent factors of the data using a variational method – models and find that the training process seems to converge to a middle-ground that prevents the model from performing well on either task. We also explore the impact of batch normalization on the pixel-rnn model and find that it greatly improves convergence speed. Additionally, we propose that separating the tasks performed by individual layers in a deep model can lead to improved model performance, which we demonstrate with improved performance on the MNIST dataset.

The next few chapters cover the central principles from which our experiments are derived. In chapter 5, we discuss experiments to shape latent model variables to adopt a convolutional structures, and to decouple latent representation topology and function. In chapter 6, we discuss experiments which aim to improve model classification performance by exploiting latent representations learned in a generative context. Finally, we explore the pixel-rnn model, a recurrent neural network which maintains latent a representation across timesteps which is conditioned on an image's pixels, with an auto-regressive dependency.

The code used in the experiments of this thesis is available at <https://bitbucket.org/zumerj/latent>.

CHAPTER 2

DATA FORMAT

Datasets are segmented into three subsets: a training set, a testing set and a validation set. The testing set is used to measure the model's ability to generalize for the task it has learned – that is, to measure how well the model performs when presented with unforeseen data, while the validation set is used to tell when the model is overfitting the train data and no longer learning information useful to the task proper; the training set being the subset used to actually train the model. The test set performance is not used to tune the model, it is only used to measure final model performance after tuning on the validation set. This is to ensure that the model does not overfit the test set by being tuned to perform best on it. Thus, training error is tracked throughout training and is the metric used to inform the model on how to improve, while test error is used to assess how the model is doing comparatively, on a set that it doesn't see during training.

2.1 MNIST

The MNIST digit dataset contains 28 by 28 pixels grayscale images. Each digit can have a label between 0 and 9 (i.e. the digit being represented). The image data represents handwritten digits. The dataset comes with a predefined train and test split, where the train set is 60,000 images and the test set is 10,000 images. 10,000 images from the train set are held out at random to form the validation set. Images are shuffled randomly.

The state of the art on the MNIST dataset is 0.21% test-set error [5, 8]. The task can essentially be considered to be solved. Nevertheless, it is a useful dataset to determine the viability of a model.

2.2 CIFAR10

The CIFAR-10 dataset contains 80 million images of size 32 by 32 pixels [1]. We use this dataset to measure model performance on a task harder than MNIST. The images come with a predefined 50,000 images train set and 10,000 images test set. 10,000 images from the train set are held out to form a validation set. The images are converted from RGB to YUV, image mean is subtracted. Finally, each pixel is divided by their image variance. CIFAR10 images can take one of 10 labels, hence its name.

Images represented in this dataset are heavily downscaled natural images of various types, such as boats, cars, dogs, etc.

The state of the art on CIFAR10 is 96.53% test-set accuracy, and it relies heavily on data augmentation and multiple passes at test time [5, 12].

2.3 Dogs vs Cats

The Dogs vs Cats Kaggle dataset is a dataset of 3000000 cat and dog images used by the Asirra (Animal Species Image Recognition for Restricting Access) CAPTCHA system [10].

The subset of the dataset that we used is the one provided by the Kaggle competition "Dogs vs Cats" [3]. The dataset contains 25000 images of dogs and cats, with a binary label (1 for dog, 0 for cat). 5000 images are randomly held out from this dataset to form a validation set. The test set is not publicly available.

The state of the art on the Dogs vs Cats dataset is 98.53% test-set accuracy [4, 40].

Since these images are large, they are first preprocessed by resizing them such that the smaller dimension is 48 pixels long. A centered 48 pixel window is cropped along the remaining dimension. The resulting images are 48 by 48 pixels, and no additional preprocessing was performed.

CHAPTER 3

LEARNING PARADIGMS

In this chapter, the main settings for which machine learning models are trained are described. The first section describes the generic setting, and further sections each describe a paradigm in more details.

3.1 Generalization and Regularization

We are interested in learning parameters for models that allow them to generalize to unforeseen data - that is, to perform accurate predictions on data not already seen by the model. In order to do that, we split our datasets into training and test sets, where the training set is used to drive the model's training process, and the test set is used to determine generalization statistics for the model, such as the classification error rate. The metrics obtained on the test set serve to estimate the model's generalization performance on unforeseen data (that is, we assume the test set is distributed in the same way as the data in the domain we are interested in).

When the model trains repetitively on the same training data, and if the model's capacity (that is, complexity of the functions it can represent, which can be described in terms of amount of parameters in the model) is sufficient, the model may learn to fit specific details of the training set, which are not necessarily relevant to data at large. This may cause overfitting - that is, training performance may continue improving while test performance decreases; thus, the model features worse generalization performance.

In order to prevent this phenomenon, we use regularization schemes, i.e., methods which penalize the model when it adapts to show properties that indicate overfitting. These properties may be a reduction in performance on a hold out

"validation" set (a set which is not used for feedback during training, and is not used for testing, but on which we track metrics such as validation performance during training to determine if the model is overfitting the training data), or properties of the weights, such as their sparsity or magnitude [34]. Some regularization schemes, such as L1 and L2 regularization [34], as well as early stopping, are introduced later in this chapter.

A model that is properly regularized may feature even better performance on the test set than the equivalent model without proper regularization right before it would start overfitting, because regularization schemes may help the model avoid local extrema in the optimization process [34].

3.2 Supervised Learning

Supervised learning consists of training parameters of a model based on observed relations, such as image to label mappings. The model makes a prediction on what the mapped value should be for a given input and the divergence between the prediction and the actual output drives learning.

A model is trained to optimize a function which acts as a proxy for the performance of the task we're interested in performing. We use a proxy function, called the loss, so as to ensure the function to optimize is reasonable smooth. Doing so ensures that areas that aren't smooth in the function are minimized, so as not to lead the model to be unable to escape a bad region of the loss function's curve.

Performance for the model can simply be evaluated by using the task the model is trying to solve. For example, in the case of recognizing digits, the recognition error rate is an obvious way to compare models. However, variants exist, based on how difficult the task is, or how ambiguous some of the choices are. Top-K error rate, where the model's top K predictions are evaluated rather than just the top 1, is one such variants. This approach has the advantage that when a datapoint has

no clear, obvious expected output, such as deciding if an image represents leaves or a tree from a close-up shot of a tree's leaves, models which make a reasonable decision are not judged to be outright wrong. This approach is also used when data is classified using a hierarchy of labels, such as all palm trees also being labeled as trees, and all trees also being labeled as plants.

3.3 Unsupervised Learning

Unsupervised learning attempts to extract useful information from unlabeled data. In other words, it learns a (usually lossy) compression scheme specific to the class of data the input belongs to. In probabilistic terms, it learns the parameters of the distribution which best fits the input data, for example.

Training the model results in capturing patterns inherent in the data, without an additional task restriction. This can be used to compress the data (by making the inner-most layer of the network much smaller than the input, and training the model to reconstruct the input: the representation at any layer is thus a compression for the input that the model knows how to decompress), to reconstruct missing data (for example, in the case of superresolution, where a small image is upscaled and the model learns to minimize distortions and artifacts), or to generate new examples (such as producing music based on a dataset of music clips).

Performance is typically evaluated using negative log-likelihood for the data under the model's learned distribution as a way to tell how well the model has really managed to capture the distribution of the dataset.

3.4 Hybrid Learning

Hybrid learning refers to learning both an unsupervised and a supervised objective at the same time, as opposed to basing a supervised learning algorithm on top of a model that has been trained in an unsupervised setting, or performing super-

vised tuning of models that were pre-trained in an unsupervised setting. The most common usage of hybrid learning is semisupervised learning, whereby only a subset of the input data contains labels. In that setting, the model uses hybrid learning to perform the standard classification task while attempting to "reconstruct" the missing labels by modeling the labels by a suitable distribution and learning this distribution's parameters during training, or by marginalizing over examples with missing labels.

In general, hybrid learning attempts to leverage unsupervised learning methods' ability to match some structure of the input data without any other input other than the data itself, in order to better perform classification, or alternatively, to improve the ability of the model to fit the data distribution by providing it with labeling information.

Several models perform this task, including the ladder network [36] and the generative semi-supervised model [19], which are described below. The experiments described in chapter 6 derive from the same principle, and make use of specifics of these models.

As the latent space representation of an input can be more informative if there are less factors that explain it than there are input dimensions (that is to say, the latent space representation matching a specific input can dilute unimportant factors to present a more informative representation of a datapoint), it is sensible to hypothesize that capturing this latent input representation would allow for easier classification, as the noise from external factors that are not part of the true data of interest are stripped from this compressed representation.

If the data contains information that is useful to the classification task in question, then it is natural to assume that learning unsupervised information about the data will help perform the classification task. For example, if input images for the same class have similarities in input space (that is, if they "look similar", such as with hand-written digits in the case of digit classification), then it is possible

to extract useful information from the data in an unsupervised way to inform the classification task and thus improve model performance, such as by acting as a regularization scheme based on the distribution of the data we are interested in.

3.5 Learning In Models With Latent Variables

When considering the layers of the model as forming latent spaces, training is the process of inferring the parameters of the model to match the distribution of the data set (or "data distribution"), and models support a generation operation, which can produce data of the kind we're interested in, by sampling from the model, using the parameters learned during inference (i.e. sampling from the "model distribution"). The training process otherwise proceeds normally; that is, an output prediction is sampled from the model in the case of classification tasks, and unsupervised tasks are often learned by training the model to reconstruct the input by sampling from the latents and matching the output to the input of the model.

3.6 Optimization

Optimizing the parameters of the model with regard to its objective function is what drives learning. The most typical solution to perform optimization in neural networks (and some probabilistic models such as the VAE) is to use backpropagation (which consists of calculating gradients in the opposite direction as the model usually operates to produce an output, minimizing the required amount of unique computations) with stochastic gradient descent (SGD). SGD consists of updating the parameters based on the gradient of the cost with regard to these parameters and a scaling factor, as described in equation 3.2 where C is the cost function.

Because the optimization problem is typically non-smooth and non-convex (although the assumption that the surface is locally convex is often made) [23], a

large learning rate η will cause training to diverge as the parameter never settles near a minimum. However, if the scaling factor is too small, the parameter will not update quickly enough and the model will not train in reasonable time. Moreover, if the surfaces formed by the function based on its parameters are too non-smooth, it is possible that the gradient takes extreme values away from useful regions of that manifold.

Several approaches exist to solve this problem. One might reduce the learning rate under specific conditions, such as when the training objective stops improving, or systematically after a certain number of training epochs, or by a small amount every epoch. Methods like momentum [33], described in equation 3.1, attempt to prevent the learning process from diverging due to proximity to non-smooth areas of performance vs parameters space by taking into account the contribution of previous gradients to the trajectory of the parameters. Methods like L1 regularization such as in equation 3.3 or L2 regularization such as in equation 3.4, which are applied after the normal SGD update of equation 3.2 can also help in preventing overfitting [34].

$$g_{t+1} \leftarrow \nu g_t + (1-\nu)\eta \cdot \frac{\partial C}{\partial \overrightarrow{\theta_t}}$$

$$\overrightarrow{\theta_{t+1}} \leftarrow \overrightarrow{\theta_t} - g_{t+1} \tag{3.1}$$

$$\overrightarrow{\theta_{t+1}} \leftarrow \overrightarrow{\theta_t} - \eta \cdot \frac{\partial C}{\partial \overrightarrow{\theta}_t} \tag{3.2}$$

$$\overrightarrow{\theta_{t+1}} \leftarrow \overrightarrow{\theta_{t+1}} + \lambda_1 \sum_{i=1}^{||\overrightarrow{\theta_{t+1}}||} |\overrightarrow{\theta}^i_{t+1}| \tag{3.3}$$

$$\overrightarrow{\theta_{t+1}} \leftarrow \overrightarrow{\theta_{t+1}} + \lambda_2 \sum_{i=1}^{||\overrightarrow{\theta_{t+1}}||} (\overrightarrow{\theta}^i_{t+1})^2 \tag{3.4}$$

Methods such as adadelta, RMSProp and ADAM [9, 18, 47] attempt to solve the learning scheduling and learning rate selection issues with adaptive algorithms.

These methods are also typically able to speed up training by using topological information in error space by maintaining statistics on the gradients observed during training (similar to how second-order methods make use of topological information such as hessians or local hessian estimates to help avoid saddle points and improve convergence speed).

A recent and surprisingly powerful method to improve optimization performance if batch normalization [15], which centers pre-activations (i.e. removes the mean and divides by the variance) based on batch statistics (the mean and variance estimates are updated as training progresses), and trains parameters γ and β to let the model remap the pre-activation (via the transformation described in equation 3.5 where α is the centered pre-activation).

$$h_i = \gamma_i(\alpha_i + \beta_i) \tag{3.5}$$

CHAPTER 4

MODELS

We describe various models on which we base our experiments. We first describe simple models such as the multilayer perceptron and convolutional models, and then move up to more complex models like the ladder network and the variational recurrent neural network.

4.1 Multilayer Perceptron

Multilayer perceptron models (MLPs) are feedforward models composed of layers of units. Units are simple elementwise operations. Layers are independent of the part of the network that lie below them, conditional on the layer immediately below. These models are said to be feedforward because their output is obtained by feeding an input forward through the network, from one layer to the next, such that the output of one layer is the input to the layer above. Layers in a MLP are typically fully-connected, i.e. each unit in a layer above is connected to each unit in the layer below.

Layers in MLPs typically perform non-linear transformations. The output of a unit is called an "activation", a reference to neuron activations in biological neural networks, and the non-linear operation they perform is referred to as the "activation function". Some popular activation functions are described in equations 4.1 through 4.4, where operations are elementwise when applicable and $\overrightarrow{\alpha}$ is the vector of pre-

activations for the layer.

$$\max(0, \overrightarrow{\alpha}) \quad \text{Rectified linear} \tag{4.1}$$

$$\frac{1}{1+e^{-\overrightarrow{\alpha}}} \quad \text{Logistic sigmoid} \tag{4.2}$$

$$\frac{\exp(\overrightarrow{\alpha}) - \exp(-\overrightarrow{\alpha})}{\exp(\overrightarrow{\alpha}) + \exp(-\overrightarrow{\alpha})} \quad \text{Tanh} \tag{4.3}$$

$$\frac{\exp(\overrightarrow{\alpha})}{\sum_{i=1}^{||\overrightarrow{\alpha}||} \exp(\overrightarrow{\alpha}_i)} \quad \text{Softmax} \tag{4.4}$$

A feedforward layer output is typically described by equation 4.5 where f is the activation function for the layer, W is a matrix of weights, $\overrightarrow{b}$ is a vector of biases, and $\overrightarrow{x}$ is the input vector. Units of layers that are not the input layer or the output layer are often called "hidden units" because their target state is not directly observed and they are free to adopt any pattern to fit the function being learned. A hidden unit's activation is thus often denoted h.

$$h_j = f(W_{ij} x_i + b_j) \tag{4.5}$$

4.1.1 Convolutional Networks

Convolutional Neural Networks (CNNs) [24] are MLP-like feedforward networks whose layers learn convolution kernels to process inputs on the premise that topologically far elements are weakly correlated, to the point that they are ignored implicitly via the convolution kernel size selection. They have shown to be powerful models to extract useful information from image-like data [13, 20, 40–42]

4.1.1.1 Convolution Operation

The discrete convolution operation is defined in equation 4.6. This operation applies the function f across the signal g. As per the convolution theorem, this is typically equivalent to the inverse Fourier transform of the Fourier transform of f

multiplied element-wise by the Fourier transform of g [7].

The 1D convolution does not conserve all topological information that may be relevant in images [6, 26] (although such a structure is not necessarily hard to learn [38]). 2D convolution (and multidimensional discrete convolution in general) is a straightforward expansion of 1D convolution and is described in 4.7. For convolutional networks operating on images, this is the preferred form of convolution [38].

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m] \tag{4.6}$$

$$(f * g)[n,k] = \sum_{m_1=-\infty}^{\infty} \sum_{m_2=-\infty}^{\infty} f[m_1,m_2]g[n-m_1,k-m_2] \tag{4.7}$$

While equations 4.6 and 4.7 formalize the computations performed by convolutional layers, the idea of convolution as used in neural networks was inspired by biology [24]. Convolution layers have some advantages over fully-connected layers when working with images: they offer shift and distortion invariance as well as reducing greatly the amount of parameters in the model by their use of local receptive fields (i.e. the part of the input which contributes to the output for a specific cell in the output matrix is not the entire image, note that this is in line with the prior that nearby pixels to a target pixel are strongly correlated, while those further apart are only weakly correlated [28]), and shared weights (i.e. the same parameters are used to perform the convolution at each location on the input). Additionally, it is customary to apply a pooling (or subsampling) operation on the output of the convolution [24].

4.2 Recurrent Models

Recurrent models are models where there is a connection between the hidden layers across timesteps [29]. A vanilla recurrent neural network (RNN) can be described such as in equation 4.8 where f is an arbitrary non-linearity.

$$\overrightarrow{h_t} = f(W\overrightarrow{x} + V\overrightarrow{h_{t-1}} + \overrightarrow{b}) \tag{4.8}$$

An LSTM model is a kind of recurrent model which adds gating units, forget, state, output and input gates, allowing the network to clear its memory, skip outputting a value, skip updating its state, or ignore the input (as well as more complex transformations). These features allow the model to learn more complex long-term dependencies [11]. The LSTM state update function is described in equations 4.9 to 4.14, where $\overrightarrow{v}$ is the output, the gates are $\overrightarrow{i}$ (input), $\overrightarrow{f}$ (forget), $\overrightarrow{o}$ (output), and $\overrightarrow{s}$ (state), g_x are arbitrary activation functions, and W_x and V_x are weight matrices. The activation functions are often the logistic sigmoid for the forget and input gate, and the hyperbolic tangeant for the output and state gates.

$$\overrightarrow{i} = g_i(W_i\overrightarrow{x} + V_i\overrightarrow{h_{t-1}} + \overrightarrow{b_i}) \tag{4.9}$$

$$\overrightarrow{s} = g_s(W_s\overrightarrow{x} + V_s\overrightarrow{h_{t-1}} + \overrightarrow{b_s}) \tag{4.10}$$

$$\overrightarrow{f} = g_f(W_f\overrightarrow{x} + V_f\overrightarrow{h_{t-1}} + \overrightarrow{b_f}) \tag{4.11}$$

$$\overrightarrow{o} = g_o(W_o\overrightarrow{x} + V_o\overrightarrow{h_{t-1}} + \overrightarrow{b_o}) \tag{4.12}$$

$$\overrightarrow{h_t} = \overrightarrow{i} \cdot \overrightarrow{s} + \overrightarrow{f} \cdot \overrightarrow{h_{t-1}} \tag{4.13}$$

$$\overrightarrow{v} = g_v(\overrightarrow{h_t}) \cdot \overrightarrow{o} \tag{4.14}$$

The $W_x\overrightarrow{x}$ term in the LSTM equations is often referred to as the "inputs-to-states" while the $V_x\overrightarrow{h_{t-1}}$ term is referred to as the "states-to-states".

RNNs are typically trained via the backpropagation through time (BPTT) al-

gorithm, which is simply the backpropagation algorithm applied on an unrolled view of the network. Additional considerations, such as gradient clipping and step limits, may be used to alleviate issues such as exploding or vanishing gradient which happen more often in these models due to the interactions between timesteps.

4.2.1 Pixel-RNN

The pixel-rnn model is an RNN that operates on images, capturing a topologically significant "context" (i.e. surrounding pixels) while ignoring pixels that the model will observe in later timesteps [45]. This is achieved by performing a convolution on the image (making use of the fact that the convolution is applied left-to-right, top-to-bottom across the image, whereas the image is processed in a recurrent way with timesteps along the height of the image). A mask is applied to the convolution kernel as a pattern extraction step. The mask used in this step is illustrated in figure 4.1 and eliminates any dependencies with pixels that haven't been visited by the convolution previously.

The model is implemented as a set of two LSTMs that process the image from opposing corners, and from the top to the bottom, in diagonal. The output of these LSTMs is combined by flipping the output of the LSTM that processes the image right-to-left, and adding it to the other LSTM's output, with an offset of one row.

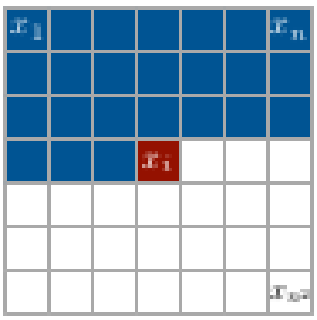


Figure 4.1: Illustration of the mask used by the pixel-rnn during the patch extraction step (image taken from [45])

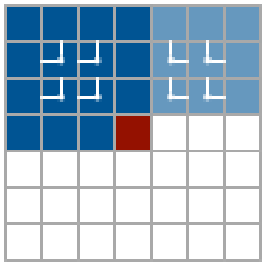


Figure 4.2: Illustration of the diagonal bilstm progress over the image through time (image taken from [45])

A convolutional network then processes this combined output into a final output for the current step. This double LSTM assembly is referred to as a "diagonal bilstm", and its processing order on the image is illustrated in figure 4.2.

The model is trained by minimizing the binary crossentropy between the input and output in the case of binary input, such as binarized MNIST, and the categorical crossentropy between the input and output otherwise, by using a discrete softmax output on the 256 possible pixel intensity values for each pixel as the final non-linearity in the output layer.

4.3 Variational Methods

Variational methods enable sampling of the posterior distribution by using an approximator distribution which can be easily sampled from, and determining that distribution's parameters so that it matches the real data distribution as closely as possible. This variational distribution is denoted $q(\cdot)$ as opposed to the model distribution $p(\cdot)$.

4.3.1 Log-likelihood Lower bound

The likelihood is typically manipulated in log-space as it is easier to manipulate log-terms than raw likelihood terms. In order to train a variational model, we thus

want to maximize the log-likelihood of the data.

$$D_{kl}(q||p) = \int_{-\infty}^{\infty} q(x) \log\left(\frac{q(x)}{p(x)}\right) \mathrm{d}x \tag{4.15}$$

The log-likelihood of a model is given in equation 4.16 [16] where D_{kl} is the Kullback-Leibler (KL) divergence as described in equation 4.15 and $\mathscr{L}$ is the "variational lowerbound" which is discussed further below. The KL divergence is always positive (as per equation 4.15), and so we want to optimize the variational lowerbound (since the expression of the KL divergence may be intractable). Equations 4.17 through 4.23 derive such a bound (based on the expression for the KL divergence), where $p(\cdot)$ is shorthand for $p(\cdot|\theta)$ for data parameters θ, and $q(\cdot)$ is shorthand for $q(\cdot|\phi)$ for model parameters ϕ. Thus, we find the variational lower bound described in equation 4.23. It is then possible to train the model by iterative sampling to optimize the bound.

$$\log p(\vec{x}) = \mathscr{L}(\vec{x}) + D_{kl}(q(\vec{z}|\vec{x})||p(\vec{z}|\vec{x})) \tag{4.16}$$

$$D_{kl}(q(\overrightarrow{z}|\overrightarrow{x})||p(\overrightarrow{z}|\overrightarrow{x})) = \sum_{\overrightarrow{z}\in Z} q(\overrightarrow{z}|\overrightarrow{x}) \ln\left(\frac{q(\overrightarrow{z}|\overrightarrow{x})}{p(\overrightarrow{z}|\overrightarrow{x})}\right) \tag{4.17}$$

$$= \sum_{\overrightarrow{z}\in Z} q(\overrightarrow{z}|\overrightarrow{x}) \ln(q(\overrightarrow{z}|\overrightarrow{x})) - \sum_{\overrightarrow{z}\in Z} q(\overrightarrow{z}|\overrightarrow{x}) \ln(p(\overrightarrow{z}|\overrightarrow{x})) \tag{4.18}$$

$$= E_{q(\overrightarrow{z}|\overrightarrow{x})}[\ln(q(\overrightarrow{z}|\overrightarrow{x}))] - \sum_{\overrightarrow{z}\in Z} q(\overrightarrow{z}|\overrightarrow{x}) \ln\left(\frac{p(\overrightarrow{z},\overrightarrow{x})}{p(\overrightarrow{x})}\right) \tag{4.19}$$

$$= E_{q(\overrightarrow{z}|\overrightarrow{x})}[\ln(q(\overrightarrow{z}|\overrightarrow{x}))] - \sum_{\overrightarrow{z}\in Z} q(\overrightarrow{z}|\overrightarrow{x}) \ln(p(\overrightarrow{z},\overrightarrow{x})) + \sum_{\overrightarrow{z}\in Z} q(\overrightarrow{z}|\overrightarrow{x}) \ln(p(\overrightarrow{x})) \tag{4.20}$$

$$= E_{q(\overrightarrow{z}|\overrightarrow{x})}[\ln(q(\overrightarrow{z}|\overrightarrow{x}))] - E_{q(\overrightarrow{z}|\overrightarrow{x})}[\ln(p(\overrightarrow{x},\overrightarrow{z}))] + \ln(p(\overrightarrow{x})) \sum_{\overrightarrow{z}\in Z} q(\overrightarrow{z}|\overrightarrow{x}) \tag{4.21}$$

$$= E_{q(\overrightarrow{z}|\overrightarrow{x})}[\ln(q(\overrightarrow{z}|\overrightarrow{x}))] - E_{q(\overrightarrow{z}|\overrightarrow{x})}[\ln(p(\overrightarrow{x},\overrightarrow{z}))] + \ln(p(\overrightarrow{x})) \tag{4.22}$$

$$\geq 0 \qquad D_{kl} \text{ is always positive}$$

$$\iff \ln(p(\overrightarrow{x})) \geq E_{q(\overrightarrow{z}|\overrightarrow{x})}[-\ln(q(\overrightarrow{z}|\overrightarrow{x})) + \ln(p(\overrightarrow{x},\overrightarrow{z}))] \tag{4.23}$$

4.4 Autoencoders

Autoencoders (AEs) are popular unsupervised models. They are comprised of an encoder pathway and a decoder pathway. The encoder pathway maps an input to a latent representation, and the decoder pathway maps the latent representation back into input space as a so-called reconstruction of the input. Such an architecture is illustrated in figure 4.4. Several variants exist, and we describe denoising autoencoder and variational autoencoder variants in the following subsections. The Pixel-RNN model is a kind of autoencoder.

4.4.1 Denoising Autoencoders

Denoising autoencoders (dAEs) [46] are a variant of autoencoders where instead of the encoder mapping the input to latent space, it instead maps a corrupted version of the input to latent space. The input is typically corrupted by injecting noise, such as gaussian noise, at the input. The decoder in such models still attempts to map the latent space to (noise-free) input space. In so doing, the latent

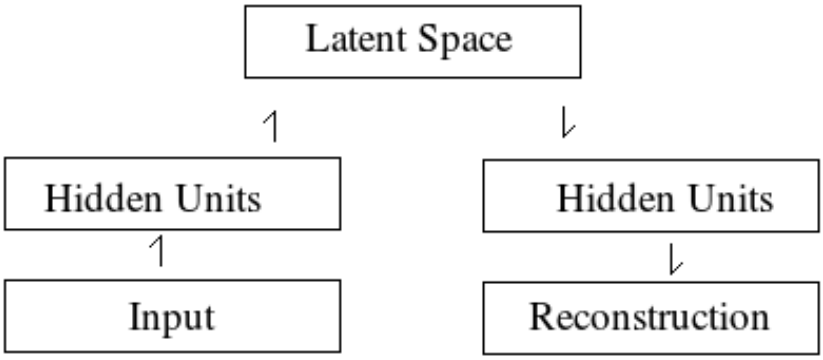


Figure 4.3: An autoencoder model. The part of the model outputting to the latent space is the encoder, while the part of the model taking its input from the latent space is the decoder.

space learned by the model is expected to encode only the factors which are truly relevant to the input while ignoring noise native to the input, such as that created by imperfect instruments in the collection of the data.

4.4.2 Variational Autoencoders

Variational autoencoders (VAEs) are autoencoder models which are trained in a variational way, but which objective can be optimized directly by back-propagation, making them very fast without accumulating bias from sampling [17].

Since we have full control of the prior distribution of the model (given that the prior distribution is determined in advance and does not depend on model or data specifics), and since we know the parameters of the model, the expression of the D_{kl} becomes simpler to compute. Because the $p(x|h)$ model is directed, it becomes trivial to sample from it. By choosing a continuous latent distribution which can easily be split to isolate the random process, it is then possible to backpropagate through even the sampling step. One such distribution is the gaussian distribution, which can be reparameterized as in equation 4.24, where the random process is

independent of the parameters and thus is constant in all gradients.

$$\mathcal{N}(\mu, \sigma^2) = \mu + \sigma\mathcal{N}(0,1) \tag{4.24}$$

It is important to note that the sampling operation mentioned here is unrelated to the sampling operations performed during training with methods such as contrastive divergence: it is instead akin to the sampling performed in order to inject noise at the input of a dAE, with the difference that the noise is instead injected at the latent space instead.

Since we know the prior, we can rewrite the variational lower-bound in terms of a KL divergence we can compute tractably. The derivation described in equations 4.25 through 4.30 shows the rewriting process, starting from the new KL divergence and relating it to the previous expression of the variational lower-bound found in section 4.3, where equation 4.30 gives the variational autoencoder's lower-bound. The reconstruction term (that is, the expectation term beside the D_{kl} in equation 4.30) is obtained by sampling. A single sample is enough to get an unbiased estimate of the gradient, which can be used in gradient descent algorithms [17]. Due to the interaction between the stochastic terms and the deterministic terms with reparameterization, it becomes easy to backpropagate.

$$D_{kl}(q(\vec{z}|\vec{x})||p(\vec{z})) = \sum_{\vec{z}\in Z} q(\vec{z}|\vec{x}) \ln\left(\frac{q(\vec{z}|\vec{x})}{p(\vec{z})}\right) \tag{4.25}$$

$$= \sum_{\vec{z}\in Z} q(\vec{z}|\vec{x}) \ln(q(\vec{z}|\vec{x})) - \sum_{\vec{z}\in Z} q(\vec{z}|\vec{x}) \ln(p(\vec{z})) \tag{4.26}$$

$$= E_{q(\vec{z}|\vec{x})}[\ln(q(\vec{z}|\vec{x}))] - \sum_{\vec{z}\in Z} q(\vec{z}|\vec{x}) \ln\left(\frac{p(\vec{z},\vec{x})}{p(\vec{x}|\vec{z})}\right) \tag{4.27}$$

$$= E_{q(\vec{z}|\vec{x})}[\ln(q(\vec{z}|\vec{x}))] - \sum_{\vec{z}\in Z} q(\vec{z}|\vec{x}) \ln(p(\vec{z},\vec{x})) + \sum_{\vec{z}\in Z} q(\vec{z}|\vec{x}) \ln(p(\vec{x}|\vec{z})) \tag{4.28}$$

$$= E_{q(\vec{z}|\vec{x})}[\ln(q(\vec{z}|\vec{x}))] - E_{q(\vec{z}|\vec{x})}[\ln(p(\vec{x},\vec{z}))] + E_{q(\vec{z}|\vec{x})}[\ln(p(\vec{x}|\vec{z}))] \tag{4.29}$$

$$\rightarrow \mathcal{L}(\vec{x}) = -D_{kl}(q(\vec{z}|\vec{x})||p(\vec{z})) + E_{q(\vec{z}|\vec{x})}[\ln(p(\vec{x}|\vec{z}))] \tag{4.30}$$

4.5 Generative Semi-Supervised Model

The generative semi-supervised model (M2 in the original paper), and its latent-feature discriminative (M1) and stacked generative (M1+M2) variants [19], leverage generative modeling to capture factors in the underlying data which are useful for classification in the context of partially missing label information. The latent-feature model is a generative model trained in an unsupervised way, on top of which a generic classifier, such as an SVM, may be used to perform classification. The combined model is a generative model that is composed of two sets of latent variables: one for the class of the data (y) and another for all other features (z), as illustrated in figure 4.5; and their combination, which first trains the latent-feature model and then trains a combined model on top of the features learned from latent-feature model [19].

The latent-feature approach lacks capacity and somewhat fails to capture the essence of what has made modern forays into machine learning so successful (namely, deep learning) as it separates the tasks in distinct steps and the classification task cannot inform the unsupervised learning task and vice-versa. The generative semi-supervised approach is more interesting as it describes the classification task as

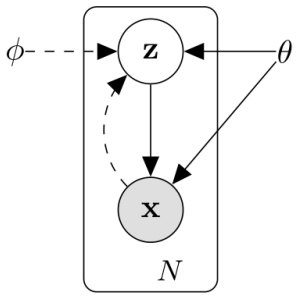


Figure 4.4: The VAE model in plate notation. Dashed lines form the inference path while solid lines form the generative path (image taken from [17]).

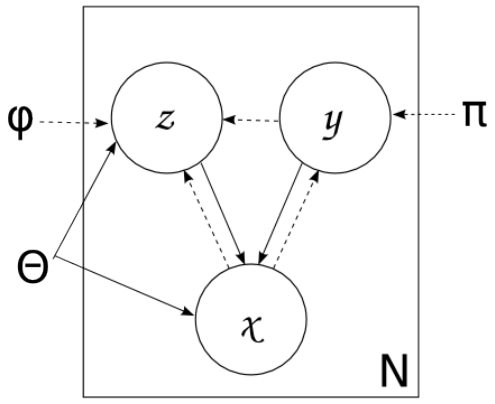


Figure 4.5: The generative semi-supervised model in plate notation. Dashed lines form the inference path while solid lines form the generative path.

an inference problem, which, with proper parameterization, allows one to both capture latent information about the form of the data (e.g. calligraphy style or colorscheme for images, or timbre or pitch for sound data) and information related to the task (i.e. the labels) simultaneously. The latent information useful to determine the data's class is learned separately and in isolation, during the inference process (the stacked model is a straightforward extension to the generative semi-supervised model that attempts to leverage depth to improve the power of the model by training a generative semi-supervised model on top of the latent feature model's learned latent representation). Thus, experiments in further chapters are mostly based on the generative semi-supervised model when appropriate, and the latent-feature model will not be discussed further here. The generative semi-supervised model is composed of two parts: the model's approximating function for the posterior distribution, conditional on both the input and the label (equation 4.31, usually an arbitrary MLP), and the approximating distribution for the labels themselves,

conditional on the input (equation 4.32, likewise).

$$q(\overrightarrow{z}|\overrightarrow{y},\overrightarrow{x}) = \mathcal{N}(\overrightarrow{z}|\mu(\overrightarrow{y},\overrightarrow{x}),\texttt{diag}(\sigma^2(\overrightarrow{x}))) \tag{4.31}$$

$$q(\overrightarrow{y}|\overrightarrow{x}) = \texttt{Cat}(\overrightarrow{y}|\pi(\overrightarrow{x})) \tag{4.32}$$

The functions μ, σ^2 and π are modeled by MLPs [19]. As this model is trained semi-supervised, two cases are considered: when the label is present along with the input, the term being optimized is an extension of the general variational objective as in equation 4.17, which also takes the label into account, as in equation 4.33 (which is simply the normal lowerbound with the likelihood term associated with the y variables added). In the case where the label is missing, it is treated as another latent parameter, and the term being optimized is described in equation 4.35. The total objective is simply the sum of the two objectives ($\mathscr{U}(x)$ and $\mathscr{L}(x,y)$) over the subsets of the training data they respectively apply to (the former for labeled inputs and the latter otherwise), but with the former cost augmented by $\lambda[-\log q_\phi(y|x)]$, where λ is a scaling factor which balances the relative weight between generative and discriminative learning.

$$-\mathscr{L}(\overrightarrow{x},\overrightarrow{y}) = -D_{kl}(q_\phi(z|x,y)||p(z)) + E_{q_\phi(z|x,y)}[\log p_\theta(x|y,z) + \log p_\theta(y)] \tag{4.33}$$

$$= E_{q(\overrightarrow{z}|\overrightarrow{x},\overrightarrow{y})}[\log p(\overrightarrow{x}|\overrightarrow{y},\overrightarrow{z}) + \log p(\overrightarrow{y}) + \log p(\overrightarrow{z}) - \log q(\overrightarrow{z}|\overrightarrow{x},\overrightarrow{y})] \tag{4.34}$$

$$-\mathscr{U}(\overrightarrow{x}) = \sum_{\overrightarrow{y}} q(\overrightarrow{y}|\overrightarrow{x})(-\mathscr{L}(\overrightarrow{x},\overrightarrow{y})) - q(\overrightarrow{y}|\overrightarrow{x})\log q(\overrightarrow{y}|\overrightarrow{x}) \tag{4.35}$$

4.6 Ladder Network

The ladder network [36] is based on the denoising autoencoder model and uses a carefully crafted ad-hoc function to fit lateral information from the encoding pathway into the corresponding part of the decoder. This model is shown in figure 4.6.

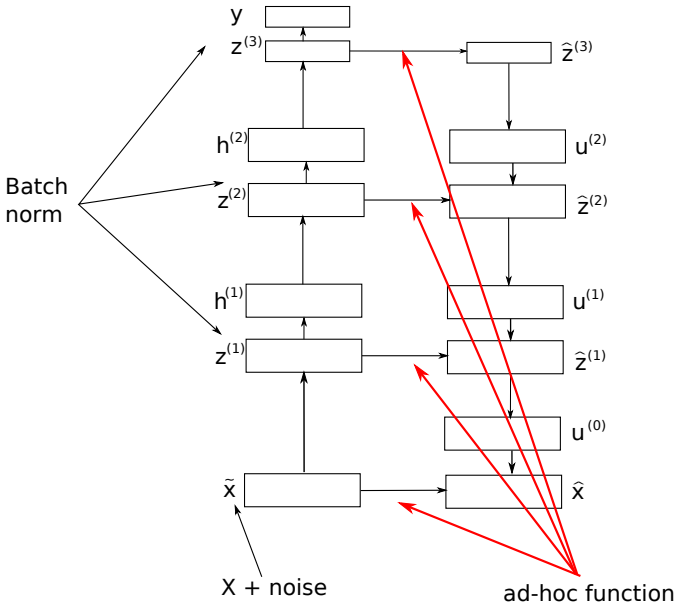


Figure 4.6: The ladder network

Because batch normalization is used, isotropic noise is added to the input, but no additional regularization method is added.

The model tries to simultaneously optimize two objectives: reconstruction (i.e. mapping the noisy model input unto the clean, real input) and classification. The objective for the classification task is given by equation 4.36, while the objective for the reconstruction task is given by equation 4.37, where $\hat{\cdot}$ is the model estimate for $\cdot$, and the $c_\cdot$ and $d_\cdot$ are trained parameters of the model.

$$C_{class} = -\frac{1}{N}\sum_{n=1}^{N}\log(P(Y = t_n|x_n)) \tag{4.36}$$

$$C_{reconst} = \frac{1}{N}\sum_{n=1}^{N}\frac{1}{|x|}||\hat{x}_n - x_n||^2 \tag{4.37}$$

$$\hat{z}_i = a_{i1}z_i + a_{i2}\sigma(a_{i3}z_i + a_{i4}) + a_{i5} \tag{4.38}$$

$$a_{ij} = c_{ij}u_i + d_{ij} \tag{4.39}$$

The full cost optimized during training is $C_{class} + \eta C_{reconst}$, where η is a hyperparmeter that determines the reconstruction task's contribution.

Finally, the layers in the encoding path perform linear transformations after the batch normalization step, and the elementwise interaction between the hierarchical and lateral connections are given by equation 4.38, where σ is the logistic sigmoid, h_0 is taken to be $\hat{x}$, and c and d are parameters of the model.

We refer to this mapping as the "ad-hoc function". It is equivalent to adding the output of a linear transformation with that of a two-layer, sigmoid-affine network, where all the weights are diagonal, and it is designed to encourage the model to learn gaussian distributions [37]. The model can also be interpreted as averaging the samples from two sources (the lateral and horizontal sources) with these activations, where the contribution of each model is weighted implicitely by the learned (diagonal) weights. The restriction on the shape of the weights in this model would create a bottleneck, ensuring perfect information is not transfered to the corresponding decoding layer, which forces the model to learn useful information in its upper layers.

Note that the mapping from $\hat{z}_i$ to u_{i-1}, which is a linear transformation, does not aim to reproduce the content of the layer at the same level in the encoding path, but rather to project $\hat{z}_i$ into the same space as that layer.

CHAPTER 5

AUGMENTED CONVOLUTION

In this chapter, we explore an augmentation to convolutional models that aims to ease training and improve classification performance by learning the mapping operation for data in the space of convolutional outputs into the space of convolutional inputs. In other words, we aim to decouple the steps of learning convolution operators suitable for classification, and of learning the position of output filters so as to accommodate training of further layers in the model by using fully-connected layers that can learn to rearrange filters correctly between convolutional layers.

It is expected that, since the fully-connected layers afford a global view of the patches from the convolution operation at the previous layer to the next convolutional layer, the joint training of convolutional and fully-connected layers ensures that the fully-connected layers will learn to assist the convolutional layers purely as a result of the training process. That is, given that convolutional layers are so powerful on image-structured data, the fully-connected layers will learn to respect this structure due to the form of the optimization problem. Fully-connected layers are not restricted in which inputs they depend on, which make them more suitable to learn global topological structures, while convolutional layers are known to extract local image information extremely well compared to other approaches.

5.1 Convolutional VAE

We used a convolutional VAE where the decoder network was a deconvolutional network (i.e. a network which aims to learn the reverse transformation of the encoder's convolution) based on standard convolutional layers.

This model would encourage the latent $\overrightarrow{z}$ to use an encoding format compatible with image data (i.e. compatible with 2D convolution). To do so, the encoder and

decoder originally followed an intuitive design. Namely, the stack elements were laid out in the same order, with the encoder input being the training data whereas the decoder input was a sample from the latent distribution.

Unexpectedly, this kind of model did not perform very well (the same model using a regular auto-encoder and hamiltonian variational inference [39] does not have this issue). However, inserting a fully-connected layer between the latent sample and the first convolutional layer of the decoder pathway "unlocked" the model, allowing it to perform significantly better.

Our hypothesis was that the fully-connected layer was remapping the sample to achieve a representation more suitable for the convolutional decoder to learn from, without losing information encoded in the latent space. That is, the learning dynamics caused the fully-connected layer to adapt to the convolutional layers above and below to convert the output of one into an input suitable for the other, without any explicit feedback, e.g. from the cost during the training process.

Informal experiments where more fully-connected layers were added between convolutional layers in both the encoder and decoder networks continued to improve model performance, which gave weight to the hypothesis.

In the following sections, we describe experimental results in classification tasks using convolutional networks augmented by interspersed fully-connected layers. In section 5.2, the experimental framework is described. In section 5.3, the model specifics and experimental results on the MNIST dataset are presented. Finally, in section 5.4, the model specifics and experimental results on the CIFAR10 dataset are presented.

5.2 Setup

In the following sections, the models are composed first from simple convolutional layers (thus, the baseline model is a simple CNN). Then, fully-connected

layers are added between pairs of previously connected convolutional layers. These fully-connected layers map an input vector $\overrightarrow{x}$ of size $||\overrightarrow{x}||$ to an output vector of size $||\overrightarrow{x}||$. Each layer except the top layer use rectified linear activations, while the top layer used a softmax activation (cf. equations 4.1 - 4.4).

This rectified linear activation was chosen due to its very good empirical performance both in terms of computation time and in helping a neural network train [30, 32, 48]. The softmax activation was chosen because it readily allows outputs to be interpreted as output probabilities for the possible labels for the data.

The convolutional layers do not use any biases and we did not find a configuration that convincingly improved performance in this setting. The size and stride of the convolutional kernels (that is, the dimensions of the kernel matrix and the offset between convolutional applications on the image, in pixels), as well as the padding to the input before convolution depends on the experiment. The subsampling layers perform maxpooling, i.e. only the maximal activation is conserved within each pooling window.

Cross-entropy loss as in equation 5.1 (where $\overrightarrow{t}$ is the vector of predicted probabilities that the input belongs to a certain class, while $\overrightarrow{y}$ is the ground truth - a zero vector with a single value of one for the class to which the input belongs) was used as the cost to optimize for every model. The optimization method and learning schedule depends on the experiment.

$$\mathscr{L}(\overrightarrow{y}, \overrightarrow{t}) = -\overrightarrow{y} \log \overrightarrow{t} - (1 - \overrightarrow{y}) \log(1 - \overrightarrow{t}) \tag{5.1}$$

5.3 MNIST

For MNIST, each convolution was followed by a pooling layer, and there was no padding of the input. The model's final output was passed through a softmax nonlinearity to yield 10 values (one per label) which can be interpreted as probabilities

that an input be labelled correspondingly. The ADAM algorithm was used for optimization because it performs very well and allows quick convergence with very little hyperparameter tuning [18]. The learning rate was decreased by decaying it to 99% of its previous value once every epoch. Training was stopped when the classification performance on the validation set didn't improve in 10 consecutive epochs. The structure of the model with the fully-connected layers is illustrated in figure 5.6. The results reported are taken from an average of 5 runs with varied seeds so as to reduce selection bias, despite the more popular approach of publishing single-best results.

On MNIST, we additionally test a variant of the proposed approach which replaces fully-connected layers with convolutional layers which feature very large receptive fields (which nevertheless do not completely cover the entire input), in a bid to decrease the computational cost of the approach. The variant of the model with fully-connected layers is refered to as "fc", while the variant using convolutional layers with large receptive fields is referred to as "conv". Baseline refers to the network without either variant applied. The results are summarized in table 5.I.

While the state of the art on MNIST is much better than our method, we do not make use of any data augmentation, preprocessing (such as patch extraction or whitening), ensembling or pretraining, and we train in a purely supervised context. To our knowledge, our model achieves a new state of the art in that category,

Model name	**Error rate on test set**
SOTA (35 convnets + elastic dist + width norm) [8]	*0.23%*
Boosted LeNet4 + distortions [27]	0.7%
Unsupervised on patches + no distortions [35]	**0.60%**
Simard convnet + affine distortions [41]	**0.60%**
Baseline	0.71%
Augmented Convolution (conv)	0.73%
Augmented Convolution (fc)	**0.61%**

Table 5.I: Test set error rate on the MNIST dataset (lower is better).

rivaling and even beating models that use pre-training, heavy preprocessing or affine transformations of the input. Moreover, it is based on a straightforward convolutional model without any tricks, trained with regular backpropagation, with the classical cross-entropy objective, making this method very easy to implement without complications, and potentially adaptable to a wide variety of convolutional structures as it does not a priori need any particular considerations.

From these results, it appears that the fully-connected layers are properly remapping the output of the previous convolutional layer into a representation that is more straightforward to learn from for the next convolutional layer – i.e. it is distorting the space of the output manifold of its previous layer. The relatively poor results when using the convolutional layers seem to imply that the full range of connectivity is required, i.e. that the fully-connected layers aren't merely boosting network capacity, but truly serving a role in restructuring the output of the previous layer.

5.4 CIFAR10

The baseline model we used was a variant of the VGG model (specifically, it is the cifar.torch model [2], and results reported with regard to the baseline come from the results the authors report on this model). Some convolutional layers were directly followed by other convolutional layers, all convolutions used a kernel of size 3×3 followed by a rectifier transformation, and their input was padded by one pixel in each direction (thus, the dimensions of the output was the same as the input along the width and height, but the number of output channels could change). Fully-connected layers were inserted between the 3rd and 4th, and the 4th and 5th convolutional groups (as illustrated in figure 5.5). No fully-connected layers were inserted before that point because of the scaling issues with this method, such that the number of parameters was so large that VRAM consumption exceeded GPU

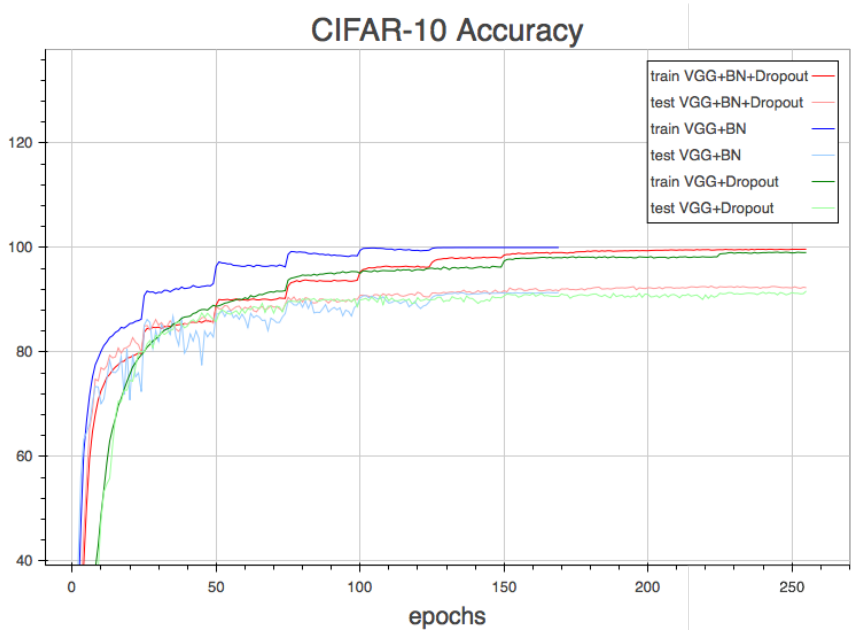


Figure 5.1: Performance on the CIFAR-10 dataset with the baseline model (image taken from [2])

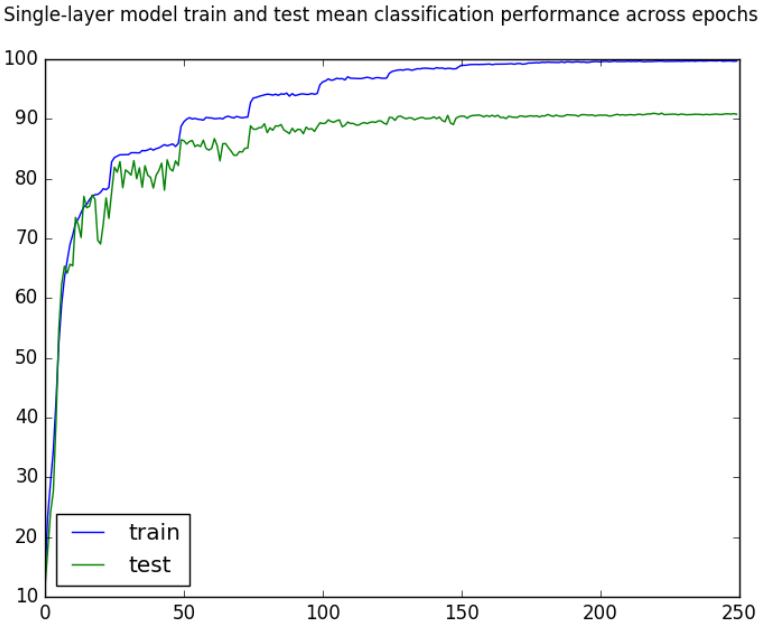


Figure 5.2: Performance on the CIFAR-10 dataset with the augmented single-layer model

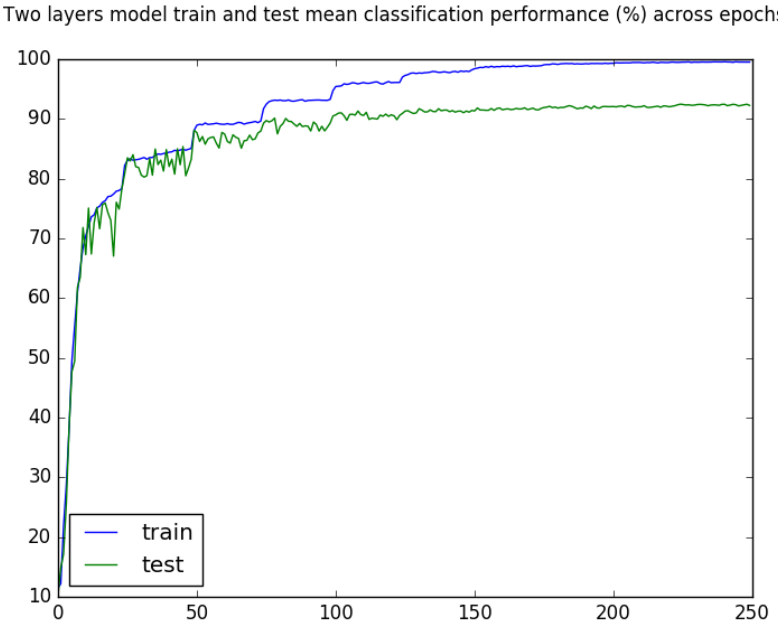


Figure 5.3: Performance on the CIFAR-10 dataset with the augmented two-layers model

capacity.

Training was performed using simple stochastic gradient descent, using a momentum factor of 0.9 and a L2 weight-decay rate of 0.0005. The learning rate was halved every 25 epochs and training was stopped after 250 epochs. Moreover, batch normalization [15] was applied before the activation function on each layer. Dropout was applied after most layers [44]. The model failed to train when batch normalization was removed, regardless of hyperparameter selection, when using stochastic gradient descent. When using the ADAM algorithm, the model was not able to obtain more than 75% classification accuracy, which is why we used SGD in this case. Results of the experiment are summarized in table 5.II.

Note that the best models for classification on CIFAR10 use data augmentation extensively. Without it, most perform about as well as the baseline model, sometimes worse.

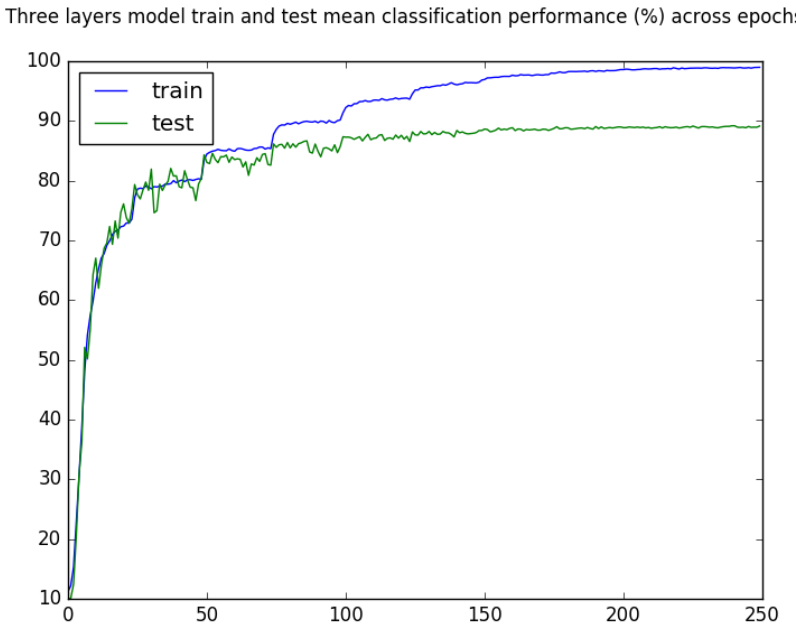


Figure 5.4: Performance on the CIFAR-10 dataset with the augmented three-layers model

Unlike with the MNIST experiment, results on CIFAR10 were not particularly convincing. Not only did the scalability issues of this method limit the experiment, but the best model obtained this way decreased performance over the baseline model (but note that the baseline model was trained without a validation set, so it might be overfitted to the test set). It is not clear why there is such a difference between the results on MNIST and on CIFAR10, but the training conditions being widely different, it is difficult to find one point of failure. Perhaps the MNIST data is more subject to poor manifold fits across convolutional layers, or perhaps this method is more sensitive to training conditions than originally thought.

Varying the amount of fully-connected layers in the network showed that the best performance was achieved by training using 2 layers, while adding an additional layer reduced performance. This suggests that beyond performance-related scaling issues due to the large amount of additional parameters added to the model,

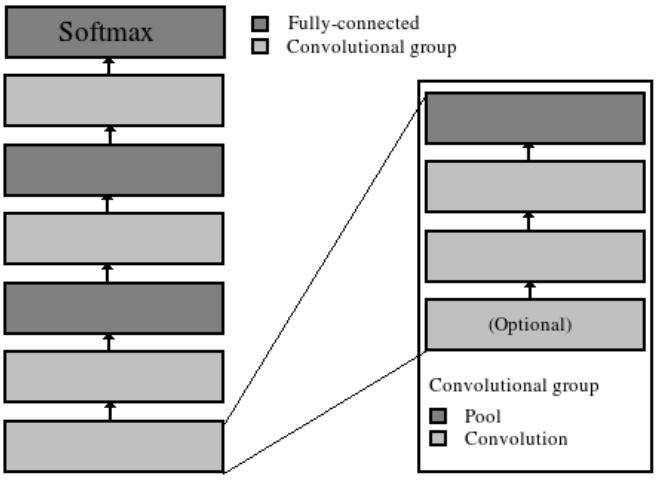


Figure 5.5: The 2-layers augmented model architecture used for CIFAR-10

Model name	**Accuracy on test set**
SOTA (data aug. + multiple passes)[12]	*96.53%*
Aggressive augmentation[43]	95.59%
Maxout + mirroring + rand. shifs[31]	94.16%
Very deep network[14]	93.57%
Baseline (VGG-like + mirroring [2])	92.45%
Augmented Convolution (1 layer)	90.7%
Augmented Convolution (2 layers)	92.3%
Augmented Convolution (3 layers)	89.8%

Table 5.II: Test set accuracy on the CIFAR10 dataset (higher is better)

this method does not scale well in terms of classification performance, at least in this setting. This indicates that models of this form might lack the flexibility required to learn in the general case. While it would be natural to suspect overfitting, regularization methods such as dropout and reducing the model capacity only reduced test-time performance, and train-time performance without these regularization methods does not appear to differ much from that of models that perform better on this task.

Figures 5.1 - 5.4 show the train set and test set performance across training epochs for the augmented models, showing that training proceeds faster with the

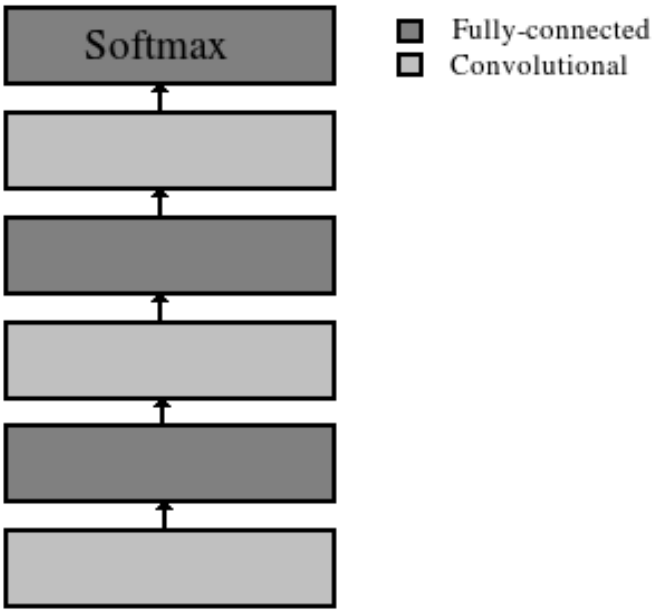


Figure 5.6: The augmented model architecture used for MNIST

augmented model even though it does not reach the same test performance. This could be a consequence of the large increase in model capacity, which help the overfitting process. While training with dropout helped keep overfitting issues mostly in check, it was still not enough to prevent its effect.

5.5 Conclusion

While we were able to obtain excellent results when training the model on the MNIST dataset, this method has shown its limits even on the CIFAR10 dataset, both in terms of scaling, where it was not possible to fully install fully-connected layers between the convolutional stages due to VRAM requirements, and in terms of performance, where the model without the fully-connected layers performed better than the one with these layers.

Due to the wide difference in setting between the CIFAR10 models and the MNIST model, it is hard to draw a conclusion from these experiments, other than

the fact that this method clearly has scaling issues and also does not adapt to arbitrary contexts, despite not being engineered to depend on model specifics.

It is possible that the CIFAR10 dataset has inherent structural properties that the fully-connected layers cannot help with. For example, there is a lot of noisy information in the background of CIFAR10 images as they are taken from natural sources, whereas there are only black pixels in the backgrounds of MNIST digits as they have been cleaned up.

It is also possible that the lack of feedback from the objective function (i.e. all layer roles, such as the expectation for the fully-connected layers to be learning topological information, were achieved without applying a penalty for misbehavior in the objective function - achievement of this goal is up to the optimization process in searching for better performing solutions) was sufficient in simpler networks, but instead induce extra stress in larger networks, where the fully-connected layers would learn too slowly to let the convolutional stages focus purely on convolution.

The fact that learning in the CIFAR10 setting was faster with the fully-connected layers than without, yet stalled sooner, could either be an indication that they do not participate in the training process properly and merely provide additional capacity to the model (the classic overfitting regime), or that they aren't taking the right role (e.g. some units from the fully-connected layers are stuck trying to propagate useful information to the next convolutional layer instead of being purely concerned with learning topology), or not quickly enough. Although the fact that we can still achieve performance similar to the baseline model despite the large addition of extra parameters in the model, which should make the training problem significantly harder, indicates that the fully-connected layers are performing some task that is useful for the model, even if it's not the best task that these layers could perform.

CHAPTER 6

GENERATIVE CLASSIFICATION

To correctly classify data, models need to perform complex transformations from the input format to the desired output format (for example, from image pixel data to a vector representing the likelihood of the image being in each class).

There is a lot of information inherent in the data regardless of the task that we are trying to solve. If the data is non-random, then it can be compressed. If it can be compressed, then the difference in bits between the compressed and original image is inherently non-informative.

Generative models fit distributions to best approximate the data, potentially yielding a much more compressed - and thus informative - representation for the data.

We investigated the use of generative models to improve classification performance on images. The generative model learns to encode the data within a latent space without information specific to any task we may want to complete. This latent space can be designed to be a compression of the input data (getting rid of factors that least explain the data, such as noise in the data caused by the quality of recording instruments, or due to unimportant background information), or can express a more useful representation than the input (such as by separating the factors that explain the data and recombining them in various ways).

The discriminative model performs a prediction based on the data, using the latent representation learned by the generative model. This approach achieves a separation of concerns (the generative model tries to find a coding of the data that is easier to manipulate regardless of the task whereas the discriminative model tries to achieve the task based on a better data representation) which could improve the overall model's ability to perform a given task.

We use the MNIST and Dogs v.s. Cats datasets [10, 25] to evaluate performance while we base our work on the generative semi-supervised semi-supervised generative model [19] and an early version of the ladder network [36]. We attempt to reconcile the classification performance of convolutional neural networks with the generative power of variational autoencoders [17].

6.1 Semi-Supervised-VAE Model

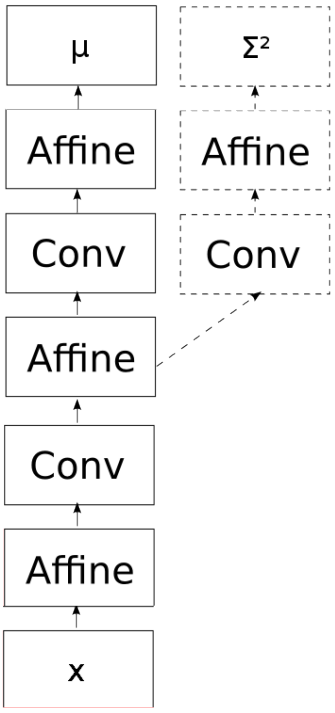


Figure 6.1: The Semi-Supervised-VAE parameter-generating architecture. The dotted path has the same root as the other path and generates Σ^2 whereas the solid path generates μ for the latent z.

The VAE model has shown to be outstanding in its generative capacity. We attempt to merge it with the generative semi-supervised model, which is able to perform classification based on a better latent representation learned during training, so as to further improve such a latent representation and thereby hopefully

improve classification performance.

The basic architecture of the model is very similar to the generative semi-supervised model. However, the encoder and decoder functions for the model (as illustrated in figure 6.1) are learned through a convolutional neural network. We describe the CNNs used in the case of the MNIST task.

For the encoder, we use interspersed affine fully-connected, and convolutional layers in alternation, starting with an affine layer, with convolutional kernel size of 7×7 and a convolutional stride of 1 across either dimension of the image. No pooling is applied, and convolution is performed using "valid" convolution in matlab parlance (i.e. no padding was added to the image). The convolutional layers use a rectified linear activation function.

For the decoder, we use much of the same architecture as in the encoder, but with "full" convolutions instead of "valid", in matlab parlance (i.e. padding is added to fill the convolutional window which leaves the boundaries of the input shape), in order to recover the amount of data that was lost due to convolution with zerod out edges in the encoding pass.

The CNN architecture is illustrated in figure 6.1. We choose this particular interspersed structure because we have found that it improves performance (cf. chapter 5).

Just as for a VAE, we choose prior distributions to sample from to produce samples for the latent factors. For the latent $\overrightarrow{z}$ (i.e. explanatory factors, such as the calligraphic style used to draw a digit), we use a gaussian distribution and learn both its parameter vectors $\boldsymbol{\mu}$ and Σ^2. For the $\overrightarrow{y}$ (i.e. the data's class), we use a multinomial and learn n_{class} parameters. For the data $\overrightarrow{x}$, we use a bernoulli vector and learn its parameter.

Finally, the classification layer is composed of two rectified linear layers followed by a sigmoid layer. The size of the first two layers is $\left\lfloor \frac{n_{visible}}{5} \right\rfloor$ and $\left\lfloor \frac{n_{visible}}{10} \right\rfloor$ respectively.

6.1.1 Model Architecture

We tried two approaches to this problem: generative semi-supervised describes the architecture based on the generative semi-supervised model's organization structure [19], but uses convolutional VAEs as the generative model. 2-headed describes a modified version of the aforementioned model that makes an independence assumption between the two classes of latent variables (the $\overrightarrow{z}$, which are the classical VAE latents, and the $\overrightarrow{y}$, which are the latent class variables).

6.1.1.1 Generative Semi-Supervised Architecture

This model follows the original generative semi-supervised model architecture with few variations. We simply use a VAE and CNNs for the encoder and decoder in the generative model (see figure 4.5). The classical VAE lower bound is modified by adding the expected log probability of the $\overrightarrow{y}$ latents to the bound as in equation 6.1 for each variable, where the probability distributions are modeled, as in the VAE and the generative semi-supervised model, with MLPs, taking the conditioned variables as inputs.

$$-\mathscr{L} = -D_{kl}(q_\phi(z|x,y)||p_\theta(z)) \tag{6.1}$$

$$+E_{q_\phi(z|x,y)}[\log q_\varepsilon(y|x)] \tag{6.2}$$

$$+E_{q_\phi(z|x,y)}[\log p_\theta(x|z,y)] \tag{6.3}$$

$$+E_{q_\varepsilon(y|x)}[\log p_\pi(y)] \tag{6.4}$$

In this setting, the generative model is described in equation 6.5 while the inference model is given in equations 6.6 and on; the $q(\cdot)$ are used during training to infer the corresponding parameters, and the $p(\cdot)$ are represented by the model itself (that is, the model family used is chosen in advance), using the parameters learned by

the inference model and used at generation time.

$$x \sim p_\theta(x|y,z)p_\pi(y)p_\theta(z) \tag{6.5}$$

$$y \sim q_\varepsilon(y|x) \tag{6.6}$$

$$z \sim q_\phi(z|x,y) \tag{6.7}$$

The convolutional architecture confers some advantages over the fully-connected one, namely parameter sharing (allowing us to build larger models) as well as the expressive power of convolutional features.

6.1.1.2 2-headed model

This model simply removes the dependency link between the $\overrightarrow{y}$ and $\overrightarrow{z}$ latent variables in the posterior approximation (see figure 6.2). The generative model is still described by equation 6.5 except with $q_\phi(z|x,y)$ becoming $q_\phi(z|x)$ in the case of one dimension (that is, the reconstruction and classification are made independent processes that share the same encoder stem), while the new inference model can now be described by equation 6.8 and on for individual variables. The independence assumption lets the $\overrightarrow{z}$ latents learn a representation that may include factors that are relevant to both the class and the form of the digit because its independence from the $\overrightarrow{y}$ latents means it does not need to care about this coupling of information when trying to produce good samples. The additional dimensions used to minimize D_{kl} in the $\overrightarrow{z}$ latent space are no longer used to accomodate for good cooperation with the latents of $\overrightarrow{y}$.

$$z \sim q_\phi(z|x) \tag{6.8}$$

$$y \sim q_\varepsilon(y|x) \tag{6.9}$$

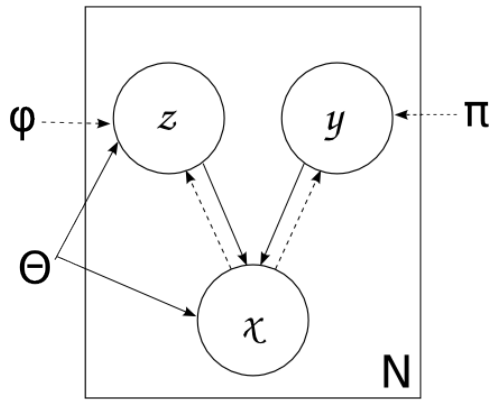


Figure 6.2: The 2-headed model in plate notation. Dashed lines represent the inference path whereas the generative path is illustrated with solid lines.

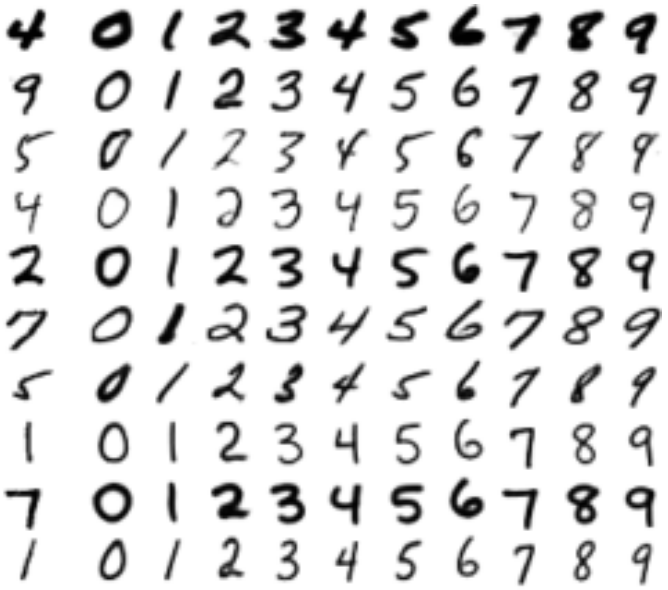


Figure 6.3: Analogies on MNIST digits: the latent z vary only per row while the latent y are varied across columns (image taken from [19]).

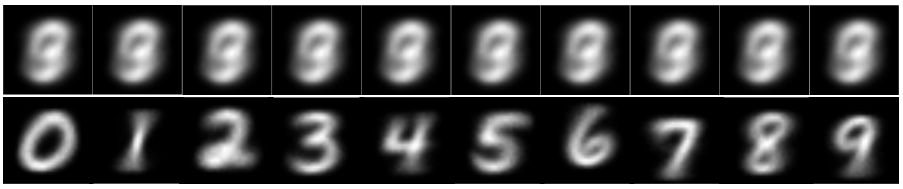


Figure 6.4: M2 (top) and 2-headed model (bottom) samples for each MNIST digit class, from 0 to 9, left to right.

Table 6.I: Test error on the MNIST dataset and the Dogs v.s. Cats dataset (lower is better)

	MNIST	**DogsVsCats**
SOTA [8, 40]	*0.21%*	*1.1%*
generative	2.644%	-
2head	1.2%	32%

6.2 Results

Using the generative semi-supervised model as described and using CNNs for the encoder and decoder did not yield positive results. The samples were pure noise (beside the biases), and while classification was reliable, the model did not seem able to pass the 2% error mark on binarized MNIST. It is unclear where the problem lies. In order to obtain sensible samples, it was necessary to modify the model by removing the dependency between the $\overrightarrow{y}$ and $\overrightarrow{z}$ latent variables. Class-conditional samples averaged over 100 trials are shown in figure 6.4.

The 2-headed model was able to yield close to 1% classification error on the binarized MNIST dataset, which is a step up from the generative semi-supervised model with VAE and CNN.

Under this model, it is possible to obtain class-conditional samples of MNIST digits. However, $\overrightarrow{z}$ seems to be completely ignored during the generative path as seeding the $\overrightarrow{z}$ to arbitrary values yield almost identical class-conditional samples. This problem has been noted by myself and others (personal communications) when using generative models such as variational autoencoders with more than one

independent layers. The original generative semi-supervised model as described in [19] suggests that it's the dependency between layers that permits learning co-adapted features, such as the latent $\overrightarrow{z}$ encoding brush strokes while the latent $\overrightarrow{y}$ encode the class, such as in figure 6.3.

This model was also applied to the kaggle competition's Dogs v.s. Cats dataset, where it was able to overfit the training set completely while performing very poorly elsewhere (at best, it could reach around 36% on a validation set obtained from holding out 10% of the original training set, at which point the training performance was of around 32% error).

The results collected on the different models are summarized in table 6.I.

6.3 Discussion of the Results

It is clear that in these experiments, the dependency between $\overrightarrow{z}$ and $\overrightarrow{y}$ is detrimental to learning, despite evidence suggesting the opposite should be true. it is possible that the fully labeled nature of these tasks cause trouble for the VAE architecture, although it is also possible that the generative semi-supervised model featuring two branches that learn independently from the same layer inside the model causes issues in the VAE framework.

6.4 VAE-Ladder Model

The ladder network is one of the few semi-supervised models that show good results in the fully-supervised case. In fact, the ladder network is able to effectively reach state-of-the-art performance on the MNIST dataset. However, it is based on a denoising autoencoder model and we would like to improve the ladder network by using a VAE instead (since VAEs are known for their generative performance).

In this section, we develop and train a VAE model hybridized with the ladder network to provide a more unified architecture where the generative and discrim-

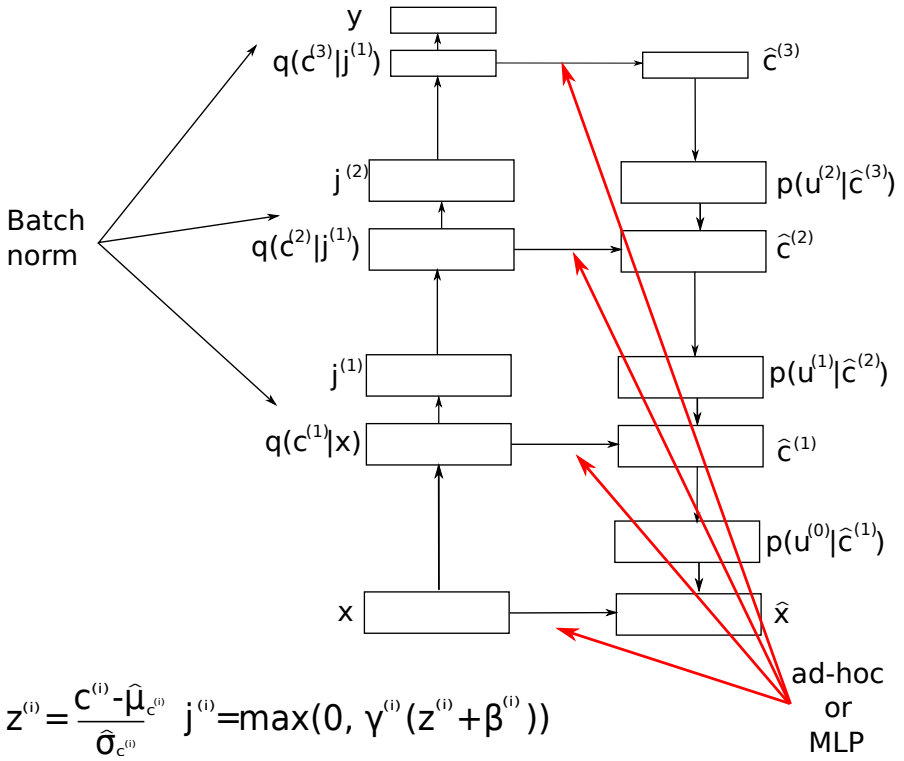


Figure 6.5: The VAE-Ladder hybrid.

inative tasks are more closely intertwined in an attempt to provide enough drive for the generative part of the model to learn useful features for discrimination, while at the same time encouraging the discriminative part to adapt to the latent representation learned by the generative part of the model.

The model in figure 6.4 shows a hybrid adaptation of the original model working inside the VAE framework: sampling is performed at every layer in the network for the lateral pathway, while the mean of the distribution is used as input to further layers on the encoder pathway. Batch-normalization is applied at every layer in the encoding path, as illustrated. Whereas in the original model, the mapping from individual units $\hat{z}_i$ to $\hat{c}_i$ is performed by a linear transformation that achieves a basis change, we now adopt a probabilistic interpretation of the model and map samples $\hat{z}_i$ to the distribution of $\hat{c}_i$ and then obtain a sample from that distribution.

Layers in the model use the rectified linear activation, except the class prediction and reconstruction. The reconstruction is output by a logistic sigmoid activation, which preserves the range of valid values for grayscale pixels. A softmax outputs the class prediction, encoding a categorical prior.

In one formulation, the ad-hoc function is used to transfer lateral information. In the other, a small MLP is used instead (that is, the lateral information is passed through a bottleneck that trains alongside the rest of the model [36]).

The lateral $\overrightarrow{z}$ use a gaussian prior in all the experiments. The KL divergence (as seen in equation 6.10, where D is the parameter from the data, and θ is the parameter of the model) with q being the gaussian distribution gives the D_{KL} term for the cost as in equation 6.11. This D_{KL} is with regard to z rather than x and thus the "data" parameter D is actually the prior parameter for the model, allowing us to use $p(\theta|D) = \mathcal{N}(0,I)$ and $q(\theta) = \mathcal{N}(\mu, \Sigma^2)$ from which we obtain the expression.

$$D_{kl}(q(\theta)||p(\theta|D)) = -\sum q(\theta)\log p(\theta|D) + \sum q(\theta)\log q(\theta) \tag{6.10}$$

$$-\frac{1}{N}\sum_{i}\sum_{n}^{N}\left[-\log(\sigma_i)-0.5+0.5\exp\{2\log(\sigma_i)\}+\mu_i^2\right] \tag{6.11}$$

When the model is tuned for generation, the part of the cost associated with reconstruction is given by equation 6.12. For classification, better results were obtained by using equation 6.13, which encodes a gaussian structure (note that the cost of equation 6.12 is only valid for values in the closed $[0,1]$ interval, whereas the cost of equation 6.13 is valid over $\mathscr{R}$. Since image data is kept in the $[0,1]$ range, we can freely use either in this case). In these equations, x_n is the input's nth value whereas $\hat{x}_n$ is the reconstruction's nth value.

$$-\sum_{i}\left[x_i\log(\hat{x}_i)+(1-x_i)\log(1-\hat{x}_i)\right] \tag{6.12}$$

$$-\sum_{i}||x_i-\hat{x}_i||^2 \tag{6.13}$$

The class penalty for the models is the binary cross-entropy cost. The total cost for training ($\mathscr{L}(x)$) is the sum of terms given by equation 6.16 in the classification case, where y_i is the label for the ith datum and $\hat{y}_i$ is the model likelihood for that class.

$$\mathscr{L}(x)=-\sum_{i}||x_i-\hat{x}_i||^2 \tag{6.14}$$

$$-\frac{1}{N}\sum_{i}\sum_{n}^{N}\left[-\log(\sigma_i)-0.5+0.5\exp\{2\log\sigma_1\}+\mu_i^2\right] \tag{6.15}$$

$$-\sum_{i}\left[y_i\log(\hat{y}_i)+(1-y_i)\log(1-\hat{y}_i)\right] \tag{6.16}$$

6.5 Results

Ztop means only the layerwise cost associated with the top z layer is actually used (the others being fixed at zero). Unfortunately, we were not able to obtain particularly good error rates on MNIST using this model. Although the ladder

Model name	Error rate on test set
VAE+Ladder+MLP+ztop	0.99%
VAE+Ladder+ad-hoc	3.86%
VAE+Ladder+MLP	1.92%

Table 6.II: Test set error rate on the MNIST dataset (lower is better). MLP uses an MLP instead of the ad-hoc function for lateral connections. Ztop disconnects lateral costs except at the top layer.

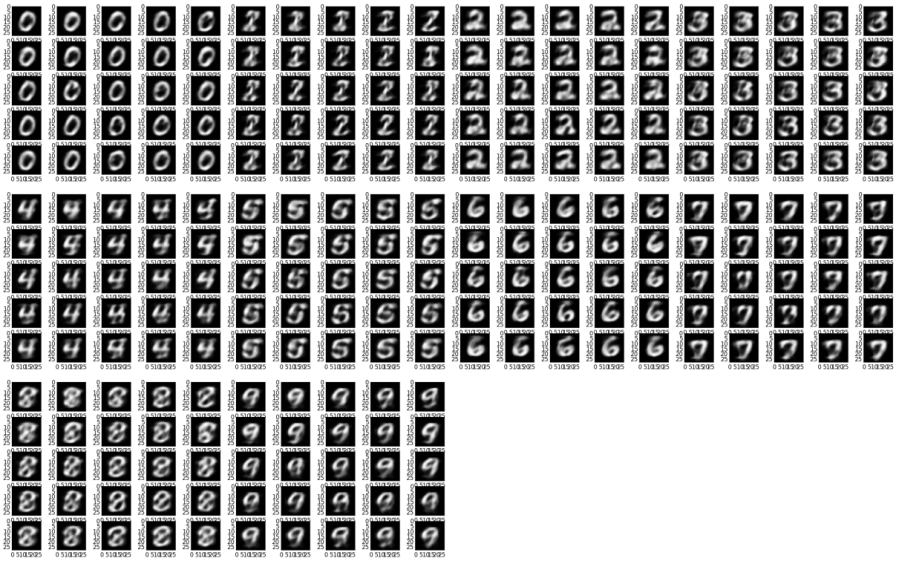


Figure 6.6: Samples from a VAE-Ladder hybrid model tuned for generation. The label is kept fixed to generate a specific digit.

network achieves state-of-the-art results in the position-invariant case for MNIST, our VAE-Ladder hybrid remained far from achieving similar performance. A summary of results is given in table 6.II. While the model could be tuned to generate decent (although still poor) samples, as in figure 6.5, the same could not be said for classification. It appears that the combination of the classification and generation objectives can benefit generation, but not classification.

As a control experiment, a completely different combination of VAE and ladder networks was tested: the model was a VAE at its core, with the encoder and decoder paths being composed of a ladder network. Samples from the latent z were input at the top of the decoding part of the network, and were computed in the usual way, where the last layer of the encoding path served as input to two independent sigmoid layers that generated the mean and variance parameters of the gaussian latent z. The ladder objective was used for all layers except the last decoding layer (that is, the one which output is $\hat{x}$). The VAE objective was added to this ladder objective to yield the total cost optimized during training.

That experiment showed significantly better results than the previous model, achieving 0.7% performance. However, the fact that this model still performed worse than the pure ladder network suggests, along with the previous results, that the VAE framework is unsuitable for combination with the ladder network.

6.6 Conclusion

We were not able to train a VAE-Ladder hybrid on MNIST data to improve performance beyond the ladder network baseline despite the various approaches we employed to attain this goal.

One possible explanation as to why is that multilayer VAE architectures are very hard to train, with early layers typically learning quickly while later layers do not learn anything at all. However, by manipulating the scaling factor for each

part of the cost, it is possible to make any amount of layers train (i.e. the per-layer D_{KL} – that is, the D_{KL} term from equation 6.11, which is applied on each layer in the same way as the standard ladder network layerwise cost – improves on each layer), so this does not seem to be the issue in this case.

Another possible explanation is that either too much, or not enough information is transmitted by the lateral connections. Since further layers can learn, it does not seem plausible that the lateral pathways would leak too much information. If they did not leak enough information for the decoder to easily perform its job, it would be natural that features that would be useful for classification are discarded by the encoder in order to distribute features useful to generation along the lateral path through multiple levels. However, manipulating the scaling factors for the cost parts again, which would encourage the model to prioritise classification over generation, has not been able to improve performance, suggesting that the VAE-Ladder forward structure itself is flawed (as otherwise, the model could lean toward a pure classification setting).

Additionally, it is possible that training to obtain great features that fit both a generative and a classification objective at the same time is not viable. Namely, since the reversed setting (i.e. optimizing the structure for generation from the get go and further tuning for generation performance) can work successfully [21], it is likely that when training for classification by learning a latent space fit for generation at the same time, the resulting latent space is a sort of middle ground that is unsuitable for either tasks (as opposed to first learning a latent space that is suitable for generation and then performing classification on it, which is known to work [19], and as opposed to training for generation by augmenting the model with a classification objective, which may simply drive the model to learn a structured latent space instead of reaching that middle ground).

CHAPTER 7

PIXEL-RNN EXPLORATION

We explore the pixel-rnn model[45] and attempt to improve upon it by applying batch-normalization to it, adapted for use in recurrent networks[22].

7.1 Architecture with Batch-Normalization

The model is organized like the pixel-rnn, where the input is first passed through a masking convolution which prevents the network from seeing future pixels. The output of this convolution is then passed through two LSTMs, forming a diagonal BiLSTM; that is, a couple of LSTMs scanning the image from opposing corners at the top toward opposing corners at the bottom, in diagonal, as in the pixel-rnn.

Batch normalization is applied to the inputs-to-state of the LSTMs from the base pixel-rnn model: at training time, batch statistics are calculated and a rolling sums for the variance and average of the inputs-to-state are maintained. Those same train statistics are applied at test time (i.e. as opposed to using testing set batch statistics), as is usual. Parameters γ and β are learned during training, and the final inputs-to-state representation is obtained as in equation 7.1.

$$\hat{i} = \gamma\left(\frac{\overrightarrow{i} - || \overrightarrow{i} ||}{\sigma_{\overrightarrow{i}}}\right) + \beta \tag{7.1}$$

Training then proceeds as with the standard pixel-rnn model.

7.2 Results

Figure 7.1 shows the training curves for the model with and without batch normalization. Additionally, test-set performance results are summarized in table 7.I.

Table 7.I: NLL on test set for binarized MNIST (lower is better) [13, 45]

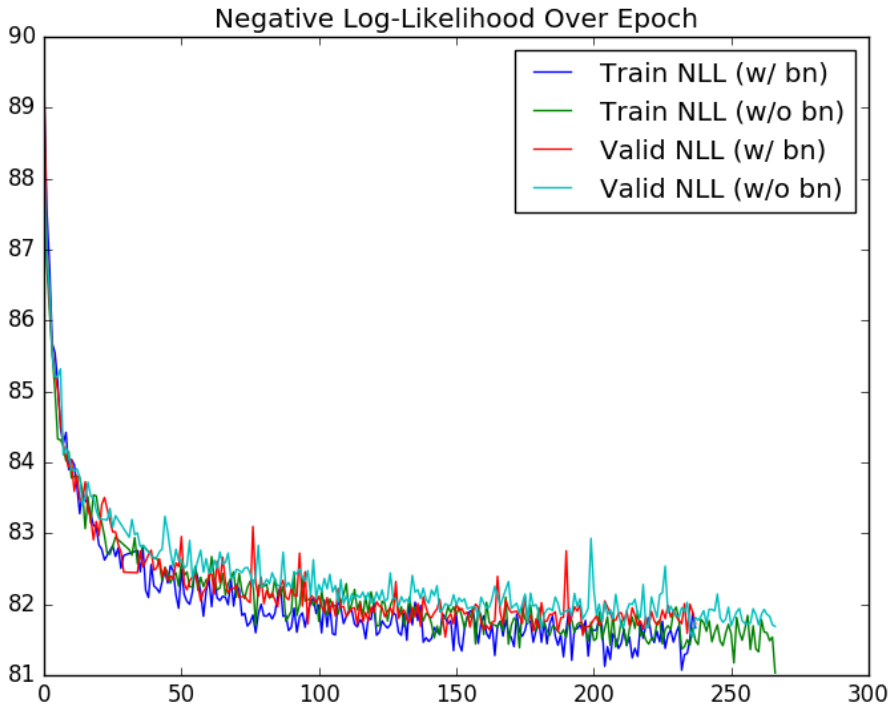
	NLL =	**NLL ≤**
MADE 2 layers, 32 masks	86.64	-
DRAW w/o attention	-	87.40
DRAW w/ attention	-	80.97
pixel-rnn 1 layer	80.75	-
pixel-rnn 7 layers	**79.20**	-
pixel-rnn reprod 1 layer	*81.2198*	-
pixel-rnn reprod, bn 1 layer	*81.2027*	-

When batch normalization is used, the model reaches train-set NLL values close to 81.5 much earlier than when batch normalization is not used in terms of epochs. Validation performance tends to match the same pattern during training. Final test performance is not significantly different when using batch-normalization or not. However, on the same GPU, the model with batch normalization took 0.573 seconds on average to complete one iteration, whereas without batch normalization, one iteration took 0.482 seconds on average.

7.3 Conclusion

We observe that adding batch normalization slows the model down in terms of wallclock time, but improves convergence speed over epoch after a "burn-in" period, before which the model trains somewhat slower. This slowdown is likely associated with the fact that the model has yet to learn good values for the γ and β batch normalization parameters. Batch normalization does not seem to help much with regard to generalization performance. Due to the increase in training speed in epochs provided by batch normalization, in the setting the model was trained, significant wallclock time is saved by applying it to the model.

Figure 7.1: Training and validation set error over epoch, with and without batch normalization



CHAPTER 8

CONCLUSION

In this thesis, we have explored various methods to influence the properties of the latent representations learned by various models. We have proposed an augmented convolution model which is capable of achieving state-of-the-art results in the setting it was employed in, on the MNIST dataset, but this model was prone to scaling issues. We have successfully modified VAEs to have their latent spaces adopt image-like structures. However, we were not able to improve the classification performance of ladder networks using the generative capacity of VAEs.

We have developed a method to decouple the structure and transformation stages of convolutional layers, providing a boost in performance on simple datasets like MNIST, but we have not been able to show generalized results on more complex datasets. This suggests that while the two tasks convolutional layers must perform simultaneously to achieve a goal is non-trivial, learning the tasks separately might be even harder as the data increase in complexity. Moreover, the approach chosen showed poor resource scaling and would not work inside larger models.

Our experiments strongly suggest that it is difficult to train a model to both perform well for generation and for classification at the same time, with the ultimate goal to improve classification performance, as if the model had to settle in a bad middle-ground. Yet, it is possible to improve upon VAEs' generative ability by augmenting them with a classification objective. The ability of the ladder network to achieve great performance by combining generative and discriminative objectives suggested that improving the generative part of the model should improve its performance for classification, but we were not able to do so. Determining the exact mechanics which makes it hard to train such a model, but not the standard ladder network, may be the key to building the next generation of high-performance

discriminative models: while we would have hoped that the generative objective of the model could act as a kind of regularizer to force the model to learn more salient features while preventing it from focusing on factors that are too specific to the data used for training (that is, we would have hoped that this would help the model generalize better), thus improving overall performance, it instead seemed that the objectives were in conflict and the model was stuck in a middleground that was good at neither task.

Further research in this direction could prove to be very useful in improving the quality of generative models, including learning in semi-supervised setting. Since we have access to very large amounts of data, but only a small fraction of it typically has useful labels, and due to the resource cost associated in labeling the data with missing labels, it is clear that improving the quality of semi-supervised or unsupervised algorithms would bring high value to machine learning efforts.

There are two main directions to pursue on this end: scalability (developing methods to separate the two convolutional tasks without requiring significantly more variables than in the initial model), and generalization (whether it is possible to generalize the findings to more complex and larger data).

Finally, we have shown the effects of batch-normalization on the Pixel-RNN model's training process, demonstrating that it can notably, though not significantly, improve network convergence speed.

We have explored many models and methods from a latent-space manipulation point of view. We have demonstrated that various latent space manipulations and cross-layer interactions can have a large impact on various characteristics of a model. Inducing certain properties in a latent space can provide boosts in performance (such as in the augmented convolution).

Further research in ways to bridge the gap between the latent space requirements for varying goals, such as models or training algorithms that enable the capture of multiple, or of much more complex latent spaces, may enable us to develop better

models for fully-supervised (and perhaps semi-supervised) tasks.

BIBLIOGRAPHY

[1] Cifar-10 and cifar-100 datasets. https://www.cs.toronto.edu/~kriz/cifar.html, . Accessed: 2016-04-22.

[2] 92.45% on cifar-10 in torch. http://torch.ch/blog/2015/07/30/cifar.html, . Accessed: 2016-04-21.

[3] Data - dogs vs. cats | kaggle. https://www.kaggle.com/c/dogs-vs-cats/data, . Accessed: 2016-04-22.

[4] Public leaderboard - dogs vs. cats | kaggle. https://www.kaggle.com/c/dogs-vs-cats/leaderboard/public, . Accessed: 2016-04-22.

[5] Classification datasets results. https://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html. Accessed: 2016-04-22.

[6] Yoshua Bengio and Yann LeCun. Scaling learning algorithms towards ai. In Leon Botton, Olivier Chapelle, Dennis DeCoste, and Jason Weston, editors, *Large-Scale Kernel Machines*. MIT Press, 2007.

[7] Ronald N. Bracewell. *The Fourier Transform and Its Applications*. McGraw-Hill, New York, 3rd edition, 1999.

[8] Dan C. Ciresan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. *CoRR*, abs/1202.2745, 2012. URL http://arxiv.org/abs/1202.2745.

[9] Yann N Dauphin, Harm de Vries, Junyoung Chung, and Yoshua Bengio. Rmsprop and equilibrated adaptive learning rates for non-convex optimization. *arXiv preprint arXiv:1502.04390*, 2015.

[10] Jeremy Elson, John R. Douceur, Jon Howell, and Jared Saul. Asirra: A captcha that exploits interest-aligned manual image categorization. In *Proceedings of 14th ACM Conference on Computer and Communications Security (CCS)*. Association for Computing Machinery, Inc., October 2007. URL http://research.microsoft.com/apps/pubs/default.aspx?id=74609.

[11] Felix A. Gers, Jürgen A. Schmidhuber, and Fred A. Cummins. Learning to forget: Continual prediction with lstm. *Neural Comput.*, 12(10):2451–2471, October 2000. ISSN 0899-7667. doi: 10.1162/089976600300015015. URL http://dx.doi.org/10.1162/089976600300015015.

[12] Benjamin Graham. Fractional max-pooling. *CoRR*, abs/1412.6071, 2014. URL http://arxiv.org/abs/1412.6071.

[13] K. Gregor, I. Danihelka, A. Graves, and D. Wierstra. Draw: A recurrent neural network for image generation. *ArXiv e-prints*, February 2015.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

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

[16] Michael I Jordan, Zoubin Ghahramani, Tommi S Jaakkola, and Lawrence K Saul. An introduction to variational methods for graphical models. *Machine learning*, 37(2):183–233, 1999.

[17] D. P Kingma and M. Welling. Auto-Encoding Variational Bayes. *ArXiv e-prints*, December 2013.

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

[19] Diederik P. Kingma, Danilo Jimenez Rezende, Shakir Mohamed, and Max Welling. Semi-supervised learning with deep generative models. *CoRR*, abs/1406.5298, 2014.

[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. URL `http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf`.

[21] Alex Lamb, Vincent Dumoulin, and Aaron Courville. Discriminative Regularization for Generative Models. *ArXiv e-prints*, February 2016.

[22] C. Laurent, G. Pereyra, P. Brakel, Y. Zhang, and Y. Bengio. Batch Normalized Recurrent Neural Networks. *ArXiv e-prints*, October 2015.

[23] Yann LeCun. Efficient learning and second order methods. In *Tutorial presented at Neural Information Processing Systems*, volume 5, page 49, 1993.

[24] Yann LeCun and Yoshua Bengio. Convolutional networks for images, speech, and time-series. In Michael A. Arbib, editor, *The handbook of brain theory and neural networks*. MIT Press, 1995.

[25] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. `http://yann.lecun.com/exdb/mnist/`. Published: 1999.

[26] Yann LeCun and John S. Denker. Natural versus universal probability complexity, and entropy. *IEEE Workshop on the Physics of Computation*, pages 122–127, 1992.

[27] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86 (11):2278–2324, 1998.

[28] Anat Levin, Nadler Boaz, Fredo Durand, and William T. Freeman. Patch complexity, finite pixel correlations and optimal denoising. *Computer Vision*, pages 73–86, 2012.

[29] Z. C. Lipton, J. Berkowitz, and C. Elkan. A Critical Review of Recurrent Neural Networks for Sequence Learning. *ArXiv e-prints*, May 2015.

[30] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *ICML*, volume 30, page 1. ICML, 2013.

[31] Dmytro Mishkin and Jiri Matas. All you need is a good init. *CoRR*, abs/1511.06422, 2015. URL `http://arxiv.org/abs/1511.06422`.

[32] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, 2010.

[33] Yurii Nesterov et al. Gradient methods for minimizing composite objective function. Technical report, UCL, 2007.

[34] Andrew Y Ng. Feature selection, l1 vs. l2 regularization, and rotational invariance. In *Proceedings of the twenty-first international conference on Machine learning*, page 78. ACM, 2004.

[35] Christopher Poultney, Sumit Chopra, Yann L Cun, et al. Efficient learning of sparse representations with an energy-based model. In *Advances in neural information processing systems*, pages 1137–1144, 2006.

[36] A. Rasmus, H. Valpola, and T. Raiko. Lateral Connections in Denoising Autoencoders Support Supervised Learning. *ArXiv e-prints*, April 2015.

[37] Antti Rasmus. Semi-supervised learning with ladder network, 2015. Presented at the 29th Annual Conference on Neural Information Processing Systems (NIPS).

[38] Nicolas L Roux, Yoshua Bengio, Pascal Lamblin, Marc Joliveau, and Balázs Kégl. Learning the 2-d topology of images. In *Advances in Neural Information Processing Systems*, pages 841–848, 2008.

[39] T. Salimans, D. P. Kingma, and M. Welling. Markov Chain Monte Carlo and Variational Inference: Bridging the Gap. *ArXiv e-prints*, October 2014.

[40] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *CoRR*, abs/1312.6229, 2013.

[41] Patrice Y Simard, Dave Steinkraus, and John C Platt. Best practices for convolutional neural networks applied to visual document analysis. In *ICDAR*, page 958. IEEE, 2003.

[42] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[43] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin A. Riedmiller. Striving for simplicity: The all convolutional net. *CoRR*, abs/1412.6806, 2014. URL `http://arxiv.org/abs/1412.6806`.

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

[45] A. van den Oord, N. Kalchbrenner, and K. Kavukcuoglu. Pixel Recurrent Neural Networks. *ArXiv e-prints*, January 2016.

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

[47] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[48] Matthew D Zeiler, Marc'Aurelio Ranzato, Rajat Monga, Min Mao, Kun Yang, Quoc Viet Le, Patrick Nguyen, Alan Senior, Vincent Vanhoucke, Jeffrey Dean, et al. On rectified linear units for speech processing. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 3517–3521. IEEE, 2013.