UNIVERSITÄT ZU LÜBECK

From the Institute for Neuro- and Bioinformatics
of the University of Lübeck
Director: Prof. Dr. rer. nat. Thomas Martinetz

Recurrent Neural Networks for Discriminative and Generative Learning

Dissertation
for Fulfillment of
Requirements
for the Doctoral Degree
of the University of Lübeck

from the Department of Computer Sciences

Submitted by
Stanislau Semeniuta
from Gomel, Belarus
Lübeck 2019

First referee: Prof. Dr.-Ing. Erhardt Barth
Second referee: Prof. Dr.-Ing. Alfred Mertins
Chairman: Prof. Dr. Philipp Rostalski

Date of oral examination: 05 July 2019

Approved for printing. Lübeck, 16 July 2019

Abstract

In this work we study Recurrent Neural Networks applied to various problems in Natural Language and Computer Vision. We propose a general purpose extension of the dropout regularization technique applicable to Recurrent Neural Networks, introduce a recurrent model for classification of natural images and discuss a number of applications of generative models to natural texts.

We present a novel approach to Recurrent Neural Network regularization, that differs from the widely adopted dropout method, which is applied to *forward* connections. Instead, we propose to drop neurons directly in *recurrent* connections in a way that does not cause loss of long-term memory. Our approach is as easy to implement and apply as the regular feed-forward dropout and we demonstrate its effectiveness for Long Short-Term Memory networks, the most popular type of Recurrent Neural Networks. Our experiments on Natural Language Processing benchmarks show consistent improvements even when combined with conventional feed-forward dropout.

We then apply a fully differentiable Recurrent Model of Visual Attention to unconstrained real-world images. We propose a deep recurrent attention model and show that it can successfully learn to jointly localize and classify objects. We evaluate our model on multiple digit images generated from MNIST data, Google Street View images, and a fine-grained recognition dataset of 200 bird species, and show that its performance is either comparable or superior to that of conventional Convolutional Neural Networks.

Lastly, we explore applications of generative models to natural texts. We show the effects of architectural choices on learning a variational autoencoder for text generation. We propose a novel hybrid architecture that blends fully feed-forward convolutional and deconvolutional components with a recurrent language model and experimentally show that our architecture exhibits several attractive properties such as faster run time and convergence, and the ability to better handle long sequences. We then discuss the evaluation of Generative Adversarial Networks when applied to language generation. In this work, we argue that this often misrepresents the true picture. We propose a novel evaluation protocol and show that, under this evaluation, and find that neither of the considered state-of-the-art models performs convincingly better than older conventional ones.

Kurzfassung

Diese Arbeit befasst sich mit der Anwendung von rekurrenten neuronalen Netzwerken auf Problemstellungen in den Domänen der Sprachverarbeitung und Computer Vision. Sie stellt eine allgemein nutzbare Erweiterung der Dropout-Regularisierungs Technik bei rekurrenten Netzwerken vor, sowie ein rekurrentes Modell zur Klassifikation natürlicher Bilder. Anschließend werden generative Modelle zur automatischen Erstellung natürlicher Texte evaluiert und diskutiert.

Zunächst präsentieren wir einen neuen Ansatz zur Regularisierung rekurrenter neuronaler Netzwerke, der sich von der allgemein bekannten Technik des Dropouts für Feed-Forward Netzwerke maßgeblich unterscheidet. Unsere Methode schaltet Neurone für den jeweiligen Trainingsschritt auf eine Art aus, die es ermöglicht, die Langzeitgedächtnis-Signale zu erhalten und dabei weiterhin so einfach zu nutzen ist wie das konventionelle Dropout. Anhand der populären Long Short-Term Memory Netzwerke wird die Effektivität der Methode verifiziert. Die Experimente mit Natural Language Processing zeigen durchgängig eine Verbesserung in der Qualität, sogar bei Benutzung des Feed Forward Dropouts.

Der darauffolgende Teil widmet sich der Verarbeitung natürlicher Bilddaten jeglicher Art mit einem vollständig differenzierbaren rekurrenten Modell, welches auf visueller Aufmerksamkeit basiert. Dieses von uns entwickelte, tiefe, rekurrente und Aufmerksamkeits-basierte Modell schafft, es Objekte in Bildern gleichzeitig zu klassifizieren und zu lokalisieren. Die Evaluation des Modells befasst sich mit mehrstelligen Zahlen aus dem MNIST-Datensatz, Hausnummern enthalten im Google Street View Datensatz, sowie einer feingranularen Erkennungsaufgabe von 200 verschiedenen Vogelarten und zeigt vergleichbare oder bessere Ergebnisse gegenüber konventionellen faltungsbasierten Methoden.

Abschließend befasst sich die Arbeit mit Anwendungen generativer Modelle bei natürlichen Texten. Zum einen wird der Effekt von Architekturunterschieden bei Text Generierung mit Variational Autoencodern untersucht. Des Weiteren wird ein hybrides Modell welches Faltungs- und Entfaltungs-Methoden der Feed Forward Netzwerke mit rekurrenten Sprach modellen kombiniert vorgestellt, das, anhand von Experimenten belegt, eine Vielzahl vorteilhafter Eigenschaften aufweist, darunter eine schnellere Laufzeit und bessere Verarbeitung langer Sequenzen. Es folgt eine Diskussion über die Evaluationsmethoden von Generative Adversarial Networks im Kontext der Sprachgenerierung. Die Arbeit zeigt auf, dass einige der genutzten Metriken die wahre Qualität der Modelle nicht akkurat wiedergeben und stellt diesen ein neues Evaluationsprotokoll gegenüber. Unter dem Gesichtspunkt einer erneuten Evaluation mit diesem Protokoll können Modelle die momentan als State of the Art betrachtet werden sich nicht überzeugend von

konventionellen Methoden abheben.

Contents

1 Introduction

1.1 Motivation

Computer Science in general is concerned with creating efficient algorithms for data processing. It is a crucial part of modern world and has enabled numerous advances in diverse set of domains, e.g. finance, health-care, logistics, etc. Traditionally, an algorithm is explicitly coded using a programming language. However, such an approach has proven to be too difficult to use for various pattern recognition tasks, e.g. classifying an object in an image. Indeed, it is virtually impossible to explicitly discover and encode relationships between pixels that make an image look like a certain object.

Among other motivations, this has led to increased interest in a subfield of Computer Science called Machine Learning. It is concerned with automatic discovery of algorithms from raw data, such as input-output pairs. Machine Learning has achieved a number of exceptional results, unavailable by conventional programming, in the past. In the recent years it has become even more prominent due to increased sizes of dataset and available computational resources. In the field of pattern recognition, especially in natural data such as images, Deep Machine Learning has achieved substantial results. In its core are a variety of Artificial Neural Networks models trained with backpropagation algorithm. One such model is a Recurrent Neural Network.

The Recurrent Neural Network (RNN) is a general purpose deep machine learning model for sequences that maintains internal state and processes inputs one sequence element at a time. The presence of a loop over the internal state makes recurrent networks similar to regular computer programs when compared to a feedforward network that implement regular functions. The downside of such an approach, however, is more difficult training. RNN come in a number of flavors, each more or less difficult to train. A notable example are Long Short-Term Memory (LSTM) networks [49] that have become a

de-facto standard tool for machine learning researchers dealing with sequences.

Given the success of Recurrent Neural Networks in different domains it is plausible that further improvements would yield even better results. Hence, in this work we attempt to improve RNNs as a general purpose algorithm and apply them in novel ways to natural texts and images.

1.2 Contributions

This work is concerned with RNN based machine learning algorithms applied to natural images and texts. Such work can be very broadly divided into three major directions: (i) general-purpose improvements of algorithms; (ii) introductions of RNNs into domains where another approach is prevalent; and (iii) improvements in domains where RNNs are already present and accepted. In our work we follow all three paths. Main contributions of the thesis are as follows:

- We introduce a novel approach to dropout [48] regularization in RNNs. Dropout is a successful algorithm allowing to improve performance of a feedforward neural network and has been used to achieve state-of-the-art results in large-scale image recognition [69]. However, its application to recurrent neural networks have been limited to the use in only forward connections [15, 132]. We identify the root cause of difficulty of successful training of RNNs with dropout and introduce a version that does not suffer from this issue. We experimentally verify its effectiveness on a number of Natural Language Processing (NLP) tasks. This work has been presented at the COLING 2016 conference [107] and has had substantial impact in the field, e.g. it has been used to achieve state-of-the-art results in character-level language modeling [44], and accumulated a total of 80 citations [1].

- We then consider the application of RNNs to natural image processing, specifically classification. The currently dominant approach to image classification is based on training a deep Convolutional Neural Network (CNN) on a large corpus of annotated images. While achieving very impressive results, CNN's single-pass approach to image analysis may not be the optimal one. Indeed, human perception is based on iterative exploration of an image through focusing on different parts of it and then building a full representation, potentially revisiting some parts to refine what has already been seen. We hypothesize that including such a scheme into an image

[1] According to Google Scholar as of 30.10.2018.

processing algorithm could help build better representations. The Recurrent Attention Model (RAM) introduced here, is a model that follows this paradigm [40, 3]. We build a deep fully differentiable RAM and show that it can outperform CNNs on synthetic data as well as natural images. We then achieve state-of-the-art results on task of transcribing unsegmented house numbers with this model. This contribution is published as a conference paper at IEEE SSCI 2016 [106].

- We consider the domain of generative modeling of natural texts. In this domain Recurrent Neural Networks outperform other models by a large margin. The dominant approach to text generation is a Language Model (LM). These models are trained in an unsupervised fashion to predict a word given all previous ones. Such an approach, however, suffers from lack of global context, defining what a model should generate. The Variational Autoencoder (VAE) is a latent variable generative model that has been applied to text generation [18] with the goal of addressing the issue of global context in LMs. However, the combination of VAE and LM suffers from the fact that LM is an overly powerful model on its own and the combination chooses to not use the global context vector. We quantify this issue and propose effective solutions. We have presented our findings at the EMNLP 2017 conference [108] and have already accumulated 40 citations [2].

- Generative Adversarial Network (GAN) based approaches have been used in combination with LMs. They form a very powerful class of models that achieved very impressive results in generative modeling of natural images. However, evaluation of such models is a non-trivial task. We have found that common metrics used to evaluate GAN-based approaches to text generation provide a meaningless comparison. We address this issue by introducing a set of metrics and empirically evaluate a set of GAN-based models for text generation. Our main finding is that Language Models are still very hard to beat baselines, since in our experiment no GAN model is convincingly better than an LM on our newly proposed set of metrics. We have presented our findings as a workshop contribution on ICML 2018 [109].

1.3 Outline

The work is structured as follows: Chapter 2 introduces basic concepts of machine learning, briefly recaps training of neural networks with backpropagation and summarizes

[2] According to Google Scholar as of 30.10.2018.

established well performing neural network architectures. In provides an in-depth discussion of weak and strong sides of different types of RNNs and briefly discusses efforts in circumventing most common issues arising during RNN training. In Chapter 3 we discuss the dropout regularization technique and present our approach to applying this technique to Recurrent Neural Networks. We present a detailed analysis of reasons for failures of previous approaches, show how our dropout avoids these issues and support it with experimental results.

Chapter 4 introduces an alternative to one-shot style image classification, performed by convolutional neural networks based on the work of Gregor et al. [40]. The model is a combination of convolutional and recurrent neural networks capable of processing an image iteratively by gradually exploring its contents locally. We present experimental results suggesting its superiority on tasks where an image contains multiple objects or when the object does not occupy a major portion of an image.

Chapter 5 provides an in depth discussion of generative models of natural texts. Specifically, we show an application of the RNN-based Variational Autoencoder algorithm to texts. Such models are known to suffer from a so called KL-collapse issue [18]. In this work we quantify this issue and propose a more effective solution by combining purely recurrent architectures with feedforward convolutional layers. The experimental results demonstrate orders of magnitude faster convergence and much better control of model's behavior. We then address a newly emerged Generative Adversarial Networks framework for generative modeling applied to textual data. Specifically, we discuss its application to fully unsupervised learning. It is known that such models are not trivial to properly compare and we show that the currently accepted evaluation metrics do not provide meaningful comparison. To remedy this, we propose a number of alternatives and perform an empirical study of a number of GAN-based models.

Chapter 6 concludes the thesis by summarizing contributions, proposed approaches, and major achieved results on various Computer Vision and Natural Language Processing benchmarks. We discuss possible extensions and outline directions for future research.

2 Background

2.1 Artificial Neural Networks

2.1.1 Feedforward Neural Networks

The Neural Network is a discriminative machine learning model. It is inspired by the structure of a brain - a large number of relatively simple units, called neurons, and connections between neurons. One neuron is a very simple building block that has a number of inputs and one output. Given a set of inputs, its output is computed with the following equation:

$$y = f\left(b + \sum_{i=0}^{k} w_i x_i\right) \tag{2.1.1}$$

where f is a scalar-to-scalar function, k is a number of inputs, b is a bias constant and x_i and w_i are i-th input and associated weight respectively. f is called the activation function. The typically used activation functions are $tanh$, $sigmoid$, rectifier ($f(x) = max(x, 0)$) and linear functions. As can be observed from the equation (2.1.1), a neuron first computes the weighted sum of all its inputs and then passes it through the activation function in order to model a nonlinear system. Afterwards its output can be either routed to an arbitrary number of other neurons or used as an output of a neural network.

The typical approach to organization of neurons in the neural network is to group them in a number of layers and the output of each neuron in the layer k is an input to each neuron in the layer $k + 1$. The first layer is called the input layer. The last layer serves as the network's output and is called the output layer. All the other layers are referred to as hidden layers. This organization is called the Feedforward Neural Network. An example of such network with three layers is presented in figure 2.1. In this case it

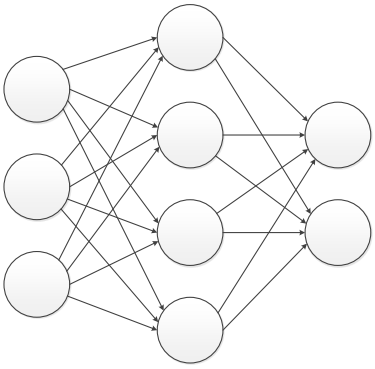


Figure 2.1: Example of a basic Feedforward Neural Network with three input, four hidden and two output units.

is a three layered network. The number of hidden layers and sizes of these layers are hyperparameters of the network and are picked prior to training. Once an architecture is selected, it becomes non-trivial to alter anything but individual weights of the network.

The neural network layer can be expressed in the matrix notation using the following equation:

$$\mathbf{y} = f(\mathbf{W}\mathbf{x} + \mathbf{b}) \tag{2.1.2}$$

where $\mathbf{x}$ is a vector of layer inputs, $\mathbf{y}$ is a vector of layer outputs, $\mathbf{b}$ is a bias vector, f is the activation function and $\mathbf{W}$ is a weight matrix. $\mathbf{W}$ contains weights of all the connections between neurons, such that element in row i and column j is the weight that neuron j uses for input i to compute the weighted sum. Such a layer is also referred to as the fully connected layer.

One of the important properties of the neural network is the way it models a phenomenon or entity. As one neuron can only produce one scalar value, which is not enough to model a complex phenomenon, the layer has to learn to build representations with sets of neurons. This effect is referred to as a distributed representation, i.e., none of the neuron's values matter by themselves. Instead, neurons form patterns that represent entities that the neural network has to model. This allows to increase the representational capacity of the network, as a single neuron can be reused to take part in different patterns, representing possibly completely unrelated entities. In addition distributed representations are robust to losses of individual neurons as long as there is sufficient amount of uncorrupted ones.

2.1.2 Backpropagation

In order for a neural network to be produce meaningful results its weights have to be picked so that the network performs the desired computation. The most common approach is to learn these weights using a supervised machine learning algorithm.

The most commonly used approach to training a neural network is a version of gradient descent in combination with the backpropagation algorithm. The algorithm allows to compute the gradient of the network's error with respect to its parameters in hidden layers. Its main idea is that after a network's output was computed during training, one can compute the error contributed by a neuron in the hidden layer by moving through the network in the backwards direction, from output to input units.

To demonstrate the algorithm we shall rewrite the equation (2.1.2):

$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b} \tag{2.1.3}$$

$$\mathbf{y} = f(\mathbf{z}) \tag{2.1.4}$$

The goal of the algorithm is to compute the gradient of the cost function with respect to a single weight. This can be achieved by applying the chain rule:

$$\frac{\partial C}{\partial w_{ij}} = \frac{\partial C}{\partial y_j}\frac{\partial y_j}{\partial z_j}\frac{\partial z_j}{\partial w_{ij}} \tag{2.1.5}$$

where C is the cost function. Computing the terms $\frac{\partial y_j}{\partial z_j}$ and $\frac{\partial z_j}{\partial w_{ij}}$ of the equation (2.1.5) is trivial: the second term is the partial derivative of the activation function. The third term is the value of the i-th input to the j-th neuron. The first term can also be computed easily in case the neuron is in the output layer. In this case one can directly differentiate the cost function and obtain the gradient. However, when computing gradients of a weight of a hidden unit, one has to use the chain rule once again and derive the following expression:

$$\frac{\partial C}{\partial y_j} = \sum_{k=0}^{L}\left(\frac{\partial C}{\partial y_k}\frac{\partial y_k}{\partial z_k}\frac{\partial z_k}{\partial y_j}\right) = \sum_{k=0}^{L}\left(\frac{\partial C}{\partial y_k}\frac{\partial y_k}{\partial z_k}w_{jk}\right) \tag{2.1.6}$$

where L is the size of the layer that follows the hidden layer being considered. The equation (2.1.6) is a recursive one. It allows to compute the error of the hidden layer when errors of the next layers are already computed.

The common practice of writing the whole backpropagation equation is to introduce an intermediate variable δ:

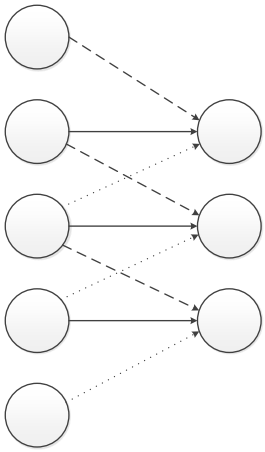


Figure 2.2: Example of a basic Convolutional Neural Network. All three neurons in the output layer share connection strengths, as shown with the type of an arrow, e.g. dashed arrows have the same weight of the connection.

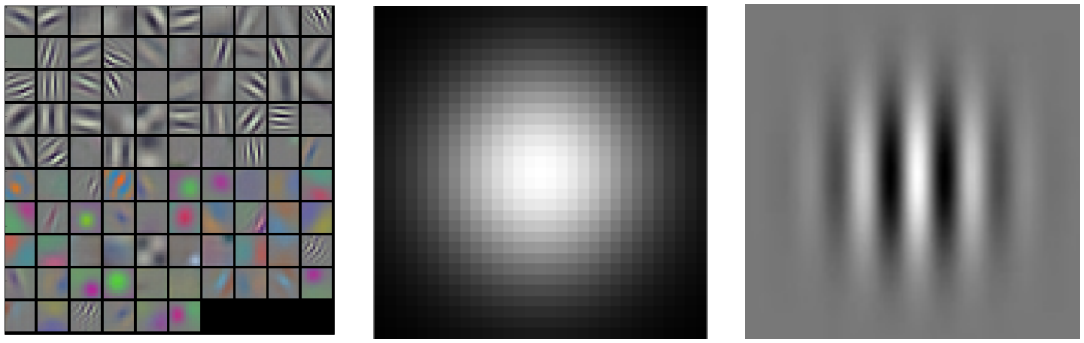
$$\delta_j = \frac{\partial C}{\partial y_j}\frac{\partial y_j}{\partial z_j} = \begin{cases} \dfrac{\partial C}{\partial y_j}\dfrac{\partial y_j}{\partial z_j}, & \text{if j is a neuron in the output layer,} \\ \sum_{k=0}^{L}\left(\delta_k w_{jk}\right)\dfrac{\partial y_j}{\partial z_j}, & \text{otherwise} \end{cases} \tag{2.1.7}$$

$$\frac{\partial C}{\partial w_{ij}} = \delta_j x_i \tag{2.1.8}$$

The backpropagation algorithm allows to efficiently train neural networks greatly outperforming less advanced strategies, such as evolutionary algorithm or random search. Since its introduction by Rumelhart et al. [103] in 1988 it has become the de-facto standard neural network learning algorithm when combined with a form of gradient descent.

2.1.3 Convolutional Neural Networks

The convolutional neural network is a model that originates from a well-known computer vision approach. The approach is to use a relatively small weight matrix, called the convolutional kernel or filter, and perform the dense convolutions with an image. The densely extracted features are then used as an input to a classification algorithm. The commonly used convolutional kernels are Gaussian, Gabor, Sobel etc. The Convolutional



(a) Convolutional kernels learned from data [25].

(b) Gaussian kernel.

(c) Gabor kernel.

Figure 2.3: Examples of convolutional kernels. Note how kernels learned from data closely resemble the widely used Gaussian and Gabor hand-designed filters. Best viewed in color.

Neural Network is conceptually similar to this approach but with a major advantage that the convolutional kernel's weights are not fixed and instead are adapted with backpropagation. This allows the network to select the kernel that is best suited to the task at hand. An example of such network is given in Figure 2.2.

When a number of convolutional layers are stacked upon one another, the resulting structure becomes capable of simultaneously learning a hierarchy of convolutional kernels, resulting in an extremely powerful model. This requires a lot of training data and computational resources, but the resulting networks are able to show very good performance when compared to other types of approaches [20]. Interestingly enough, even though networks are not constrained in any way in the choice of kernels, the learning process yields filters that are very similar to the commonly used ones. Example of the learned kernels is presented in figure 2.3(a). The important advantage of the Convolutional Networks is that they are able to learn a number of layers of filters. This allows them to learn a very high-level kernels that are tailored specifically to, for example, faces or wheels. We refer the reader to the work by Zeiler and Fergus [133] for the visualization of the higher level filters.

2.1.4 Normalization in Feedforward Neural Networks

It is widely known that normalization of input data to be 0 mean and 1 variance is helpful for machine learning algorithms in general. It is also true for Neural Networks. There has been a lot of research directed towards how to maintain these properties in representations of intermediate layers of a Neural Network. One such approach is to

carefully choose how weights of a Neural Network (NN) are initialized [35]. Glorot and Bengio [35] propose to initialize the weights such that the variance of features extracted by a layer does not change. However, this has no effect on the backward pass, where the variance of error vectors can increase or decrease arbitrarily. It is possible to design a similar approach by keeping the backward pass in mind, but it would still provide no guarantees that the weights would continue to preserve variance during training.

To address this issue, Ioffe and Szegedy [51] propose to normalize intermediate representations of a Neural Network explicitly by simply computing mean and variance over a batch of features. They note, however, that naive normalization hurts the final results of a network. Instead, in addition to explicit normalization, they propose to include learnable scales and shifts that would follow the transformation. More formally, the Batch Normalization layer performs the following computation:

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x_i \tag{2.1.9}$$

$$\sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu)^2 \tag{2.1.10}$$

$$x_i' = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}} \tag{2.1.11}$$

$$y_i = \gamma x_i' + \beta \tag{2.1.12}$$

where m is the batch size, x_i is a data point, ϵ is a small scalar for numerical stability and γ and β are the learnable scale and shift respectively. These operations are continuous and can thus be backpropagated through. It is also possible to obtain the gradient of γ and β with respect to an arbitrary loss function and then update them jointly with all the other parameters of a network. We refer the reader to the work of Ioffe and Szegedy [51] for derivations of the gradients.

While the introduced batch normalization is suited to Fully-Connected Neural Networks, it is straightforward to generalize it to CNNs. To do so, one can apply the Batch Normalization function similarly to the convolution operation, that is to share the scale and shift positions across every spacial location and normalize using mean over both batch and location dimensions. In this formulation BN is incredibly effective in increasing the convergence speed of a CNN. Figure 2.4 shows examples of networks with the same architecture and with or without BN.

Lastly, it is required to obtain global mean and variance of every feature in order for BN to be used during inference. To do so, one can either collect these values offline after

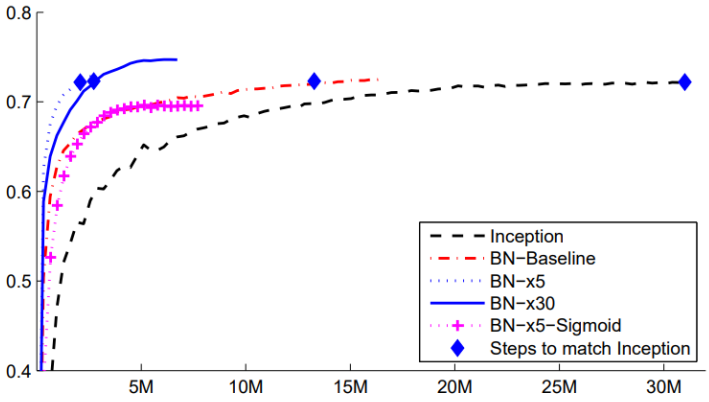


Figure 2.4: Learning curves of deep neural networks trained with and without Batch Normalization. Note that some variants with BN achieve the same result as unnormalized baseline in ten times less iterations and then continue to improve. Figure by Ioffe and Szegedy [51].

training is complete by computing mean and variance over the full training dataset, or maintain running mean and variance over batch-level moments. Once these values are collected they can be used in place of per-batch μ and σ^2 from Equation 2.1.11. The two approaches produce similar results in practice. Due to its effectiveness and simplicity BN is a de-facto standard tool for training Deep Convolutional Neural Networks and every state-of-the-art system is trained with the help of Batch Normalization.

Other types of normalization have been considered, such as Layer Normalization [5], Weight Normalization [105], Local Response Normalization [69], etc, but none is as efficient as BN when applied to CNNs.

2.1.5 Residual Networks

Once the depths of trained networks have reached the order of hundreds, researchers have noticed that such networks suffer from issues similar to those of Recurrent Neural Networks trained on long sequences, specifically vanishing and exploding gradients [49]. To address these issues, He et al. [45] propose to use what the call Residual Networks. These networks perform the following computation:

$$\mathbf{y} = \mathbf{x} + f(\mathbf{x}) \tag{2.1.13}$$

where f is an arbitrary function, x is the input and y is the output. This formulation allows to much more efficiently backpropagate gradients through very deep networks. The authors demonstrate it is possible to train networks with depths of more than a

(a) Vanilla Neural Layer. (b) Residual Layer.

Figure 2.5: Side by side comparison of standard and residual neural layers. Note that residual layers model a difference between inputs and outputs, thus giving the name "Residual".

thousand layers with this technique. Figure 2.5 shows a side-by-side comparison of a vanilla Neural Layer with the Residual Layer. The approach have then been used to establish state-of-the-art result on a large scale image recognition benchmark [45].

It should be noted that since the output of function f from Equation 2.1.13 is summed up with a layer's input, it is required to be of the exact same size than the input. However, it is desirable to increase the amount of convolutional kernels further in the hierarchy of filters. A typical approach to address this issue, also used by the authors, is to employ NxN convolutions with as many filters as required followed by a 1x1 convolution compressing outputs of the previous convolution to the size of the input. BN further improves convergence of the whole system. A common building block for image recognition systems would then be formally expressed as follows:

$$\mathbf{x}' = ReLU(BN(conv\_NxN(\mathbf{x}))) \tag{2.1.14}$$

$$\mathbf{y} = \mathbf{x} + ReLU(BN(conv\_1x1(\mathbf{x}'))) \tag{2.1.15}$$

where BN is the Batch Normalization function, $ReLU$ is the rectifier nonlinearity and $conv\_NxN$ and $conv\_1x1$ are convolutional layers with kernel sizes N and 1 respectively. Stacking a number of such blocks allows to achieve very impressive results on natural images.

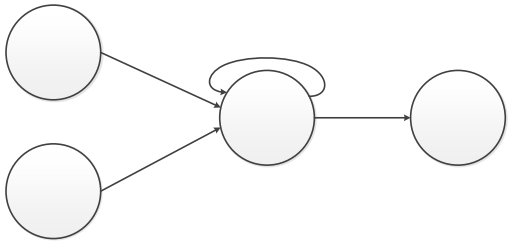


Figure 2.6: Example of a basic Recurrent Neural Network with one hidden neuron.

2.2 Recurrent Neural Networks

2.2.1 Vanilla Recurrent Networks

The Recurrent Network is an extension of the feedforward one. It can be expressed using the following equations:

$$\mathbf{h_0} = 0 \tag{2.2.1}$$

$$\mathbf{h_t} = f(\mathbf{W_{ih}x_t} + \mathbf{W_{hh}h_{t-1}} + \mathbf{b_h}) \tag{2.2.2}$$

$$\mathbf{y_t} = g(\mathbf{W_{ho}h_t} + \mathbf{b_o}) \tag{2.2.3}$$

where f and g are activation functions of hidden and output layers respectively. The typically used functions are *tanh* and *sigmoid*, as they are saturating functions and automatically prevent the absolute values of the hidden layer activations from growing to exponentially large values. $\mathbf{W_{ih}}$, $\mathbf{W_{hh}}$ and $\mathbf{W_{ho}}$ are weight matrices of the input-to-hidden, hidden-to-hidden and hidden-to-output connections respectively. $\mathbf{b_h}$ and $\mathbf{b_o}$ are bias units of hidden and output layers. The difference from feedforward network can be seen in Equation (2.2.2). This equation has a term $\mathbf{W_{hh}h_{t-1}}$, indicating that activations of a hidden layer at the previous time step are also an input at the current time step. Equation (2.2.1) sets the initial state of hidden layer activations to zero. In general, this can be of arbitrary value or can even be learned for a conditioning vector extracted from data. An example of an RNN is given in Figure 2.6.

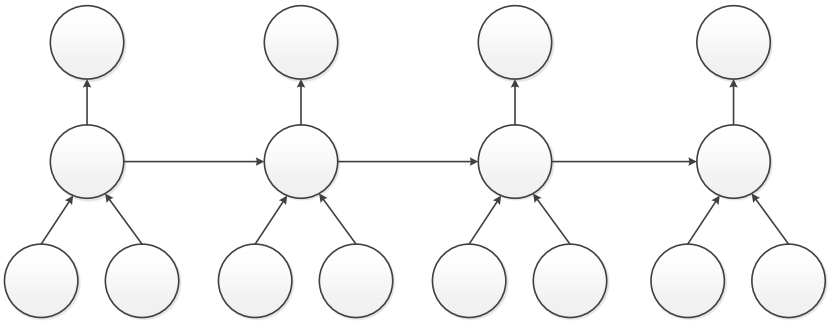


Figure 2.7: Visual representation of the RNN from Figure 2.6, unrolled over time for computation of the gradients.

2.2.2 Backpropagation-Through-Time

The most commonly used method for training Recurrent Networks is a version of gradient descent in conjunction with Backpropagation-Through-Time (BPTT). BPTT is an extension of the standard backpropagation algorithm discussed in Section 2.1.2 for computing gradients in a neural network. Its idea is schematically depicted in figure 2.7. An RNN is unrolled through time and is treated as a very deep feedforward network with weights shared across all timesteps. This allows to seamlessly apply standard backpropagation to a recurrent network and compute gradients of its parameters.

This approach has one major problem - when the gradient is backpropagated through the recurrent connections, it gets repeatedly multiplied by the hidden weights matrix. When computing the values of δ of the recurrent neurons at time step t, δ_t, we multiply δ_{t+1} by the weight of corresponding connection. However, δ_{t+1} have also been multiplied by this weight when it was computed from δ_{t+2}. When δ has to be propagated to many time steps back, it can become exponentially small or large, depending on the properties of the hidden-to-hidden weight matrix. This problem has received the name of vanishing and exploding gradients, respectively [49]. There have been a number of attempts to solve this problem, such as echo-state networks [55], leaky integrator units [56] and second-order optimization algorithms [81]. One of the most promising ones is the Long Short-Term Memory unit [49]. These units are specifically designed to avoid the problem of vanishing and exploding gradients and were able to achieve very good results in handwriting recognition [38], speech recognition [39] and machine translation

(a) Long Short-Term Memory Unit. (b) Gated Recurrent Unit.

Figure 2.8: Graphical illustrations of LSTM and GRU networks. We omit connections from input data for brevity.

[117], and are becoming more and more popular among researchers.

2.2.3 Long Short-Term Memory Networks

Initially introduced by Hochreiter and Schmidhuber [49], Long Short-Term Memory units address the problems of exploding and vanishing gradients. The authors propose to formulate the forward pass of the RNN in such a way that the two problems would be greatly reduced. To do so, they propose to compose networks from so-called cells, whose interactions with themselves are additive instead of multiplicative. More formally, a single step of an LSTM network can be defined as follows:

$$\begin{pmatrix} \mathbf{i}_t \\ \mathbf{f}_t \\ \mathbf{o}_t \\ \mathbf{g}_t \end{pmatrix} = \begin{pmatrix} \sigma(\mathbf{W}_i \begin{bmatrix} \mathbf{x}_t, \mathbf{h}_{t-1} \end{bmatrix} + \mathbf{b}_i) \\ \sigma(\mathbf{W}_f \begin{bmatrix} \mathbf{x}_t, \mathbf{h}_{t-1} \end{bmatrix} + \mathbf{b}_f) \\ \sigma(\mathbf{W}_o \begin{bmatrix} \mathbf{x}_t, \mathbf{h}_{t-1} \end{bmatrix} + \mathbf{b}_o) \\ f(\mathbf{W}_g \begin{bmatrix} \mathbf{x}_t, \mathbf{h}_{t-1} \end{bmatrix} + \mathbf{b}_g) \end{pmatrix} \tag{2.2.4}$$

$$\mathbf{c}_t = \mathbf{f}_t * \mathbf{c}_{t-1} + \mathbf{i}_t * \mathbf{g}_t \tag{2.2.5}$$

$$\mathbf{h}_t = \mathbf{o}_t * f(\mathbf{c}_t), \tag{2.2.6}$$

where σ is the *sigmoid* activation function and f is an arbitrary nonlinearity, typically the *tanh* function. $\mathbf{x}_t$ is the network's input on current step. $\mathbf{c}_t$ and $\mathbf{c}_{t-1}$ are cell values on current and previous steps respectively. $\mathbf{h}_t$ and $\mathbf{h}_{t-1}$ are cell outputs on current and previous steps respectively. Vectors $\mathbf{i}$, $\mathbf{f}$, $\mathbf{o}$ and $\mathbf{g}$ are gates controlling, for example, how cell values are exposed to the rest of the network. One step of the LSTM cell is graphically shown in Figure 2.8(a).

In LSTM the recurrence is defined by Equation 2.2.5. Note that the current state c_t is a gated copy of the previous state c_{t-1} plus the contribution coming from inputs on current step. Use of summation instead of multiplication allows to easily backpropagate error vectors during the backward pass. In this regard, LSTMs and Residual Networks,

discussed in Section 2.1.5, are conceptually close since they are based on the same observation that the product between state and a weight matrix during forward pass makes training very deep networks problematic. They differ in the "direction" of depth – Residual Networks are architecturally deep, that is they have a lot of layers, while LSTMs are temporally deep, i.e. they work on very long sequences. It should be noted, that historically LSTMs were introduced two decades earlier than the Residual Networks. One could thus view Residual Networks as architecturally deep LSTMs with simplified single step.

2.2.4 Gated Recurrent Unit Networks

Since LSTM networks have been shown to be a very successful instantiation of RNNs, various attempts at improving their architecture have been made [59, 23]. However, the LSTM architecture has been shown to be a very successful variant that is very difficult ot improve upon. One successful attempt at improving over LSTMs is a GRU [22]. They are built on a similar principle, that is additive recurrence rather than multiplicative. More formally, one step of a GRU network can be expressed as follows:

$$\begin{pmatrix} \mathbf{z}_t \\ \mathbf{r}_t \end{pmatrix} = \begin{pmatrix} \sigma(\mathbf{W}_z \begin{bmatrix} \mathbf{x}_t, \mathbf{h}_{t-1} \end{bmatrix} + \mathbf{b}_z) \\ \sigma(\mathbf{W}_r \begin{bmatrix} \mathbf{x}_t, \mathbf{h}_{t-1} \end{bmatrix} + \mathbf{b}_r) \end{pmatrix} \tag{2.2.7}$$

$$\mathbf{g}_t = f(\mathbf{W}_g \begin{bmatrix} \mathbf{x}_t, \mathbf{r}_t * \mathbf{h}_{t-1} \end{bmatrix} + \mathbf{b}_g) \tag{2.2.8}$$

$$\mathbf{h}_t = (1 - \mathbf{z}_t) * \mathbf{h}_{t-1} + \mathbf{z}_t * \mathbf{g}_t \tag{2.2.9}$$

where σ is the *sigmoid* activation function and f is an arbitrary nonlinearity, typically the *tanh* function. $\mathbf{x}_t$ is the current network's input. $\mathbf{h}_t$ and $\mathbf{h}_{t-1}$ are network states on current and previous steps respectively. Vectors $\mathbf{z}$, $\mathbf{r}$, and $\mathbf{g}$ are gates with functions similar to those of LSTM. GRU enjoys a better computational cost when compared to LSTMs due to three gates instead of four and achieves comparable results. A Gated Recurrent Unit is graphically shown in Figure 2.8(b).

2.2.5 Normalization in Recurrent Neural Networks

Normalization has been proven to be extremely effective when applied to Feedforward Neural Networks, what gave rise to works applying normalization to RNNs as well. Initial attempts to apply BN to Recurrent Neural Networks have shown that application of BN to RNNs is not straightforward [73]. Subsequent work [27] has demonstrated that diffi-

culty of applying BN to RNNs is caused by incorrect initialization of γ parameter causing severe vanishing gradients. The authors demonstrate better convergence and generalization of RNNs with Batch Normalization, but at the cost of cumbersome maintenance of per-step mean and variance statistics.

Layer Normalization [5] has been introduced to address both issues. It is conceptually similar to BN, but instead of normalizing over batch dimension, that does not guarantee that a specific feature vector has zero mean and one variance, Layer Normalization explicitly rescales hidden states of an RNN:

$$\mu = \frac{1}{D}\sum_{i=1}^{D} x[i] \tag{2.2.10}$$

$$\sigma^2 = \frac{1}{D}\sum_{i=1}^{D}(x[i] - \mu)^2 \tag{2.2.11}$$

$$x' = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} \tag{2.2.12}$$

$$y = \gamma x' + \beta \tag{2.2.13}$$

where D is the dimensionality of the hidden state, $x[i]$ is a single feature from the hidden state. γ, β and ϵ have the same functions as in BN. Such a formulation removes all issues of Recurrent Batch Normalization, while maintaining it benefits. It should be noted that while improving training of RNNs, no normalization yields improvements on the order of those that BN brings to CNNs. Thus, despite being a useful tool to use, normalization is not as imprortant for RNNs as it is to Feedforward Networks.

2.3 Generative Modeling

Machine learning models can be broadly divided into discriminative and generative by the type of problems that they attempt to solve. The goal of discriminative models is to find a decision boundary between datapoints belonging to different classes. This can be achieved by minimizing the Negative Log Likelihood (NLL) of the conditional distribution $p(y|x)$, where x is a data point and y is the class label associated with this data point, $-\log(p(y|x))$. There are other approaches to this problem, but they are out of scope of this work. In contrast, generative models attempt to discover the data distribution, instead of just a simple boundary between classes, by minimizing $-log(p(x))$. Note that in this formulation an algorithm only has access to data points and not to the associated

class labels. Since such models approximate a probability distribution, they have a number of applications, e.g. estimation of probability of a single data point. It should be noted that in some cases the two classes of machine learning models are relatively close, e.g. in Machine Translation, where y is a sentence in a certain language with it's own grammar that does not depend on the input sentence x. In this section we provide an overview of the most prominent modern generative models.

2.3.1 Autoregressive Models

To introduce the class of autoregressive models, we shall first rewrite the objective function of the generative model:

$$-\log(p(\mathbf{x})) = -\sum_{i=1}^{N} p(x_i|x_1..x_N/x_i) \tag{2.3.1}$$

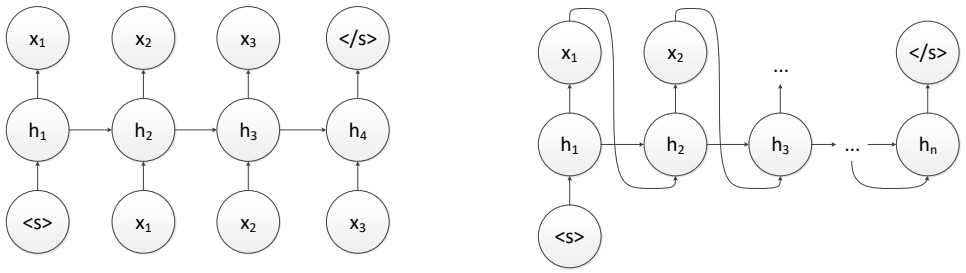
where N is the dimensionality of a data point and x_i is the i-th element of a data point. Equation 2.3.1 decomposes the joint probability $p(x)$ into product of individual probabilities $p(x_i)$. Since there is no prior knowledge about how data elements interact, in this equation x_i is conditioned on every other data element.

Autoregressive models make an assumption that there exists a certain order of elements, e.g. words, in a data point and one data element only depends on other elements that come prior to it:

$$-\log(p(\mathbf{x})) = -\sum_{i=1}^{N} p(x_i|\mathbf{x_{<i}}) \tag{2.3.2}$$

This formulation is attractive because it allows to (i) straightforwardly train generative models with the help of so-called Teacher Forcing; (ii) straightforwardly sample new data points from a trained model by sampling one data element at a time and feeding them back into the model and (iii) the ordering assumptions fits very well to some types of data, e.g. natural language or audio. The function $p(x_i|\mathbf{x_{<i}})$ is usually parametrized with a Neural Network whose parameters are then minimized with respect to the decomposed objective function in Equation 2.3.2.

Teacher Forcing. An efficient approach to training of such models is Teacher Forcing. In this regime the model's inputs always come from the natural data that it should model. The model is then trained to predict a data element, e.g. a word, given all elements that precede it. When applied to natural languages this approach is referred to as a Language Model and has received a lot of attention from the NLP community. Neural



(a) Training an autoregressive RNN in teacher forcing mode.

(b) Sampling from an RNN based autoregressive model.

Figure 2.9: Graphical illustrations of training and inference of autoregressive models. Note that the length of a sequence is not fixed and instead defined by the model itself by emitting the special symbol for end of a sequence.

Language Models have recently achieved very impressive results and became state-of-the-art generative models of natural texts. The process is schematically depicted in Figure 2.9(a).

Sampling. Sampling from a trained model can be performed by generating one data element at a time. For example, if data elements are characters, a model can generate sentences one character at a time until a special `<end_of_sequence>` charecter is generated. At this point the process is complete. Figure 2.9(b) presents a graphical illustration of this process. It is additionally possible to add a conditioning information to this process to achieve, for example, image captioning. Combining this generation process with a state-of-the-art image recognition systems yields state-of-the-art image captioning algorithms [122].

2.3.2 Variational Autoencoders

Variational Autoencoders are directed graphical models where data x is generated from the latent vector z, graphically shown in Figure 2.10. In such models one of the goals is to learn the inference function $p(z|x)$. A typical approach to this problem is to minimize the Kullback-Leibler (KL) divergence between the true posterior $p(z|x)$ and the approximating distribution $q(z|x)$. However, this has the downside that it would involve computation of an intractable term $p(x)$ that requires integration of all possible values of z. Instead, one can derive a so-called Evidence Lower Bound (ELBO) term, that would be an upper bound on the target KL divergence. This approach broadly falls

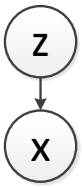


Figure 2.10: Variational Autoencoder as a probabilistic graphical model.

into the class of Variational Inference algorithms, thus giving its name to the Variational Autoencoder. Specifically, the derivation of the VAE objective is as follows:

$$\begin{aligned} KL(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x}) = \sum_{\mathbf{z}} q(\mathbf{z}|\mathbf{x}) \log \frac{q(\mathbf{z}|\mathbf{x})}{p(\mathbf{z}|\mathbf{x})} = \\ \mathbb{E}\Big[\log \frac{q(\mathbf{z}|\mathbf{x})}{p(\mathbf{z}|\mathbf{x})} \Big] = \\ \mathbb{E}[\log q(\mathbf{z}|\mathbf{x}) - \log(p(\mathbf{z}|\mathbf{x})] = \\ \mathbb{E}\Big[\log q(\mathbf{z}|\mathbf{x}) - \log \frac{p(\mathbf{z}|\mathbf{x})p(\mathbf{z})}{p(\mathbf{x})} \Big] = \\ \mathbb{E}[\log q(\mathbf{z}|\mathbf{x}) - \log p(\mathbf{x}|\mathbf{z}) - \log p(\mathbf{z}) + \log p(\mathbf{x})] \end{aligned}$$

Note, that expectation over $p(\mathbf{x})$ is independent of $\mathbf{z}$ and can thus be moved outside of the expectation:

$$KL(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x}) - \log p(\mathbf{x}) = \mathbb{E}[\log q(\mathbf{z}|\mathbf{x}) - \log p(\mathbf{x}|\mathbf{z}) - \log p(\mathbf{z})]$$

$$\begin{aligned} \log p(\mathbf{x}) - KL(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x}) = \mathbb{E}[-\log q(\mathbf{z}|\mathbf{x}) + \log p(\mathbf{x}|\mathbf{z}) + \log p(\mathbf{z})] = \\ \mathbb{E}[\log p(\mathbf{x}|\mathbf{z}) - (\log q(\mathbf{z}|\mathbf{x}) - \log p(\mathbf{z}))] = \\ \mathbb{E}[\log p(\mathbf{x}|\mathbf{z})] - \mathbb{E}[\log q(\mathbf{z}|\mathbf{x}) - \log p(\mathbf{z})] = \\ \mathbb{E}[\log p(\mathbf{x}|\mathbf{z})] - KL(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z})) \end{aligned}$$

Since $\log p(x)$ is a negative constant depending on a dataset, we can write the following equation:

$$KL(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x})) \leq -\mathbb{E}[\log p(\mathbf{x}|\mathbf{z})] + KL(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z})) \tag{2.3.3}$$

Closely examining Equation 2.3.3 reveals that it consists of (i) a reconstruction term and (ii) a regularization term promoting posterior $q(\mathbf{z}|\mathbf{x})$ to be close to a prior over $\mathbf{z}$. The second term can be trivially computed if both these distributions are carefully constructed, for example, if $q(\mathbf{z}|\mathbf{x})$ is diagonal Gaussian and $p(\mathbf{z})$ is a unit Gaussian distribution. The first term closely resembles the objective function of a typical Autoencoder, but with a stochastic bottleneck.

Thus, from a neural network perspective, a VAE is an autoencoder with a stochastic bottleneck and an additional regularization term restricting outputs of the encoder network $q(\mathbf{z}|\mathbf{x})$. Once this network has predicted the parameters of the posterior distribution, the generating model, or decoder, reconstructs the input from a sampled latent code, thus allowing the model to be efficiently trained with Stochastic Gradient Descent.

A final note is on the so-called reparametrization trick. When sampling a latent vector from the posterior distribution $q(\mathbf{z}|\mathbf{x})$, one can naively sample directly from $N(\mu, \sigma)$, where μ and σ are predicted by the inference model:

$$\mu, \sigma = f(\mathbf{x}) \tag{2.3.4}$$

$$\mathbf{z} \sim N(\mu, \sigma) \tag{2.3.5}$$

where f if the inference model. This formulation, however, results in a high variance of the gradients estimated with backpropagation. Instead, the authors propose to reparametrize the sampling procedure:

$$\mu, \sigma = f(\mathbf{x}) \tag{2.3.6}$$

$$\mathbf{z}' \sim N(0, 1) \tag{2.3.7}$$

$$\mathbf{z} = \mu + \sigma\mathbf{z}' \tag{2.3.8}$$

When sampling with Equation 2.3.8, the forward pass of a model remains unchanged, while the backward pass becomes much more stable when compared to sampling with Equation 2.3.5. This allows to successfully train large-scale VAE models in reasonable time.

2.3.3 Generative Adversarial Networks

Training a generative model involves minimizing a divergence between a data distribution $p(x)$ and model distribution $q(x)$. Training with NLL is equivalent to minimizing the KL-divergence between these distributions. However, it is not obvious that this divergence is the perfect one to optimize. One issue, for example, is that it can be viewed as recall based, that is by definition a model should put non-zero mass on data points that occur in the training dataset. However, it does not penalize spurious high-probability data points if they never appear in the training set.

Generative Adversarial Network is a two player game where one player is referred to as the discriminator and the other as the generator formally expressed as:

$$\min_G \max_D J(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{data}}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}}[\log(1 - D(G(\mathbf{z})))] \tag{2.3.9}$$

where D is the discriminator function and G is the generator function. It can be shown [37] that this formulation is equivalent to minimizing the Jensen-Shannon (JS) divergence between two distributions:

$$m = \frac{p + q}{2} \tag{2.3.10}$$

$$JS(p||q) = \frac{1}{2}(KL(p||m) + KL(q||m) \tag{2.3.11}$$

where KL is the KL-divergence. Nowozin et al. [91] further show that depending on specific variants of Equation 2.3.9, the minimax optimization process will minimize different kinds of divergences.

While the framework allows to train models capable of producing convincing samples [13] and learning good feature representations [96], it is very sensitive to the choice of hyperparameters. In addition, it suffers from a so-called mode collapse problem [37] and often exhibits unstable training. A notable modification of the framework [2] reviews the game based on optimal transport distances between distributions and changes Equation 2.3.9 to

$$\min_G \max_{D \in L} J(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{data}}[D(\mathbf{x})] - \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}}[D(G(\mathbf{z}))] \tag{2.3.12}$$

where L is a set of 1-Lipshitz functions. In practice, constraining the discriminator to be such a function is a non-trivial task and the authors instead propose to use k-Lipshitz functions by clipping the weights of the discriminator. Gulrajani et al. [42] improve

over the work of Arjovsky et al. [2] by proposing a soft version of ensuring that the discriminator belongs to the set of 1-Lipshitz by directly penalizing its gradients with respect to the input:

$$\min_G \max_D J(D,G) = \mathbb{E}_{\mathbf{x}\sim p_{data}}[D(\mathbf{x})] - \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}}[D(G(\mathbf{z}))] - \lambda \mathbb{E}_{\hat{\mathbf{x}}\sim p_{\hat{\mathbf{x}}}}[(||\nabla_{\hat{\mathbf{x}}} D(\hat{\mathbf{x}})||_2 - 1)^2] \tag{2.3.13}$$

where λ is a hyperparameter and $\hat{\mathbf{x}}$ is a specifically chosen point from the joint data and generator space. The authors demonstrate robustness to various choices of hyperparameters and architectures and show that their system is one of the first ones capable of generating texts and sequences of discrete elements, which have been difficult to model with traditional GANs.

An elegant application of GANs is to use them in place of the KL-divergence of the VAE objective function. The VAE objective function is:

$$J_{vae} = KL(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z})) - \mathbb{E}_{q(\mathbf{z}|\mathbf{x})}[\log\ p(\mathbf{x}|\mathbf{z})] \tag{2.3.14}$$

Makhzani et al. [79] propose to use a GAN discriminating between samples from prior $p(z)$ and posterior $q(z|x)$. Specifically:

$$J_{aae} = -\alpha \mathbb{E}_{\mathbf{z}\sim q_{\mathbf{z}|\mathbf{x}}}[\log(1 - D(G(\mathbf{z})))] - \mathbb{E}_{q(\mathbf{z}|\mathbf{x})}[\log\ p(\mathbf{x}|\mathbf{z})] \tag{2.3.15}$$

where α is a hyperparameter. This formulation has the disadvantage of losing the theoretical justification. Indeed, while the objective function of the VAE has a clear interpretation, it is not obvious what exactly is being minimized in Equation 2.3.15. However, this reformulation allows to move away from diagonal Gaussian priors and use arbitrarily complex distribution in their place.

3

Recurrent Dropout

3.1 Introduction

Recurrent Neural Networks, LSTMs in particular, have recently become a popular tool among NLP researchers for their superior ability to model and learn from sequential data. These models have shown state-of-the-art results on various public benchmarks ranging from sentence classification [124, 52, 76] and various tagging problems [29] to language modelling [65, 137], text generation [136] and sequence-to-sequence prediction tasks [117].

Having shown excellent ability to capture and learn complex linguistic phenomena, RNN architectures are prone to overfitting. Among the most widely used techniques to avoid overfitting in neural networks is the dropout regularization [48]. Since its introduction it has become, together with the L2 weight decay, the standard method for neural network regularization. While showing significant improvements when used in feed-forward architectures, e.g., Convolutional Neural Networks [69], the application of dropout in RNNs has been somewhat limited. Indeed, so far dropout in RNNs has been applied in the same fashion as in feed-forward architectures: it is typically injected in input-to-hidden and hidden-to-output connections, i.e., along the input axis, but not between the recurrent connections (time axis). Given that RNNs are mainly used to model sequential data with the goal of capturing short- and long-term interactions, it seems natural to also regularize the recurrent weights. This observation has led us and other researchers [88, 33] to the idea of applying dropout to the recurrent connections in RNNs.

In this work we propose a novel *recurrent dropout* technique and demonstrate how our method is superiour to other recurrent dropout methods recently proposed in [88, 33]. Additionally, we answer the following questions which helps to understand how to best

apply recurrent dropout: (i) how to apply the dropout in recurrent connections of the LSTM architecture in a way that prevents possible corruption of the long-term memory; (ii) what is the relationship between our *recurrent dropout* and the widely adopted dropout in input-to-hidden and hidden-to-output connections; (iii) how the dropout mask in RNNs should be sampled: once per step or once per sequence. The latter question of sampling the mask appears to be crucial in some cases to make the recurrent dropout work and, to the best of our knowledge, has received very little attention in the literature. Our work is the first one to provide empirical evaluation of the differences between these two sampling approaches.

Regarding empirical evaluation, we first highlight the problem of information loss in memory cells of LSTMs when applying *recurrent dropout*. We demonstrate that previous approaches of dropping *hidden state* vectors cause loss of memory while our proposed method to use dropout mask in *hidden state update* vectors does not suffer from this problem. Our experimental results demonstrate that our *recurrent dropout* helps to achieve better regularization and yields improvements across all the tasks, even when combined with the conventional feed-forward dropout. Furthermore, we compare our dropout scheme with the recently proposed alternative recurrent dropout methods and show that our technique is superior in almost all cases.

3.2 Related Work

Neural Network models often suffer from overfitting, especially when the number of network parameters is large and the amount of training data is small. This has led to a lot of research directed towards improving their generalization ability. Below we primarily discuss some of the methods aimed at improving regularization of RNNs.

Pham et al. [95] and Zaremba et al. [132] have shown that LSTMs can be effectively regularized by using dropout in forward connections. While this already allows for effective regularization of recurrent networks, it is intuitive that introducing dropout also in the hidden state may force it to create more robust representations. Indeed, Moon et al. [88] have extended the idea of dropping neurons in forward direction and proposed to drop cell states as well showing good results on a Speech Recognition task. Bluche et al. [15] carry out a study to find where dropout is most effective, e.g. input-to-hidden or hidden-to-output connections. The authors conclude that it is more beneficial to use it once in the correct spot, rather than to put it everywhere. Bengio et al. [10] have proposed an algorithm called scheduled sampling to improve performance of recurrent networks on sequence-to-sequence labeling tasks. A disadvantage of this work is that the

scheduled sampling is specifically tailored to this kind of tasks, what makes it impossible to use in, for example, sequence-to-label tasks. Gal [33] uses insights from variational Bayesian inference to propose a variant of LSTM with dropout that achieves consistent improvements over a baseline architecture without dropout.

The main contribution discussed in this chapter is a new *recurrent dropout* technique, which is most useful in gated recurrent architectures such as LSTMs and GRUs. We demonstrate that applying dropout to arbitrary vectors in LSTM cells may lead to loss of memory thus hindering the ability of the network to encode long-term information. In other words, our technique allows for adding a strong regularizer on the model weights responsible for learning short and long-term dependencies without affecting the ability to capture long-term relationships, which are especially important to model when dealing with natural language. Finally, we compare our method with alternative *recurrent dropout* methods recently introduced in [88, 33] and demonstrate that our method allows to achieve better results.

3.3 Recurrent Dropout

In this section we first show how the idea of feed-forward dropout [48] can be applied to recurrent connections in vanilla RNNs. We then introduce our *recurrent dropout* method specifically tailored for gated architectures such as LSTMs and GRUs. We draw parallels and contrast our approach with alternative recurrent dropout techniques recently proposed in [88, 33] showing that our method is favourable when considering potential memory loss issues in long short-term architectures.

3.3.1 Dropout

The Dropout is a regularization method originally proposed by Hinton et al. [48]. The method is motivated by the observation that it is possible for neurons in a network to co-adapt to each others presence. This makes neurons in a hidden layer as a whole less robust to small changes of a single unit. To overcome this issue, the authors propose to stochastically zero-out units in a hidden layer. Intuitively, this makes the neurons more independent from one another and forces a layer to build representations that are more robust to loss of a single unit. As the network cannot rely on a single neuron being active, it learns to make use of the corrupted distributed representations. This helps the network to generalize, as when it is presented a new input, it is very likely that it will be different from ones used during training and its representation will be corrupted by

some degree.

More formally the algorithm can be expressed with the following equations:

$$mask = Bernoulli(p) \tag{3.3.1}$$

$$d(\mathbf{x}) = \begin{cases} mask * \mathbf{x}, & \text{if train phase} \\ (1-p)\mathbf{x} & \text{otherwise,} \end{cases} \tag{3.3.2}$$

where p is the probability to turn off an individual neuron, $mask$ is a vector of values, sampled from the Bernoulli distribution with success probability p, $*$ is the element wise product operator.

The Dropout is currently one of the most widely used regularization methods for Neural Networks. It have been used by the state-of-the-art systems in image classification [69, 113] and activity recognition [114].

3.3.2 Dropout in vanilla RNNs

Vanilla RNNs process the input sequences as follows:

$$\mathbf{h_t} = f(\mathbf{W_h}[\mathbf{x_t}, \mathbf{h_{t-1}}] + \mathbf{b_h}), \tag{3.3.3}$$

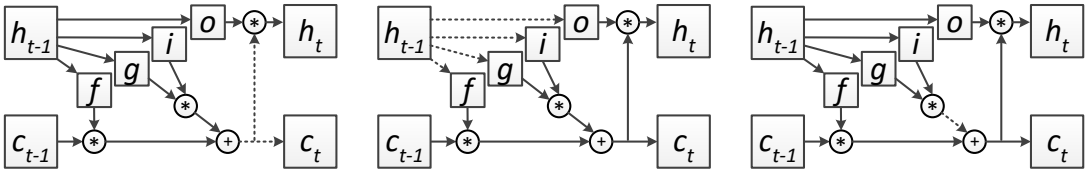
where $\mathbf{x_t}$ is the input at time step t; $\mathbf{h_t}$ and $\mathbf{h_{t-1}}$ are hidden vectors that encode the current and previous states of the network; $\mathbf{W_h}$ is parameter matrix that models input-to-hidden and hidden-to-hidden (recurrent) connections; $\mathbf{b}$ is a vector of bias terms, and f is the activation function.

As RNNs model sequential data by a fully-connected layer, dropout can be applied by simply dropping the previous hidden state of a network. Specifically, we modify Equation 3.3.3 in the following way:

$$\mathbf{h_t} = f(\mathbf{W_h}[\mathbf{x_t}, d(\mathbf{h_{t-1}})] + \mathbf{b_h}), \tag{3.3.4}$$

where d is the dropout function from Equation 3.3.2.

This modification leads to a reduction the validation error of vanilla RNNs. However, plain LSTM networks without dropout still outperform RNNs with dropout. Hence, it is important to design a solution suitable for LSTM networks as well.



(a) Moon et al., 2015 (b) Gal, 2015 (c) Ours

Figure 3.1: Illustration of the three types of dropout in recurrent connections of LSTM networks. Dashed arrows refer to dropped connections. Input connections are omitted for clarity.

3.3.3 Dropout in LSTM networks

Long Short-Term Memory networks [49] have introduced the concept of gated inputs in RNNs, which effectively allow the network to preserve its memory over a larger number of time steps during both forward and backward passes, thus alleviating the problem of vanishing gradients [11]. Formally, it is expressed with the following equations:

$$\begin{pmatrix} \mathbf{i}_t \\ \mathbf{f}_t \\ \mathbf{o}_t \\ \mathbf{g}_t \end{pmatrix} = \begin{pmatrix} \sigma(\mathbf{W}_i \begin{bmatrix} \mathbf{x}_t, \mathbf{h}_{t-1} \end{bmatrix} + \mathbf{b}_i) \\ \sigma(\mathbf{W}_f \begin{bmatrix} \mathbf{x}_t, \mathbf{h}_{t-1} \end{bmatrix} + \mathbf{b}_f) \\ \sigma(\mathbf{W}_o \begin{bmatrix} \mathbf{x}_t, \mathbf{h}_{t-1} \end{bmatrix} + \mathbf{b}_o) \\ f(\mathbf{W}_g \begin{bmatrix} \mathbf{x}_t, \mathbf{h}_{t-1} \end{bmatrix} + \mathbf{b}_g) \end{pmatrix} \tag{3.3.5}$$

$$\mathbf{c}_t = \mathbf{f}_t * \mathbf{c}_{t-1} + \mathbf{i}_t * \mathbf{g}_t \tag{3.3.6}$$

$$\mathbf{h}_t = \mathbf{o}_t * f(\mathbf{c}_t), \tag{3.3.7}$$

where $\mathbf{i}_t, \mathbf{f}_t, \mathbf{o}_t$ are input, output and forget gates at step t; $\mathbf{g}_t$ is the vector of cell updates and $\mathbf{c}_t$ is the updated cell vector used to update the hidden state $\mathbf{h}_t$; σ is the sigmoid function and $*$ is the element-wise multiplication.

Gal [33] proposes to drop the previous *hidden state* vectors when computing values of gates and updates of the current step, where he samples the dropout mask once for every sequence:

$$\begin{pmatrix} \mathbf{i}_t \\ \mathbf{f}_t \\ \mathbf{o}_t \\ \mathbf{g}_t \end{pmatrix} = \begin{pmatrix} \sigma(\mathbf{W}_i \begin{bmatrix} \mathbf{x}_t, d(\mathbf{h}_{t-1}) \end{bmatrix} + \mathbf{b}_i) \\ \sigma(\mathbf{W}_f \begin{bmatrix} \mathbf{x}_t, d(\mathbf{h}_{t-1}) \end{bmatrix} + \mathbf{b}_f) \\ \sigma(\mathbf{W}_o \begin{bmatrix} \mathbf{x}_t, d(\mathbf{h}_{t-1}) \end{bmatrix} + \mathbf{b}_o) \\ f(\mathbf{W}_g \begin{bmatrix} \mathbf{x}_t, d(\mathbf{h}_{t-1}) \end{bmatrix} + \mathbf{b}_g) \end{pmatrix} \tag{3.3.8}$$

Moon et al. [88] propose to apply dropout directly to the cell values and use per-

sequence sampling as well:

$$\mathbf{c}_t = d(\mathbf{f}_t * \mathbf{c}_{t-1} + \mathbf{i}_t * \mathbf{g}_t) \tag{3.3.9}$$

In contrast to dropout techniques proposed by Gal [33] and Moon et al. [88], we propose to apply dropout to the *cell update* vector $\mathbf{g}_t$ as follows:

$$\mathbf{c}_t = \mathbf{f}_t * \mathbf{c}_{t-1} + \mathbf{i}_t * d(\mathbf{g}_t) \tag{3.3.10}$$

Different from methods of [88, 33], our approach does not require sampling of the dropout masks once for every training sequence. On the contrary, as we will show in Section 3.4, networks trained with a dropout mask sampled per-step achieve results that are at least as good and often better than per-sequence sampling. Figure 3.1 shows differences between approaches to dropout.

The approach of [33] differs from ours in the overall strategy – they consider network's hidden state as input to subnetworks that compute gate values and cell updates and the purpose of dropout is to regularize these subnetworks. Our approach considers the architecture as a whole with the hidden state as its key part and regularize the whole network. The approach of [88] on the other hand is seemingly similar to ours. In Section 3.3.4 we argue that our method is a more principled way to drop recurrent connections in gated architectures.

It should be noted that while being different, the three discussed dropout schemes are not mutually exclusive. It is in general possible to combine our approach and the other two. We expect the merge of our scheme and that of [33] to hold the biggest potential. The relations between recurrent dropout schemes are however out of scope of this thesis and we rather focus on studying the relationships of different dropout approaches with the conventional forward dropout.

Lastly, we note that our dropout is also applicable to the recently introduced Gated Recurrent Unit (GRU) networks [22]. GRU networks are built on the same design principles as LSTM networks and our dropout technique applies in a similar fashion.

3.3.4 Dropout and memory

We found that an intuitive idea to drop previous hidden states directly, as proposed in Moon et al. [88], produces mixed results. We have observed that it helps the network to generalize better when not coupled with the forward dropout, but is usually no longer beneficial when used together with a regular forward dropout.

The problem is caused by the scaling of neuron activations during inference. Consider the hidden state update rule in the test phase of an LSTM network. For clarity, we assume every gate to be equal to 1:

$$\mathbf{h}_t = (\mathbf{h}_{t-1} + \mathbf{g}_t)p, \tag{3.3.11}$$

where $\mathbf{g}_t$ are update vectors computed by Eq. 3.3.5 and p is the probability to not drop a neuron. As h_{t-1} was, in turn, computed using the same rule, we can rewrite this equation as:

$$\mathbf{h}_t = ((\mathbf{h}_{t-2} + \mathbf{g}_{t-1})p + \mathbf{g}_t)p \tag{3.3.12}$$

Recursively expanding $\mathbf{h}$ for every timestep results in the following equation:

$$\mathbf{h}_t = ((((\mathbf{h}_0 + \mathbf{g}_0)p + \mathbf{g}_1)p + ...)p + \mathbf{g}_t)p \tag{3.3.13}$$

Pushing p inside parenthesis, Eq. 3.3.13 can be written as:

$$\mathbf{h}_t = p^{t+1}\mathbf{h}_0 + \sum_{i=0}^{t} p^{t-i+1}\mathbf{g}_i \tag{3.3.14}$$

Since p is a value between zero and one, sum components that are far away in the past are multiplied by a very low value and are effectively removed from the summation. Thus, even though the network is able to learn long-term dependencies, it is not capable of exploiting them during test phase. Note that our assumption of all gates being equal to 1 helps the network to preserve hidden state, since in a real network gate values lie within (0, 1) interval. In practice trained networks tend to saturate gate values [63] what makes gates to behave as binary switches. The fact that Moon et al. [88] have achieved an improvement can be explained by the experimentation domain. Le et al. [74] have proposed a simple yet effective way to initialize vanilla RNNs and reported that they have achieved a good result in the Speech Recognition domain while having an effect similar to the one caused by Eq. 3.3.14. One can reduce the influence of this effect by selecting a low dropout rate. This solution however is partial, since it only increases the number of steps required to completely forget past history and does not remove the problem completely.

One important note is that the dropout function from Eq. 3.3.2 can be implemented as:

$$d(\mathbf{x}) = \begin{cases} mask * \mathbf{x}/p, & \text{if train phase} \\ \mathbf{x} & \text{otherwise} \end{cases} \tag{3.3.15}$$

In this case the above argument holds as well, but instead of observing exponentially decreasing hidden states during testing, we will observe exponentially increasing values of hidden states during training.

Our approach addresses the problem discussed previously by dropping the update vectors $\mathbf{g}$. Since we drop only candidates, we do not scale the hidden state directly. This allows for solving the scaling issue, as Eq. 3.3.14 becomes:

$$\mathbf{h}_t = p\mathbf{h}_0 + \sum_{i=0}^{t} p\ \mathbf{g}_i = p\mathbf{h}_0 + p\sum_{i=0}^{t} \mathbf{g}_i \tag{3.3.16}$$

Moreover, since we only drop differences that are added to the network's hidden state at each time-step, this dropout scheme allows us to use per-step mask sampling while still being able to learn long-term dependencies. Thus, our approach allows to freely apply dropout in the recurrent connections of a gated network without hindering its ability to process long-term relationships.

We note that the discussed problem does not affect vanilla RNNs because they overwrite their hidden state at every timestep. Lastly, the approach of Gal [33] is not affected by the issue as well.

3.4 Experiments

First, we empirically demonstrate the issues linked to memory loss when using various dropout techniques in recurrent nets (see Sec. 3.3.4). For this purpose we experiment with training LSTM networks on one of the synthetic tasks from [49], specifically the Temporal Order task. We then validate the effectiveness of our *recurrent dropout* on three public benchmarks: word and character-level Language Modeling and Named Entity Recognition comparing directly to alternative *recurrent dropout* methods from [88, 33].

3.4.1 Synthetic Task

Data. In this task the input sequences are generated as follows: all but two elements in a sequence are drawn randomly from {C, D} and the remaining two symbols from {A, B}. Symbols from {A, B} can appear at any position in the sequence. The task is to classify a sequence into one of four classes ({AA, AB, BA, BB}) based on the order of the symbols. We generate data so that every sequence is split into three parts with the same size and emit one meaningful symbol in first and second parts of a sequence. The prediction is taken after the full sequence has been processed. We use two modes in our

Sampling	Moon et al. [88]				Gal [33]; Ours			
	short sequences		medium sequences		short sequences		medium sequences	
	Train	Test	Train	Test	Train	Test	Train	Test
per-step	100%	100%	25%	25%	100%	100%	100%	100%
per-sequence	100%	25%	100%	<25%	100%	100%	100%	100%

Table 3.1: Accuracies on the Temporal Order task.

Dropout rate	Sampling	Moon et al. [88]		Gal [33]		Ours	
		Valid	Test	Valid	Test	Valid	Test
0.0	–	130.0	125.2	130.0	125.2	130.0	125.2
0.25	per-step	**113.0**	**108.7**	119.8	114.2	106.1	100.0
0.5	per-step	124.0	116.5	**118.3**	**112.5**	102.8	98.0
0.25	per-sequence	121.0	113.0	120.5	114.0	106.3	100.7
0.5	per-sequence	137.7	126.2	125.2	117.9	**103.2**	**96.8**
0.0	–	**94.1**	**89.5**	94.1	89.5	94.1	89.5
0.25	per-step	113.5	105.8	**92.9**	**88.4**	**91.6**	**87.0**
0.5	per-step	140.6	130.1	98.6	92.5	100.6	95.5
0.25	per-sequence	105.7	99.9	94.5	89.7	92.4	87.6
0.5	per-sequence	125.4	117.4	98.4	92.5	107.8	101.8

Table 3.2: Perplexity scores of the LSTM network on word level Language Modeling task (lower is better). Upper and lower parts of the table report results without and with forward dropout respectively. Networks with forward dropout use 0.2 and 0.5 dropout rates in input and output connections respectively. Values in bold show best results for each of the recurrent dropout schemes with and without forward dropout.

experiments: `Short` with sequences of length 15 and `Medium` with sequences of length 30.

Setup. We use LSTM with one layer that contains 256 hidden units and *recurrent dropout* with 0.5 strength. Network is trained by SGD with a learning rate of 0.1 for 5k epochs. The networks are trained on 200 mini-batches with 32 sequences and tested on 10k sequences.

Results. Table 3.1 reports the results on the Temporal Order task when *recurrent dropout* is applied using our method and methods from [88] and [33]. Using dropout from [88] with per-sequence sampling, networks are able to discover the long-term dependency, but fail to use it on the test set due to the scaling issue. Interestingly, in `Medium` case results on the test set are worse than random. Networks trained with per-step sampling exhibit different behaviour: in `Short` case they are capable of capturing the temporal

dependency and generalizing to the test set, but require 10-20 times more iterations to do so. In `Medium` case these networks do not fit into the allocated number of iterations. This suggests that applying dropout to hidden states as suggested in [88] corrupts memory cells hindering the long-term memory capacity of LSTMs.

In contrast, using our *recurrent dropout* methods, networks are able to solve the problem in all cases. We have also ran the same experiments for longer sequences, but found that the results are equivalent to the `Medium` case. We also note that the approach of [33] does not seem to exhibit the memory loss problem.

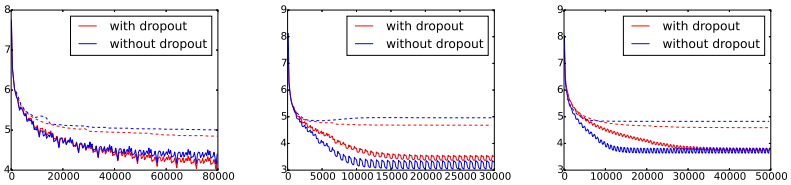
3.4.2 Word Level Language Modeling

Data. Following Mikolov et al. [84] we use the Penn Treebank Corpus to train our Language Modeling (LM) models. The dataset contains approximately 1 million words and comes with pre-defined training, validation and test splits, and a vocabulary of 10k words.

Setup. In our LM experiments we use recurrent networks with a single layer with 256 cells. Network parameters were initialized uniformly in [-0.05, 0.05]. For training, we use plain SGD with batch size 32 with the maximum norm gradient clipping [94]. Learning rate, clipping threshold and number of Backpropagation Through Time (BPTT) steps were set to 1, 10 and 35 respectively. For the learning rate decay we use the following strategy: if the validation error does not decrease after each epoch, we divide the learning rate by 1.5. The aforementioned choices were largely guided by the work of Mikolov et al. [86].

Results. Table 3.2 reports the results for LSTM networks. We also present results when the dropout is applied directly to hidden states as in [88] and results of networks trained with the dropout scheme of [33]. In addition, we report results of networks trained with no regularization and with dropout in only forward connetions in first rows of upper and lower parts of the table respectivelly. We make the following observations: (i) our approach shows better results than the alternatives; (ii) per-step mask sampling is better when dropping hidden state directly; (iii) on this task our method using per-step sampling seems to yield results similar to per-sequence sampling; (iv) in this case forward dropout yields better results than any of the three recurrent dropouts; and finally (v) both our approach and that of [33] are effective when combined with the forward dropout, though ours is more effective.

To demonstrate the effect of our approach on the learning process, we also present learning curves of recurrent networks trained with and without *recurrent dropout* (Fig. 5.10).



(a) RNN (b) LSTM (c) GRU

Figure 3.2: Learning curves when training without and with 0.25 per-step recurrent dropout. Solid and dashed lines show training and validation errors respectively. Best viewed in color.

As expected, models trained using our *recurrent dropout* scheme have slower convergence than models without dropout and usually have larger training error and lower validation errors. This behaviour is consistent with what is expected from a regularizer and is similar to the effect of the feed-forward dropout applied to non-recurrent networks [48].

3.4.3 Character Level Language Modeling

Data. We train our networks on the dataset described in the previous section. It contains approximately 6 million characters, and a vocabulary of 50 characters. We use the provided partitions train, validation and test partitions.

Setup. We use networks with 1024 units to solve the character level LM task. The characters are embedded into 256 dimensional space before being processed by the LSTM. All parameters of the networks are initialized uniformly in [-0.01, 0.01]. We train our networks on non-overlapping sequences of 100 characters. The networks are trained with the Adam [66] algorithm with initial learning rate of 0.001 for 50 epochs. We decrease the learning rate by 0.97 after every epoch starting from epoch 10. To avoid exploding gradients, we use MaxNorm gradient clipping with threshold set to 10.

Results. Results of our experiments are given in Table 3.3. Note that on this task regularizing only the recurrent connections is more beneficial than only the forward ones. In particular, LSTM networks trained with our approach and the approach of [33] yield a lower bit-per-character (bpc) score than those trained with forward dropout only. We attribute it to pronounced long term dependencies. In addition, our approach is the only one that improves over baseline LSTM with forward dropout. The overall best result is achieved by a network trained with our dropout with 0.25 dropout rate and per-step sampling, closely followed by network with Gal [33] dropout.

Dropout rate	Sampling	Moon et al. [88]		Gal [33]		Ours	
		Valid	Test	Valid	Test	Valid	Test
0.0	–	1.460	1.457	1.460	1.457	1.460	1.457
0.25	per-step	1.435	1.394	1.345	1.308	**1.338**	**1.301**
0.5	per-step	1.610	1.561	1.387	1.348	1.355	1.316
0.25	per-sequence	**1.433**	**1.390**	**1.341**	**1.304**	1.356	1.319
0.5	per-sequence	1.691	1.647	1.408	1.369	1.496	1.450
0.0	–	**1.362**	**1.326**	**1.362**	**1.326**	1.362	1.326
0.25	per-step	1.471	1.428	1.381	1.344	**1.358**	**1.321**
0.5	per-step	1.668	1.622	1.463	1.425	1.422	1.380
0.25	per-sequence	1.455	1.413	1.387	1.348	1.403	1.363
0.5	per-sequence	1.681	1.637	1.477	1.435	1.567	1.522

Table 3.3: Bit-per-character scores of the LSTM network on character level Language Modelling task (lower is better). Upper and lower parts of the table report results without and with forward dropout respectively. Networks with forward dropout use 0.2 and 0.5 dropout rates in input and output connections respectively. Values in bold show best results for each of the recurrent dropout schemes with and without forward dropout.

3.4.4 Named Entity Recognition

Data. To assess our recurrent Named Entity Recognition (NER) taggers when using *recurrent dropout* we use a public benchmark from CONLL 2003 [119]. The dataset contains approximately 300k words split into train, validation and test partitions. Each word is labeled with either a named entity class it belongs to, such as `Location` or `Person`, or as being not named. The majority of words are labeled as not named entities. The vocabulary size is about 22k words.

Setup. Previous state-of-the-art NER systems have shown the importance of using word context features around entities. Hence, we slightly modify the architecture of our recurrent networks to consume the context around the target word by preprocessing the inputs by a convolutional layer. The size of the convolutional kernel is fixed to 5 words (the word to be labeled, two words before and two words after) and the number of filters is fixed to 256. The recurrent layer size is 1024 units. The network inputs include word embeddings (initialized with pretrained word2vec embeddings [85] and kept static) and capitalization features. For training we use the Adam algorithm [66] with initial learning rate of 0.001. We train for 50 epochs and multiply the learning rate by 0.95 after every epoch starting at epoch 10. We also combine our *recurrent dropout* with the conventional forward dropout with the rate 0.2 in input and 0.5 in output connections. Lastly, we

Dropout rate	Sampling	Moon et al. [88]		Gal [33]		Ours	
		Valid	Test	Valid	Test	Valid	Test
0.0	–	88.56	84.46	88.56	84.46	88.56	84.46
0.25	per-step	**88.79**	**84.80**	88.95	84.34	89.27	84.78
0.5	per-step	88.68	84.43	88.66	84.33	89.06	84.39
0.25	per-sequence	88.71	84.33	**88.54**	**84.88**	**89.32**	**84.95**
0.5	per-sequence	88.06	83.92	89.05	84.22	88.94	84.32
0.0	–	90.53	86.99	90.53	86.99	90.53	86.99
0.25	per-step	90.86	87.19	91.06	87.05	91.02	87.03
0.5	per-step	90.71	87.03	**90.76**	**87.23**	90.78	87.31
0.25	per-sequence	**90.73**	**87.32**	90.86	86.89	**90.99**	**87.33**
0.5	per-sequence	89.61	86.39	90.76	86.68	90.40	86.82

Table 3.4: F1 scores (higher is better) of the LSTM network on NER task (average scores over 3 runs). Upper and lower parts of the table report results without and with forward dropout respectively. Values in bold show best results for each of the recurrent dropout schemes with and without forward dropout.

found that using $relu(x) = max(x, 0)$ nonlinearity resulted in higher performance than $tanh(x)$. We train our network on randomly extracted samples up to 15 words long and use full sentences for testing.

Results. Table 3.4 reports the results of networks trained with and without forward dropout and compares our algorithm to approaches of [88] and [33]. We make the following observations: (i) forward dropout provides a much bigger improvement than recurrent one, what can be explained by the fact that long term dependencies are much less important in the NER task, in contrast to the Language Modeling; (ii) the results of our approach and dropout of [33] are comparable and both better than those of [88]; and (iii) all three approaches consistently outperform baseline networks without dropout in recurrent connections.

3.4.5 Twitter Sentiment Analysis

Data. We use Twitter sentiment corpus from SemEval-2015 Task 10 (subtask B) [102]. It contains 15k labeled tweets split into training and validation partitions. The total number of words is approximately 330k and the vocabulary size is 22k. The task consists of classifying a tweet into three classes: `positive`, `neutral`, and `negative`. Performance of a classifier is measured by the average of F1 scores of `positive` and `negative` classes. We evaluate our models on a number of datasets that were used for benchmarking during

Dropout rate	Twitter13	LiveJournal14	Twitter15	Twitter14	SMS13	Sarcasm14
RNN						
0	67.54	71.20	59.35	68.90	64.51	53.58
0.25	66.35	67.70	59.91	67.76	64.46	48.74
LSTM						
0	67.97	69.82	57.84	67.95	61.47	53.49
0.25	69.11	71.39	61.35	68.08	65.45	53.80
GRU						
0	67.09	70.80	59.07	67.02	67.11	51.01
0.25	69.04	72.10	60.34	69.65	65.73	54.77

Table 3.5: F1 scores (higher is better) on Sentiment Evaluation task

the last years.

Setup. We use recurrent networks in the standard sequence labeling manner - we input words to a network one by one and take the label at the last step. Similarly to [112], we use 1 million of weakly labeled tweets to pre-train our networks. We use networks composed of 500 neurons in all cases. Our models are trained with the RMSProp algorithm with a learning rate of 0.001. We use our *recurrent dropout* regularization with *per-step* mask sampling. All the other settings are equivalent to the ones used in the NER task.

Results. The results of these experiments are presented in Table 3.5. Note that in this case our algorithm decreases the performance of the vanilla RNNs while this is not the case for LSTM and GRU networks. This is due to the nature of the problem: differently from LM and NER tasks, a network needs to aggregate information over a long sequence. Vanilla RNNs notoriously have difficulties with this and our dropout scheme impairs their ability to remember even further. The best result over most of the datasets is achieved by the GRU network with *recurrent dropout*. The only exception is the Twitter2015 dataset, where the LSTM network shows better results.

3.5 Conclusions

This chapter presents a novel *recurrent dropout* method specifically tailored to the gated recurrent neural networks. Our approach is easy to implement and is even more effective when combined with conventional forward dropout. We have shown that applying dropout to arbitrary cell vectors results in suboptimal performance. We discuss in detail the cause of this effect and propose a simple solution to overcome it. The effectiveness

of our approach is verified on three public NLP benchmarks.

Our findings along with our empirical results help us to answer the questions posed in Section 3.1: (i) while is straight-forward to use dropout in vanilla RNNs due to their strong similarity with the feed-forward architectures, its application to LSTM networks is not so straightforward. We demonstrate that *recurrent dropout* is most effective when applied to *hidden state update* vectors in LSTMs rather than to *hidden states*; (ii) we observe an improvement in the network's performance when our *recurrent dropout* is coupled with the standard forward dropout, though the extent of this improvement depends on the values of dropout rates; (iii) per-step mask sampling is at least as good as per-sequence mask sampling when using our *recurrent dropout* method, with the most pronounced difference in the character level LM experiments, while the results of [88] and [33] are mixed.

4

Recurrent Attention Models

4.1 Introduction

CNNs have recently achieved excellent results in various of visual recognition tasks [34, 62, 69]. However, CNNs suffer from poor scalability with respect to the input image size. Due to high computational costs, despite the fact that images are almost always rescaled to a smaller size, most of the current state-of-the-art CNNs are trained either on multiple high-end GPUs or in a highly distributed environment [118]. One way to overcome this issue is to allow a network to dynamically attend to most informative regions of an image. The dynamic attention approach has received a lot of attention in recent years [3, 4, 30, 40, 87, 98] and prior to CNNs achieving state-of-the-art results [53, 70].

This work continues the line of research directed towards learning a RAM. It processes an image sequentially, gradually collecting information present in the image. The model achieves this by extracting a *glimpse* from an image, extracting features from the *glimpse* and updating its internal state. The most simple and straightforward form of *glimpse* extraction is cropping an area from an image. However, this procedure is not differentiable and thus prevents end-to-end learning with backpropagation. There are two major ways to overcome this issue. One is to use a different learning approach [87, 4, 98] and the other is to define a differentiable *glimpse* extraction procedure [54, 40]. Our work follows the second approach.

In this chapter we describe a deep differentiable Recurrent Attention Model, show that it can consistently learn to jointly locate and classify objects in an image, and demonstrate that it achieves performance superior to that of reinforcement learning based RAM. We then describe ways to increase the speed of learning, improve generalization, and prevent the learning of bad attention policies. We use these techniques to establish

a new state-of-the-art on the task of transcribing tightly cropped unsegmented house numbers.

4.2 Related Work

The computational efficiency of recognition algorithms has always received a lot of attention. It is especially important for object detection algorithms, since they usually need to perform multiple inferences per image. The amount of computation is mostly affected by the number of regions to be examined. This number is the largest for the simple sliding window approach and researchers have made a number of attempts to find alternatives to the sliding window approach such as a model that internally performs object detection step.

The most straightforward way to integrate object detection is to enable a model to extract crops from an image and process them subsequently. However, this approach poses a problem to backpropagation based learning, since the cropping is not differentiable. A number of attempts to overcome this issue have been made. Mnih et al. [87] have shown that it is possible to use reinforcement learning to successfully train an attention model. Ba et al. [3] and Sermanet et al. [111] extended this work and showed that RAM trained with reinforcement learning can be effective in challenging real world applications. Their work differs from ours in the training method: the authors have used a Reinforcement Learning based algorithm, while our model can be trained end-to-end with backpropagation. Ranzato [98] have used a two phase algorithm that optimizes location estimation and target predictions independently. Gregor et al. [40] have proposed a generative model that creates an image in an iterative fashion. Among other contributions, the authors have presented a differentiable crop operation and thus completely removed the need of dedicated attention learning algorithms. They have demonstrated that their model outperforms the one by Mnih et al. [87] on a synthetic classification task. Our work extends the model and investigates the classification performance it can achieve. Jaderberg et al. [54] have proposed a trainable module called Spatial Transformer, that is very similar in nature to a recurrent attention model. The authors use an affine transformation matrix to parametrize grid sampling points and a differentiable interpolation kernel to produce a crop from an image. Sønderby et al. [116] combine the Spatial Transformer modules with a Recurrent Neural Network. This work is the closest to ours in terms of the methodology, however the direct comparison is difficult due to a limited amount of experimental results in Sønderby et al. [116].

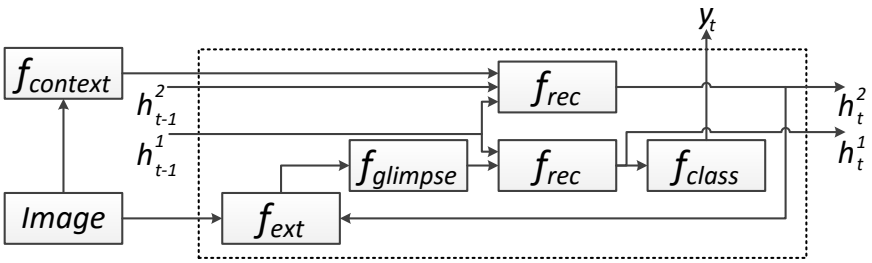


Figure 4.1: Illustration of one processing step of the proposed model. $f_{context}$, f_{ext}, $f_{glimpse}$, f_{class} and f_{rec} denote context, extraction, glimpse, classification and recurrent networks respectively. See text for details on their structure and tasks. The blocks inside the dashed rectangle are iterated.

4.3 Recurrent Attention Model

A Recurrent Attention Model is a model that processes information in a sequential manner. Its processing consists of three steps: choosing where to look, extracting features from the chosen location, and updating the RAM's internal state. These steps form one iteration, which is then repeated for as many times as required by the task at hand. This process is very different from one-shot approach of CNNs and has a number of potential advantages. It allows the model to ignore any kind of unnecessary information present in an image, collect data scattered over an image and naturally model object detection.

Figure 4.1 provides a graphical description of our model, which mostly follows the iterative scheme described before, with two notable differences. Firstly, we extract features from the whole frame before iterations start. These features encode a rough representation of an image and are used by the model to guide its attention. Secondly, we explicitly split the hidden state into two parts, one responsible for modeling attention and the other for aggregating features over multiple iterations. As shown in Figure 4.1, our model can be split into a number of relatively independent modules. Most of these modules are neural networks, each serving its distinct purpose. Now we will introduce these modules and comment on their structure and the tasks they perform.

Context network. The context network receives the whole image as input and outputs a feature vector. The job of the context network is to provide clues on where to deploy the model's attention. We usually parametrize this network by a CNN with two or three layers. In addition, it is possible to downsample an image before processing it with the context network to save computations.

Recurrent network. The core of our model is a two-layer recurrent neural network that aggregates information over multiple glimpses and controls the locations of the

glimpses. We initialize the states of both layers to zeros. In general, any kind of recurrent function can be used to implement this module. We use a Gated Recurrent Unit (GRU) [22] network for its good trade-off between stable learning and computational costs. There is a clear separation of tasks between the two recurrent layers: the first aggregates data over multiple iterations and is used as an input to the classification network, while the second one only affects attention. In contrast to the widely adopted bottom-up computation order, we compute activations of the second layer before activations of the first layer. The main reason for this top-down order of computation is that the context network is only connected to the second layer. Since we compute activations of the second layer prior to the first one, the context network features are able to affect the first glimpse. When the allowed number of iterations is small, i.e. 2 or 3, having the location of the first glimpse fixed can use a large portion of the allowed computations. We have experimented with a variant of our model with a single recurrent layer, but encountered severe with issues exploding gradients.

Glimpse extraction network. The extraction network maps the hidden state of the second recurrent layer to attention parameters and uses them to yield the current glimpse. We use a linear mapping ($\mathbf{y} = W\mathbf{x} + b$) from the hidden state of the second recurrent layer to the attention parameters. The module then extracts a NxN glimpse following the procedure described by Gregor et al. [40]. The authors use a NxN grid of Gaussian filters placed upon an image to create a glimpse. The grid is parametrized by the location of its center c_x and c_y, the distance between adjacent points δ, the variance of Gaussian filters σ and the sharpening coefficient γ. We make two alterations to the grid parametrization. First, we use rectangular attention and replace the δ parameter with a pair δ_x and δ_y. Second, we make sure that the predicted center of the sampling grid is always within the image by restricting c_x and c_y to be between -1 and 1.

Glimpse network. The glimpse network is a function that receives the current extracted glimpse and outputs a feature vector. Its responsibility is to extract a set of useful features that are later used by the recurrent network. In most of our experiments this function is implemented with a CNN.

Classification network. The classification network produces the prediction given a current state of the first recurrent layer. In general it outputs a prediction on every step. However the exact moments when to use supervision, i.e. only on last step, can vary. We have observed very little influence of this module on the final result, so we have used small networks, such as logistic regression or a fully connected network with one hidden layer.

Full model. Lastly, we put all the discussed modules together. Let $f_{context}$, f_{ext},

$f_{glimpse}$, f_{class} and f_{rec} denote context, extraction, glimpse, classification and recurrent networks respectively. Given input image I, our model can be defined by the following equations:

$$\mathbf{h}_t^2 = f_{rec}(\left[\mathbf{h}_{t-1}^1, f_{context}(I)\right], \mathbf{h}_{t-1}^2) \tag{4.3.1}$$

$$g = f_{ext}(I, \mathbf{h}_t^2) \tag{4.3.2}$$

$$\mathbf{h}_t^1 = f_{rec}(f_{glimpse}(g), \mathbf{h}_{t-1}^1) \tag{4.3.3}$$

$$\mathbf{y}_t = f_{class}(\mathbf{h}_t^1) \tag{4.3.4}$$

where $\mathbf{h}_t^1$, $\mathbf{h}_t^2$ and $\mathbf{y}_t$ are the hidden state of first and second layers and the model's output on step t respectively. Note that the second recurrent layer uses the hidden state of the first layer from previous step as input. Since each module of our model is differentiable, the whole model can be trained end-to-end with backpropagation. The most important module in this regard is the glimpse extraction network.

4.4 Experiments

First, our model is evaluated on synthetic tasks, derived from the MNIST dataset. We show that it achieves either superior or competitive results compared to a reinforcement learning based alternative [3]. We then train it to read house numbers from the publicly available Street View House Numbers (SVHN) dataset [89] and show that our model achieves state-of-the-art performance on the task of reading house numbers from an image. Lastly, we demonstrate that the attention mechanism is capable of learning a decent object localizer when trained on the Birds-200-2011 dataset [123].

We have used Theano [9] and Caffe [58] frameworks to implement our experiments. Unless otherwise noted, the Adam [66] algorithm with an initial learning rate of 0.0001 was used to determine the step size of the gradient descent. β_1, β_2 and ϵ hyperparameters were set to 0.9, 0.999, and 1e-8 respectively.

4.4.1 Cluttered MNIST

Data. The dataset is derived from the popular MNIST benchmark by randomly placing one or two randomly selected digits in a 100x100 image. In addition, to simulate clutter present in natural images, 8x8 fragments of other digits are randomly placed in the same image as well. We define three subtasks based on this dataset: classifying an image

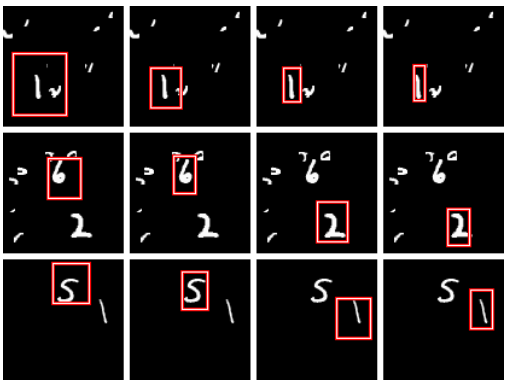


Figure 4.2: Examples of learned attention on digit recognition (first row), pair recognition (second row) and digit summation (third row). Best viewed in color.

Model	Error
Mnih et al. [87], 4 glimpses	9.41%
Mnih et al. [87], 8 glimpses	8.11%
Gregor et al. [40], 4 glimpses	4.18%
Gregor et al. [40], 8 glimpses	3.36%
Ours, without context	3.29%
Ours, with context	1.80%

Table 4.1: Test classification errors on the single digit recognition task

with one digit into 10 classes, classifying an image with two digits into 55 classes, each corresponding to a particular pair of digits and classifying an image with two digits into 19 classes based on the sum of the digits. In the last task we follow [3] and use an empty background. Note that since digits are sampled randomly, the distribution of classes is not uniform.

Setup. We have chosen the glimpse and classification networks to be fully connected networks with one hidden layer with 256 units. In this case the model's performance directly depends on its ability to discover a good attention policy. In addition, it allows for a fair comparison with the work of [3]. The number iterations was set to 4 in all

Model	Pair	Sum
Ba et al. [3]	5%	2.5%
Ours	4.3%	2.55%

Table 4.2: Test classification errors on the pair recognition task

experiments in this section. We have used a three layer CNN as context network.

We have found that it is difficult for our model to learn on weakly supervised images with two digits. There are two main problems that prevent it from learning a good attention policy. Firstly, the model tends to choose attention regions that include both digits instead of focusing on one and then switching to the other. This issue can be solved by imposing an upper limit on the size of the attention regions. However, it does not solve the problem completely, as in this case the network focuses on only one digit and never switches to the other. We have solved this issue by reparametrizing the reader's sampling grid and adding a regularization term to the cost function. The extraction network, discussed in Section 4.3, originally regresses absolute values of locations of grid centers. We have modified it to regress relative shifts $\triangle c_x$ and $\triangle c_y$ that are added to the previous center location. We then use Gaussian RBFs depending on relative shifts as regularization term:

$$R = ae^{-(\triangle c_x^2 + \triangle c_y^2)/b} \tag{4.4.1}$$

where a and b are hyperparameters that we set to 0.1 in our experiments. This term promotes the model to choose locations that are far from the current one. We use this term only on the third processing step to force the model that after two steps it should deploy its attention elsewhere.

Results. The results of our model on single digit recognition task can be found in Table 4.1. Our model achieves an almost two times smaller error than the model of [40]. It should be noted that the authors did not strive to optimize the performance of their model on the digit recognition task. To highlight the importance of the context network, we note that errors in this task are caused by either incorrect digit localization or errors of the classification network. When the context network is used the percentage of errors caused by mislocalization is less than 1% in contrast to about 10% without the context network.

Results of our model on pair classification and digit summation are given in Table 4.2. Our RAM outperforms the model of [3] on the pair classification task. This is not the case for digit summation and our model performs slightly worse on this task. We find it somewhat surprising, since the images are very similar for these two tasks. In fact, it is possible to use a model trained on pair recognition to initialize weights of all components of a model for digit summation except for the last classification layer, and vice versa. Such a model converges orders of magnitude faster than a randomly initialized one. Examples of learned attention mechanisms are presented in Figure 4.2.

Figure 4.3: Example of an image processed by forward (top row) and backward (bottom row) models. Computation proceeds from left to right, i.e. the first step corresponds to leftmost image in each row. Image was taken from the test partition. Best viewed in color.

4.4.2 Street View House Numbers

Data. In this section we apply our model to the medium sized dataset of house number images [89]. We follow Goodfellow et al. [36] and form a validation set by randomly choosing 5000 images from train and extra partitions. We also follow the same work in terms of data preprocessing and generate 64x64 tightly cropped images that contain all digits of a single house number. To do that, we use the ground truth bounding boxes, provided with the dataset, and compute a bounding box that includes all digits present in an image. We then increase the bounding box by 30% of its original size, crop the resulting area and rescale it to 64x64. We also follow Ba et al. [3] and generate 128x128 loosely cropped images by expanding the bounding boxes that enclose all digits in an image by 130%. In both cases we then convert the images to grayscale, subtract the mean image and divide by 128.

Setup. This task is different from the ones discussed in Section 4.4.1 since images contain variable numbers of objects. To address this, we train our model to read the image and output digits one by one. The model is trained to read an image from left to right. We add an extra class to encode that no more objects are present in an image and run our model until we receive the terminal label. The model is allowed to perform 3 glimpses per object. Since the dataset contains at most 5 objects in one image, the maximum possible amount of iterations is 18.

The SVHN dataset is more diverse than the one used in the previous section, so it is no longer sufficient to use a simple fully connected glimpse network. Following Ba et al. [3], we use a three layer CNN. However, we have observed that it leads to relatively slow convergence. We address this issue by training our model in two steps: first, we use a very simple glimpse network and train the model for a small number of epochs. After that we switch the simple glimpse network with the one that we actually would like to use. We have observed that it significantly speeds up training of the full model. Figure 4.4 shows learning curves of models trained from scratch and after 2 epochs of pretraining.

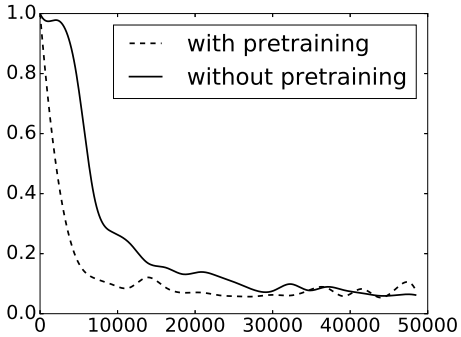


Figure 4.4: Validation errors of full models with and without pretraining for first 25 epochs. We omit training errors since they follow the same pattern

Computational costs of pretraining are negligible with respect to the amount of time required to train the full model, so we use this approach in all experiments presented in this section. Note that such curriculum approach does not affect the final result, as the full model trained from scratch achieves similar results but requires more iterations to do so.

This phenomenon can be explained by expressive power of the CNN: it is possible that it can extract reasonable features even when the attention mechanism is not very well trained. Thus, even though we train the full model jointly, the training actually happens in two steps. First, the attention mechanism learns to focus on important parts of an image and only then can the glimpse network learn to extract features that are good for classification. During the first phase of learning attention, regions constantly change and thus the nature of extracted glimpses changes as well, preventing the network from learning effectively. This effect is somewhat similar to the Internal Covariate Shift issue, discussed by Ioffe and Szegedy [51] and addressed by Batch Normalization.

We regularize our model with 0.5 dropout in the outputs of the glimpse network and 0.5 dropout in recurrent connections [107]. Data augmentation does not provide satisfactory improvements of the final result, so we do not use it and always feed the network with the original 64x64 images. This can be explained by the presence of the attention mechanism that focuses on the same part of an image and effectively counteracts data augmentation. To address this, we add noise to the sampling grid parameters. We uniformly sample noise coefficient n from [-α, α] and add it to the location of grid center ($c_x = c_x + n$, $c_y = c_y + n$). This corresponds to small random shifts of attention regions. In addition, we multiply the distance between sampling points by $1 + n$ ($\delta_x = \delta_x * (1 + n)$, $\delta_y = \delta_y * (1 + n)$), that results in either increase or decrease of corresponding

Model	64x64	128x128
Goodfellow et al. [36]	3.96%	–
Ba et al. [3]	3.9%	4.46%
Jaderberg et al. [54]	3.6%	3.9%
forward, single pass	4.5%	5.5%
forward, average	4.3%	5.2%
both, single pass	3.9%	4.45%
both, average	3.65%	4.35%
5 layers, both, average	3.4%	4.34%

Table 4.3: Test classification errors on full sequence house number recognition task

dimensions. We have used $\alpha = 0.1$ in all our experiments and sample noise independently for every parameter. This modification can be viewed as the data augmentation applied to attention regions instead of the full image. There are two options of how to use noise during testing. One is to not use corrupted attention parameters. The other is to allow noise injection, run the model multiple times and average the results over all runs. The second approach is similar to Markov Chain Monte Carlo (MCMC) averaging used by Ba et al. [3] and widely adopted oversampling [69]. It allows to achieve lower error rate at the cost of higher computational expenses when compared to the first option.

Results. The results of our model on transcribing house numbers are given in Table 4.3. Similarly to Ba et al. [3], we have observed that our model tends to overestimate the number of digits in an image. To address this, we train an additional model that reads an image from right to left. We show an example of a single run of both models in Figure 4.3. During testing we choose the smaller sequence length from two models and average predictions from them. This heuristic allows us to match the performance of the model with MCMC averaging from Ba et al. [3] with our model without averaging over multiple runs with sampling grid noise. When we allow injection of noise during testing and average over 10 runs, the performance of our model improves and almost achieves the performance of a Spatial Transformer based model [54]. All of the discussed models outperform a conventional CNN with eleven layers [36] while requiring much less computation. Lastly, we have found that our model with the three layer glimpse network slightly underfits training data due to regularization. This allowed us to add two more convolutional layers. The resulting five layer model yields a result better than that of Jaderberg et al. [54] and achieves state-of-the-art performance. However, it should be noted that the result of [54] is achieved with a single pass of a single model, while we use model averaging. In the case of loosely cropped house numbers our model achieves

Model	Accuracy
Jaderberg et al. [54]	84.1%
CNN, w/o localization	78.57%
CNN, GT localization	79.63%
CNN, RAM localization	79.77%

Table 4.4: Test classification accuracies on the bird recognition task

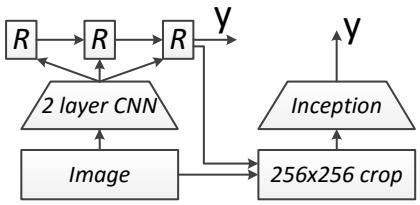


Figure 4.5: Illustration of our approach to fine grained classification. R blocks denote one iteration of the model.

a better result than that of [3] by a similar margin. Interestingly, in this case the model with a 5 layer glimpse network achieves only marginally better result. However, the Spatial Transformer based model of Jaderberg et al. [54] is still superior to both RAMs on loosely cropped images.

4.4.3 Fine-Grained Bird Recognition

Data. In this section we apply our model to the CUB-200-2011 dataset [123]. The dataset contains 12k images of various bird species split into train and test partitions of approximately the same size. We preprocess images by resizing to 256x256 and subtracting per-channel mean values.

Setup. We consider an Inception architecture with Batch Normalization [51] as our baseline model. We train it on the ImageNet1000 dataset and finetune it on the CUB-200-2011 dataset. We use the standard data augmentation during training and randomly crop 224x224 areas of an image and mirror them with a 0.5 probability. We do not use multiscale augmentation. During testing we average outputs of the model applied to ten crops as described by [69]. The baseline Inception network achieves 78.57% accuracy.

In this case, the object of interest is relatively easy to locate and usually occupies a large portion of an image. Hence, we do not use the context network for these experiments. In contrast to our previous experiments, in this case we apply the attention mechanism to convolutional images. In particular, we first process an image with two

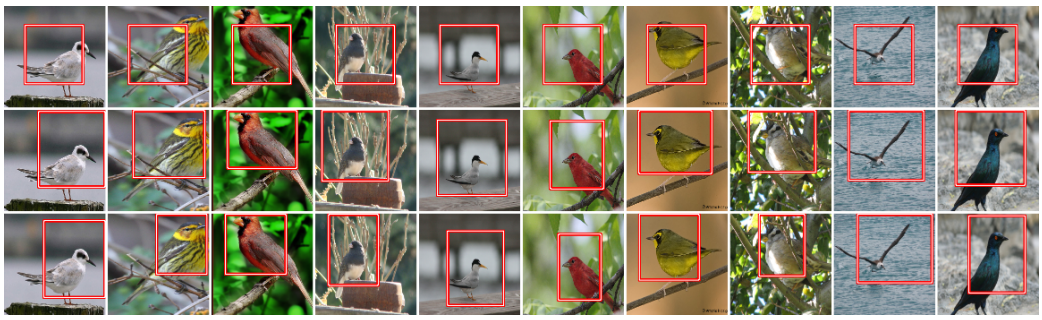


Figure 4.6: Examples of learned attention policy. Computation proceeds from top to bottom, i.e. the first step corresponds to topmost image in each column. Since in this experiment our model acts on a convolutional image, shown bounding boxes are expanded to approximately cover the corresponding area of the raw image. Note that the attention region on the first step is always the same, since we do not use the context network. Images were taken from the test partition. Best viewed in color.

sets of convolution-relu-pooling operations and only then use attention. Parameters of convolutional layers are pretrained on the ImageNet1000 dataset as part of a larger CNN with five convolutional layers and kept fixed throughout all experiments on this dataset. Our initial experiments have shown a substantial amount of overfitting. Therefore, we have changed the input of the classification network to be the output of the glimpse network instead of the hidden state of the first recurrent layer. In addition, we have hypothesized that the main source of overfitting is the classification network and that the full model does not have enough time to learn a good attention policy before the classification network overfits to the data. To address this issue, we have reset the weights of the last 200-way classification layer after every 10 epochs. We reset the weights of the classification network 10 times and then allow the model to converge. Indeed, this simple procedure reduces the validation error. However, while these modifications substantially reduce overfitting when compared to the original model, the baseline CNN still achieves a superior result. Thus, we use our model for an object localization step before baseline CNN.

To do that, we compute the attention region selected by the network on the last processing step and expand it to approximately cover the corresponding area of the raw image, extract this area from the image and resize it to 256x256 size. We process the whole dataset in this manner and train our baseline CNN on extracted images following exactly the same procedure as described before. This approach is schematically depicted in Figure 4.5. Note that the purpose of this experiment is to demonstrate that our model is capable of learning attention mechanisms on natural images. We do not achieve the

current state-of-the-art results [54] in these experiments.

Results. Results of our experiments are presented in Table 4.4. The CNN that uses our model for localization achieves 1.2% improvement with respect to the baseline model trained on whole images. Examples of localizations are presented in Figure 4.6. To put performance of localization into context, we use the provided ground truth (GT) bounding boxes as a "perfect" localizer. Interestingly, CNN with RAM localization slightly outperforms the one with GT localization. The difference between these two is rather small, however.

4.5 Discussion

In our experiments the Recurrent Attention Model consistently learns good attention policies, although in some cases it requires some extra built-in knowledge. Reinforcement learning based RAM is superior to ours in this regard, since it can learn to switch between two digits based on the classification error only. This issue deserves further attention, since objects in natural scenes are very likely to appear at different locations of an image.

Ba et al. [3] have observed that their model does not benefit much from dropout regularization. In contrast, we have found that our model does not achieve the results of alternative models when we do not. We speculate that this is due to a certain amount of stochasticity built in into the model of Ba et al. [3], while ours is fully deterministic. Our model does not benefit much from data augmentation. We attribute it to a certain amount of spatial invariance built into any RAM. If the localization mechanism is robust to small distortions of the input image, then it will always extract the same glimpses and diminish any kind of positive effects of data augmentation. However, data augmentation can help with learning to localize better by requiring a model to choose a different attention region for each crop.

Our model has a number of further applications. While it shows reasonable results on image classification, CNN based models are already very well suited for this task and we do not expect RAM to significantly outperform them in terms of classification accuracy. RAM can rather yield an improvement for tasks that have some dependencies between predicted labels, such as optical character recognition or object detection. In addition, our model is easily extendable to temporal data. Some of our early experiments suggest that RAM trained on static images can learn to follow an object in a video with its attention region. In OCR a recurrent attention model can jointly learn to locate characters, classify them and model language to resolve cases that are ambigious without context, i.e. whether a vertical stroke is a letter i or l. In object detection it can learn

relationships between objects, i.e. a whole-part relationship or the fact that certain objects are more or less likely to appear in the same image.

4.6 Conclusions

We have presented a deep recurrent model based on recent advances in fully differentiable RAMs and have experimentally shown that it achieves competitive results on the MNIST based synthetic benchmark and state-of-the-art result on transcribing house numbers from an image. We have shown that it can operate on outputs of convolutional layers and learn object localizers from image labels only. We have demonstrated that learned attention can be used subsequently in a recognition pipeline to boost the final result. Lastly, we have shown a set of procedures that decrease the testing error of our model and improve convergence speed. The two most important techniques are pretraining with a simple glimpse network and injection of noise into attention parameters.

5

Generative Models of Natural Texts

5.1 Introduction

Generative models of texts are currently at the cornerstone of natural language understanding enabling recent breakthroughs in machine translation [7, 126], dialogue modelling [110], abstractive summarization [104], etc.

Currently, RNN-based generative models hold state-of-the-art results in both unconditional [60, 44] and conditional [122] text generation. At a high level, these models represent a class of autoregressive models that work by generating outputs sequentially one step at a time where the next predicted element is conditioned on the history of elements generated thus far.

VAEs, recently introduced by [67, 101], offer a different approach to generative modeling by integrating stochastic latent variables into the conventional autoencoder architecture. The primary purpose of learning VAE-based generative models is to be able to generate realistic examples as if they were drawn from the input data distribution by simply feeding noise vectors through the decoder. Additionally, the latent representations obtained by applying the encoder to input examples give a fine-grained control over the generation process that is harder to achieve with more conventional autoregressive models. Similar to compelling examples from image generation, where it is possible to condition generated human faces on various attributes such as hair, skin color and style [128, 72], in text generation it should be possible to also control various attributes of the generated sentences, such as, for example, sentiment or writing style.

While training VAE-based models seems to pose little difficulty when applied to the tasks of generating natural images [6, 41] and speech [32], their application to natural text generation requires additional care [18, 83]. As discussed by Bowman et al. [18], the core difficulty of training VAE models is the collapse of the latent loss (represented by

the KL divergence term) to zero. In this case the generator tends to completely ignore latent representations and becomes a standard language model. This is largely due to the high modeling power of the RNN-based decoders which with sufficiently small history can achieve low reconstruction errors while not relying on the latent vector provided by the encoder.

In this work, we propose a novel VAE model for texts that is more effective in forcing the decoder to make use of latent vectors. Contrary to existing work, where both encoder and decoder layers are LSTMs, the core of our model is a feed-forward architecture composed of one-dimensional convolutional and deconvolutional [134] layers. This choice of architecture helps to gain more control over the KL term, which is crucial for training a VAE model. Given the difficulty of generating long sequences in a fully feed-forward manner, we augment our network with an RNN language model layer.

While being very successful, both autoregressive and VAE-based models still suffer from a number of problems. Arguably, the most prominent are exposure bias [10] and a mismatch between the NLL objective used during training and a task-specific metric that we would like to minimize [8]. Exposure bias stems from the fact that there is a mismatch between training and inference procedures. During training a model always receives histories that come from the well-behaved ground-truth sequences, whereas at inference it is conditioned on its own imperfect predictions.

Reinforcement Learning techniques that have recently received increased interest in the NLP community carry the promise of addressing both of these issues by allowing for task-specific (even non-differentiable) loss functions and incorporating sampling directly in the training process. However, previously used manually designed metrics based on n-gram matching such as BLEU [93] or ROUGE [75], are crude approximations for the true objective of generating samples that are perceptually indistinguishable from the real data.

The recently proposed GAN framework [37] goes beyond optimizing a manually designed objective by leveraging a discriminator that learns to distinguish between real and generated data samples. It thus mitigates both issues of NLL training, since it includes sampling into the training procedure and aims at generating samples that cannot be discriminated from the real data points. Despite their recent success in the image generation domain (both unconditional [13, 64] and conditional [99, 92]), applying GANs to text generation is still a challenging task. One of the many challenges that slows down the progress is the lack of proper evaluation, which is a largely unsolved problem and is an active area of research. Previous works studying GANs for text generation have either reported metrics specific to a family of algorithms [108] or resorted to BLEU

scores where a validation set is used as a reference [131, 43] to asses the quality of the generated samples. In this work we study the currently adopted evaluation approach to GAN models for language generation, explore their shortcomings and propose novel solutions.

5.2 Related Work

So far, the majority of neural generative models of text are built on the autoregressive assumption [71, 120]. Such models assume that the current data element can be accurately predicted given sufficient history of elements generated thus far. The conventional RNN based language models fall into this category and currently dominate the language modeling and generation problem in NLP. Neural architectures based on recurrent [60, 140, 44] or convolutional decoders [61, 28] provide an effective solution to this problem.

A recent work by Bowman et al. [18] tackles language generation problem within the VAE framework [67, 101]. The authors demonstrate that with some care it is possible to successfully learn a latent variable generative model of text. Although their model is slightly outperformed by a traditional LSTM [49] language model, their model achieves a similar effect as in computer vision where one can (i) sample realistic sentences by feeding randomly generated novel latent vectors through the decoder and (ii) linearly interpolate between two points in the latent space. Miao et al. [83] apply VAE to bag-of-words representations of documents and the answer selection problem achieving good results on both tasks. Yang et al. [129] discuss a VAE consisting of RNN encoder and CNN decoder so that the decoder's receptive field is limited. They demonstrate that this allows for a better control of KL and reconstruction terms. Hu et al. [50] build a VAE for text generation and design a cost function that encourages interpretability of the latent variables. Zhang et al. [135], Serban et al. [110] and Zhao et al. [139] apply VAE to sequence-to-sequence problems, improving over deterministic alternatives. Chen et al. [21] propose a hybrid model combining autoregressive convolutional layers with the VAE. The authors make an argument based on the Bit-Back coding [47] that when the decoder is powerful enough the best thing for the encoder to do is to make the posterior distribution equivalent to the prior. While they experiment on images, this argument is very relevant to the textual data. A recent work by Bousquet et al. [16] approaches VAEs and GANs from the optimal transport point of view. The authors show that commonly known blurriness of samples from VAEs trained on image data are a necessary property of the model. While the implications of their argument to models combining latent variables and autoregressive layers trained on non-image data

are still unclear, the argument supports the hypothesis of Chen et al. [21] that difficulty of training a hybrid model is not caused by a simple optimization difficulty but rather may be a more principled issue.

Various techniques to improve training of VAE models where the total cost represents a trade-off between the reconstruction cost and KL term have been used so far: KL-term annealing and input dropout [18, 115], imposing structured sparsity on latent variables [130] and more expressive formulations of the posterior distribution [100, 68]. A work by [82] follows the same motivation and combines GANs and VAEs allowing a model to use arbitrary complex formulations of both prior and posterior distributions. In Section 5.3.4 we propose another efficient technique to control the trade-off between KL and reconstruction terms.

GANs are a promising algorithm specifically designed to generate samples indistinguishable from real data. This has led to an increased interest in systematic comparison of different algorithms, both for images [78] and texts [77]. A recent study [78] shows that careful hyperparameter tuning is very important for a fair comparison of different image GAN models and significant improvement can be achieved with a larger computational budget rather that from a better algorithm. Another recent work studies a set of GANs targeted specifically at text generation [77]. However, it has all the drawbacks of the accepted evaluation approach, namely using n-gram based metrics and reporting only a single best result. It is thus difficult to draw a convincing conclusion based on this kind of comparison. Another related work conducts a study on the properties of Variational and Adversarial Autoencoder-based generative models [24].

Currently, the evaluation protocol adopted by the previous work on GAN-based text generation [131, 43] is primarily based on metrics derived from n-gram matching, e.g., BLEU and self-BLEU, which are used to assess sample quality and diversity. Additionally, a single best metric [131, 43] is reported which does not convey how sensitive various models are w.r.t. random initialization and hyperparameter choices. In this evaluation, we demonstrate that n-gram based metrics are inadequate for evaluation of unsupervised text generation models. Furthermore, we demonstrate that GAN models are extremely sensitive to random initialization and careful hyperparameter tuning is a must to have a meaningful comparison.

In this work we first address the issue of metrics used for evaluation of textual GANs. We then introduce a number of alternatives to the widely used BLEU and self-BLEU scores and demonstrate that they are capable of detecting a number of failure modes that the BLEU score does not capture. We then demonstrate the need of extensive hyperparameter tuning and conduct an initial set of experiments comparing a number of

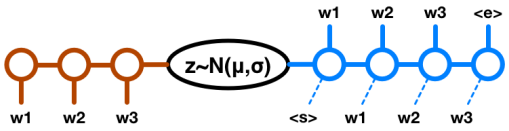


Figure 5.1: LSTM VAE model of [18]

recent GAN-based approaches to text generation.

5.3 Hybrid Variational Autoencoder Model

In this section we first briefly explain the VAE framework of Kingma and Welling [67], then describe our hybrid architecture where the feed-forward part is composed of a fully convolutional encoder and a decoder that combines deconvolutional layers and a conventional RNN. Finally, we discuss optimization recipes that help VAE to respect latent variables, which is critical training a model with a meaningful latent space and being able to sample realistic sentences.

5.3.1 Variational Autoencoder

The VAE is a recently introduced latent variable generative model, which combines variational inference with deep learning. It modifies the conventional autoencoder framework in two key ways. Firstly, a deterministic internal representation $\mathbf{z}$ (provided by the encoder) of an input $\mathbf{x}$ is replaced with a posterior distribution $q(\mathbf{z}|\mathbf{x})$. Inputs are then reconstructed by sampling $\mathbf{z}$ from this posterior and passing them through a decoder. To make sampling easy, the posterior distribution is usually parametrized by a Gaussian with its mean and variance predicted by the encoder. Secondly, to ensure that we can sample from any point of the latent space and still generate valid and diverse outputs, the posterior $q(\mathbf{z}|\mathbf{x})$ is regularized with its KL divergence from a prior distribution $p(\mathbf{z})$. The prior is typically chosen to be also a Gaussian with zero mean and unit variance, such that the KL term between posterior and prior can be computed in closed form [67]. The total VAE cost is composed of the reconstruction term, i.e., negative log-likelihood of the data, and the KL regularizer:

$$\begin{aligned} J_{vae} &= KL(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z})) \\ &\quad -\mathbb{E}_{q(\mathbf{z}|\mathbf{x})}[\log\ p(\mathbf{x}|\mathbf{z})] \end{aligned} \tag{5.3.1}$$

Kingma and Welling [67] show that the loss function from Eq (5.3.1) can be derived from the probabilistic model perspective and it is an upper bound on the true negative

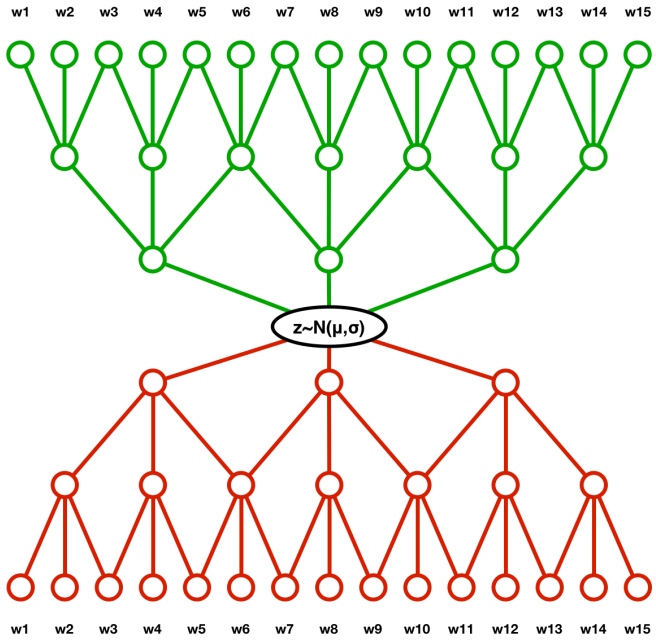
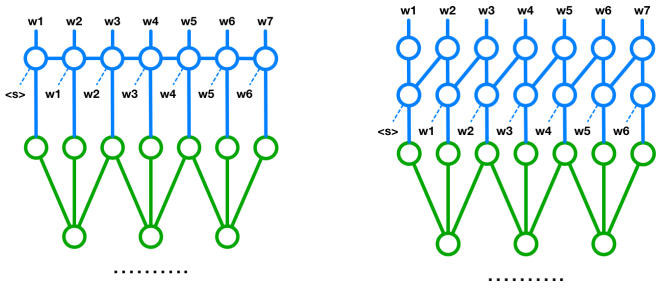


Figure 5.2: Fully feedforward component of the proposed model.

likelihood of the data. One can view a VAE as a traditional Autoencoder with some restrictions imposed on the internal representation space. Specifically, using a sample from the $q(\mathbf{z}|\mathbf{x})$ to reconstruct the input instead of a deterministic $\mathbf{z}$, forces the model to map an input to a region of the space rather than to a single point. The most straightforward way to achieve a good reconstruction error in this case is to predict a very sharp probability distribution effectively corresponding to a single point in the latent space [97]. The additional KL term in Eq (5.3.1) prevents this behavior and forces the model to find a solution with, on one hand, low reconstruction error and, on the other, predicted posterior distributions close to the prior. Thus, the decoder part of the VAE is capable of reconstructing a sensible data sample from every point in the latent space that has non-zero probability under the prior. This allows for straightforward generation of novel samples and linear operations on the latent codes. Bowman et al. [18] demonstrate that this does not work in the fully deterministic Autoencoder framework . In addition to regularizing the latent space, KL term indicates how much information the VAE stores in the latent vector.

Bowman et al. [18] propose a VAE model for text generation where both encoder and decoder are LSTM networks (Figure 5.1). We will refer to this model as LSTM VAE in the remainder of the thesis. The authors show that adapting VAEs to text generation is more challenging as the decoder tends to ignore the latent vector (KL term is close to



(a) Hybrid model with LSTM decoder

(b) Hybrid model with ByteNet decoder

Figure 5.3: Illustrations of our proposed models.

zero) and falls back to a language model. Two training tricks are required to mitigate this issue: (i) KL-term annealing where its weight in Eq (5.3.1) gradually increases from 0 to 1 during the training; and (ii) applying dropout to the inputs of the decoder to limit its expressiveness and thereby forcing the model to rely more on the latent variables. We will discuss these tricks in more detail in Section 5.3.4. Next we describe a deconvolutional layer, which is the core element of the decoder in our VAE model.

5.3.2 Deconvolutional Networks

A deconvolutional layer (also referred to as transposed convolutions [41] and fractionally strided convolutions [96]) performs spatial up-sampling of its inputs and is an integral part of latent variable generative models of images [96, 41] and semantic segmentation algorithms [90]. Its goal is to perform an "inverse" convolution operation and increase the spatial size of the input while decreasing the number of feature maps. This operation can be viewed as a backward pass of a convolutional layer and can be implemented by simply switching the forward and backward passes of the convolution operation. In the context of generative modeling based on global representations, the deconvolutions are typically used as follows: the global representation is first linearly mapped to another representation with small spatial resolution and large number of feature maps. A stack of deconvolutional layers is then applied to this representation, each layer progressively increasing spatial resolution and decreasing the amount of feature channels. The output of the last layer is an image or, in our case, a text fragment. A notable example of such a model is the deep network of [96] trained with adversarial objective. Our model uses a similar approach but is instead trained with the VAE objective.

There are two primary motivations for choosing deconvolutional layers instead of

the dominantly used recurrent ones: firstly, such layers have extremely efficient GPU implementations due to their fully parallel structure. Secondly, feed-forward architectures are typically easier to optimize than their recurrent counterparts, as the number of back-propagation steps is constant and potentially much smaller than in RNNs. Both points become significant as the length of the generated text increases. Next, we describe our VAE architecture that blends deconvolutional and RNN layers in the decoder to allow for better control over the KL-term.

5.3.3 Hybrid Convolutional-Recurrent VAE

Our model is composed of two relatively independent modules. The first component is a standard VAE where the encoder and decoder modules are parametrized by convolutional and deconvolutional layers respectively (see Figure 5.2). This architecture is attractive for its computational efficiency and simplicity of training.

The other component is a recurrent language model consuming activations from the deconvolutional decoder concatenated with the previous output characters. We consider two flavors of recurrent functions: a conventional LSTM network (Figure 5.3(a)) and a stack of masked convolutions also known as the ByteNet decoder from Kalchbrenner et al. [61] (Figure 5.3(b)). The primary reason for having a recurrent component in the decoder is to capture dependencies between elements of the text sequences – a hard task for a fully feed-forward architecture. Indeed, the conditional distribution $P(\mathbf{x}|\mathbf{z}) = P(x_1, \ldots, x_n|\mathbf{z})$ of generated sentences cannot be richly represented with a feed-forward network. Instead, it factorizes as: $P(x_1, \ldots, x_n|\mathbf{z}) = \prod_i P(x_i|\mathbf{z})$ where components are independent of each other and are conditioned only on $\mathbf{z}$. To minimize the reconstruction cost the model is forced to encode every detail of a text fragment. A recurrent language model instead models the full joint distribution of output sequences without having to make independence assumptions $P(x_1, \ldots, x_n|\mathbf{z}) = \prod_i P(x_i|x_{i-1}, \ldots, x_1, \mathbf{z})$. Thus, adding a recurrent layer on top of our fully feed-forward encoder-decoder architecture relieves it from encoding every aspect of a text fragment into the latent vector and allows it to instead focus on more high-level semantic and stylistic features.

Note that the feed-forward part of our model is different from the existing fully convolutional approaches of Dauphin et al. [28] and Kalchbrenner et al. [61] in two respects: firstly, while being fully parallelizable during training, these models still require predictions from previous time steps during inference and thus behave as a variant of recurrent networks. In contrast, expansion of the z vector is fully parallel in our model (except for the recurrent component). Secondly, our model down- and up-samples a

text fragment during processing while the existing fully convolutional decoders do not. Preserving spatial resolution can be beneficial to the overall result, but comes at a higher computational cost. Lastly, we note that our model imposes an upper bound on the size of text samples it is able to generate. While it is possible to model short texts by adding special padding characters at the end of a sample, generating texts longer than certain thresholds is not possible by design. This is not an unavoidable restriction, since the model can be extended to generate variable sized text fragments by, for example, variable sized latent codes. These extensions however are out of scope of this work.

5.3.4 Optimization Difficulties

The addition of the recurrent component results in optimization difficulties that are similar to those described by Bowman et al. [18]. In most cases the model converges to a solution with a vanishingly small KL term, thus effectively falling back to a conventional language model. Bowman et al. [18] have proposed to use input dropout and KL term annealing to encourage their model to encode meaningful representations into the $\mathbf{z}$ vector. We found that these techniques also help our model to achieve solutions with non-zero KL term.

KL term annealing can be viewed as a gradual transition from conventional deterministic Autoencoder to a full VAE. In this work we use linear annealing from 0 to 1. We have experimented with other schedules but did not find them to have a significant impact on the final result. As long as the KL term weight starts to grow sufficiently slowly, the exact shape and speed of its growth does not seem to affect the overall result. We have found the following heuristic to work well: we first run a model with KL weight fixed to 0 to find the number of iterations it needs to converge. We then configure the annealing schedule to start after the unregularized model has converged and last for no less than 20% of that amount.

While helping to regularize the latent vector, input dropout tends to slow down convergence. We propose an alternative technique to encourage the model to compress information into the latent vector: in addition to the reconstruction cost computed on the outputs of the recurrent language model, we also add an auxiliary reconstruction term computed from the activations of the last deconvolutional layer:

$$\begin{aligned} J_{aux} &= -\mathbb{E}_{q(\mathbf{z}|\mathbf{x})}[\log\ p(\mathbf{x}|\mathbf{z})] \\ &= -\mathbb{E}_{q(\mathbf{z}|\mathbf{x})}[\sum_t \log\ p(x_t|\mathbf{z})]. \end{aligned} \tag{5.3.2}$$

Since at this layer the model does not have access to previous output elements it needs to rely on the $\mathbf{z}$ vector to produce a meaningful reconstruction. The final cost minimized by our model is:

$$J_{hybrid} = J_{vae} + \alpha J_{aux} \tag{5.3.3}$$

where α is a hyperparameter, J_{aux} is the intermediate reconstruction term and J_{vae} is the bound from Eq (5.3.1). Expanding the two terms from Eq (5.3.3) gives:

$$\begin{aligned} J_{hybrid} &= KL(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z})) \\ -\mathbb{E}_{q(\mathbf{z}|\mathbf{x})}[\sum_t \log\ & p(x_t|\mathbf{z}, x_{<t})] \\ -\alpha\mathbb{E}_{q(\mathbf{z}|\mathbf{x})}[\sum_t & \log p(x_t|\mathbf{z})]. \end{aligned} \tag{5.3.4}$$

The objective function from Eq (5.3.4) puts a mild constraint on the latent vector to produce features useful for historyless reconstruction. Since the autoregressive part reuses these features, it also improves the main reconstruction term. We are thus able to encode information in the latent vector without hurting expressiveness of the decoder.

One can view the objective function in Eq 5.3.4 as a joint objective for two VAEs: one only feed-forward, as in Figure 5.2, and the other combining feed-forward and recurrent parts, as in Figures 5.3(a) and 5.3(b), that partially share parameters. Since the feed-forward VAE is incapable of producing reasonable reconstructions without making use of the latent vector, the full architecture also gains access to the latent vector through shared parameters. We note that this trick comes at a cost of worse result on the density estimation task, since part of the parameters of the full model are trained to optimize an objective that does not capture all the dependencies that exist in the textual data. However, the gap between purely deterministic LM and our model is small and easily controllable by the α hyperparameter. We refer the reader to Figure 5.6 for quantitative results regarding the effect of α on the performance of our model on the LM task.

5.4 GAN Models for Natural Language

GANs have had big success in generating real-valued data such as images. This has led to a huge number of newly proposed GAN-based approaches for image generation. While text generating GANs are not as numerous, they also differ significantly from each other. In our evaluation we have opted for benchmarking individual core GAN techniques that we decouple from the original models to ensure they can be compared on equal footing. In

this section we briefly review the models and introduce components that we benchmark. The models can be broadly divided into two large subclasses – continuous and discrete.

5.4.1 Continuous models

Continuous models for text closely follow how GANs are applied to images, i.e. they treat a sequence of tokens as a one-dimensional signal and directly backpropagate from the discriminator into the generator. We adopt the architecture of a continuous GAN model for language generation from [42]. The generator is a stack of one-dimensional transposed convolutions and the discriminator is a stack of one-dimensional convolutional layers. The use of continuous generator outputs allows for straightforward application of GANs to text generation. To train this model we use the proposed WGAN-GP [42] objective:

$$\min_G \max_D J(D,G) = \mathbb{E}_{\mathbf{x}\sim p_{data}}[D(\mathbf{x})] - \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}}[D(G(\mathbf{z}))] - \lambda \mathbb{E}_{\hat{\mathbf{x}}\sim p_{\hat{\mathbf{x}}}}[(||\nabla_{\hat{\mathbf{x}}} D(\hat{\mathbf{x}})||_2 - 1)^2] \tag{5.4.1}$$

where D and G are the discriminator and the generator functions respectively. D is a stack of convolutional layers that consumes the outputs from G. The authors [42] use a feedforward network as a generator, which consists of a stack of transposed convolutional layers (Conv-Deconv). Such a generator, however, does not properly model the sequential structure of language. Thus, we also consider an RNN-based generator. To ensure it remains continuous and gradients from D can be backpropagated into G, instead of taking argmax or sampling from the output distribution at each step, we feed the entire softmax output as the input for the next step of G. This follows the generation process of RNN-based Language Models with the difference that it models the conditional distribution $p(x_t|x_{<t})$ implicitly. Another option is to make use of annealed softmax temperature, gumbel softmax [57] or straight-through estimator [12], but we leave it for the future research.

5.4.2 Discrete models

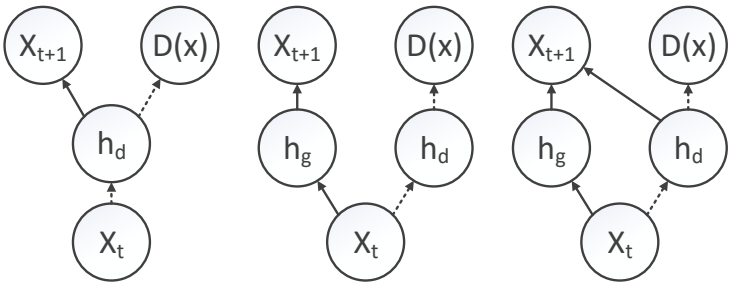
Discrete models learn the distribution over the next token $p(x_t|x_{<t})$ explicitly and thus sample (or take argmax) from the output distribution at each step. This makes the generator output non-differentiable and gradients from D can no longer be backpropagated through G.

To train such a non-differentiable generator one can use Reinforcement Learning (RL) where scores from D are treated as rewards. The majority of discrete GAN models for

text generation employ RL to train their models [131, 43, 31]. However, in addition to instability of GAN training one has to also address problems of RL training such as reward sparsity, credit assignment, large action space, etc. For example, one approach to the credit assignment issue is to use Monte-Carlo(MC) rollouts [131], which allows for providing a training signal to the generator at each step. Most commonly adopted solution to avoid reward sparsity is to pre-train the generator with the NLL objective, since sampling from a randomly initialized model in large action spaces makes it extremely hard to discover a well formed sequence. Many other RL techniques have been applied to NLP problems, for instance actor-critic methods, e.g., [8], or hierarchical learning [43]. However, these are out of the scope of this work.

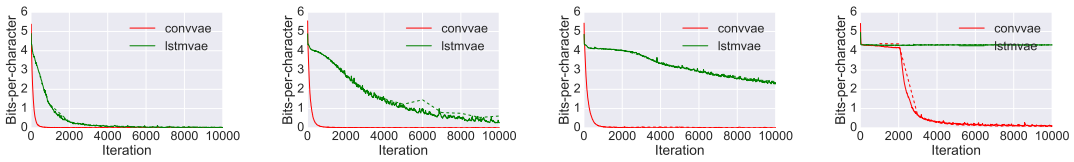
SeqGAN. In its simplest form RL-based GAN would employ a generator and a discriminator scoring the full sequence. The generator can then be trained with the REINFORCE objective $J_g = \sum_t D(y) * \log(p(y_t|y_{<t}))$. We refer to this variant as SeqGAN-reinforce. While this objective is enough in theory, in practice it has a number of problems. One such problem is credit assignment, where single per-sequence decision is an overly coarse feedback to train a generator. To address this, we consider two options. In SeqGAN-step we make use of the discriminator that outputs a decision at every step following previous research that has addressed credit assignment with this approach [31]. The generator's loss is then given by $J_g = \sum_t R_t * \log(p(y_t|y_{<t}))$, where $R_t = \gamma * R_{t+1} + D(y_{1:t})$. Such a formulation allows us to more accurately perform credit assignment and make sure that the generator does not behave greedily and take into account the long term effect a generated token might have. The issue however is that scoring an incomplete sequence might be difficult. To address this we follow the SeqGAN model [131] and employ MC rollouts to continue a sequence till the end. We then score these rollouts with a per-sequence discriminator and use its output as a reward. We will refer to this variant as SeqGAN-rollout in the rest of the chapter. The three considered variants are close to the original SeqGAN model and differ in their approach to the credit assignment problem.

LeakGAN. To address GAN instability in the RL training setup, a recent work [43] proposes to reveal discriminator's state to the generator. We decouple this idea from the complicated RL training setup used by the authors and study the utility of passing discriminator's state to the generator during training. Our initial experiments have shown that it is important to fuse discriminator's and generator's state with a non-linear function and thus we use a one-layer MLP to predict the distribution over next token. In this setup we only use per-step discriminators. We consider three variants of the *LeakGAN* model that differ in how a hidden state of D is made available to G: *leak*,



(a) LeakGAN-leak (b) LeakGAN-noleak (c) LeakGAN-mixed

Figure 5.4: Schematic description of the three considered LeakGAN models. Solid and dashed arrows represent weights learned in generator and discriminator phases respectively. h_g and h_d represent hidden states of the generator and discriminator respectively. Note that h_g is absent in LeakGAN-leak case. x_t and x_{t+1} are current and predicted tokens. $D(x)$ is output of the discriminator.



(a) 10 characters (b) 20 characters (c) 30 characters (d) 50 characters

Figure 5.5: Training curves of LSTM autoencoder and our model on samples of different length. Solid and dashed lines show training and validation curves respectively. Note that the model exhibits little to no overfitting since the validation curve follows the training one almost perfectly.

noleak and *mixed*, where the generator has access to the discriminator's, generator's and both hidden states respectively. Note that in LeakGAN-leak generator is an MLP and it does not maintain its own hidden state, only consuming that of the discriminator. Lastly, we do not update discriminator weights during generator's update phase. This way these features are external to the generator. We note that these variants are simpler when compared to the original LeakGAN model [43] since we do not use the complicated RL technique used by the authors and do not interleave GAN and NLL training. These simplifications allow us to decouple the influence of the architectural changes from other dimensions. Figure 5.4 presents graphical illustration of the three models.

5.5 Experiments on VAE-based Models

We use KL term annealing and input dropout when training the LSTM VAE models from Bowman et al. [18] and KL term annealing and regularized objective function from

Eq (5.3.3) when training our models. All models were trained with the Adam optimization algorithm [66] with decaying learning rate. We use Layer Normalization [5] in LSTM layers and Batch Normalization [51] in convolutional and deconvolutional layers.

Data. Our first task is character-level language generation performed on the standard Penn Treebank dataset [80]. One of the goals is to test the ability of the models to successfully learn the representations of long sequences. For training, fixed-size data samples are selected from random positions in the standard training and validation sets.

5.5.1 Comparison with LSTM VAE

Historyless decoding. We start with an experiment where the decoder is forced to ignore the history and has to rely fully on the latent vector. By conditioning the decoder only on the latent vector $\mathbf{z}$ we can directly compare the expressiveness of the compared models. For the LSTM VAE model historyless decoding is achieved by using the dropout on the input elements with the dropout rate equal to 1 so that its decoder is only conditioned on the $\mathbf{z}$ vector and, implicitly, on the number tokens generated so far. We compare it to our fully-feedforward model without the recurrent layer in the decoder (Figure 5.2). Both networks are parametrized to have comparable number of parameters.

To test how well both models can cope with the stochasticity of the latent vectors, we minimize only the reconstruction term from Eq. (5.3.1). This is equivalent to a pure autoencoder setting with stochastic internal representation and no regularization of the latent space. This experiment corresponds to an initial stage of training with KL term annealing when its weight is set to 0. We pursue two goals with this experiment: firstly, we investigate how do the two alternative encoders behave in the beginning of training and establish a lower bound on the quality of the reconstructions. Secondly, we attempt to put the Bit Back coding argument from Chen et al. [21] in context. The authors assume the encoder to be powerful enough to produce a good representation of the data. One interpretation of this argument applied to textual data is that factorizing the joint probability as $p(\mathbf{x}) = \prod_t p(x_t|x_{<t})$ provides the model with a sufficiently powerful decoder that does not need the latent variables. However, our experimental results suggest that LSTM encoder may not be a sufficiently expressive encoder for VAEs for textual data, potentially making the argument inapplicable.

The results are presented in Figure 5.5. Note that when the length of input samples reaches 30 characters, the historyless LSTM autoencoder fails to fit the data well, while the convolutional architecture converges almost instantaneously. The results appear even

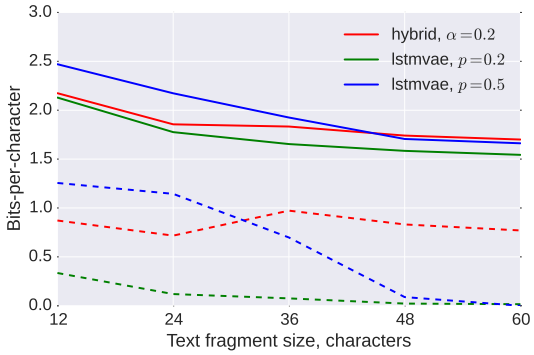


Figure 5.6: The full cost (solid lines) and its KL component (dashed lines) in bits-per-character of our Hybrid model trained with 0.2 α hyper-parameter vs. LSTM based VAE trained with 0.2 and 0.5 input dropout, measured on validation partition.

worse for LSTMs on sequences of 50 characters. To make sure that this effect is not caused by optimization difficulties, i.e. exploding gradients [94], we have searched over learning rates, gradient clipping thresholds and sizes of LSTM layers but were only able to get results comparable to those shown in Figure 5.5. Note that LSTM networks make use of Layer Normalization [5] which has been shown to make training of such networks easier. These results suggest that our model is easier to train than the LSTM-based model, especially for modeling longer pieces of text. Additionally, our model is computationally faster by a factor of roughly two, since we run only one recurrent network per sample and time complexity of the convolutional part is negligible in comparison.

Decoding with history. We now move to a case where the decoder is conditioned on both the latent vector and previous output elements. In these experiments we pursue two goals: firstly, we verify whether the results obtained on the historyless decoding task also generalize to a less restricted case. Secondly, we study how well the models cope with stochasticity introduced by the latent variables. Note that we do not attempt to improve the state-of-the-art result on the Language Modeling task but instead focus on providing an approach capable of generating long and diverse sequences. We experiment on the task to obtain a detailed picture of how are our model and LSTM VAE affected by various choices and compare the two models, focusing on how effective is the encoder at producing meaningful latent vector. However, we note that our model performs fairly well on the LM task and is only slightly worse than purely deterministic Language Model, trained in the same environment, and is comparable to the one of Bowman et al. [18] in this regard.

We fix input dropout rates at 0.2 and 0.5 for LSTM VAE and use auxiliary recon-

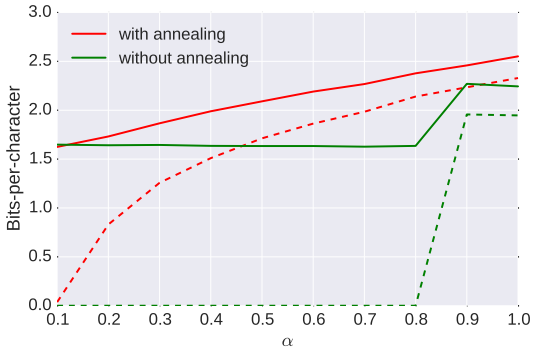


Figure 5.7: The full cost (solid line) and the KL component (dashed line) of our Hybrid model with LSTM decoder trained with various α, with and without KL term weight annealing, measured on the validation partition.

struction loss (Section 5.3.4) with 0.2 weight in our Hybrid model. The bits-per-character scores on differently sized text samples are presented in Figure 5.6. As discussed in Section 5.3.1, the KL term value indicates how much information the network stores in the latent vector. We observe that the amount of information stored in the latent vector by our model and the LSTM VAE is comparable when we train on short samples and largely depends on hyper-parameters α and p. When the length of a text fragment increases, LSTM VAE is able to put less information into the latent vector (i.e., the KL component is small) and for texts longer than 48 characters, the KL term drops to almost zero while for our model the ratio between KL and reconstruction terms stays roughly constant. This suggests that our model is better at encoding latent representations of long texts since the amount of information in the latent vector does not decrease as the length of a text fragment grows. In contrast, there is a steady decline of the KL term of the LSTM VAE model. This result is consistent with our findings from the historyless decoding experiment. Note that in both of these experiments the LSTM VAE model fails to produce meaningful latent vectors with inputs over 50 characters long. This further suggests that our Hybrid model encodes long texts better than the LSTM VAE.

5.5.2 Controlling the KL term

We study the effect of various training techniques that help control the KL term which is crucial for training a generative VAE model.

Aux cost weight. First, we provide a detailed view of how optimization tricks discussed in Section 5.3.4 affect the performance of our Hybrid model. Figure 5.7 presents

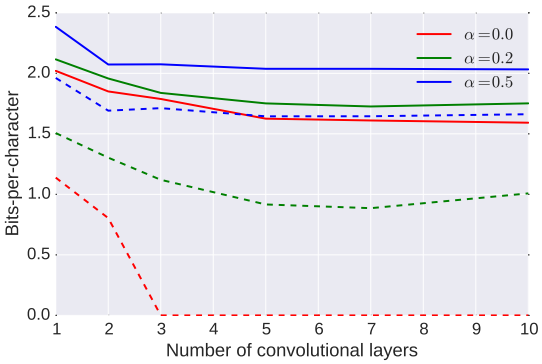


Figure 5.8: The full cost (solid line) and the KL component (dashed line) of our Hybrid model with ByteNet decoder trained with various number of convolutional layers, measured on the validation partition

@userid @userid @userid @userid @userid ...
I want to see you so much @userid #FollowMeCam ...
@userid @userid @userid @userid @userid ...
Why do I start the day today?
@userid thanks for the follow back
no matter what I'm doing with my friends they are so cute
@userid Hello How are you doing
I wanna go to the UK tomorrow!! #feelinggood #selfie #instago
@userid @userid I'll come to the same time and it was a good day too xx

Table 5.1: Random sample tweets generated by LSTM VAE (top) and our Hybrid model (bottom).

results of our model trained with different values of α from Eq. (5.3.3). Note that the inclusion of the auxiliary reconstruction loss slightly harms the bound on the likelihood of the data but helps the model to rely more on the latent vector as α grows. A similar effect on model's bound was observed by Bowman et al. [18]: increased input dropout rates force their model to put more information into the $\mathbf{z}$ vector but at the cost of increased final loss values. This is a trade-off that allows for sampling outputs in the VAE framework. Note that our model can find a solution with non-trivial latent vectors when trained with the full VAE loss provided that the α hyper-parameter is large enough. Combining it with KL term annealing helps to find non-zero KL term solutions at smaller α values.

	Rec	KL
LSTM, $p = 0.2$	67.4	1.0
LSTM, $p = 0.5$	77.1	2.1
LSTM, $p = 0.8$	93.7	3.8
Hybrid, $\alpha = 0.2$	58.5	12.5

Table 5.2: Breakdown into KL and reconstruction terms for char-level tweet generation. p refers to input dropout rate.

Receptive field. The goal of this experiment is to study the relationship between the KL term values and the expressiveness of the decoder. Without KL term annealing and input dropout, the RNN decoder in LSTM VAE tends to completely ignore information stored in the latent vector and essentially falls back to an RNN language model. To have a full control over the receptive field size of the recurrent component in our decoder, we experiment with masked convolutions (Figure 5.3(b)), which is similar to the decoder in ByteNet model from Kalchbrenner et al. [61]. We fix the size of the convolutional kernels to 2 and do not use dilated convolutions and skip connections as in the original ByteNet.

The resulting receptive field size of the recurrent layer in our decoder is equal to $N+1$ characters, where N is the number of convolutional layers. We vary the number of layers to find the amount of preceding characters that our model can consume without collapsing the KL term to zero.

Results of these experiments are presented in Figure 5.8. Interestingly, with the receptive field size larger than 3 and without the auxiliary reconstruction term from Eq. (5.3.3) ($\alpha = 0$) the KL term collapses to zero and the model falls back to a pure language model. This suggests that the training signal received from the previous characters is much stronger than that from the input to be reconstructed. Using the auxiliary reconstruction term, however, helps to find solutions with non-zero KL term component irrespective of receptive field size. Note that increasing the value of α results in stronger values of KL component. This is consistent with the results obtained with LSTM decoder in Figure 5.7.

5.5.3 Generating Tweets

In this section we present qualitative results on the task of generating tweets.

Data. We use 1M tweets[1] to train our model and test it on a held out dataset of 10k samples. We minimally preprocess tweets by only replacing user ids and urls with “@userid" and “url".

Setup. We use 5 convolutional layers with the ReLU non-linearity, kernel size 3 and stride 2 in the encoder. The number of feature maps is [128, 256, 512, 512, 512] for each layer respectively. The decoder is configured equivalently but with the amount of feature maps decreasing in each consecutive layer. The top layer is an LSTM with 1000 units. We have not observed significant overfitting. The baseline LSTM VAE model contained two distinct LSTMs both with 1000 cells. The models have comparable number of parameters: 10.5M for the LSTM VAE model and 10.8M for our hybrid model.

Results. Both VAE models are trained on the character-level generation. The breakdown of total cost into KL and reconstruction terms is given in Table 5.2. Note that while the total cost values are comparable, our model puts more information into the latent vector, further supporting our observations from Section 5.5.1. This is reflected in the random samples from both models, presented in Table 5.1. We perform greedy decoding during generation so any variation in samples is only due to the latent vector. LSTM VAE produces very limited range of tweets and tends to repeat "@userid" sequence, while our model produces much more diverse samples.

5.6 Evaluation of GANs for Natural Texts

Data. We perform our experiments on the Stanford Natural Language Inference (SNLI) [17] and MultiNLI datasets [125]. SNLI is a dataset of pairs of sentences where each pair is labeled with semantic relationship between two sentences. We discard these labels and use all unique sentences to train our model. The size of the resulting dataset is 600k unique sentences. We preprocess the data with the SentencePiece model with a vocabulary size equal to 4k. MultiNLI follows the SNLI structure but also provides a topic that a sentence pair comes from. This allows us to emulate mode collapse and thus measure the recall. We use SNLI for model comparison and MultiNLI for metric evaluation.

5.6.1 Metrics

Previous work on GANs for text generation has used unique n-grams [127] and dataset-level BLEU scores [131, 43] . Some works have also performed human evaluations of the

[1] a random sample collected using the Twitter API

generated samples [31]. While the ultimate goal is to obtain models that generate samples indistinguishable by a human rater from real ones, human evaluation of unsupervised models may not tell the full story. Most commonly, raters are presented with one sample at a time, which makes it possible to measure only precision but not recall. If a model produces only a few very good samples, it will score well on human evaluation, but models with severe mode-drop will not be penalized. Additionally, human evaluation is expensive and hence cannot be used for model tuning.

Thus, to make faster progress in GAN research for text we require an automatic metric that is: (i) fast to compute so that it can be used for model tuning; (ii) able to capture both precision and recall or ideally measure the distance between distributions of the real and generated data. Hence, our goal is to thoroughly evaluate the utility of various evaluation metrics for unsupervised text generation, as it is not obvious that previously reported n-gram matching scores computed by BLEU can be used to measure and compare GAN models in a fair way. In our study, we additionally propose to use a variant of Frechet Inception Distance (FID) [46] for text, scores assigned to real data by a Language Model trained on generated data [138], and scores assigned to samples by a pretrained Language Model.

N-gram based metrics. Typical metrics that researchers have used to evaluate textual GANs are the number of unique n-grams [127] and dataset level BLEU scores [131, 43]. We use BLEU4 throughout the experiments since we found the results to be similar for different sizes of N-grams. While they do give some insight into a model's behavior they have a number of drawbacks. The main criticism is that they do not capture semantic variations in generated texts and can only detect relatively simple problems with syntax.

Language Model score. Another feasible way to evaluate a model is to estimate the likelihood of samples under a pretrained Language Model. This, however, has a drawback that a model that always generates a few highly likely sentences will score very well. Despite this, it is still a useful metric reacting only on the quality of generated samples and thus is a good proxy for a model's precision.

Reverse Language Model score. A more general approach is to train a Language Model on samples from a model and then evaluate its performance on a held out set of real texts [138]. In this setting, however, the score is biased due to two factors. One is model bias caused by the imperfection of the LM that may not be good enough to model the data distribution. The other is data bias caused by the fact that we use a data sample to train a LM that will serve as a proxy for the true data distribution.

Frechet InferSent Distance. Another approach to evaluate a generative model is

through an embedding model. Originally proposed in the computer vision community, Frechet Inception Distance (FID) [46] computes the distance between distributions of features extracted from real and generated samples. Inception refers to a specific image classifier architecture [118]. To evaluate FID, the authors first make an assumption that features extracted by a classifier are normally distributed. They then propose to sample a number of data points from real and estimated distributions and compute

$$FID(r, g) = ||\mu_r - \mu_g||_2^2 + Tr(\Sigma_r + \Sigma_g - 2(\Sigma_r\Sigma_g)^{0.5}) \tag{5.6.1}$$

where (μ_r, Σ_r) and (μ_g, Σ_g) are the mean and covariance of embeddings of samples from data and model distributions respectively. While researchers have pointed out that FID has its drawbacks, namely that it is a biased metric [78, 14] and it makes unnecessary assumptions about feature distributions [14], it is a widely accepted metric in the Computer Vision community. In this work we adapt this metric for text by using InferSent text embedding model [26], which is a bidirectional LSTM with max pooling trained in a supervised manner. Unless otherwise noted, we use InferSent embedding model to compute sentence embeddings. However, we note that training a sequence embedding model is an ongoing research which will likely affect the quality of the discussed metric. Thus we omit specific embedding model from the metric name refer to it as Frechet Distance (FD).

Human Evaluation. Since the goal of a text generation model is to create samples that are indistinguishable from real ones, it is important to also perform human evaluation to assess their quality. We send 200 samples from each model (uniformly sampling from random restarts) to the human raters (using 3 raters per sample) asking them to score if the presented sentence is grammatically correct and understandable on a scale from 1 to 5 (with being 5 the best score).

5.6.2 Parameter optimization procedure

Since GANs are very sensitive to the choice of hyperparameters, we optimize these parameters using random search limiting the computational budget to 100 trials. Once we have discovered the best performing set of hyperparameters, we retrain a model with these hyperparameters 7 times and report mean and standard deviation for each metric to quantify how sensitive the model is to random initialization. To justify the need of such a procedure we show distributions of results achieved by three models during one run in Figure 5.11. As expected, GAN-based models are considerably less stable than Language Model.

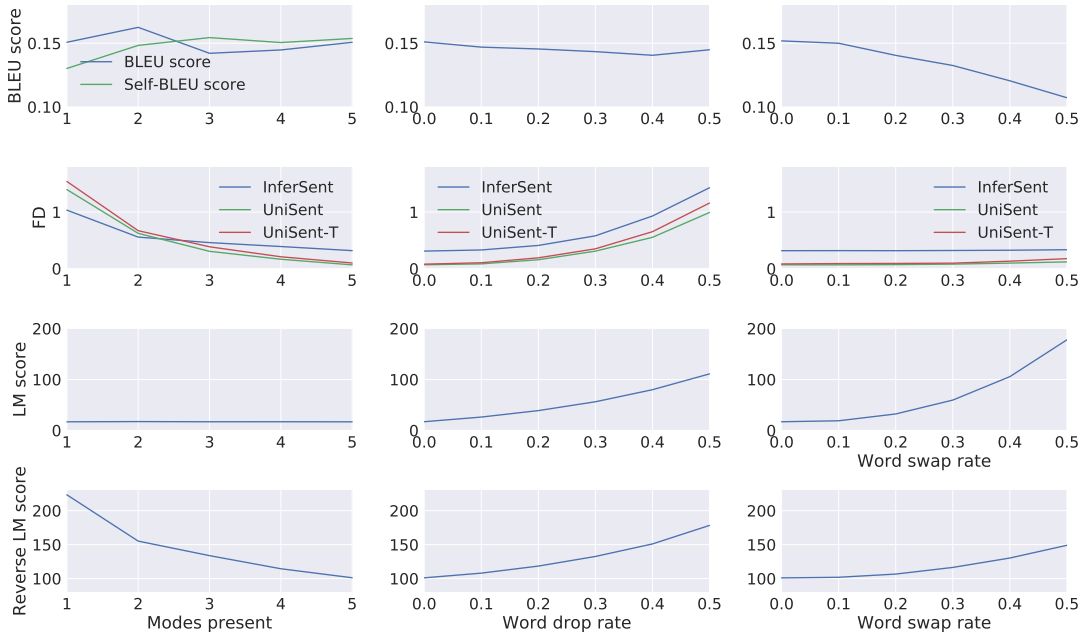


Figure 5.9: Scores assigned by four considered metrics for data with controllable amount of quality deterioration. Each row shows one metric and each column one task. Note that neither BLEU nor self-BLEU scores capture semantic deterioration of the data. For BLEU higher is better. For other metrics lower is better. We increase FDs obtained with UniSent and LM score embedding by a factor of 10 and decrease LM scores by the same factor for visualization purposes.

In addition, we generally see that the best results achieved in a run are usually obtained with a fortunate random seed supporting the second step of our procedure where we retrain a number of models and then report both mean and standard deviation using the best hyperparameters found during model tuning. We use the Adam optimizer [66] to train our models and tune its hyperparameters separately for the generator and the discriminator. When training models with per-step discriminators we also tune the discount factor γ.

5.6.3 Metric Evaluation

In the following experiments we measure how well BLEU, Language Model and FD scores capture syntactic and semantic variations.

Mode collapse. To emulate samples with varying degrees of diversity, we sample sentences from the train set using a fixed set of allowed topics. We then use the development

Figure 5.10: Learning curves of three differently sized Language Models. For all metrics lower is better.

set containing the full set of topics as a reference. An evaluation metric should be able to capture the fact that some topics, e.g. fictional sentences, are present in the reference but not in samples. Results of this experiment are shown in Figure 5.9. We find that results vary when the number of topics is small, so we run the evaluation 5 times and report the average. Note that BLEU and LM score fail to capture semantic variations. FD, on the other hand, drastically increases as we remove more and more topics. This also holds for the reversed LM score. To test robustness of FD to the choice of embedding model we use two additional sequence encoders [19] on the same data. One model is a mean pooled uni- and bigram embeddings followed by a feedforward network (UniSent). The other is a more computationally expensive Transformer [121] based model (UniSent-T). The models are trained with a combination of supervised and unsupervised learning. We find that all three models show comparable results suggesting that FD is robust to the choice of a sequence embedding model. We also evaluate the self-BLEU score that has been used to evaluate the degree of mode collapse [77]. To compute this metric we sample from a model twice and then compute BLEU score of one set of samples with respect to the other one. If a model suffers from mode collapse, then its samples are similar to each other and such a metric will produce high values. In this experiment, however, we observe that self-BLEU cannot detect this kind of mode collapse.

Sample quality. To measure metric sensitivity to the changes in the sample quality we introduce two types of perturbations in the samples. One is word dropout where we remove words with certain probability p that controls the quality of the samples. The other is word swapping where we take a fraction of words present in a sentence and randomly swap their places. Results of these experiments are presented in columns 2 and 3 of Figure 5.9. Interestingly, the BLEU score is not very sensitive to word dropping. FD, on the other hand, significantly worsens under heavy word dropout. The situation

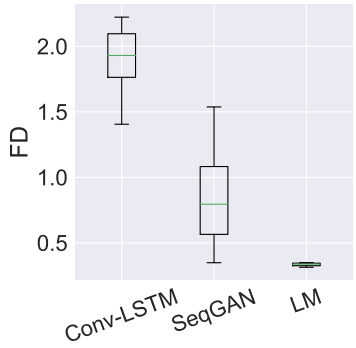


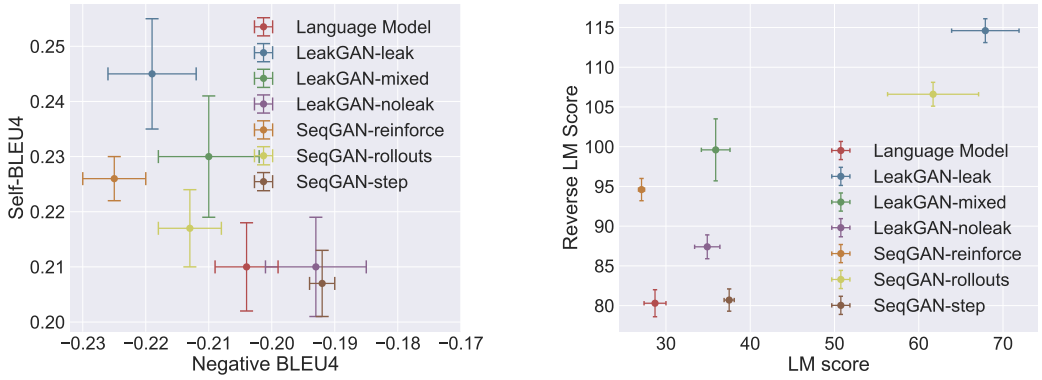
Figure 5.11: Distributions of FDs achieved by 30 best trials of three different models during hyperparameter search.

Conv-LSTM GAN (BLEU4=0.197, FD=1.464)
a young woman is sitting on a into into, on his sit
young woman woman woman while a group of son
the people are hair with sons
a little girl is wearing dogss
the children is at a red
man man white whiteing
Language Model (BLEU4=0.204, FD=0.273)
a man is competing in his ski class
the man is playing the accordion
she is the baby's sisters
the man is walking towards the fountain
a boy is climbing a tree lined
a man uses what looks to be a lawn mower

Table 5.3: Random samples from two models with close BLEU scores and considerably different FD.

is the opposite for word swapping, where the BLEU score is reacting more than FD. We attribute this behavior of FD to the underlying sequence embedding model. Since we use a bi-directional LSTM with max-pooling, it might have learned to be position-invariant due to pooling and is thus having difficulties detecting this kind of syntactic perturbations. Further research on better sequence embedding models is likely to improve the quality of evaluation with FD. LM score successfully captures decreased quality of samples but does not react to decreased diversity. Reversed LM score is sensitive to all three types of deteriorations.

In our second experiment we train three LSTM Language Models with one hidden layer with sizes 32, 256 and 1024. In this setting a larger model consistently achieves lower perplexity scores and thus we expect a metric to be able to detect that larger model produces better samples. In addition, we evaluate the models during training to also get the FD and LM score curves. Results of this experiment are shown in Figure 5.10. Note that all three metrics exhibit strong correlation and generally maintain ordering between differently sized models and different checkpoints of the same model. Our experiments suggest that both FD and reverse LM score can be successfully used as a metric for unsupervised sequence generation models. We generally observe reverse LM score to be more sensitive. However, it is prohibitively expensive to use during tuning. We thus opt for FD as a metric to optimize during hyperparameter searches. We report both for fully trained models and encourage other researchers to also make use of these metrics for evaluation. Lastly, we note that neither of the proposed metrics is perfect since they do



(a) BLEU scores

(b) LM scores

Figure 5.12: Results of best models shown on two complementary axes. We show negative values of BLEU4 for visualization purposes. Note that according to BLEU scores three models have comparable results, while LM scores show significantly better results for one model. We omit Conv-Deconv and Conv-LSTM models from these Figures since they show results considerably worse than those of other models.

not detect overfitting. Indeed, if a model simply remembers the training set and samples from it uniformly then this model will score perfectly on each of the proposed metrics. The same observation holds for BLEU scores. In addition, no metric provides breakdown into precision and recall. We leave these issues for further research.

5.6.4 GAN model comparison

For all GAN models that we compare we fix the generator architecture to be a one-layer Long Short-Term Memory (LSTM) network (except for the Conv-Deconv model). Other types of generators show promise [121], but we leave them for further research.

Table 5.4 shows the results obtained by various models using our evaluation procedure. We make the following observations: (i) discrete GAN models outperform continuous ones, which could be attributed to the pretraining step – most discrete models barely achieve non-random results without supervised pretraining; (ii) SeqGAN-reinforce achieves lower LM score and higher human ratings than the Language Model but higher reverse LM scores, suggesting improved precision at large cost to recall; (iii) Most of GAN models achieve higher BLEU scores than the LM, while other metrics disagree, showing that looking only at BLEU scores would put the LM at a significant disadvantage; (iv) No GAN model is convincingly better than the LM. However, the LM is not convincingly better than SeqGAN-reinforce either. While the LM achieves lower FD,

Metric	Language Model	Conv-LSTM	Conv-Deconv
Unique 4grams ↑	$43.5k \pm 1.7k$	$35k \pm 1.4k$	$24.9k \pm 1.6k$
BLEU4 ↑	0.204 ± 0.005	0.197 ± 0.003	0.08 ± 0.02
Self-BLEU4 ↓	0.21 ± 0.008	0.34 ± 0.02	0.45 ± 0.11
FD ↓	0.273 ± 0.001	1.464 ± 0.087	1.81 ± 0.11
LM score ↓	28.7 ± 1.3	221 ± 15	2800 ± 1100
Reverse LM score ↓	80.3 ± 1.7	2273 ± 358	4000 ± 0.3
Human evaluation ↑	3.37 ± 0.08	1.4 ± 0.1	1.88 ± 0.2
	SeqGAN-reinforce	SeqGAN-step	SeqGAN-rollouts
Unique 4grams ↑	$34.9k \pm 0.7k$	$56.2k \pm 1.6k$	$38.2k \pm 0.8k$
BLEU4 ↑	0.225 ± 0.005	0.192 ± 0.002	0.213 ± 0.005
Self-BLEU4 ↓	0.226 ± 0.004	0.207 ± 0.006	0.217 ± 0.007
FD ↓	0.316 ± 0.005	0.364 ± 0.01	0.348 ± 0.006
LM score ↓	27.1 ± 0.36	37.5 ± 0.6	61.7 ± 5.4
Reverse LM score ↓	94.6 ± 1.4	80.7 ± 1.4	106.6 ± 1.5
Human evaluation ↑	3.49 ± 0.22	3.27 ± 0.16	2.78 ± 0.08
	LeakGAN-leak	LeakGAN-noleak	LeakGAN-mixed
Unique 4grams ↑	$45k \pm 1.3k$	$54.4k \pm 2.8k$	$45.3k \pm 2.4k$
BLEU4 ↑	0.219 ± 0.007	0.193 ± 0.008	0.21 ± 0.008
Self-BLEU4 ↓	0.245 ± 0.01	0.21 ± 0.009	0.23 ± 0.011
FD ↓	0.4 ± 0.009	0.385 ± 0.02	0.352 ± 0.008
LM score ↓	67.9 ± 4	34.9 ± 1.5	35.9 ± 1.7
Reverse LM score ↓	114.3 ± 1.6	87.4 ± 1.5	99.5 ± 3.9
Human evaluation ↑	2.47 ± 0.28	3.35 ± 0.22	3.22 ± 0.15

Table 5.4: Results of best models obtained with our evaluation procedure. For brevity, we report only BLEU4 scores in this table. We have measured scores humans assign to real samples for reference and obtained a value of 4.27. ↓ means lower is better, ↑ higher is better.

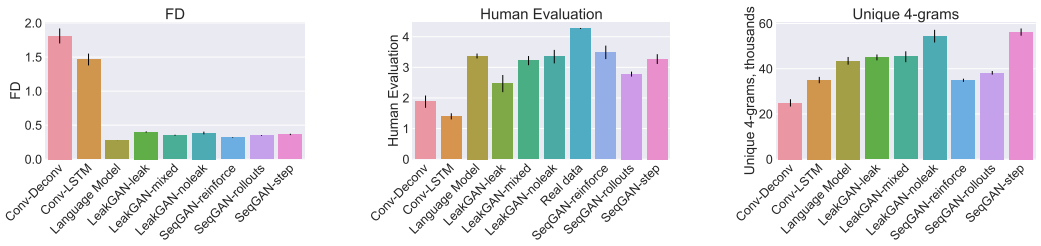


Figure 5.13: Results of models on FD, Human evaluation and Unique 4-grams.

LM score and human evaluations prefer the GAN model. This further supports that it is important to report different metrics – reporting only FD would make the comparison biased towards the LM; (v) We do not observe improvements of models with access to the discriminator's state, suggesting that the previously reported good result [43] may be due to the RL setup; (vi) Supervised pretraining of the generator is extremely important, since training of every GAN model that achieves reasonable results includes pretraining step.

In addition, we generally observe that hyperparameter search favors low values of generator learning rates. This suggests that lower learning rates help to keep the generator weights close to a Language Model used to initialize the weights. However, we note that BLEU scores of the generated sequences improve suggesting, higher precision for GAN models. We expect metrics that are capable of revealing trade-offs between precision and recall to allow better understanding of what kind of generators GANs learn.

To further demonstrate that BLEU scores are not representative of a model's quality we present samples from the Conv-LSTM GAN and the Language Model in Table 5.3. We make the following observations: Conv-LSTM GAN's samples are qualitatively worse than those of the Language Model due to spelling and syntactic errors. Its sentences are also generally less coherent. However, the difference in BLEU score between these two models is less than 1 point, as shown in Table 5.4. It is thus difficult to draw conclusions from BLEU scores alone whether SeqGAN-rollout produces better samples than a Language Model since the difference in BLEU scores for these two models is also less than 1 point. FD and reverse LM score, on the other hand, reveal that samples from Conv-LSTM GAN are considerably worse than those from a Language Model.

Human evaluation supports FD and the reverse LM score and also assigns better scores to the Language Model. Note that in this particular case simply inspecting samples from Conv-LSTM GAN and a Language Model would suffice. We are, however, interested

in automated comparison of models, where BLEU scores seem to not show reliable results.

5.7 Conclusions

We have introduced a novel generative model of natural texts based on the VAE framework. Its core components are a convolutional encoder and a deconvolutional decoder combined with a recurrent layer. We have shown that the feed-forward part of our model architecture makes it easier to train a VAE and avoid the problem of KL-term collapsing to zero, where the decoder falls back to a standard language model thus inhibiting the sampling ability of VAE.

Additionally, we propose an efficient way to encourage the model to rely on the latent vector by introducing an additional cost term in the training objective. We observe that it works well on long sequences which is hard to achieve with purely RNN-based VAEs using the previously proposed tricks such as KL-term annealing and input dropout. Finally, we have extensively evaluated the trade-off between the KL-term and the reconstruction loss.

In addition, in this work we discuss a proper evaluation of GANs for language generation. We have discussed drawbacks of previously adopted evaluation using BLEU scores and focused on the Frechet Distance and reverse Language Model scores. Our results suggest that BLEU scores are insufficient to evaluate textual GAN systems. In contrast, we have shown that both FD and reverse LM scores can successfully detect deteriorations that BLEU is not sensitive to. In addition, we have proposed a more systematic evaluation protocol and shown evidence that it provides a better picture than just reporting the single best result.

We used the proposed protocol and metrics to evaluate a number of adversarial text generation systems. We found that properly tuned conventional Language Models yield better results than any of the considered GAN-based systems. In fact, with proper hyperparameter tuning we find that when evaluated with FD the best results are achieved when the learning rate of the GAN generator after pre-training is the lowest, which corresponds to not performing GAN training at all, further supporting the need of reporting a number of metrics. These results generally agree with those obtained by a recent study [77]. The authors find that most models yield worse results than a simple Language Model. However, they do not perform hyperparameter tuning and report only BLEU scores, which makes it difficult to draw a convincing conclusion from the proposed comparison.

6 Conclusions

In this work we have introduced a number of improvements to RNNs applied to natural texts and images. In this chapter we briefly summarize our contributions and outline directions for further research.

We have generalized the successful dropout regularization to RNNs by identifying the core issue of naive application of this regularization scheme. We have then experimentally demonstrated its effectiveness across a number of NLP tasks, including tagging, classification and language modeling. The approach have then been used by other researchers to establish state-of-the-art results in NLP tasks [44]. The introduced algorithm is readily available in Tensorflow, one of the major frameworks for Deep Learning research [1].

We have introduced a fully differentiable Recurrent Attention Model and have experimentally validated its effectiveness. The model was able to achieve a significantly lower error rate on a specifically designed semi-synthetic task when compared to a strong baseline. Furthermore, we have achieved state-of-the-art result on the task of transcribing unsegmented house number images. However, our model was not able to outperform a CNN based baseline on the large-scale Imagenet dataset. We attribute this to the nature of the dataset – most images already contain a single centered object occupying the majority of an image. Under these conditions, an attention mechanism cannot do much more than simply scan the whole image. We observe this behavior in our trained models, suggesting that an object detection task would be more suited to this kind of models. This suggest one direction of future research, where it is possible to train a model to perform object detection without explicit and expensive supervision, since it is capable of learning object detection only from classification error. Another direction is to apply the proposed Recurrent Attention Model to more challenging datasets.

Lastly, we have focused our research on generative models of natural texts based on combinations of conventional Language Models with Variational Autoencoders and

Generative Adversarial Networks. We have shown that a purely recurrent VAE for text suffers from poor convergence on long text snippets and addressed this issue with a hybrid architecture combining both recurrent and feedforward layers. This has allowed us to better control the KL-term of the VAE objective function leading to better generations on challenging text datasets. An immediate extension of this work is to build a system capable of generating texts based on external attributes provided by a user, such as sentiment or topic. A recent work [50] discusses such a model using similar approach.

Researchers working on GAN based models for text generation have been using BLEU scores to compare various models. In our work we demonstrate that this is a flawed metric that can yield very misleading results. In addition, it is not capable of detecting certain failure modes of text generation models. To address these issues, we have introduced a number of metrics to compare such models. Furthermore, we have introduced a common comparison protocol to ensure that every model being compared is allocated a similar amount of computational budget for hyperparameter tuning. We have then used the proposed metrics and protocol to perform an empirical evaluation of a number of GAN based models and to compare these models with a LM baseline. Somewhat surprisingly, we found that none of the considered models consistently outperforms the Language Model. Further research in this direction involves further improvements to becnhmarking protocol, design of a unified metric for evaluation of languge generation and better coverage of evaluated models.

In conclusion, in this work we have introduced a number of improvements to RNN based Machine Learning models. They boost regularization capabilities of Recurrent Neural Networks and open new directions of research in image analysis. We have studied the problem of generative modeling of natural texts and made a number of contributions, ranging from architectural improvements to better evaluation metrics.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Gregory S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian J. Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Józefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Gordon Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul A. Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda B. Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *CoRR*, abs/1603.04467, 2016. URL `http://arxiv.org/abs/1603.04467`.

[2] Martín Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein GAN. *CoRR*, abs/1701.07875, 2017. URL `http://arxiv.org/abs/1701.07875`.

[3] Jimmy Ba, Volodymyr Mnih, and Koray Kavukcuoglu. Multiple object recognition with visual attention. *CoRR*, abs/1412.7755, 2014.

[4] Jimmy Ba, Ruslan R Salakhutdinov, Roger B Grosse, and Brendan J Frey. Learning wake-sleep recurrent attention models. In *NIPS*. 2015.

[5] Lei Jimmy Ba, Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *CoRR*, abs/1607.06450, 2016.

[6] Philip Bachman. An architecture for deep, hierarchical generative models. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *NIPS*, pages 4826–4834. 2016.

[7] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014.

[8] Dzmitry Bahdanau, Philemon Brakel, Kelvin Xu, Anirudh Goyal, Ryan Lowe, Joelle Pineau, Aaron C. Courville, and Yoshua Bengio. An actor-critic algorithm for sequence prediction. *CoRR*, abs/1607.07086, 2016. URL `http://arxiv.org/abs/1607.07086`.

[9] Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian J. Goodfellow, Arnaud Bergeron, Nicolas Bouchard, and Yoshua Bengio. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

[10] Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. *CoRR*, abs/1506.03099, 2015. URL http://arxiv.org/abs/1506.03099.

[11] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[12] Yoshua Bengio, Nicholas Léonard, and Aaron C. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *CoRR*, abs/1308.3432, 2013. URL http://arxiv.org/abs/1308.3432.

[13] David Berthelot, Tom Schumm, and Luke Metz. BEGAN: boundary equilibrium generative adversarial networks. *CoRR*, abs/1703.10717, 2017. URL http://arxiv.org/abs/1703.10717.

[14] Mikolaj Binkowski, Dougal J. Sutherland, Michael Arbel, and Arthur Gretton. Demystifying MMD gans. *CoRR*, abs/1801.01401, 2018. URL http://arxiv.org/abs/1801.01401.

[15] Theodore Bluche, Christopher Kermorvant, and Jérôme Louradour. Where to apply dropout in recurrent neural networks for handwriting recognition? In *13th International Conference on Document Analysis and Recognition, ICDAR 2015, Tunis, Tunisia, August 23-26, 2015*, pages 681–685, 2015. doi: 10.1109/ICDAR.2015.7333848. URL http://dx.doi.org/10.1109/ICDAR.2015.7333848.

[16] Olivier Bousquet, Sylvain Gelly, Ilya Tolstikhin, Carl-Johann Simon-Gabriel, and Bernhard Schoelkopf. From optimal transport to generative modeling: the vegan cookbook. *CoRR*, abs/1705.07642, 2017. URL https://arxiv.org/abs/1705.07642.

[17] Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 2015.

[18] Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew M. Dai, Rafal Józefowicz, and Samy Bengio. Generating sentences from a continuous space. In *CONLL*, pages 10–21, 2016.

[19] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, Yun-Hsuan Sung, Brian Strope, and Ray Kurzweil. Universal sentence encoder. *CoRR*, abs/1803.11175, 2018. URL http://arxiv.org/abs/1803.11175.

[20] Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Return of the devil in the details: Delving deep into convolutional nets. *CoRR*, abs/1405.3531, 2014. URL http://arxiv.org/abs/1405.3531.

[21] Xi Chen, Diederik P. Kingma, Tim Salimans, Yan Duan, Prafulla Dhariwal, John Schulman, Ilya Sutskever, and Pieter Abbeel. Variational lossy autoencoder. *CoRR*, abs/1611.02731, 2016. URL http://arxiv.org/abs/1611.02731.

[22] KyungHyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *CoRR*, abs/1409.1259, 2014. URL http://arxiv.org/abs/1409.1259.

[23] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014. URL http://arxiv.org/abs/1412.3555.

[24] Ondrej Cífka, Aliaksei Severyn, Enrique Alfonseca, and Katja Filippova. Eval all, trust a few, do wrong to none: Comparing sentence generation models. *CoRR*, abs/1804.07972, 2018. URL http://arxiv.org/abs/1804.07972.

[25] Stanford CS class CS231n: Convolutional Neural Networks for Visual Recognition. Visualizing what convnets learn. https://github.com/BVLC/caffe/blob/master/examples/filter_visualization.ipynb , Accessed: 15 Aug, 2018.

[26] Alexis Conneau, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes. Supervised learning of universal sentence representations from natural language inference data. *CoRR*, abs/1705.02364, 2017. URL http://arxiv.org/abs/1705.02364.

[27] Tim Cooijmans, Nicolas Ballas, César Laurent, and Aaron C. Courville. Recurrent batch normalization. *CoRR*, abs/1603.09025, 2016. URL http://arxiv.org/abs/1603.09025.

[28] Yann N. Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks. *CoRR*, abs/1612.08083, 2016.

[29] Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and A. Noah Smith. Transition-based dependency parsing with stack long short-term memory. In *ACL*, pages 334–343. Association for Computational Linguistics, 2015. URL http://aclweb.org/anthology/P15-1033.

[30] S. M. Ali Eslami, Nicolas Heess, Theophane Weber, Yuval Tassa, Koray Kavukcuoglu, and Geoffrey E. Hinton. Attend, infer, repeat: Fast scene understanding with generative models. *CoRR*, abs/1603.08575, 2016.

[31] William Fedus, Ian J. Goodfellow, and Andrew M. Dai. Maskgan: Better text generation via filling in the _____. *CoRR*, abs/1801.07736, 2018. URL http://arxiv.org/abs/1801.07736.

[32] Marco Fraccaro, Søren Kaae Sø nderby, Ulrich Paquet, and Ole Winther. Sequential neural models with stochastic layers. In *NIPS*, pages 2199–2207. 2016.

[33] Yarin Gal. A theoretically grounded application of dropout in recurrent neural networks. *arXiv:1512.05287*, 2015.

[34] Ross B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015.

[35] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR. URL http://proceedings.mlr.press/v9/glorot10a.html.

[36] Ian J. Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, and Vinay D. Shet. Multi-digit number recognition from street view imagery using deep convolutional neural networks. *CoRR*, abs/1312.6082, 2013.

[37] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial networks. *CoRR*, abs/1406.2661, 2014.

[38] Alex Graves, Marcus Liwicki, Horst Bunke, Jürgen Schmidhuber, and Santiago Fernández. Unconstrained on-line handwriting recognition with recurrent neural networks. In *NIPS*, pages 577–584. 2008. URL http://papers.nips.cc/paper/3213-unconstrained-on-line-handwriting-recognition-with-recurrent-neural-networks.pdf.

[39] Alex Graves, Abdel-rahman Mohamed, and Geoffrey E. Hinton. Speech recognition with deep recurrent neural networks. *CoRR*, abs/1303.5778, 2013. URL http://arxiv.org/abs/1303.5778.

[40] Karol Gregor, Ivo Danihelka, Alex Graves, and Daan Wierstra. DRAW: A recurrent neural network for image generation. *CoRR*, abs/1502.04623, 2015.

[41] Ishaan Gulrajani, Kundan Kumar, Faruk Ahmed, Adrien Ali Taiga, Francesco Visin, David Vázquez, and Aaron C. Courville. Pixelvae: A latent variable model for natural images. *CoRR*, abs/1611.05013, 2016.

[42] Ishaan Gulrajani, Faruk Ahmed, Martín Arjovsky, Vincent Dumoulin, and Aaron C. Courville. Improved training of wasserstein gans. *CoRR*, abs/1704.00028, 2017. URL http://arxiv.org/abs/1704.00028.

[43] Jiaxian Guo, Sidi Lu, Han Cai, Weinan Zhang, Yong Yu, and Jun Wang. Long text generation via adversarial training with leaked information. *CoRR*, abs/1709.08624, 2017. URL http://arxiv.org/abs/1709.08624.

[44] David Ha, Andrew M. Dai, and Quoc V. Le. Hypernetworks. *CoRR*, abs/1609.09106, 2016.

[45] 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.

[46] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, Günter Klambauer, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a nash equilibrium. *CoRR*, abs/1706.08500, 2017. URL http://arxiv.org/abs/1706.08500.

[47] Geoffrey E. Hinton and Drew van Camp. Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the Sixth Annual ACM Conference on Computational Learning Theory, COLT 1993, Santa Cruz, CA, USA, July 26-28, 1993.*, pages 5–13, 1993. doi: 10.1145/168304.168306. URL http://doi.acm.org/10.1145/168304.168306.

[48] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012. URL http://arxiv.org/abs/1207.0580.

[49] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL http://dx.doi.org/10.1162/neco.1997.9.8.1735.

[50] Zhiting Hu, Zichao Yang, Xiaodan Liang, Ruslan Salakhutdinov, and Eric P. Xing. Controllable text generation. *CoRR*, abs/1703.00955, 2017. URL https://arxiv.org/abs/1703.00955.

[51] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

[52] Ozan Irsoy and Claire Cardie. Opinion mining with deep recurrent neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 720–728. Association for Computational Linguistics, 2014. URL http://aclweb.org/anthology/D14-1080.

[53] Laurent Itti, Christof Koch, and Ernst Niebur. A model of saliency-based visual attention for rapid scene analysis. *IEEE PAMI*, 1998. ISSN 0162-8828. doi: 10.1109/34.730558.

[54] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and koray kavukcuoglu. Spatial transformer networks. In *NIPS*. 2015.

[55] Herbert Jaeger and Harald Haas. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science*, pages 78–80, 2004.

[56] Herbert Jaeger, Mantas Lukosevicius, Dan Popovici, and Udo Siewert. Optimization and applications of echo state networks with leaky- integrator neurons. *Neural Networks*, 20(3):335 – 352, 2007. ISSN 0893-6080. doi: http://dx.doi.org/10.1016/j.neunet.2007.04.016. URL http://www.sciencedirect.com/science/article/pii/S089360800700041X. Echo State Networks and Liquid State Machines.

[57] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *CoRR*, abs/1611.01144, 2016. URL http://arxiv.org/abs/1611.01144.

[58] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[59] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 2342–2350. JMLR.org, 2015. URL http://dl.acm.org/citation.cfm?id=3045118.3045367.

[60] Rafal Józefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. Exploring the limits of language modeling. *CoRR*, abs/1602.02410, 2016.

[61] Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aäron van den Oord, Alex Graves, and Koray Kavukcuoglu. Neural machine translation in linear time. *CoRR*, abs/1610.10099, 2016.

[62] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *CVPR*, 2014.

[63] Andrej Karpathy, Justin Johnson, and Fei-Fei Li. Visualizing and understanding recurrent networks. *CoRR*, abs/1506.02078, 2015. URL http://arxiv.org/abs/1506.02078.

[64] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *CoRR*, abs/1710.10196, 2017. URL http://arxiv.org/abs/1710.10196.

[65] Yoon Kim, Yacine Jernite, David Sontag, and Alexander M. Rush. Character-aware neural language models. *CoRR*, abs/1508.06615, 2015. URL http://arxiv.org/abs/1508.06615.

[66] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[67] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. *CoRR*, abs/1312.6114, 2013.

[68] Diederik P. Kingma, Tim Salimans, and Max Welling. Improving variational inference with inverse autoregressive flow. *CoRR*, abs/1606.04934, 2016. URL http://arxiv.org/abs/1606.04934.

[69] 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.

[70] Hugo Larochelle and Geoffrey E. Hinton. Learning to combine foveal glimpses with a third-order Boltzmann machine. In *NIPS*. 2010.

[71] Hugo Larochelle and Iain Murray. The neural autoregressive distribution estimator. In *AISTATS*, pages 29–37, 2011.

[72] Anders Boesen Lindbo Larsen, Søren Kaae Sønderby, and Ole Winther. Autoencoding beyond pixels using a learned similarity metric. *CoRR*, abs/1512.09300, 2015. URL http://arxiv.org/abs/1512.09300.

[73] C. Laurent, G. Pereyra, P. Brakel, Y. Zhang, and Y. Bengio. Batch normalized recurrent neural networks. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2657–2661, March 2016. doi: 10.1109/ICASSP.2016.7472159.

[74] Quoc V. Le, Navdeep Jaitly, and Geoffrey E. Hinton. A simple way to initialize recurrent networks of rectified linear units. *CoRR*, abs/1504.00941, 2015. URL http://arxiv.org/abs/1504.00941.

[75] Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. In *Proc. ACL workshop on Text Summarization Branches Out*, page 10, 2004. URL http://research.microsoft.com/~cyl/download/papers/WAS2004.pdf.

[76] Pengfei Liu, Xipeng Qiu, Xinchi Chen, Shiyu Wu, and Xuanjing Huang. Multi-timescale long short-term memory neural network for modelling sentences and documents. In *ACL*. Association for Computational Linguistics, 2015.

[77] Sidi Lu, Yaoming Zhu, Weinan Zhang, Jun Wang, and Yong Yu. Neural text generation: Past, present and beyond. *CoRR*, abs/1803.07133, 2018. URL http://arxiv.org/abs/1803.07133.

[78] Mario Lucic, Karol Kurach, Marcin Michalski, Sylvain Gelly, and Olivier Bousquet. Are gans created equal? A large-scale study. *CoRR*, abs/1711.10337, 2017. URL http://arxiv.org/abs/1711.10337.

[79] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, and Ian J. Goodfellow. Adversarial autoencoders. *CoRR*, abs/1511.05644, 2015. URL http://arxiv.org/abs/1511.05644.

[80] Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19(2):313–330, 1993.

[81] James Martens and Ilya Sutskever. Learning recurrent neural networks with hessian-free optimization. In Lise Getoor and Tobias Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, ICML '11, pages 1033–1040, New York, NY, USA, June 2011. ACM. ISBN 978-1-4503-0619-5.

[82] Lars M. Mescheder, Sebastian Nowozin, and Andreas Geiger. Adversarial variational bayes: Unifying variational autoencoders and generative adversarial networks. *CoRR*, abs/1701.04722, 2017. URL http://arxiv.org/abs/1701.04722.

[83] Yishu Miao, Lei Yu, and Phil Blunsom. Neural variational inference for text processing. *CoRR*, abs/1511.06038, 2015.

[84] T. Mikolov, S. Kombrink, L. Burget, J.H. Cernocky, and Sanjeev Khudanpur. Extensions of recurrent neural network language model. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pages 5528–5531, May 2011. doi: 10.1109/ICASSP.2011.5947611.

[85] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In C.J.C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc., 2013. URL http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf.

[86] Tomas Mikolov, Armand Joulin, Sumit Chopra, Michaël Mathieu, and Marc'Aurelio Ranzato. Learning longer memory in recurrent neural networks. *CoRR*, abs/1412.7753, 2014. URL http://arxiv.org/abs/1412.7753.

[87] Volodymyr Mnih, Nicolas Heess, Alex Graves, and Koray Kavukcuoglu. Recurrent models of visual attention. *CoRR*, abs/1406.6247, 2014.

[88] Taesup Moon, Heeyoul Choi, Hoshik Lee, and Inchul Song. Rnndrop: A novel dropout for rnns in asr. *Automatic Speech Recognition and Understanding (ASRU)*, 2015.

[89] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011.

[90] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. *CoRR*, abs/1505.04366, 2015.

[91] Sebastian Nowozin, Botond Cseke, and Ryota Tomioka. f-gan: Training generative neural samplers using variational divergence minimization. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 271–279, 2016. URL http://papers.nips.cc/paper/6066-f-gan-training-generative-neural-samplers-using-variational-divergence-minimization.

[92] Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional image synthesis with auxiliary classifier gans. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 2642–2651, 2017. URL http://proceedings.mlr.press/v70/odena17a.html.

[93] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, July 6-12, 2002, Philadelphia, PA, USA.*, pages 311–318, 2002. URL http://www.aclweb.org/anthology/P02-1040.pdf.

[94] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, pages 1310–1318, 2013. URL http://jmlr.org/proceedings/papers/v28/pascanu13.html.

[95] Vu Pham, Christopher Kermorvant, and Jérôme Louradour. Dropout improves recurrent neural networks for handwriting recognition. *CoRR*, abs/1312.4569, 2013. URL http://arxiv.org/abs/1312.4569.

[96] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2015.

[97] Tapani Raiko, Mathias Berglund, Guillaume Alain, and Laurent Dinh. Techniques for learning binary stochastic feedforward neural networks. *CoRR*, abs/1406.2989, 2014.

[98] Marc'Aurelio Ranzato. On learning where to look. *CoRR*, abs/1405.5488, 2014.

[99] Scott E. Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. Generative adversarial text to image synthesis. *CoRR*, abs/1605.05396, 2016. URL http://arxiv.org/abs/1605.05396.

[100] Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 1530–1538, 2015. URL http://jmlr.org/proceedings/papers/v37/rezende15.html.

[101] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *ICML*, pages 1278–1286, 2014.

[102] Sara Rosenthal, Preslav Nakov, Svetlana Kiritchenko, Saif Mohammad, Alan Ritter, and Veselin Stoyanov. Semeval-2015 task 10: Sentiment analysis in twitter. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pages 451–463, Denver, Colorado, June 2015. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/S15-2078.

[103] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Neurocomputing: Foundations of research. chapter Learning Representations by Back-propagating Errors, pages 696–699. MIT Press, Cambridge, MA, USA, 1988. ISBN 0-262-01097-6. URL http://dl.acm.org/citation.cfm?id=65669.104451.

[104] Alexander M. Rush, Sumit Chopra, and Jason Weston. A neural attention model for abstractive sentence summarization. *CoRR*, abs/1509.00685, 2015.

[105] Tim Salimans and Diederik P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *CoRR*, abs/1602.07868, 2016. URL http://arxiv.org/abs/1602.07868.

[106] Stanislau Semeniuta and Erhardt Barth. Image classification with recurrent attention models. In *2016 IEEE Symposium Series on Computational Intelligence, SSCI 2016, Athens, Greece, December 6-9, 2016*, pages 1–7, 2016. doi: 10.1109/SSCI.2016.7850113. URL https://doi.org/10.1109/SSCI.2016.7850113.

[107] Stanislau Semeniuta, Aliaksei Severyn, and Erhardt Barth. Recurrent dropout without memory loss. *CoRR*, abs/1603.05118, 2016.

[108] Stanislau Semeniuta, Aliaksei Severyn, and Erhardt Barth. A hybrid convolutional variational autoencoder for text generation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*, pages 627–637, 2017. URL https://aclanthology.info/papers/D17-1066/d17-1066.

[109] Stanislau Semeniuta, Aliaksei Severyn, and Sylvain Gelly. On accurate evaluation of gans for language generation. *TFAGDM ICML workshop*, 2018.

[110] Iulian Vlad Serban, Alessandro Sordoni, Ryan Lowe, Laurent Charlin, Joelle Pineau, Aaron C. Courville, and Yoshua Bengio. A hierarchical latent variable encoder-decoder model for generating dialogues. *CoRR*, abs/1605.06069, 2016.

[111] Pierre Sermanet, Andrea Frome, and Esteban Real. Attention for fine-grained categorization. *CoRR*, abs/1412.7054, 2014.

[112] Aliaksei Severyn and Alessandro Moschitti. Twitter sentiment analysis with deep convolutional neural networks. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval, Santiago, Chile, August 9-13, 2015*, pages 959–962, 2015. doi: 10.1145/2766462.2767830. URL http://doi.acm.org/10.1145/2766462.2767830.

[113] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[114] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. In *NIPS*, pages 568–576. 2014.

[115] Casper Kaae Sønderby, Tapani Raiko, Lars Maaløe, Søren Kaae Sønderby, and Ole Winther. Ladder variational autoencoders. *CoRR*, abs/1602.02282, 2016. URL https://arxiv.org/abs/1602.02282.

[116] Søren Kaae Sønderby, Casper Kaae Sønderby, Lars Maaløe, and Ole Winther. Recurrent spatial transformer networks. *CoRR*, abs/1509.05329, 2015. URL http://arxiv.org/abs/1509.05329.

[117] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *NIPS*, pages 3104–3112, 2014.

[118] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[119] Erik F. Tjong Kim Sang and Fien De Meulder. Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003 - Volume 4*, CONLL '03, pages 142–147, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics. doi: 10.3115/1119176.1119195. URL http://dx.doi.org/10.3115/1119176.1119195.

[120] Aäron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. *CoRR*, abs/1601.06759, 2016.

[121] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 6000–6010, 2017. URL http://papers.nips.cc/paper/7181-attention-is-all-you-need.

[122] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. *CoRR*, abs/1411.4555, 2014.

[123] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The Caltech-UCSD Birds-200-2011 Dataset. Technical report, 2011.

[124] Xin Wang, Yuanchao Liu, Chengjie SUN, Baoxun Wang, and Xiaolong Wang. Predicting polarities of tweets by composing word embeddings with long short-term memory. In *ACL*, pages 1343–1353. Association for Computational Linguistics, 2015. URL http://aclweb.org/anthology/P15-1130.

[125] Adina Williams, Nikita Nangia, and Samuel R. Bowman. A broad-coverage challenge corpus for sentence understanding through inference. *CoRR*, abs/1704.05426, 2017. URL http://arxiv.org/abs/1704.05426.

[126] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016. URL http://arxiv.org/abs/1609.08144.

[127] Jingjing Xu, Xu Sun, Xuancheng Ren, Junyang Lin, Bingzhen Wei, and Wei Li. DP-GAN: diversity-promoting generative adversarial network for generating informative and diversified text. *CoRR*, abs/1802.01345, 2018. URL http://arxiv.org/abs/1802.01345.

[128] Xinchen Yan, Jimei Yang, Kihyuk Sohn, and Honglak Lee. Attribute2image: Conditional image generation from visual attributes. *CoRR*, abs/1512.00570, 2015. URL http://arxiv.org/abs/1512.00570.

[129] Zichao Yang, Zhiting Hu, Ruslan Salakhutdinov, and Taylor Berg-Kirkpatrick. Improved variational autoencoders for text modeling using dilated convolutions. *CoRR*, abs/1702.08139, 2017. URL http://arxiv.org/abs/1702.08139.

[130] Serena Yeung, Anitha Kannan, Yann Dauphin, and Li Fei-Fei. Epitomic variational autoencoder. *In submission to ICLR 2017*, 2016. URL http://ai.stanford.edu/~syyeung/resources/YeuKanDauLi16.pdf.

[131] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: Sequence generative adversarial nets with policy gradient. *CoRR*, abs/1609.05473, 2016. URL http://arxiv.org/abs/1609.05473.

[132] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *CoRR*, abs/1409.2329, 2014. URL http://arxiv.org/abs/1409.2329.

[133] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *ECCV*, pages 818–833, 2014. doi: 10.1007/978-3-319-10590-1_53. URL http://dx.doi.org/10.1007/978-3-319-10590-1_53.

[134] Matthew D. Zeiler, Dilip Krishnan, Graham W. Taylor, and Robert Fergus. Deconvolutional networks. In *CVPR*, pages 2528–2535, 2010.

[135] Biao Zhang, Deyi Xiong, and Jinsong Su. Variational neural machine translation. *CoRR*, abs/1605.07869, 2016.

[136] Xingxing Zhang and Mirella Lapata. Chinese poetry generation with recurrent neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 670–680. Association for Computational Linguistics, 2014. URL http://aclweb.org/anthology/D14-1074.

[137] Xingxing Zhang, Liang Lu, and Mirella Lapata. Tree recurrent neural networks with application to language modeling. *CoRR*, abs/1511.00060, 2015. URL http://arxiv.org/abs/1511.00060.

[138] Junbo Jake Zhao, Yoon Kim, Kelly Zhang, Alexander M. Rush, and Yann LeCun. Adversarially regularized autoencoders for generating discrete structures. *CoRR*, abs/1706.04223, 2017. URL http://arxiv.org/abs/1706.04223.

[139] Tiancheng Zhao, Ran Zhao, and Maxine Eskenazi. Learning discourse-level diversity for neural dialog models using conditional variational autoencoders. *CoRR*, abs/1703.10960, 2017. URL https://arxiv.org/abs/1703.10960.

[140] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. *CoRR*, abs/1611.01578, 2016.

Acronyms

RNN Recurrent Neural Network

CNN Convolutional Neural Network

RAM Recurrent Attention Model

VAE Variational Autoencoder

NLP Natural Language Processing

GAN Generative Adversarial Network

LM Language Model

NN Neural Network

BN Batch Normalization

LSTM Long Short-Term Memory

GRU Gated Recurrent Unit

BPTT Backpropagation-Through-Time

NLL Negative Log Likelihood

KL Kullback-Leibler

ELBO Evidence Lower Bound

MCMC Markov Chain Monte Carlo

List of Figures

List of Tables