Latent Variable Models in Neural Machine Translation

Author: *John Isak Texas Falk*
Supervisor: *Pr. David Barber*

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Master of Science

of

Computational Statistics and Machine Learning.

The Centre for Computational Statistics and Machine Learning

University College London

This report is submitted as part requirement for the MSc Degree in CSML at University College London. It is substantially the result of my own work except where explicitly indicated in the text. The report may be freely copied and distributed provided the source is explicitly acknowledged.

September 4, 2017

Abstract

Neural machine translation is the application of neural networks to perform translation between two languages, natural language generation concerns itself with the generation of sentences using a probabilistic model. Employing latent variable models for language generation enables generation of sentences from an underlying latent cause. Latent variable models using neural networks are very powerful but hard to optimise since the log-likelihood is intractable. Variational Autoencoder is a framework for optimising a stochastic approximation of the ELBO, a lower bound on the log-likelihood, letting us train these models. While variational autoencoders have been applied with some success to natural language processing, it is still unclear if this could be used as a way of translating sentence between two languages. We apply the variational autoencoder framework to a latent variable model mapping the latent variable to a sentence in two distinct languages, showing that after completing training, we can perform translation through the use of the recognition model. We evaluate how the quality of the generated and translated sentences differ with the power of the generative model by qualitative inspection, BLEU, upper bounded perplexity and an estimate of the ELBO on a small and big dataset. The results show that while the model manages to perform well on shorter sentences it has problems generating coherent sentences over longer sequences and that translation is possible but unlikely to be competitive with the currently best models.

Acknowledgements

I would like to thank the help from my supervisor Harshil Shah for helping me make this thesis possible, and my parents, for always being there for me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Natural language processing (NLP) is an old field with roots in many different disciplines including but not exclusive to electrical engineering, artificial intelligence and linguistics [2, p. 10-15]. Machine learning often view NLP as a statistical problem. A language model is a statistical model that trained on a training corpus assign probabilities to new sentences [3]. Latent variable models enables these language models to generate novel sentences from an underlying stochastic variable. While using probabilistic models enables us to do inference in a principled manner, it is non-trivial to optimise the log-likelihood specified by a latent variable model. Variational Inference bounds the log-likelihood from below using the Evidence Lower Bound (ELBO) by introducing an approximate distribution, $q(\boldsymbol{z})$, of the conditional probability of the latent variable, $p(\boldsymbol{z}|\boldsymbol{x})$ [4].

Although Variational Inference specifies ways of approximating the log-likelihood by a variational distribution q it does not specify the form of this q. Variational Autoencoder (VAE) [5], also under the name of stochastic backpropagation [6], is a framework for training deep generative models with latent variables by introducing a recognition model $q_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x})$. Except for the most basic of models the ELBO function is not tractable. However, it is possibly to approximate the ELBO by sampling the latent variable instead of integrating it out. Using the reparameterization trick to sample this latent variable gives us the differentiable unbiased Stochastic Gradient Variational Bayes (SGVB) estimator of the ELBO, leading to the Auto-Encoding Variational Bayes (AEVB) algorithm. AEVB optimises the introduced recognition model $q_{\varphi}(z|\boldsymbol{x})$ jointly with the generative model $p_{\theta}(z, \boldsymbol{x})$ with respect to the SGVB objective [5]. As the recognition model maps the input sentences to the latent space, it effectively tries to compress information about the sentences through z, resulting in distributed latent representations of sentences [7].

In this thesis we introduce a deep generative model which generates sentences in two different languages, $\mathcal{X}$ (EN) and $\mathcal{Y}$ (FR), from one latent variable, z. The generative model is based on the WaveNet neural network [1] and is traicned using the AEVB algorithm. We show how different recognition models (MLP, RNN, WaveNet) affect the final performance and training time in the mono-language generative model and based on this performance we use the MLP recognition model for the twin-language generative model.

Learning the recognition model $q_{\varphi}(z|\boldsymbol{x}, \boldsymbol{y})$ effectively lets us do translation. By omitting the dependency of one of the languages, here $\mathcal{Y}$, we force $q_{\varphi}(z|\boldsymbol{x}, \boldsymbol{y}) \propto q_{\varphi}(z|\boldsymbol{x})$ giving us a way to perform mappings of $\mathcal{X} \rightarrow \mathcal{Y}$. We use a variety of measures to show that our model may perform translation and reconstruction, but that it does not perform as well as the current best models.

The thesis follows the following structure. In Chapter 2 we review the necessary background knowledge to understand the theory and experiments conducted. Chapter 3 lays out how variational inference and how the AEVB algorithm works, and also specifies the models we will be using. Chapter 4 presents the experimental setup and a discussion of results and chapter 5 the conclusion and future work. The appendix shows an excerpt of generated sentences for all the different models.

Chapter 2

Background Knowledge

This chapter will introduce the necessary background knowledge to follow the theoretical derivations later.

2.1 Probability Theory and Statistics

We go over the rules of probability and statistics.

2.1.1 Rules and Theorems

We here lay out definitions and theorems that we will make use of in the thesis. For all parts below, assume that $x \in \mathcal{X}$ and $y \in \mathcal{Y}$ are random variables and $p(x)$ respectively $p(y)$ the probability distributions of these variables. We interpret the integral over x to be a sum over $\mathcal{X}$ if x is discrete and a regular Lebesgue integral over $\mathcal{X}$ if x is continuous. Then:

Definition 2.1.1 (Unit Volume)**.**

$$\int_{\mathcal{X}} p(x) \, \mathrm{d}x = 1$$

Definition 2.1.2 (Non-negativity)**.**

$$p(x) \geq 0$$

Most manipulations of random variables may be reduced to the following three rules of probability:

Theorem 2.1.3 (Sum Rule)**.**

$$p(x) = \int_{\mathcal{Y}} p(x, y)\, \mathrm{d}y$$

Theorem 2.1.4 (Product Rule)**.**

$$p(x, y) = p(y|x)p(x)$$

Theorem 2.1.5 (Bayes Rule)**.**

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)}$$

Two very important operations involving probabilities of random variables are those of *Expectation* and *Covariance.*

Definition 2.1.6 (Expectation)**.** *Let $f : \mathcal{X} \to \mathbb{R}$, then*

$$\mathbb{E}_x[f] = \int_{\mathcal{X}} f(x)p(x)\, \mathrm{d}x$$

Definition 2.1.7 (Covariance)**.**

$$\mathrm{Cov}(x, y) = \mathbb{E}_{xy}[(x - \mathbb{E}_x[x])(y - \mathbb{E}_y[y])]$$

The variance operator is straightforwardly defined:

Definition 2.1.8 (Variance)**.**

$$\mathrm{Var}(x) = \mathrm{Cov}(x, x)$$

The generalisation of expectation from $f : \mathcal{X} \to \mathbb{R}$ to $f : \mathcal{X} \to \mathbb{R}^n$ is defined in a direct manner. If $\boldsymbol{f} = f(x)$ then:

$$\mathbb{E}_x \begin{bmatrix} f_1 \\ \vdots \\ f_n \end{bmatrix} = \begin{bmatrix} \mathbb{E}_x f_1 \\ \vdots \\ \mathbb{E}_x f_n \end{bmatrix}$$

similarly $\mathrm{Cov}(\boldsymbol{f})$ is a $D \times D$-dimensional matrix where $\mathrm{Cov}(\boldsymbol{f})_{i,j} = \mathrm{Cov}(f_i, f_j)$[8, 9].

2.1.2 The Gaussian Distribution

For a D-dimensional random vector $\boldsymbol{x}$, the multivariate Gaussian distribution takes the form

$$\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}, \Sigma) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu})^T \Sigma^{-1} (\boldsymbol{x} - \boldsymbol{\mu})\right), \tag{2.1}$$

where $\boldsymbol{\mu}$ is a D-dimensional mean vector, Σ is a $D \times D$ dimensional positive definite covariance matrix and $|\Sigma|$ denotes the determinant of Σ. It is straightforward to show that these parameters correspond to the mean and covariance as defined in 2.1.6 and 2.1.7[9].

The Gaussian distribution can be seen as a unit D-dimensional cube which is translated, sheared and rotated, giving rise to the fact that we can write any Gaussianly distributed random variable $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}, \Sigma)$ as a linear combination of a unit Gaussian random variable $\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{x}|\boldsymbol{0}, \boldsymbol{I})$. If we let $\Lambda\Lambda^\top = \Sigma$ be the Cholesky decomposition[10, p. 100-102] of Σ, then we also have that

$$\boldsymbol{x} = \boldsymbol{\mu} + \Lambda \boldsymbol{z}, \tag{2.2}$$

where the equality is in terms of distribution. If we further assume that $\boldsymbol{x}$ is parametrised by $\boldsymbol{\mu}$ and Σ such that Σ is diagonal positive definite with diagonal

$\boldsymbol{\sigma}$ then this means that if we want to sample a random variable $\boldsymbol{x}$ with diagonal covariance structure, we can do this by sampling a unit normal $\boldsymbol{z}$ which we then transform, which may be expressed as

$$\boldsymbol{x} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{z} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}), \tag{2.3}$$

where we define a vector $\boldsymbol{\sigma}$ as the covariance matrix Σ to mean that Σ is diagonal positive definite with diagonal $\boldsymbol{\sigma}$.

As the Gaussian distribution is part of the exponential family [9], the density of the joint distribution of iid Gaussian variables are themselves Gaussian distributed where the natural parameters of this joint distribution is the sum of the natural parameters of each random variable in the joint. In particular for the Gaussian distribution, this means that if we have a collection of iid gaussian random variables $\{\boldsymbol{x}_i\}_i^n$, such that $\boldsymbol{x}_i \sim \mathcal{N}(\boldsymbol{x}_i|\boldsymbol{\mu}_i, \Sigma_i)$, then the joint can be found to be Gaussian distributed as

$$\mathcal{N}(\boldsymbol{\mu}, \Sigma)$$

, where

$$\Sigma = \left(\sum_i^n \Sigma_i^{-1}\right)^{-1} \tag{2.4}$$

$$\boldsymbol{\mu} = \Sigma\left(\sum_i^n \Sigma^{-1}\boldsymbol{\mu}_i\right) \tag{2.5}$$

[8, p. 78-84].

In the case of two independent random variables distributed according to the form as laid out in (2.3), $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{x}}, \boldsymbol{\sigma}_{\boldsymbol{x}})$ and $\boldsymbol{y} \sim \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{y}}, \boldsymbol{\sigma}_{\boldsymbol{y}})$, we have that the resulting distribution $p(\boldsymbol{x}, \boldsymbol{y}) = p(\boldsymbol{x})p(\boldsymbol{y})$ is distributed such that

$$p(\boldsymbol{x}, \boldsymbol{y}) = \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{x},\boldsymbol{y}}, \boldsymbol{\sigma}_{\boldsymbol{x},\boldsymbol{y}})$$

where

$$\boldsymbol{\sigma}_{\boldsymbol{x},\boldsymbol{y}} = \frac{1}{\boldsymbol{\sigma}_{\boldsymbol{x}}^{-1} + \boldsymbol{\sigma}_{\boldsymbol{y}}^{-1}} \tag{2.6}$$

$$\boldsymbol{\mu}_{\boldsymbol{x},\boldsymbol{y}} = \frac{\boldsymbol{\sigma}_{\boldsymbol{x}}^{-1}\boldsymbol{\mu}_{\boldsymbol{x}} + \boldsymbol{\sigma}_{\boldsymbol{y}}^{-1}\boldsymbol{\mu}_{\boldsymbol{y}}}{\boldsymbol{\sigma}_{\boldsymbol{x}}^{-1} + \boldsymbol{\sigma}_{\boldsymbol{y}}^{-1}} \tag{2.7}$$

with the inverse and division operators being done elementwise.

2.1.3 Maximum Likelihood Estimation

Assume we have a model $\mathcal{M}$ parametrised by $\boldsymbol{\theta}$ constrained to live in the parameter space $\boldsymbol{\Theta}$. Given data $\mathcal{D} = \{\boldsymbol{x}_i\}_{i=1}^{n}$ we want to be able to fit the parameters $\boldsymbol{\theta}$ such that the model generalise to unseen data.

We define the likelihood function using the common assumption of i.i.d datapoints

$$\mathcal{L}(\boldsymbol{\theta}|\mathcal{D}) = p(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n|\boldsymbol{\theta}) = \prod_i^n p(\boldsymbol{x}_i|\boldsymbol{\theta}). \tag{2.8}$$

The MLE of of the parameters of the model is then defined to be

$$\hat{\boldsymbol{\theta}}_{ML} = \operatorname*{argmax}_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} \mathcal{L}(\boldsymbol{\theta}|\mathcal{D}). \tag{2.9}$$

While the original MLE is defined in terms of the likelihood function $\mathcal{L}(\boldsymbol{\theta}|\mathcal{D})$, it's often more practical to work with the logarithm of this function, the log-likelihood function $\ell(\boldsymbol{\theta}|\mathcal{D})$. Using the log-likelihood we transform this product into a form involving sums

$$\ell(\boldsymbol{\theta}|\mathcal{D}) = \log \mathcal{L}(\boldsymbol{\theta}|\mathcal{D}) = \sum_i^n \log p(\boldsymbol{x}_i). \tag{2.10}$$

[11]

Besides from simplifying notation and calculation, the log-likelihood has the added benefit of reducing the risk of arithmetic underflow due to the small magnitude of individual probabilities. In models with long dependency such as language models, working in the log-space is needed to enable computation efficiently and robustly [].

2.1.4 Graphical models

Graphical models are powerful tools for specifying the dependency relations between random variables in models. Each graph specifies the class of possible distributions for the model. In particular they provide a clear way of reasoning about model assumptions and provide a framework for doing inference and learning [9].

Directed graphical models specify how the joint distribution of a set of random variables $\mathcal{X}$ factors in a conditional manner. In general, the relationship between a given directed graph and the corresponding distributions over the variables in $\mathcal{X}$ is such that the join distribution defined by the graph is given by the product over all vertices of the graph. The conditional distribution for each vertex is conditioned on the variables corresponding to the parents of that vertex in the graph. So for a graph with K vertices, the joint distribution is give by

$$p(\mathcal{X}) = \prod_{k=1}^{K} p(x_k|\text{pa}(x_k)) \tag{2.11}$$

where $\text{pa}(x_k)$ is defined as the set of random variables corresponding to the parent vertices of the random variable x_k.

Although a graphical model is completely defined in terms of it's vertex and edge set, it is really the most powerful when visualized as a diagram. I will show this with an example using all of the graphical model parts found in the later section when we specify our models.

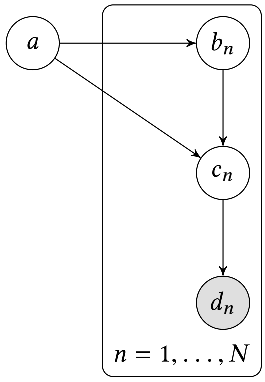


Figure 2.1: Example of a graphical model

There are two different vertices in this graphical model, the greyed out vertex indicates that the random variable is *observed* such that it's value is fixed and known. The white vertices indicates *latent* random variables which are unobserved.

The edge around some of the vertices specifying that $n = 1, \ldots, N$ is an instance of plate notation. Plate notation signifies that the plate is duplicated N times and is used for cases where the same subgraph reoccurs many times as is the case with for example iid data.

For this example we have the following factorisation of the joint distribution, following the rules laid out in equation (2.11),

$$p(a, \{(b_n, c_n, d_n)_{n=1}^N\}) = p(a) \prod_{n=1}^{N} p(d_n|c_n)p(c_n|b_n, a)p(b_n|a).$$

2.2 Deep Learning

The field of NLP were originally dominated by older machine learning techniques utilising linear models trained over very high-dimensional and sparse feature vectors. Recently the field has switched over to neural networks over dense inputs using word embeddings [12, p. 1 - 2].

Neural networks may be seen from many angles, but from a mathematical point of view a neural network parametrised by parameters (weights) $\boldsymbol{\theta}$ specifies a non-linear functional relationship between the input to the network $\boldsymbol{x}$ and the output $\boldsymbol{y}$. As such each neural network with unspecified parameters $\boldsymbol{\theta}$ in a parameter space Θ specifies a set of functions [8]. It has been shown that this set of functions of several architectures are universal approximators and thus are able to approximate a big set of nice functions [13, 14], justifying their use theoretically.

2.2.1 Multilayer Perceptron

Multilayer Perceptrons (MLP's) are neural networks represented by functional composition, where each function is interpreted as a layer of the network. The original MLP can be defined in terms of a recurrence relation such that if we have input vectors of the form $\boldsymbol{x} \in \mathbb{R}^{d_{in}}$ and output vectors of the form $\boldsymbol{y} \in \mathbb{R}^{d_{out}}$ and $\sigma_i(\cdot)$ represents an arbitrary activation function, then an MLP with L layers have the functional form

$$f(\boldsymbol{x}|\boldsymbol{\theta}) = \sigma_L(\boldsymbol{W}_L \boldsymbol{z}_{L-1} + \boldsymbol{b}_L) \tag{2.12}$$

where for any $l \in \{2, \dots, L-1\}$

$$\boldsymbol{z}_l = \sigma_l(\boldsymbol{W}_l \boldsymbol{z}_{l-1} + \boldsymbol{b}_l) \tag{2.13}$$

with base case

$$\boldsymbol{z}_1 = \sigma_1(\boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1). \tag{2.14}$$

[15]

$\boldsymbol{W}_l$ and $\boldsymbol{b}_l$ may be of any dimension as long as it is dimensionally consistent with the input and output of the previous and next layers and conform to the original input and output dimensions. In this case we have that the parameters of the network are all of the biases and weights for the layers, $\boldsymbol{\theta} = \{(\boldsymbol{W}_l, \boldsymbol{b}_l)_{l=1}^L\}$.

Activation functions generally are monotonically increasing function mapping real numbers to some interval. Activation functions which will be useful to us are

Sigmoid

$$\sigma(x) = \frac{1}{1 + \exp(-x)} \tag{2.15}$$

The sigmoid activation function maps any real number to a value in $(0, 1)$.

Hyperbolic tangent

$$\tanh(x) = \frac{2}{1 + \exp(-2x)} - 1 = 2\sigma(2x) - 1 \tag{2.16}$$

SELU Scaled Exponential Linear Units [16] have the form

$$\text{selu}(x) = \lambda \begin{cases} x & \text{if } x > 0 \\ \alpha(\exp(x) - 1) & \text{if } x \leq 0 \end{cases} \tag{2.17}$$

Softmax The Softmax is a mapping from an input space $\mathbb{R}^{d_{in}}$ to an output space $\mathbb{R}^{d_{out}}$, the form given by

$$\text{softmax}(z)_i = \frac{\exp(z_i)}{\sum_{k=1}^{d_{out}} \exp(z_k)} \quad \text{where} \quad i \in \{1, \ldots, d_{out}\} \tag{2.18}$$

Since it is normalized it is a natural candidate for defining categorical distributions [9] and we will use it for this purpose in our models.

2.2.2 Recurrent Neural Networks

RNN's have traditionally been the neural network architecture of choice for NLP due to it being able to handle long-term dependencies well [17].

For in depth treatment of the architecture with respect to nlp see [18, 19].

2.2.3 Convolutional Neural Networks

The CNN we will use will be highly specialised for our case. We use the WaveNet as laid out in the paper by DeepMind [1].

WaveNet works on data of the form of vectors, $\boldsymbol{x}$, were we describe the distribution as

$$p(\boldsymbol{x}) = \prod_{l=1}^{L} p(\boldsymbol{x}_l | \boldsymbol{x}_1, \ldots, \boldsymbol{x}_{l-1}),$$

the form common in NLP and NMT. The output of the model has the same time dimensionality as the input, meaning that the model has the ability to output a categorical distribution over the next value x_l with a softmax layer. In our case this is a parametrisation of the output probability at point l.

WaveNet uses dilated convolutions, a type of convolution where the filter is applied to an area larger than its length by only considering input values such that at each step you skip k inputs, where k is the dilation. Stacking layers and using dilations that increase exponentially with the depth of the layer, it is possible to get a receptive field which grows exponentially with the depth of the network [1].

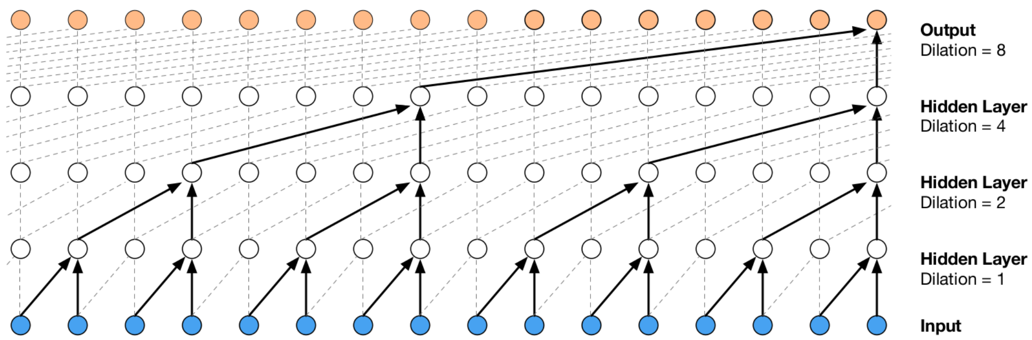


Figure 2.2: Dilated Convolutions in WaveNet [1]

2.3 Natural Language Processing

Humans use natural language every day to convey concepts and abstractions to each other in an efficient manner. Compared to formal languages found in mathematics and programming, the natural languages we use are often ambiguous systems filled with rules and exceptions [20, 21].

Natural Language Processing is an old field that for a long time developed in parallel with the field of machine learning and computational statistics which deals with how to process information coming from human languages and is split up into several subfields such as Machine Translation, statistical parsing and sentiment analysis [21].

We will here lay out the necessities for understanding the neural language model we will build.

2.3.1 Language model

We define a sentence to be a vector of words $\boldsymbol{x}_{1:L} = (\mathrm{x}_1, \ldots, \mathrm{x}_L)^\top$ such that each word is an atomic element $\mathrm{x}_i \in V$, where V is the dictionary of words in our language. Repeated use of the rule laid out in 2.1.4 enables us to rewrite the probability of a sentence in terms of how it is composed of words

$$P(\boldsymbol{x}_{1:L}) = \prod_{l=1}^{L} P(\mathrm{x}_l | \mathrm{x}_1, \ldots, \mathrm{x}_{l-1}) \tag{2.19}$$

where x_l is the l'th word of the sentence $\boldsymbol{x}_{1:L}$ [3].

This form of the probability of a sentence lays bare how the words in a sentence relate to previously occurred words within the same sentence. We will assume that all sentences in the dataset $\mathcal{D} = \{(\boldsymbol{x}^n)_{n=1}^N\}$ are iid which enables us to express the log-likelihood in the form of equation (2.10).

2.3.2 Word embeddings

Breaking down sentences at a word level and processing them into a form that encodes information efficiently is a problem which has gained notorious recognition [22], leading to algorithms such as word2vec [23] and Glove [24]. However, these techniques work less well in a neural network setting where instead the preferred technique is to find the best embeddings jointly with the parameters of the model using backpropagation [12, p. 5-7].

A straightforward way to represent the various words of the dictionary is as one-hot-encoded vectors such that a word $\mathrm{x} \in V$, where the size of V is $|V|$, with an index i given by its place in the dictionary sorted alphabetically in descending order will have the vector representation

$$\boldsymbol{x}_{one-hot} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{2.20}$$

such that $\boldsymbol{x}_{one-hot,j} = \delta_{ij}$ [12, p. 6].

While this is a form conceptually easy to understand, it fails to account for the curse of dimensionality as the size of the vocabulary grows very big in practice and the fact that the cosine similarity of two words $\mathrm{x}_1, \mathrm{x}_2 \in V$ is zero unless they are the same word,

$$\cos_{similarity}(\boldsymbol{x}_{1,one-hot}, \boldsymbol{x}_{2,one-hot}) = \delta_{\mathrm{x}_1\mathrm{x}_2}. \tag{2.21}$$

This means that no meaning is embedded in the vector space except for the location in the sorted dictionary. Instead we would like to associate each word in the vocabulary with a distributed *word feature vector*, a dense, real-valued vector in $\mathbb{R}^m$ where m is the dimension of our embedding space. After training this embedding jointly with the parameters of the network words which share similarities such as `Dog, Puppy` would have a higher similarity score than with an unrelated concepts such as `Dog, Bulwark` [3].

We may represent this in a mathematical form by trying to find a linear map C from any element $\mathrm{x} \in V$ such that $C(\mathbf{x}) \in \mathbb{R}^m$. Using the canonical basis of the Euclidean space, we can express this linear map in terms of a matrix $\boldsymbol{C} \in \mathbb{R}^{m \times |V|}$, thus the word feature vector of the learned embedding can be represented by the matrix multiplication $\boldsymbol{C}\boldsymbol{x}_{one-hot}$.

2.3.3 Neural machine translation

For a long time the dominant paradigm within machine translation was to use phrase based machine translation systems [25, 26], recently however, modelling the word or character level directly with neural networks has overtaken phrase based translation systems [27, 28]. These systems are called Neural Machine Translation.

Most NMT models work in terms of an encoder-decoder architecture where the encoder extracts a fixed length representation $\boldsymbol{c}$, often called a context vector, from a variable length input sentence $\boldsymbol{x} \in \mathcal{X}$, and the decoder uses this representation to generate a correct translation $\boldsymbol{y} \in \mathcal{Y}$ from this representation [29] as can be seen in Figure 2.3.

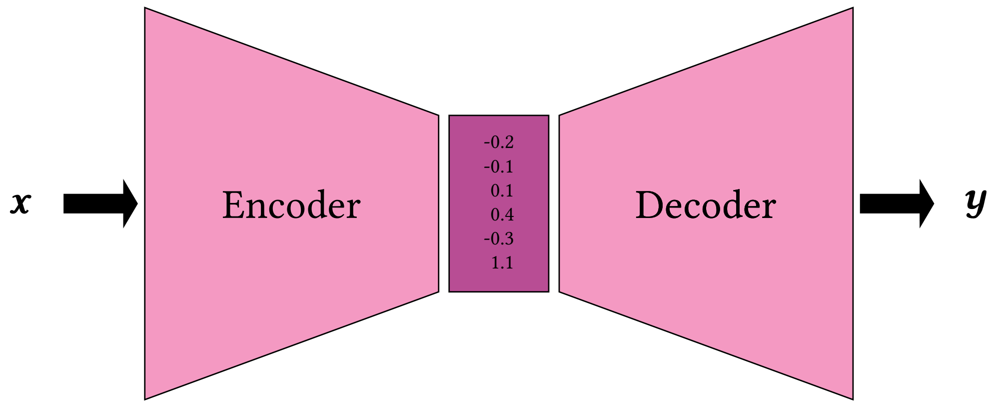


Figure 2.3: Encoder decoder schematic

Our model builds on this paradigm but instead of considering context vectors $\boldsymbol{c}$ we instead consider distributions over a latent vector $\boldsymbol{z}$ which in theory should give the model a greater expressive power since the distribution relays more knowledge than a direct point estimate.

2.4 Optimization

Most parts of machine learning reduces to optimization of a loss function. While optimisation of convex problems are well-understood there has been considerable interest in ways to optimize non-linear loss functions such as those used traditionally in deep learning.

While theoretical results are lacking, there has been substantial advances in various optimisation techniques fit to attack the highly non-linear, non-convex and high-dimensional optimization problems of learning in deep models, particularly from Stochastic Gradient Descent and its friends. This has led to numerous gradient descent-like algorithms used machine learning [30].

2.4.1 Stochastic Gradient Descent

For a normal probabilistic machine learning problem, we have data $\mathcal{D}$ that we try to model with a model $\mathcal{M}$ parametrised by parameters $\boldsymbol{\theta} \in \Theta$. The optimization problem in our case can be recast as an effort to find the parameters $\boldsymbol{\theta}_{ML}$ that maximizes the log-likelihood function (2.10) or equally minimizes the negative log-likelihood.

Gradient descent takes steps in the direction of the gradient, which also is the direction of steepest descent. If we let $\mathcal{D} = \{\boldsymbol{x}_i\}_{i=1}^n$ be our set of sentences and $-\ell$ be the negative log-likelihood that we are trying to minimize then the updates are of the form

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \gamma \frac{1}{n} \sum_{i=1}^{n} \nabla_{\theta}(-\ell(\boldsymbol{\theta}|\boldsymbol{x}_i)), \tag{2.22}$$

where γ is the learning rate of the algorithm.

Stochastic Gradient Descent is a similar algorithm to GD that instead of calculating the gradient with respect to the whole dataset calculates an approximate gradient, as it only takes a subset of the data into account at each update. If we let I_t be a random subset of the indices of $\{(\boldsymbol{x}_n)_{n=1}^N\}$ of size m then SGD does the following update

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \gamma \frac{1}{m} \sum_{i \in I_t} \nabla_{\boldsymbol{\theta}}(-\ell(\boldsymbol{\theta}|\boldsymbol{x}_i)) \tag{2.23}$$

[31][8, p. 240].

2.4.2 ADAM

SGD has found widespread use within the machine learning community due to strong experimental results and ease of use, especially in deep learning. However there are notable alternatives that try to improve on SGD such as RMSProp[32] and AdaGrad[33].

Adam takes inspiration from RMSProp and AdaGrad. Technically, Adam keeps an exponential running average of the first and second order statistics of the gradient, using these to calculate an adaptive learning rate.

As laid out in the original paper by Kingma et al. [34], ADAM operates on a stochastic objective function $f_t(\boldsymbol{\theta})$, where t denotes the time step, similarly $g_t = \nabla_{\boldsymbol{\theta}} f_t(\boldsymbol{\theta})$. m_t and v_t denotes the first and second order moments of the gradients, however, these are biased and are corrected in the algorithm. The whole algorithm is as follows.

Algorithm 2 The ADAM algorithm

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates of the moment estimates
Require: $f(\boldsymbol{\theta})$: Stochastic objective function with parameters $\boldsymbol{\theta}$
Require: $\boldsymbol{\theta}_0$: Initial parameter vector

$m_0 \leftarrow 0$ (Initialize 1$^{\text{st}}$ moment vector)
$v_0 \leftarrow 0$ (Initialize 2$^{\text{nd}}$ moment vector)
$t \leftarrow 0$ (Initialize timestep)
while $\boldsymbol{\theta}_t$ not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_{\boldsymbol{\theta}} f_t(\boldsymbol{\theta}_{t-1})$ (get gradients w.r.t stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\hat{m}_1 \leftarrow m_t/(1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\hat{v}_1 \leftarrow v_t/(1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \alpha \cdot \hat{m}_i/(\sqrt{\hat{v}_t} + \epsilon)$ (Update parameters)
return $\boldsymbol{\theta}_t$ (Resulting parameters)

Experimentally Adam has shown very good results on training various deep learning models such as MLP's, CNN's and RNN's so we will use it to train our models throughout this dissertation [34].

2.4.3 Automatic Differentiation

The AutoDiff framework is a general theory for efficiently calculating gradients with respect to a loss function in general computational graphs. It includes the famed backpropagation [8, 35] algorithm as a special instance.

In machine learning we most often deal with many-to-one mappings with a high-dimensional input $\boldsymbol{x}$ to a scalar output in the form of a loss. In terms of machine learning models, AutoDiff takes a loss function $L(\boldsymbol{\theta})$ and returns an exact value up to machine accuracy for the gradient $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})|_{\boldsymbol{\theta}}$. There are two different modes of AutoDiff, forward and reverse, although in practice reverse is preferred.

Reverse mode AutoDiff is efficient in the way that calculating the gradient of $L(\boldsymbol{\theta})$ is guaranteed to take at most 5 times the time it takes to compute $L(\boldsymbol{\theta})$ [36], enabling neural network libraries to calculate the gradients needed for optimization algorithms such as SGD and ADAM in a swift and automatic manner.

Chapter 3

Methods and Theory

This chapter deals with the theory needed in order to use VAE in our setting. It will also deal with extending the theory and necessary calculations.

3.0.1 Variational Inference

Consider the following general graphical model of a latent variable $\boldsymbol{z}$ and an observed variable $\boldsymbol{x}$ parametrised by $\boldsymbol{\theta}$:

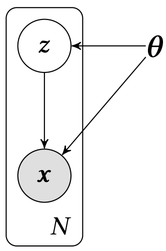


Figure 3.1: Latent variable model

implying the joint distribution

$$p(\boldsymbol{x}, z) = p(\boldsymbol{x}|z)p(z). \tag{3.1}$$

In order to fit the parameters of the model using optimisation of $\ell(\boldsymbol{\theta}|\mathcal{D})$ we need to be able to to find the gradient and value of $\ell(\boldsymbol{\theta}|\boldsymbol{x})$ to use ADAM. We can rewrite the likelihood for a datapoint $\boldsymbol{x}$ as

$$\mathcal{L}(\theta|\boldsymbol{x}) = \int_{\mathcal{Z}} p(\boldsymbol{x}, z)\,\mathrm{d}z \tag{3.2}$$

by using the sum rule of 2.1.3. For many models, this integral is either not available in a closed form or requires exponential time to compute [4]. While for some special cases it is possible to find a local optimum of the likelihood by using the EM algorithm [37], in general it is too restrictive since we need to be able to find $p(z|\boldsymbol{x})$ analytically.

Generally it is possible to use sampling based techniques such as MCMC [38] to sample from $p(z|\boldsymbol{x})$ but in practice it is often slow and depending on the problem may be less than ideal. Especially for problems where we use complex models and have large datasets it's not possible in practice to use MCMC [4].

Variational inference approaches the problem of optimizing the intractable log-likelihood by positing a variational family of distributions, $\mathcal{Q}$ and introducing a variational distribution $q \in \mathcal{Q}$. The log-likelihood is then bounded from below using the concavity of the $\log(\cdot)$ function in order to apply Jensen's inequality and the fact that q is a distribution:

$$\begin{aligned}
\log p_\theta(\boldsymbol{x}) &= \log \int_{\mathcal{Z}} p_\theta(z, \boldsymbol{x})\,\mathrm{d}z \\
&= \log \int_{\mathcal{Z}} q_\varphi(z) \frac{p_\theta(\boldsymbol{x}, z)}{q_\varphi(z)}\,\mathrm{d}z \\
&= \log \mathbb{E}_{q_\varphi(z)} \left[\frac{p_\theta(\boldsymbol{x}, z)}{q_\varphi(z)} \right] \\
&\geq \mathbb{E}_{q_\varphi(z)} \left[\log \frac{p_\theta(\boldsymbol{x}, z)}{q_\varphi(z)} \right] \\
&= \mathbb{E}_{q_\varphi(z)} \left[\log p_\theta(\boldsymbol{x}, z) - \log q_\varphi(z) \right].
\end{aligned}$$

We call the lower bound the Evidence Lower Bound Objective (ELBO),

$$\mathrm{ELBO}(q_{\boldsymbol{\varphi}}(z)) = \mathbb{E}_{q_{\boldsymbol{\varphi}}(z)}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}, z) - \log q_{\boldsymbol{\varphi}}(z)\right] \tag{3.3}$$

and we get the following inequality

$$\log p_{\boldsymbol{\theta}}(\boldsymbol{x}) \geq \mathrm{ELBO}(q_{\boldsymbol{\varphi}}(z)). \tag{3.4}$$

Rewriting the ELBO in terms of the log-likelihood

$$\mathrm{ELBO}(q_{\boldsymbol{\varphi}}(z)) = \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) - D_{KL}(q_{\boldsymbol{\varphi}}(z)||p_{\boldsymbol{\theta}}(z|\boldsymbol{x})) \tag{3.5}$$

we see how the choice of $q_{\boldsymbol{\varphi}}(z)$ affects the tightness of the lower bound through the closeness to the true posterior $p_{\boldsymbol{\theta}}(z|\boldsymbol{x})$ in terms of the KL-divergence [4].

3.1 VAE and SGVB

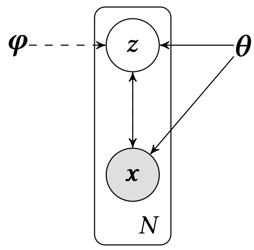


Figure 3.2: Variational Autoencoder

The Variational Autoencoder by Kingma et al. [5] introduces a recognition model $q_{\boldsymbol{\varphi}}(z|\boldsymbol{x})$ that serves as an approximation to the true posterior $p_{\boldsymbol{\theta}}(z|\boldsymbol{x})$. We let $q_{\boldsymbol{\varphi}}(z|\boldsymbol{x}) \sim \mathcal{N}(z|\boldsymbol{\mu}_{\boldsymbol{\varphi}}(\boldsymbol{x}), \boldsymbol{\sigma}_{\boldsymbol{\varphi}}(\boldsymbol{x}))$ such that the pair of vectors $(\boldsymbol{\mu}_{\boldsymbol{\varphi}}(\boldsymbol{x}), \boldsymbol{\sigma}_{\boldsymbol{\varphi}}(\boldsymbol{x}))$ is the output of a neural network, and let this Gaussian have a diagonal covariance structure with the vector $\boldsymbol{\sigma}_{\boldsymbol{\varphi}}(\boldsymbol{x})$ as the diagonal.

In the following we assume that $\boldsymbol{x}$ is an observed datapoint from our dataset and z is the corresponding latent variable. If we consider the ELBO again we can rewrite it in terms that only involve known functions and quantities

$$\mathrm{ELBO}(q_{\boldsymbol{\varphi}}(z|\boldsymbol{x})) = \mathbb{E}_{q_{\boldsymbol{\varphi}}(z|\boldsymbol{x})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|z)\right] - D_{KL}(q_{\boldsymbol{\varphi}}(z|\boldsymbol{x})||p_{\boldsymbol{\theta}}(z)). \tag{3.6}$$

Since this depends on both the parameters of the generative model and the recognition model, $(\boldsymbol{\theta}, \boldsymbol{\varphi})$ we make this explicit and write

$$\mathcal{L}^{\mathrm{ELBO}}(\boldsymbol{\theta}, \boldsymbol{\varphi}|\boldsymbol{x}) = \mathbb{E}_{q_{\varphi}(z|\boldsymbol{x})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|z)\right] - D_{KL}(q_{\boldsymbol{\varphi}}(z|\boldsymbol{x})||p_{\boldsymbol{\theta}}(z)). \tag{3.7}$$

Note that for most models we are not able to get an analytical form of the expectation over $q_{\boldsymbol{\varphi}}(z|\boldsymbol{x})$ due to the non-linear relationship between $\boldsymbol{x}$ and $\boldsymbol{z}$. Instead we sample $\boldsymbol{z}$ to estimate this expectation using normal Monte Carlo estimation.

While it would be possible to differentiate and optimize the lower bound $\mathcal{L}^{\mathrm{ELBO}}(\boldsymbol{\theta}, \boldsymbol{\varphi}|\boldsymbol{x})$ jointly with respect to both the variational and generative parameters $\boldsymbol{\varphi}$ and $\boldsymbol{\theta}$ by using a straightforward Monte Carlo gradient estimator with respect to $q_{\boldsymbol{\varphi}}(z|\boldsymbol{x})$, this gradient exhibits very high variance and is impractical compared to the reparametrisation trick which we will introduce below [5].

3.1.1 Reparametrisation Trick

Under certain mild regularity conditions, for a chosen approximate posterior $q_{\boldsymbol{\varphi}}(z|\boldsymbol{x})$ we can express the distribution of $\boldsymbol{z} \sim q_{\boldsymbol{\varphi}}(z|\boldsymbol{x})$ in terms of a simpler distribution using reparametrisation, called the reparametrisation trick [5]. For our case, let transformation $g_{\boldsymbol{\varphi}}(\boldsymbol{\epsilon}, \boldsymbol{x})$ such that

$$\boldsymbol{z} = g_{\boldsymbol{\varphi}}(\boldsymbol{\epsilon}, \boldsymbol{x}), \quad \text{such that} \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}). \tag{3.8}$$

In our case we use (2.3) in order to write $g_{\boldsymbol{\varphi}}(\boldsymbol{\epsilon}, \boldsymbol{x}) = \boldsymbol{\mu}_{\boldsymbol{\varphi}}(\boldsymbol{x}) + \boldsymbol{\sigma}_{\boldsymbol{\varphi}}(\boldsymbol{x}) \odot \boldsymbol{\epsilon}$.

This means that we can form Monte Carlo estimates of the ELBO by sampling

z through sampling ϵ and using the differentiable transformation

$$\mathbb{E}_{q_{\varphi}(z|x)}\left[\log p_{\theta}(\boldsymbol{x}, z) - \log q_{\varphi}(z|\boldsymbol{x})\right] \simeq \frac{1}{L}\sum_{l=1}^{L} \log p_{\theta}(\boldsymbol{x}, z^{l}) - \log q_{\varphi}(z^{l}|\boldsymbol{x}) \quad (3.9)$$

where $z^l = g_{\varphi}(\epsilon^l, \boldsymbol{x})$ and $\epsilon^l \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$.

Equation (3.9) is the Stochastic Gradient Variational Bayes estimator of the ELBO, which we call $\mathcal{L}^{SGVB}(\boldsymbol{\theta}, \boldsymbol{\varphi}|\boldsymbol{x})$ and is an unbiased estimator of the ELBO since we sample z from the recognition model.

3.1.2 AEVB algorithm

The AEVB algorithm optimises the SGVB equation (3.9) in order to in drive up the ELBO and push the lower bound of the the log-likelihood up. If the recognition model and the prior are chosen to be from appropriate distributions it is possible to get an analytical form of the KL-divergence. Choosing $p_{\theta}(z)$ and $q_{\varphi}(z|\boldsymbol{x})$ to be Gaussianly distributed enables us to write the SGVB in an alternative form which typically has less variance than the SGVB form of equation (3.9):

$$\mathcal{L}^{SGVB}(\boldsymbol{\theta}, \boldsymbol{\varphi}) = -D_{KL}(q_{\varphi}(z|\boldsymbol{x})||p_{\theta}(z)) + \frac{1}{L}\sum_{l=1}^{L} \log p_{\theta}(\boldsymbol{x}|z^{l}) \quad (3.10)$$

where $z^l = g_{\varphi}(\epsilon^l, \boldsymbol{x})$ and $\epsilon^l \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$. We let $L = 1$ following the advice of Kingma et al. [5].

3.2 Models

Here we lay out the graphical models we will use in order to model language generation. All of our models are latent variable models with the prior on z being unit normal, $p(z) = \mathcal{N}(z|\mathbf{0}, \boldsymbol{I})$.

3.2.1 Reconstruction

The generative model is the same as in the figure for latent variable models , except that the distribution of z does not depend on $\boldsymbol{\theta}$:

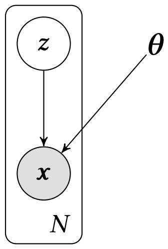


Figure 3.3: Reconstruction language model

where z is the latent variable representing the latent sentence structure and the joint factorizes as $p_{\boldsymbol{\theta}}(\boldsymbol{x}, z) = p_{\boldsymbol{\theta}}(\boldsymbol{x}|z)p(z)$.

The reconstruction model is the same model specified in [5]. Bowman et al. has successfully showed that it can be applied to natural language, learning meaningful representations of language in the latent dimension [7] while improving on previous models such as RNNLM [39].

3.2.2 Translation

Translation adds and extra observed variable $\boldsymbol{y}$, a sentence belonging to $\mathcal{Y}$. The probabilistic model looks like:

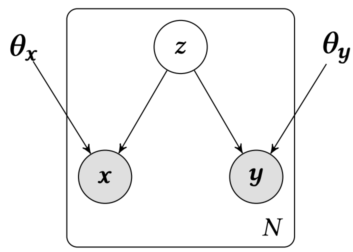


Figure 3.4: Translation language model

The graphical model implies that the joint distribution factorises as

$$p(z, \boldsymbol{x}, \boldsymbol{y}) = p(z)p(\boldsymbol{x}|z)p(\boldsymbol{y}|z) \tag{3.11}$$

and we make the following distributional assumption

$$p(\boldsymbol{x}|z) = \mathcal{N}(\mu_{\theta_x}(z), \sigma_{\theta_x}(z)) \tag{3.12}$$

$$p(\boldsymbol{y}|z) = \mathcal{N}(\mu_{\theta_y}(z), \sigma_{\theta_y}(z)) \tag{3.13}$$

The notation $\mathcal{N}(\mu_\theta(z), \sigma_\theta(z))$ means that the mean and variance diagonal of the normal are represented by a functional relationship $f : \boldsymbol{z} \mapsto (\boldsymbol{\mu}, \boldsymbol{\sigma})$ where f is a function represented a a learnable neural network with parameters $\boldsymbol{\theta}$.

The recognition model parametrises the probability distribution $q_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{y})$. We are free to choose this parametrisation and distributional form however we like, but the closer this is to the actual conditional distribution over the latent, $p(z|\boldsymbol{x}, \boldsymbol{y})$ in the KL sense, the tighter the inequality in (3.4) will be as shown by equation (3.5). We let the distributions be Gaussians to have analytical tractability,

$$\begin{aligned} q_{\varphi}(z|\boldsymbol{x}, \boldsymbol{y}) &= q_{\varphi_x}(z|\boldsymbol{x}) q_{\varphi_y}(z|\boldsymbol{y}) \\ &= \mathcal{N}(\mu_{\varphi_x}(\boldsymbol{x}), \sigma_{\varphi_x}(\boldsymbol{x})) \mathcal{N}(\mu_{\varphi_y}(\boldsymbol{y}), \sigma_{\varphi_y}(\boldsymbol{y})). \end{aligned}$$

This is again a Gaussian distribution

$$q(z|\boldsymbol{x}, \boldsymbol{y}) = \mathcal{N}(z|\mu(\boldsymbol{x}, \boldsymbol{y}), \sigma(\boldsymbol{x}, \boldsymbol{y})) \tag{3.14}$$

which reduces to the expressions (2.5) for the mean and (2.4) for the covariance as shown in the background section on the Gaussian distribution. Pulling all of these statements together, we get that the SGVB of the translation case reduces to the equation

$$\mathcal{L}^{SGVB}(\boldsymbol{\theta}, \boldsymbol{\varphi}|\boldsymbol{x}, \boldsymbol{y}) = (\frac{1}{L}\sum_{l=1}^{L}\log(\mathcal{N}(\boldsymbol{x}|\mu_{\theta_y}(z), \sigma_{\theta_y}(z))) + \log(\mathcal{N}(\boldsymbol{y}|\mu_{\theta_y}(z), \sigma_{\theta_y}(z))))+ \\ \frac{1}{2}\sum_{j=1}^{J}(1 + \log((\sigma_j^{(i)})^2) - (\mu_j^{(i)})^2 - (\sigma_j^{(i)})^2)) \tag{3.15}$$

where the parameters μ, σ are from the normal in equation (3.14), with $\mu_j^{(i)}, \sigma_j^{(i)}$ being the i'th entry in the mean and covariance of the j'th latent sampled. The second expression on the right is the analytical form of $-D_{KL}(q_{\boldsymbol{\varphi}}(z|\boldsymbol{x}, \boldsymbol{y})||p(z))$ [5].

3.2.3 Training

Variational Autoencoders in NLP have a tendency to collapse the KL term in the SGVB objective to zero, effectively being reduced a RNNLM. As the KL can be seen as a measure of how much information is encoded in z, the KL-collapse makes the use of VAE's to model language-agnostic features in the latent space useless. One of the objectives is to make sure this KL-collapse doesn't happen.

We use KL-annealing where we anneal the ELBO by a pseudo-objective

$$ELBO_{\beta_t} = \mathbb{E}_{q_{\boldsymbol{\varphi}}(z|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|z)] - \beta_t * D_{KL}(q_{\boldsymbol{\varphi}}(z|\boldsymbol{x})||p_{\boldsymbol{\theta}}(z)) \tag{3.16}$$

where β_t is an annealing term going from 0 to 1 as we train. We do this by choosing a warm-up β and let $\beta_t = \min(1, \frac{t}{\beta})$ [7], letting this warm up be done linearly until we recover the original ELBO objective, similar to normal simulated annealing [40].

The training is then carried out using ADAM on the objective $\mathcal{L}^{SGVB}$ where we take into account the KL-annealing. In order to calculate the gradients efficiently we use Theano which has automatic support for AutoDiff 2.4.3.

Chapter 4

Experiments

We compare how the different models fare for reconstruction and translation.

4.1 Data

4.1.1 Dataset

We evaluate the models on two different datasets.

Europarl A parallel corpus consisting of the proceedings of the European Parliament, comprising in total the 11 official languages of the European Union [41]. We use the French-English language pair and preprocess it. The English dataset used for reconstruction has 625213 train sentences and 69469 test sentences. For the French-English dataset used for translation has 534635 train sentences and 59404 test sentences. The reason why the reconstruction dataset is bigger than the translation dataset is because for translation we take the intersection of the sentences that is between 2 and 20 words long for French and English, while in reconstruction we only need to consider the English sentences on their own.

Custom Our custom dataset is made from merging three different datasets; Europarl [41], News Commentary and UN corpus [42] [1]. This dataset is consid-

[1]Taken from the wmt14 translation challenge: `http://www.statmt.org/wmt14/translation-task.html`

erably bigger than the Europarl dataset. We only use it on the twin-language model. After preprocessing the dataset it has 4268791 train sentences and 474311 test sentences.

4.1.2 Preprocessing

The raw data is unfit for use directly with the model. For one thing the raw data is in the form of strings and in order to use neural networks we need to transform this into a form using vectors in some space $\mathbb{R}^m$.

We do the following preprocessing steps:

- We specify the maximum and minimum length of the words. This enables us to pad sentences to be the same length.

- We calculate the word frequencies in order to sort all of the words in the dataset in terms of how often it appear in absolute terms. This is then used to only retain the 30000 most common words. Words which are not part of this list gets replaced by an `<UNK>` token, specifying that it's an unknown word outside of the dictionary.

- Newline characters were removed and replaced by `<EOS>`, end-of-sentence tokens, signifying the end of a sentence.

- When preprocessing, we let the preprocessed data from English and French be the intersection of the sentences in English and French that pass the above criteria.

- Unicode is needed for French due to diacretics and accents which are not included in the normal ASCII encoder-decoder scheme for characters. We make sure that encoding and decoding is done correctly.

It is important to note that due to how languages differ, even though the dataset might consist of sentence pairs this will still not mean that in general both of the languages will have the same dictionary of words. Partially this is due to the different ways that languages are built up when expressing meaning, but on a more basic level, there are no bijection between languages as words have slightly different meaning and contexts, with some words only existing in one language but not the other. Similarly the sentence length of the same sentence in English and French will in general be different.

4.2 software

All of the models were trained using Theano [43] and Lasagne [44] built on top of the Python (2.7) programming language. All of the plots were generated using Matplotlib [45] and Seaborn [46]. Data processing was done with Numpy [47] and Pandas [48]. The software repository can be found at `http://www.github.com/isakfalk/project`.

4.3 Design

Each model needs to specify a recognition model $q_\varphi(\boldsymbol{z}|\boldsymbol{x})$ and a generative model $p_\theta(\boldsymbol{x}|\boldsymbol{z})$. Below we specify all of the different models we trained and the hyperparameters we chose.

4.3.1 Reconstruction

We let the latent dimension be 50 and the embedding dimension 300. For Reconstruction we only use the Europarl dataset. These dimensions were tuned by evaluating performance over short runs.

4.3.1.1 Generative model

As specified before we let the prior be a unit normal:

$$p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z}|\boldsymbol{0}, \boldsymbol{I}). \tag{4.1}$$

The generative model of the observed variable $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ is chosen to be the output from the WaveNet, as specified in 2.2.3. The distribution of each word is parametrised as a categorical distribution dependent on all previously outputted probability vectors for the preceding words and the latent vector $\boldsymbol{z}$.

4.3.1.2 Recognition model

We let the recognition model for all reconstruction models be a normal distribution

$$q_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x}) \sim \mathcal{N}(\boldsymbol{z}|\boldsymbol{\mu}_{\boldsymbol{\varphi}}(\boldsymbol{x}), \boldsymbol{\sigma}_{\boldsymbol{\varphi}}(\boldsymbol{x})), \tag{4.2}$$

with the parameters of the Gaussian being the output from different neural networks depending on the model such that $\boldsymbol{\varphi}$ are the parameters of the neural network. We train three different models.

Chapter 5

Conclusions

Possible extensions:

The Authors of WaveNet has introduced ByteNet, might be worthwhile looking into that as well as it is explicitly created to work well on character level generative models (https://arxiv.org/pdf/1610.10099.pdf).

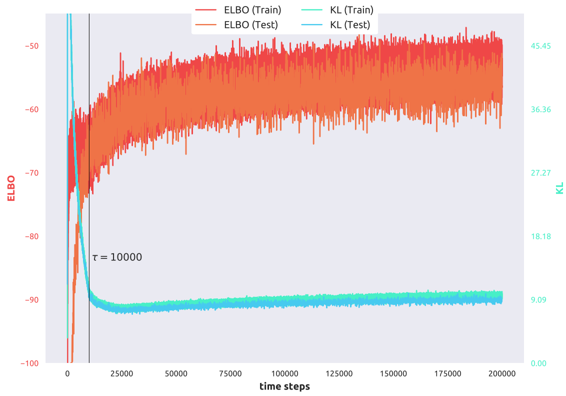
Kingma et. al has come with a new extension to VAE's which uses normalising flows https://arxiv.org/pdf/1606.04934.pdf.

Appendix A

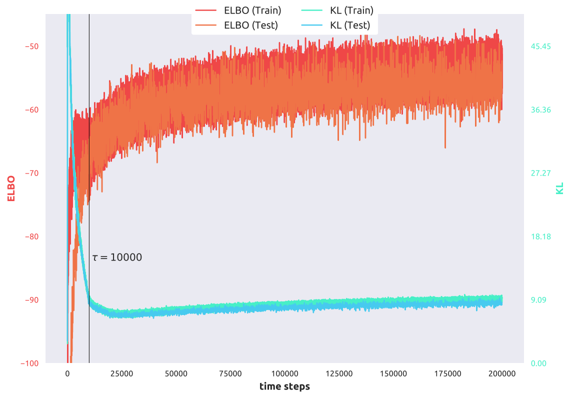
Plots and generated output

A.1 Mono-Reconstruction

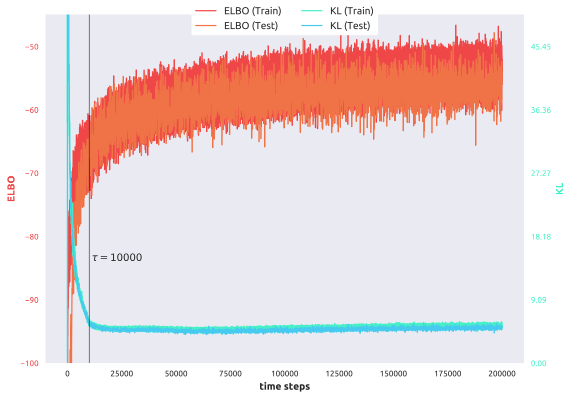
A.1.1 Plots



(a) Plot for $\mathcal{M}_{M1}$



(b) Plot for $\mathcal{M}_{M2}$



(c) Plot for $\mathcal{M}_{M3}$

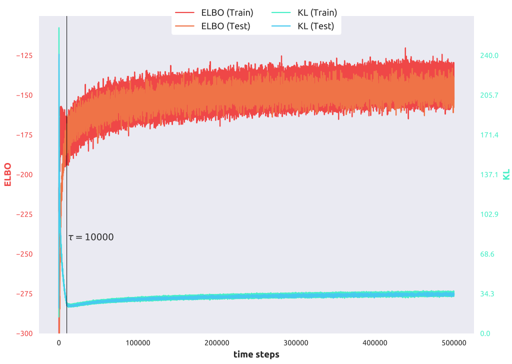
Figure A.1: Plots of the SGVB estimate of ELBO (Labelled ELBO in plots) together with the KL divergence $D_{KL}(q_{\varphi}(z|\boldsymbol{x})||p_{\theta}(z))$ for the different models for mono-reconstruction. The vertical line specifies the end of the KL-annealing.

A.1.2 Generated Sentences

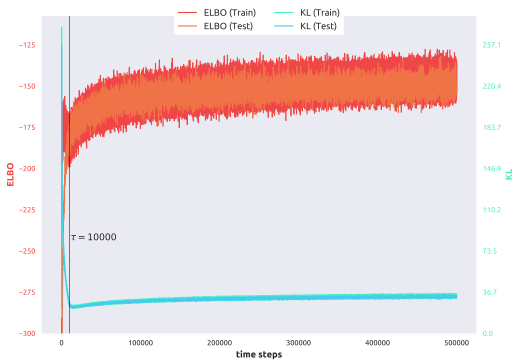
Sampled sentences from prior (English)
Sentences 1
Sentences 2
Sentences 3
Sentences 4
Sentences 5

A.2 Twin-Reconstruction

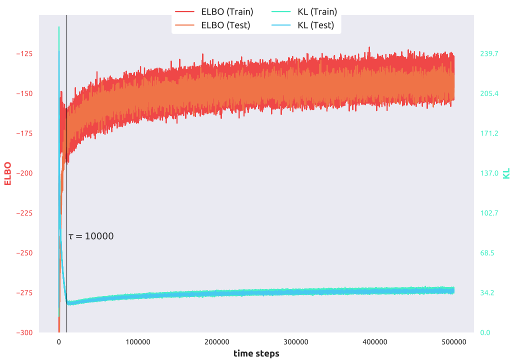
A.2.1 Plots



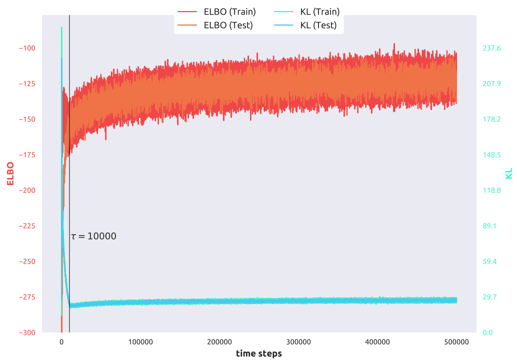
(a) Plot for $\mathcal{M}_{T1}$



(b) Plot for $\mathcal{M}_{T2}$



(c) Plot for $\mathcal{M}_{T3}$



(d) Plot for $\mathcal{M}_{T4}$

Figure A.2: Plots of the SGVB estimate of ELBO (Labelled ELBO in plots) together with the KL divergence $D_{KL}(q_{\varphi}(z|\boldsymbol{x})||p_{\theta}(z))$ for the different models for twin-reconstruction. The vertical line specifies the end of the KL-annealing.

A.2.2 Generated Sentences

A.2.2.1 Model $\mathcal{M}_{M1}$

English

Sampled sentences from prior
in my view the european union has been said .
now we are all UNK !
as far ’ s and UNK to being UNK to UNK UNK , to the UNK UNK .

We let (T) stand for the true sentence, and (G) for the generated sentence.

Sampled sentences from posterior
(T) this has been a most useful debate . (G) this has been a debate was not .
(T) mr president , commissioner , ladies and gentlemen , what is this debate actually about ? (G) mr president , commissioner , ladies and gentlemen , what is going to do about ?
(T) mrs ¡UNK¿ was the rapporteur at the time . (G) mrs ¡UNK¿ has to be of no to ¡UNK¿ .

French

Sampled sentences from prior
madame la présidente , ce problème n’est pas tout une intervention des femmes .
pouviez , notre groupe aussi pour un communication pour cette approche .
lançons , ¡UNK¿ a la parole de la commission .

Model $\mathcal{M}_{M2}$

English

French

Model $\mathcal{M}_{M3}$

English

French

Model $\mathcal{M}_{M4}$

English

French

Bibliography

[1] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W. Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *CoRR*, abs/1609.03499, 2016.

[2] Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2000.

[3] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155, March 2003.

[4] David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe. Variational Inference: A Review for Statisticians. *Journal of the American Statistical Association*, 112(518):859–877, April 2017. arXiv: 1601.00670.

[5] Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes. *arXiv:1312.6114 [cs, stat]*, December 2013. arXiv: 1312.6114.

[6] D. Jimenez Rezende, S. Mohamed, and D. Wierstra. Stochastic Backpropagation and Approximate Inference in Deep Generative Models. *ArXiv e-prints*, January 2014.

[7] Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew M. Dai, Rafal Jozefowicz, and Samy Bengio. Generating Sentences from a Continuous Space. *arXiv:1511.06349 [cs]*, November 2015. arXiv: 1511.06349.

[8] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[9] David Barber. *Bayesian Reasoning and Machine Learning*. Cambridge University Press, New York, NY, USA, 2012.

[10] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 3 edition, 2007.

[11] George Casella and Roger Berger. *Statistical Inference*. Duxbury Resource Center, June 2001.

[12] Yoav Goldberg. A primer on neural network models for natural language processing, 2015. cite arxiv:1510.00726.

[13] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Netw.*, 2(5):359–366, July 1989.

[14] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314, December 1989.

[15] David Barber. Lecture notes in applied machine learning, September 2016.

[16] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. *CoRR*, abs/1706.02515, 2017.

[17] Andrej Karpathy. The unreasonable effectiveness of recurrent neural networks, May 2015.

[18] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[19] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. Lstm neural networks for language modeling. In *Thirteenth Annual Conference of the International Speech Communication Association*, 2012.

[20] Ronald Rosenfeld. Two decades of statistical language modeling: Where do we go from here. In *Proceedings of the IEEE*, page 2000, 2000.

[21] Lenhart Schubert. Computational linguistics. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, spring 2015 edition, 2015.

[22] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[23] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems*, NIPS'13, pages 3111–3119, USA, 2013. Curran Associates Inc.

[24] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *In EMNLP*, 2014.

[25] Philipp Koehn, Franz Josef Och, and Daniel Marcu. Statistical phrase-based translation. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, NAACL '03, pages 48–54, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics.

[26] Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondřej Bojar, Alexandra Constantin, and Evan Herbst. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions*, ACL '07, pages 177–180, Stroudsburg, PA, USA, 2007. Association for Computational Linguistics.

[27] Krzysztof Wo lk and Krzysztof Marasek. Neural-based machine translation for medical text domain. Based on European Medicines Agency leaflet texts. *Procedia Computer Science*, 64:2–9, 2015. arXiv: 1509.08644.

[28] 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. *arXiv:1609.08144 [cs]*, September 2016. arXiv: 1609.08144.

[29] Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. *arXiv:1409.1259 [cs, stat]*, September 2014. arXiv: 1409.1259.

[30] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2016.

[31] Léon Bottou. Stochastic gradient descent tricks. In Grégoire Montavon, Genevieve B. Orr, and Klaus-Robert Müller, editors, *Neural Networks: Tricks*

of the Trade (2nd ed.), volume 7700 of *Lecture Notes in Computer Science*, pages 421–436. Springer, 2012.

[32] T. Tieleman and G. Hinton. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

[33] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. Technical Report UCB/EECS-2010-24, EECS Department, University of California, Berkeley, Mar 2010.

[34] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*, December 2014. arXiv: 1412.6980.

[35] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. chapter Learning Internal Representations by Error Propagation, pages 318–362. MIT Press, Cambridge, MA, USA, 1986.

[36] David Barber. Deep learning: Autodiff, parameter tying and backprop through time, 2015.

[37] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B*, 39(1):1–38, 1977.

[38] Steve Brooks, Andrew Gelman, Galin Jones, and Xiao-Li Meng. *Handbook of Markov Chain Monte Carlo*. CRC press, 2011.

[39] Tomas Mikolov, Stefan Kombrink, Lukás Burget, Jan Cernocký, and Sanjeev Khudanpur. Extensions of recurrent neural network language model. In *ICASSP*, pages 5528–5531. IEEE, 2011.

[40] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[41] Philipp Koehn. Europarl: A Parallel Corpus for Statistical Machine Translation. In *Conference Proceedings: the tenth Machine Translation Summit*, pages 79–86, Phuket, Thailand, 2005. AAMT, AAMT.

[42] Micha l Ziemski, Marcin Junczys-Dowmunt, and Bruno Pouliquen. The united nations parallel corpus v1.0. In Nicoletta Calzolari (Conference Chair), Khalid Choukri, Thierry Declerck, Sara Goggi, Marko Grobelnik, Bente Maegaard, Joseph Mariani, Helene Mazo, Asuncion Moreno, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*, Paris, France, may 2016. European Language Resources Association (ELRA).

[43] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

[44] Sander Dieleman, Jan Schlüter, Colin Raffel, Eben Olson, Søren Kaae Sønderby, Daniel Nouri, et al. Lasagne: First release., August 2015.

[45] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007.

[46] Michael Waskom, Olga Botvinnik, Paul Hobson, John B. Cole, Yaroslav Halchenko, Stephan Hoyer, Alistair Miles, Tom Augspurger, Tal Yarkoni, Tobias Megies, Luis Pedro Coelho, Daniel Wehner, cynddl, Erik Ziegler, diego0020, Yury V. Zaytsev, Travis Hoppe, Skipper Seabold, Phillip Cloud, Miikka Koskinen, Kyle Meyer, Adel Qalieh, and Dan Allan. seaborn: v0.5.0 (november 2014), November 2014.

[47] Stefan van der Walt, S. Chris Colbert, and Gael Varoquaux. The numpy array: A structure for efficient numerical computation. *Computing in Science and Engg.*, 13(2):22–30, March 2011.

[48] Wes McKinney. Data structures for statistical computing in python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 51 – 56, 2010.