Latent Variable Models in Neural Machine Translation

John Isak Texas Falk

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Master of Science

of

University College London.

The Centre for Computational Statistics and Machine Learning

University College London

August 4, 2017

I, John Isak Texas Falk, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the work.

Abstract

Neural Machine translation is a direction in automated translation which learns the mapping from the source to the target language directly in an end-to-end fashion using the framework of neural networks. As NMT hinges on advances in neural network methods and architectures, it is able to directly use improvements there in its own domain.

We use recent improvements in latent variables models in order to train a completely probabilistic generative model with a latent variable representing a language-agnostic representation of a sentence mapping through deep neural networks to two different output languages. Following recent advances in training deep generative latent variable models we approximate the posterior of the latents given output with a recognition model, mimicking a VAE. Following the SGVB method to find a stochastic lower bound to the true log-likelihood of the observed data, we train the parameters of the generative model and the variational recognition model jointly to optimise this bound.

Since the recognition model acts as a pseudo-posterior (it approximates it given constraints on the distributional form of the recognition model) we can use this to translate from one language to another by finding the posterior q-distribution over z and then from sampled z find the most likely output of the languages.

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, hereafter called NLP, is a subfield within artificial intelligence almost as old as the field itself. Briefly, NLP can be defined as the study of the properties of natural language and how these properties may be used to answer questions that humans who master the language can do naturally. Clearly, if we are to enable machines to cooperate with human beings, it is of utmost importance that they can speak our language, since we are not very apt in speaking theirs!

NLP has a plethora of different subfields, however, in this thesis we will limit ourselves to the field of machine translation. Machine translation has long been a cornerstone of NLP and has undergone many different guises from the beginning of the 1940's until today.

Machine translation can essentially be seen as a problem of learning a model that maps from one language $\mathscr{X}$ to another language $\mathscr{Y}$. Formally, we have a dataset $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^{N}$, such that $\boldsymbol{x}_i \in \mathscr{X}, \boldsymbol{y}_i \in \mathscr{Y}$ represent the same sentence in two different languages. The goal is then to find a mapping $f : \mathscr{X} \to \mathscr{Y}$ such that $f(\boldsymbol{x}) \approx \boldsymbol{y}$ with regards to some chosen metric.

From this several different ways of looking at language has emerged. The two different ways depends on how granular you are going. For latin based languages and other languages with a phonetic writing system, this means deciding if the atomic symbols should be characters of words. Character level gives the advantage that your dictionary of characters i finite, it consists of your alphabet. Since it's finite there's less of a worry of new characters being introduced. The data also gets bigger. For words you have a problem that the dictionary swells, often being bigger than 100000 unique words, which are troublesome when it comes to calculate the normalizer of the softmax in order to get a distribution of the output from the raw model.

From the chain rule of probability we have that any sentence $\boldsymbol{x}$ is such that the joint probability of the atomic units may be rewritten in a recursive form,

$$\begin{aligned} p(\boldsymbol{x}) &= p(x_1, \ldots, x_n) \\ &= \prod_{i=1}^{n} p(x_i | x_{i-1}, \ldots, x_1) \end{aligned}$$

, which shows that one way of modelling language is to try to capture this temporal relationship using models catered for long term dependencies. The main problem is one of scale. This can be seen from the Markovian models called *N-grams* which simply assumes that the dependency in languages can be explained by N-dimensional tables, meaning that N-grams can be seen as N-step markov models applied to language data. In practice, this means that we assume the relationship

$$p(x_i | x_{i-1}, \ldots, x_1) = p(x_i | x_{i-1}, \ldots, x_{i-N})$$

.

Although this is a reasonable assumption from a modelling point of view, given that we take N to be large enough, it doesn't work well in practice. Due to sparsity, these N-dimensional tables will be mostly filled with zeros. This is simple to see whether working on a word or character level, I will just give an example using words for maximum impact:

Assume we are trying to model the english language using a Tri-gram model, meaning we are only considering grouped word of 3. Putting together random words from the dictionary of words in the English language gives us by a huge margin gibberish, in terms of both semantic meaning and of grammar:

moucharaby epithelium sonlike
Bacchides lulliloo oneiromancer
actinology dihydroxy nonmineralogical
Homalonotus Vened dyspepsy
uncessant twee femorofibular

. This is due to the fact that only extremely few tuples of word triplets are actually valid in the sense that they can be said to exist naturally in the English language. Mathematically, this means that the tables we get from running maximum likelihood on these tables to find the actual probabilities, which is just a matter of counting the number of times the triplets occur compared to the number of times that the starting symbol occur in the corpus we have at hand.

Neural Machine Translation, hereafter called NMT, is the use of Neural Networks as the models inside the machine translation systems. NMT can be trained end-to-end by specifying the data, architecture and the various other components that make up the model specification. While NMT is very data-hungry, mostly getting its power from being able to unearth the various rules and constructs in a language (semantically and grammatically) through the use of the statistical information existing within it, it is extremely well-suited for learning these rules given enough data. This black box approach means that people without any knowledge of language $\mathscr{X}$ and $\mathscr{Y}$ can train sophisticated translation systems on par with state of the art results, solely relying on the data to speak for itself.

In this thesis we will explore fully probabilistic models, meaning that any statement about output languages can be given a probability score. Using the language of probability enables use to make statements about the plausibility of sentences and logical statements about these sentences using the laws of probability. Practically, it means that we get a model which is generative, that is, we can sample random variables of the hidden variable that encodes the sentences in a language-agnostic which through the models can map back to the sentences in the original languages. This gives us some hope that the model have some kind of internal language model and not only learns the specific output relation when mapping from $\mathscr{X}$ to $\mathscr{Y}$.

In essence, we will use recent techniques that enables us to train latent-variable models, that is models where the observed output, in our case the language sentence tuples depend on a hidden variable $\boldsymbol{z}$ that encodes the information about the sentence in a language-agnostic way. Consider the sentence *The quick brown fox jumps over the lazy dog*. The sentence is written using the English language, but it's easy to imagine the if we disregard the language we are saying it in, whether it be German, *Der schnelle braune Fuchs sprang über den faulen Hund* or in Latin, *Lorem ipsum vulpes salit super piger canis*, there is some underlying meaning which all languages are trying to convey. Our model tries to encode this meaning in terms of a hidden stochastic variable $\boldsymbol{z}$.

This leads to the realisation given that we can encode $\boldsymbol{z}$ properly we should be able to translate from language $\mathscr{X}$ to $\mathscr{Q}$ without the model having ever seen a sentence pair of the form $(\boldsymbol{x},\boldsymbol{q}), \boldsymbol{x} \in \mathscr{X}, \boldsymbol{q} \in \mathscr{Q}$.

1.1 Things to talk about

- NLP (History, challenges)
- Deep Learning (What it is)
- Neural Machine Translation

Chapter 2

Background Knowledge

From here on I will assume familiarity with some concepts which will be important for the experiments that we will conduct and analyse.

2.1 Probability Theory and Statistics

2.1.1 Probability Theory as a Reasoning System

Although often seen as a self-contained mathematical theory of stochastic systems and reasoning about the random, probability theory has a prominent role within machine learning since it gives us a principled way of reasoning about the world. As proved by Cox, if we are to accept that any theory of reasoning is to satisfy the Cox desiderata,

1. Degrees of plausibility are represented by real numbers.

2. Qualitative correspondence with common sense

3. If a conclusion can be reasoned out in more than one way, then every possible way must lead to the same result.

, then we must accept that this theory is isomorphic to probability theory and effectively the same.

This is a formal way of stating that probability theory is the best way to organize our reasoning if we want to make sure that we are consistent in the way we reason.[1, p. 3-23]

2.1.2 Rules and Theorems

Rules of probability that will be useful to us are the following axioms

Unit volume

$$\int_{\mathscr{X}} p(X) = 1 \tag{2.1}$$

Non-negativity

$$p(X) \geq 0 \tag{2.2}$$

where $\mathscr{X}$ is the domain of X and the generalized integral is interpreted as the Lebesgue integral if X is continuous and as a sum over the possible values of X if it is discrete.

I will assume the notion of a random variable X in an intuitive sense. However, this X might be represented in many forms and in our case there is not reason to not be able to put probabilities over sentences, words and/or characters and conditioning thereof. Similarly I will assume familiarity with the different notions of continuous, discrete and categorical random variables. Finally

Most manipulation of statements about random variables can be stated as a consequence of the two following fundamental rules

Sum rule

$$p(X) = \int_{\mathscr{Y}} p(X,Y) \tag{2.3}$$

Product rule

$$p(X,Y) = p(Y|X)p(X) \tag{2.4}$$

such that X,Y are two random variables defined on the domains $\mathscr{X}, \mathscr{Y}$. The integral $\int_{\mathscr{Y}}$ is to be understood in the general sense, if Y is continuous it is the ordinary Lebesgue integral as commonly used throughout mathematics and calculus, while if Y is discrete then it is the sum over the possible values of Y. $p(X,Y)$ is the joint distribution of X and Y, $p(Y|X)$ is the probability of Y conditioned on nX and $p(X)$ is the marginal distribution of X. Using these rules it is easy to Bayes theorem, one of the most integral (and simple) theorem of probability theory

Bayes Theorem

$$p(Y|X) = \frac{p(X|Y)p(Y)}{p(X)} \tag{2.5}$$

Two very important operations involving probabilities of random variables are those of *Expectation* and *Covariance*. These take as input a function f and maps to the real number line $\mathbb{R}$, and are defined implicitly with regards to some random variable X and its probability distribution $p(X)$.

Expectation

$$\mathbb{E}_X[f] = \int_{\mathscr{X}} p(x)f(x) \tag{2.6}$$

Covariance

$$\mathrm{Cov}(X,Y) = \mathbb{E}_{XY}[(X - \mathbb{E}_X[X])(Y - \mathbb{E}_Y[Y])] \tag{2.7}$$

We then define the variance operator as

$$\mathrm{Var}(X) = \mathrm{Cov}(X,X) \tag{2.8}$$

The generalisation from $f : \mathscr{X} \to \mathbb{R}$ to $f : \mathscr{X} \to \mathbb{R}^n$ is defined in the straightforward manner such that if $\boldsymbol{f} = f(X)$ then

$$\mathbb{E}_X \begin{bmatrix} \boldsymbol{f}_1 \\ \vdots \\ \boldsymbol{f}_n \end{bmatrix} = \begin{bmatrix} \mathbb{E}_X \boldsymbol{f}_1 \\ \vdots \\ \mathbb{E}_X \boldsymbol{f}_n \end{bmatrix}$$

similarly $\mathrm{Cov}(\boldsymbol{f})$ is a $D \times D$-dimensional matrix where $\mathrm{Cov}(\boldsymbol{f})_{i,j} = \mathrm{Cov}(\boldsymbol{f}_i, \boldsymbol{f}_j)$.

2.1.3 The Gaussian Distribution

A very common distribution in machine learning and statistics in general is the Gaussian distribution, also called the Normal distribution. The Gaussian distribution satisfies some properties that makes it an ideal candidate from a modeling perspective including the Central Limit Theorem which says that sums of independent random variables of finite mean and variance will tend to a normal distribution, and the fact that the joint distribution of many random variables that are Gaussianly distributed is itself Gaussian which means that conditionals, posteriors and other distributions are themselves Gaussian when we deal with Gaussian random variables.

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

$$\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu},\boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{D/2}}\frac{1}{|\boldsymbol{\Sigma}|^{1/2}}\exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right) \tag{2.9}$$

where $\boldsymbol{\mu}$ is a D-dimensional mean vector, $\boldsymbol{\Sigma}$ is a $D \times D$ dimensional covariance matrix, and $|\boldsymbol{\Sigma}|$ denotes the determinant of $\boldsymbol{\Sigma}$. The meaning of these parameters can be shown to correspond to the operations of the expected value and covariance of $\boldsymbol{x}$, $\mathbb{E}_{\boldsymbol{x}}[\boldsymbol{x}] = \boldsymbol{\mu}$ and $\mathrm{Cov}(\boldsymbol{x}) = \boldsymbol{\Sigma}$.

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 $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu},\boldsymbol{\Sigma})$ as a linear combination of a unit Gaussian random variable $\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{x}|\boldsymbol{0}_D, \boldsymbol{I}_{D\times D})$. If we let $\boldsymbol{\Lambda\Lambda} = \boldsymbol{\Sigma}$ be the Cholesky decomposition[2, p. 100-102] of $\boldsymbol{\Sigma}$, then we also have that

$$p(\boldsymbol{x}) = p(\boldsymbol{\mu} + \boldsymbol{\Lambda z})$$

. If we further assume that $\boldsymbol{x}$ is parametrised by $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ such that $\boldsymbol{\Sigma}$ is diagonal positive definite with diagonal $\boldsymbol{\sigma}$, then $\boldsymbol{\Sigma} = \boldsymbol{\sigma} \odot \boldsymbol{I}_{D\times D}$. Finally this means that if we want to sample a random variable $\boldsymbol{x}$ with diagonal covariance structure, then we can do this by sampling a unit normal $\boldsymbol{z}$ which we then transform, which we can express as

$$\boldsymbol{x} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{z} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma} \odot \boldsymbol{I}_{D\times D}) \tag{2.10}$$

As the Gaussian distribution is part of the exponential family, the density of 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 $\boldsymbol{x}_i \sim \mathscr{N}(\boldsymbol{x}_i|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$, then the joint can be found to be Gaussian distributed as

$$\mathscr{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \tag{2.11}$$

, where

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

.[3, p. 78-84]

In the case of two random variables distributed according to the form as laid out in 2.10, $\boldsymbol{x} \sim \mathscr{N}(\mathscr{N}(\boldsymbol{\mu}_{\boldsymbol{x}}, \boldsymbol{\sigma}_{\boldsymbol{x}} \odot \boldsymbol{I}))$ and $\boldsymbol{y} \sim \mathscr{N}(\mathscr{N}(\boldsymbol{\mu}_{\boldsymbol{y}}, \boldsymbol{\sigma}_{\boldsymbol{y}} \odot \boldsymbol{I}))$ we have that the resulting distribution $p(\boldsymbol{x}, \boldsymbol{y}) = p(\boldsymbol{x})p(\boldsymbol{y})$ is distributed such that

$$p(\boldsymbol{x}, \boldsymbol{y}) = \mathscr{N}(\boldsymbol{\mu}_{\boldsymbol{x},\boldsymbol{y}}, \boldsymbol{\sigma}_{\boldsymbol{x},\boldsymbol{y}}) \tag{2.12}$$

where

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

.A

2.1.4 Maximum Likelihood Estimation

An often used estimator for parameters is the *Maximum Likelihood Estimator*, hereafter called the MLE.

Assume we have a model $\mathscr{M}$ parametrised by $\boldsymbol{\theta}$ constrained to live in the parameter space $\boldsymbol{\Theta}$. Given data $\mathscr{D}$ we want to be able to fit the parameters $\boldsymbol{\theta}$ such that these somehow explains the data, and thus should be able to work on new data in a manner that generalises well. The MLE of of the parameters of the model is defined to be

$$\hat{\boldsymbol{\theta}}_{ML} = \underset{\boldsymbol{\theta}\in\boldsymbol{\Theta}}{\operatorname{argmax}} \mathscr{L}(\boldsymbol{\theta};\mathscr{D})$$

.

While the original MLE is defined in terms of the likelihood function $\mathscr{L}(\boldsymbol{\theta};\mathscr{D})$, it's often more practical to work with the logarithm of this function, the log-likelihood function $\ell(\boldsymbol{\theta};\mathscr{D})$. Using the common assumption of i.i.d datapoints, the joint distribution becomes a product of individual probabilities for each datapoint,

$$\mathscr{L}(\boldsymbol{\theta}|\mathscr{D}) = p(\boldsymbol{x}_1,\dots,\boldsymbol{x}_n|\boldsymbol{\theta}) = \prod_i^n p(\boldsymbol{x}_i|\boldsymbol{\theta}) \tag{2.13}$$

. Using the log-likelihood we transform this product into a form involving sums

$$\ell(\boldsymbol{\theta}|\mathscr{D}) = \log\mathscr{L}(\boldsymbol{\theta}|\mathscr{D}) = \sum_i^n \log p(\boldsymbol{x}_i) \tag{2.14}$$

. Besides from simplifying notation and calculation, it has the added benefit of reducing the risk of arithmetic underflow due to the small magnitude of individual probabilities[1]

In a sense, the MLE is the best fit to the data given that we put a flat prior over the parameters $\boldsymbol{\theta}$, that is we let the data speak for itself. The MLE only takes into account the information given by the available data $\mathscr{D}$ and thus makes a choice of $\boldsymbol{\theta}$ based upon the information in the data set and nothing else. Besides this intuitive justification of the estimator, from a frequentist perspective it comes with a number of advantageous properties that we like an estimator to have, making it a natural candidate for training models.

The main problem with MLE's are that they are defined implicitly in terms of the likelihood function of the data. In that sense every setting is different as there

[1] Also, with the use of the log-sum-exp-trick,

$$\log\sum_{i=1}^n \exp(x_n) = \max_i x_i + \log\sum_{i=1}^n \exp(x_n - \max_i x_i)$$

this problem can be reduced even further is no straightforward way to solve for the MLE as we in most cases have to resort to numerical solutions to try to find the global maximum of $\mathscr{L}(\boldsymbol{\theta};\mathscr{D})$, a function which is often non-linear, multimodal and with a complex surface, leading to local maximums being found instead of the global maximum in many cases.[4]

2.1.5 Graphical models

A very convenient tool for probabilistic modelling is the notion of using diagrams to specify the conditional relationships between random variables. A graphical model is a diagrammatic way of specifying this relationship by creating a Directed Acyclic Graph, a directed graph without any cycles. This representation is called a *Graphical Model* and provide a powerful way to visualize the structure of the probabilistic model and also how to use and abuse the structure of the model in order to infer variables in a computationally efficient way.

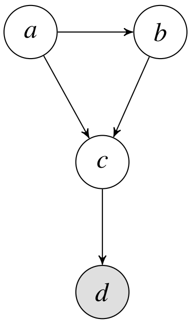
A graph in this setting consists of a set of *vertices* connected by *edges*, following the notation and nomenclature of graph theory as used in mathematics. While there are many different kinds of graphical models depending on the type of graph structure used (directed graphical models, undirected graphical models, factor graphs, etc.), we will only focus on the subset of graphical models called directed graphical models.

Directed graphical models specify how the joint distribution of a set of random variables $\mathscr{X}$ factors in a conditional manner. In general, the relationship between a given directed graph and the corresponding distribution over the variables in $\mathscr{X}$ is such that the join distribution defined by the graph is given by the product, over all vertices of the graph, of a conditional distribution for each vertex 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

$$p(\mathscr{X}) = \prod_{k=1}^{K} p(x_k|\mathrm{pa}(x_k)) \tag{2.15}$$

where $\mathrm{pa}(x_k)$ 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. As an example I will repeat the above formulation in the context of the directed 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 we don't observe.

For this example we have the following factorisation of the joint distribution, following the rules laid out in equation 2.15,

$$p(a,b,c,d) = p(d|c)p(c|a,b)p(b|a)p(a)$$

.

2.1.6 Approximate Inference

While MLE is in many ways the optimal way that we can fit the model, it's only analytically and/or computationally feasible for very simple models which relies simple transformations and tractable distributional relationships. In cases where more powerful models are used it is very hard to find the MLE or even local maximum to the likelihood $\mathscr{L}$, or equivalently the log-likelihood ℓ.

While in theory most of these problems can be resolved by MCMC sampling, which also practically have been implemented in the way of probabilistic programming with some success[5, Ch. 1][6, 7], most often this is too computationally intensive and has a large cost in terms of time. Approximate inference forms an alternative to MCMC for solving intractable densities by recasting this problem into an optimization problem instead of a sampling one as in MCMC.

Approximate inference form a group of methods for approximating probability densities and is widely used to approximate posterior densities for Bayesian models. Compared to MCMC it tends to be faster and easier to scale to larger datasets, but lacks in generality as every new problem has to be solved again within the approximate inference framework.

The most common setting is in latent variable models, where a latent variable, often denoted by $\boldsymbol{z}$ is introduced to explain underlying causes to the observed variables, such as different clusters for Mixture of Gaussians or a lower-dimensional manifold in terms of the Factor Analysis model[3, page. 430-439, 583-586]. Latent variable models which rely on Gaussian distributions and linear relationships may be learned in an exact manner in the context of the EM algorithm or its many variations which guarantees parameters $\hat{\boldsymbol{\theta}}$ such that this point in the parameter space will yield a local maximum of the likelihood function[8, 9].

As theory and computing power has progressed, so has the advances in more powerful models. Compared to older models for which solutions often could be found analytically, these models were too non-linear, non-gaussian and complex to train in a straightforward manner, which can bee seen directly from the numerous heuristics that exist with regards how to train deep neural networks[10, 11]. As neural network architectures are becoming commonplace in bayesian modelling, techniques to train these models has also become more prevalent.

The problem setting of variational inference is a latent variable model such that it can be split up into the observed variables $\boldsymbol{x}$ and latent variables $\boldsymbol{z}$ such that

$$p(\boldsymbol{z},\boldsymbol{x}) = p(\boldsymbol{z})p(\boldsymbol{x}|\boldsymbol{z}) \tag{2.16}$$

. Indeed this describes a plethora of different models showing how broadly the variational framework can be applied. Variational inference has enjoyed much success in the past over other domains as well[12], but we will focus on this special case.

Technically, Approximate inference is a way of approximating a complicated distribution $p(\boldsymbol{z}|\boldsymbol{x})$ by a distribution $q(\boldsymbol{z})$ belonging to some constrained family of distributions $\mathscr{Q}$, for continuous distributions often such that $\mathscr{Q} = \{\mathscr{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) | \boldsymbol{mu} \in \mathbb{R}^d, \text{p.d } \boldsymbol{\Sigma} \in \mathbb{R}^{d \times d}\}$. The goal is then to find the elementfof $\mathscr{Q}$ that minimizes some distance from p to q, most often the Kullback-Leibler divergence,

$$q^*(\boldsymbol{z}) = \underset{q(\boldsymbol{z}) \in \mathscr{Q}}{\operatorname{argmin}} KL(q(\boldsymbol{z}||p(\boldsymbol{z}|\boldsymbol{x}))) \tag{2.17}$$

. This optimal q^* may then be used as a pseudo-correct distribution in order to calculate other statistics and quantities.

This also means that we have recast the original MLE into a simpler optimisation problem involving terms which lends itself to analytical solutions and computational tractability. For MLE we are interested in optimizing the log-likelihood

$$p(\boldsymbol{x}|\boldsymbol{\theta}) = \int_{\mathscr{Z}} p(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})$$

. For many models this is not possible to evaluate.

2.2 Deep Learning

A ubiquitous classifier within statistics is logistic regression. Logistic regression uses an input vector $\boldsymbol{x}$ in order to give importance scores in form of probabilities to different classes $y \in \{c_1, \dots, c_k\}$. It gets its name from the logistic function

$$\sigma(a) = \frac{1}{1 + e^a} \tag{2.18}$$

which together with an affine transformation $\boldsymbol{Wx}$ yields the layer

$$\sigma(\boldsymbol{Wx})$$

which transforms values from a feature space $\boldsymbol{X} \subset \mathbb{R}^m$ into probabilities[3].

Deep learning builds upon this intuition by recursively applying transformations and activation functions, functions which in some sense maps input on to *ON/OFF* states. These functions take their functionality from an abstraction from how neurons function when firing with regards to input, mirroring how artificial neural networks have taken inspiration from how the brain operates in the past. On a very basic level, neural networks are characterised by stacked layers of affine transformations followed by activation functions, where the output of one layer serves as the input to the next layer. The final layer outputs $\hat{y}$ where the form of $\hat{y}$ depends on the application. The hope is that after training the model using backpropagation[13] that the model is able to predict satisfactory and drive down the specified loss.

Deep models are very powerful in that they are able to model complex functional relationships. In our case we are looking at Supervised and Semi-supervised learning, trying to find the relationship between $\boldsymbol{x} \in bmX$ and $\boldsymbol{y} \in \boldsymbol{Y}$ of some kind of functional form $f(\boldsymbol{x}) \approx \boldsymbol{y}$.

Besides from the straightforward models where we stack logistic regressors serially, neural networks have extended well beyond this into an extremely diverse set of models that can capture different aspects of data such as long term-dependencies through the architecture of Recurrent Neural Networks and invariances by using convolutions. Many of these models have also found use in NLP, especially in the form of RNN's which are well-suited for handling language due to how it enables information to flow through time[14][15] and more recently CNN's for finding representation over many different scales[16][17][18].

In a Bayesian setting each graphical model codifies how different random variables relate to each other in terms of independency. This is specified by the Directed Acyclic Graph where each arrow signifies a conditional relationship between $\boldsymbol{x}$ and $\boldsymbol{y}$. A full description of how graphical models ,

2.3 Natural Language Processing

2.3.1 Language statistics

2.3.2 Word embeddings

2.3.3 Neural machine translation

2.4 Optimization

2.4.1 Problem formulation

Most of machine learning may be recasted as an optimization problem. In a fully probabilistic setting, the function to optimise is simply the joint probability of the data, parametrised by the model. In a likelihood setting this reduces to finding the maximizer of equations of the form 2.14.

Numerical optimization has been applied to many parts of science and thus many different algorithms have been developed to tackle different problems. However, most of the theoretical results that exist deal with convex optimization, where the function we are trying to optimize is particularly well-behaved, leading to theoretical guarantees on the solution converged to. The surface of the function we are trying to optimize in machine learning in general and deep learning in particular is not well-behaved, being highly non-linear and non-convex, meaning most theoretical results that exist do not apply to this domain[19].

Nevertheless, 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 the Stochastic Gradient Descent. This has led to numerous gradient descent-like algorithms used machine learning[20].

2.4.2 Stochastic Gradient Descent

For a normal probabilistic machine learning problem, we have data $\mathscr{D}$ that we try to model with a model $\mathscr{M}$ parametrised by parameters $\boldsymbol{\theta} \in \Theta$. The optimization problem can in the most general case be recast as an effort to find the parameters $\boldsymbol{\theta}_{ML}$ that maximizes the log-likelihood function 2.14.

As this can't be found analytically we have to resort to numerical schemes to find good candidates which hopefully will be close to the true solution $\boldsymbol{\theta}_{ML}$. The first candidate is called Gradient Descent (GD). GD approximates the gradient as if it was linear and takes steps to maximize the probability through a hill-climbing scheme. If we let $\mathscr{D} = \{\boldsymbol{z_i}\}_{i=1}^{n}$ such that $\boldsymbol{z_i}$ is any general datapoint, Q an objective function that we try to maximize, such that $Q : \Theta \times \mathscr{Z} \to \mathbb{R}$, then the update using GD looks as follows

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \gamma_t \frac{1}{n} \sum_{i=1}^{n} \nabla_{\boldsymbol{\theta}} Q(z_i, \boldsymbol{\theta}_t) \tag{2.19}$$

. While γ_t may vary with t, it is often fixed for practical reasons. As we can see GD takes into account all of the datapoints in the set, in some sense updating the parameters with respect to the true linear approximation of the gradient.

However, with the huge size of data existing today, it is often not feasible to calculate the update for all datapoints in the dataset due to the time it would take and memory it would take to store the gradients.

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, hence the word *stochastic*, as it picks a random subset of the dataset to update $\boldsymbol{\theta}$ with regards to. If we let I_t be a random subset of the indices of $\mathscr{D}$ of size m then SGD does the following update

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \gamma_t \frac{1}{m} \sum_{i \in I_t} \nabla_{\boldsymbol{\theta}} Q(z_i, \boldsymbol{\theta}_t) \tag{2.20}$$

[21][3, p. 240].

The stochasticity has been shown to act as a regularizer and several analyses of this in terms of a Bayesian framework has been done, explaining the stationary behaviour of SGD with constant learning rate after convergence[22, 23]

This solves the problem of memory and time, but also improves on GD in terms of acting as a sort of regulariser for the model if γ_t is kept fixed. Experimental results also show that SGD yields extremely good practical convergence on many tasks, and was often the first algorithm to be tried on deep learning problems before other algorithms improved on the issue of picking a good learning rate.

2.4.3 ADAM

SGD has found widespread use within the machine learning community due to strong experimental results and ease of use, especially in deep learning. Lately though, a number of alternatives have sprung up that aims to improve the vanilla SGD, such as RMSProp[24] and AdaGrad[25].

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, ADAM operates on a stochastic objective function $f(\boldsymbol{\theta})$, which in our case would be the probability averaged over our input batch. $f_t(\boldsymbol{\theta})$ denotes this function over different timesteps, equally $g_t = \nabla_{\boldsymbol{\theta}} f_t(\boldsymbol{\theta})$. 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 1 ADAM

```
Require: $\alpha$: 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 1st moment vector)
$v_0 \leftarrow 0$ (Initialize 2nd 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)
    $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \alpha \cdot \hat{m}_i/(\sqrt{\hat{v}_t} + \varepsilon)$ (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[26].

Chapter 3

Methods and Theory

Here we build on the theory laid out in Background Knowledge, and take it further, and tell how we use it for our experiments.

Chapter 4

Experiments

4.1 Data

4.1.1 Dataset

The dataset we have chosen to evaluate the model on is the Europarl dataset between languages English and French. Europarl is a dataset of the proceedings of the European Parliament, comprising in total of the 11 official languages of the European Union.

The dataset was chosen as the number of sentences for English and French is enough to be able to generalise (the uncompressed size of the full dataset is 619MB, 288MB for English, 311MB for French) to new sentences, and furthermore has established baseline for NMT in the form of BLEU scores for all the different language pairs in the full dataset, English-French in particular.[27]

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 leverage the mathematics easily we need to translate the raw form into a form which take place in a high-dimensional space instead, here $\mathbb{R}^N$. Equally we remove aspects of the data that will make it harder for the model to learn due to sparsity and other statistical peculiarities of the data and NLP in general.

Preprocessing the data we make the following simplifications

English: I declare resumed the session of the European Parliament adjourned on Friday 17 December 1999, and I would like once again to wish you a happy new year in the hope that you enjoyed a pleasant festive period.
French: Je déclare reprise la session du Parlement européen qui avait été interrompue le vendredi 17 décembre dernier et je vous renouvelle tous mes vux en espérant que vous avez passé de bonnes vacances.

Table 4.1: A randomly sampled sentence from the Europarl corpus

4.1.2.1 Character Level

4.1.2.2 Word Level

The problem with words is that there exist an immense quantity of them, if even just due to grammatical constructs (example: run, running, ran etc.). Similarly, for any given point in time, words go in and out of use and this necessitates choosing which words to include in the dictionary. The dictionary consists of all of the words that we consider part of the language, everything not in the dictionary are either too rare or for some other reason excluded from use.

- We only include sentences of length between 2 and 30. This makes sure that the model have long enough sentences such that it may learn from the dependencies between words, but short enough so that the parameters are able to capture the long-term dependencies of sentences.

- 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 80000 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. This makes sure that only words which are prevalent enough such that the model can derive its relation to other words are part of the dictionary.

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

- In parts where we have a dataset of 2 or more language sentences in parallel, we make sure that both of the the languages both satisfy the above criteria.

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.

4.2 Scores

We will evaluate our models on a variety of scores:

ELBO ELBO is the lower bound of the actual log-likelihood of the observed data

$$\sum_{i=1}^{N} \log p_\theta(\boldsymbol{x}_i)$$

Qualitative Since natural language is not a formal in the sense that it is ambiguous, inconsistent and with exceptions to rules; any of these scores will be imperfect insofar as taking into account the feel of the generated sentences. Due to this we will inspect the sentences manually.

BLEU BLEU compares the generated sentences with sentences translated by professional translators, yielding a score telling us how well the generated translation does in relation to the translated benchmarks for each sentence.

KL Part of our investigation is about building models that take into account the latent space, enforcing the model to encode the information in the latent variable z instead of the encoder/decoder part. Luckily, we have a quantitative measure of this, the KL divergence between the prior and the posterior q-distribution over $\boldsymbol{z}$,

$$KL[q_\phi(\boldsymbol{z}|\boldsymbol{x})||p_\theta(\boldsymbol{z})]$$

, where the KL is a measure of how much information is put into the q-distribution compared to just using the prior isotropic gaussian over $\boldsymbol{z}$, $p_{\theta}(\boldsymbol{z})$.

4.3 layout

We will perform the following experiments, building up the order of doing them from least complex to more complex.

We first have different models, depending on how we choose

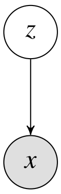
Recognition model
- WaveNet
- RNN
- MLP

Factorisation of Rec model
- Independence of $\boldsymbol{x}, \boldsymbol{y}$
- diagonal sigma
- opposite of these

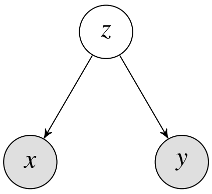
Generative model
- AUTR
- WaveNet
- RNN

.

We then use these models on different types of language modelling:



Reconstruction



Translation

While using the recognition model to do translation (We let $\boldsymbol{x}$ encode all information about $\boldsymbol{z}$, and then see what the generated $\boldsymbol{x}, \boldsymbol{y}$ correspond to).

Chapter 5

Conclusions

What have we learned from all of this?

Appendix A

Proofs

These are the proofs for some of the results.

Appendix B

Another Appendix About Things

Appendix C

Colophon

This is a description of the tools you used to make your thesis. It helps people make future documents, reminds you, and looks good.

(example) This document was set in the Times Roman typeface using LaTeX and BibTeX, composed with a text editor.

[28]

Bibliography

[1] E.T. Jaynes and G.L. Bretthorst. *Probability Theory: The Logic of Science*. Cambridge University Press, 2003.

[2] 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.

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

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

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

[6] Bob Carpenter, Daniel Lee, Marcus A. Brubaker, Allen Riddell, Andrew Gelman, Ben Goodrich, Jiqiang Guo, Matt Hoffman, Michael Betancourt, and Peter Li. Stan: A probabilistic programming language.

[7] John Salvatier, Thomas V. Wiecki, and Christopher Fonnesbeck. Probabilistic programming in python using pymc3. *PeerJ Computer Science*, 2:e55, 2016.

[8] 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.

[9] Radford Neal and Geoffrey E. Hinton. A view of the em algorithm that justifies incremental, sparse, and other variants. In *Learning in Graphical Models*, pages 355–368. Kluwer Academic Publishers, 1998.

[10] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. *arXiv:1206.5533 [cs]*, June 2012. arXiv: 1206.5533.

[11] Hugo Larochelle, Yoshua Bengio, Jérôme Louradour, and Pascal Lamblin. Exploring strategies for training deep neural networks. *J. Mach. Learn. Res.*, 10:1–40, June 2009.

[12] Matthew J. Beal. *Variational Algorithms for Approximate Bayesian Inference*. PhD thesis, Gatsby Computational Neuroscience Unit, University College London, 2003.

[13] David E. Rumelhart, Richard Durbin, Richard Golden, and Yves Chauvin. Backpropagation. chapter Backpropagation: The Basic Theory, pages 1–34. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 1995.

[14] Alex Graves. Generating Sequences With Recurrent Neural Networks. *arXiv:1308.0850 [cs]*, August 2013. arXiv: 1308.0850.

[15] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *arXiv:1406.1078 [cs, stat]*, June 2014. arXiv: 1406.1078.

[16] Stanislau Semeniuta, Aliaksei Severyn, and Erhardt Barth. A Hybrid Convolutional Variational Autoencoder for Text Generation. *arXiv:1702.02390 [cs]*, February 2017. arXiv: 1702.02390.

[17] Zichao Yang, Zhiting Hu, Ruslan Salakhutdinov, and Taylor Berg-Kirkpatrick. Improved Variational Autoencoders for Text Modeling using Dilated Convolutions. *arXiv:1702.08139 [cs]*, February 2017. arXiv: 1702.08139.

[18] Jonas Gehring, Michael Auli, David Grangier, and Yann N. Dauphin. A Convolutional Encoder Model for Neural Machine Translation. *arXiv:1611.02344 [cs]*, November 2016. arXiv: 1611.02344.

[19] Anna Choromanska, Mikael Henaff, Michael Mathieu, Gérard Ben Arous, and Yann LeCun. The Loss Surfaces of Multilayer Networks. *arXiv:1412.0233 [cs]*, November 2014. arXiv: 1412.0233.

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

[21] 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.

[22] Stephan Mandt, Matthew D. Hoffman, and David M. Blei. Stochastic Gradient Descent as Approximate Bayesian Inference. *arXiv:1704.04289 [cs, stat]*, April 2017. arXiv: 1704.04289.

[23] Stephan Mandt, Matthew D. Hoffman, and David M. Blei. A Variational Analysis of Stochastic Gradient Algorithms. *arXiv:1602.02666 [cs, stat]*, February 2016. arXiv: 1602.02666.

[24] 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.

[25] 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.

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

[27] 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.

[28] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. *arXiv:1409.0473 [cs, stat]*, September 2014. arXiv: 1409.0473.