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 14, 2017

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

Nomenclature **9**

1 Background Knowledge **13**

1.1 Probability Theory and Statistics 13

1.1.1 Rules and Theorems 13

1.1.2 The Gaussian Distribution 15

1.1.3 Maximum Likelihood Estimation 16

1.1.4 Graphical models 17

1.1.5 Approximate Inference 19

1.2 Deep Learning 20

1.2.1 Multilayer Perceptron 21

1.2.2 Recurrent Neural Networks 21

1.2.3 Convolutional Neural Networks 21

1.3 Natural Language Processing 21

1.3.1 Language model 22

1.3.2 Word embeddings 22

1.3.3 Neural machine translation 24

1.4 Optimization 24

1.4.1 Problem formulation 24

1.4.2 Stochastic Gradient Descent 25

1.4.3 ADAM 26

2 Methods and Theory **28**

2.1 ELBO 28

2.1.1 Translation case 29

Bibliography **31**

List of Figures

1.1 Encoder decoder schematic 24

List of Tables

Nomenclature

GD	Gradient Descent
MCMC	Markov Chain Monte Carlo
MLE	Maximum Likelihood Estimation
NLP	Natural Language Processing
NMT	Neural Machine Translation
SGD	Stochastic Gradient Descent
VI	Variational Inference

$\boldsymbol{\theta} \in \boldsymbol{\Theta}$	Parameters $\boldsymbol{\theta}$ belonging to the parameter space $\boldsymbol{\Theta}$
$\boldsymbol{v}_{\mathrm{w}}$	Word embedding of w
$\boldsymbol{w}$	One-hot encoding of w
$\boldsymbol{x}_i \in \mathcal{Y}$	Sentence $\boldsymbol{x}_i$ belonging to language X
$\boldsymbol{y}_i \in \mathcal{Y}$	Sentence $\boldsymbol{y}_i$ belonging to language Y
$\mathcal{M}$	A model $\mathcal{M}$
$\mathcal{X}$	Set of sentences belonging to language X

$\mathcal{Y}$	Set of sentences belonging to language Y
w	The word as a symbol
$\sigma(x)$	An activation function with respect to x
N	Number of data points
V	Dictionary; set of all words in vocabulary

$\mathbf{0}_d$	The d-dimensional zero vector
$\mathbf{0}_{d\times d}$	The $d \times d$-dimensional zero matrix
$\boldsymbol{A} \in \mathbf{R}^{n\times m}$	Real matrix $\boldsymbol{A}$ of dimensions $n \times m$
$\boldsymbol{A} \odot \boldsymbol{B}$	Elementwise multiplication of matrices $\boldsymbol{A}$ and $\boldsymbol{B}$
$\boldsymbol{I}_d$	The $d \times d$ unit matrix
$\boldsymbol{x}$	A real column vector
$\mathbb{R}$	Set of real numbers
$\mathbb{R}^d$	Real vector space of dimension d
$\nabla_{\boldsymbol{x}} Q(\boldsymbol{x})$	Gradient of function Q with respect to vector $\boldsymbol{x}$
$\mathrm{pa}(x)$	Set of parent nodes for node x in a graphical model
$\mathbf{0}_d$	The d-dimensional zero vector
$\mathbf{0}_{d\times d}$	The $d \times d$-dimensional zero matrix
$\boldsymbol{A} \in \mathbf{R}^{n\times m}$	Real matrix $\boldsymbol{A}$ of dimensions $n \times m$
$\boldsymbol{A} \odot \boldsymbol{B}$	Elementwise multiplication of matrices $\boldsymbol{A}$ and $\boldsymbol{B}$
$\boldsymbol{I}_d$	The $d \times d$ unit matrix

$\boldsymbol{x}$	A real column vector
$\mathbb{R}$	Set of real numbers
$\mathbb{R}^d$	Real vector space of dimension d
$\nabla_{\boldsymbol{x}} Q(\boldsymbol{x})$	Gradient of function Q with respect to vector $\boldsymbol{x}$
$\mathrm{pa}(x)$	Set of parent nodes for node x in a graphical model

$\boldsymbol{\theta} \in \boldsymbol{\Theta}$	Parameters $\boldsymbol{\theta}$ belonging to the parameter space $\boldsymbol{\Theta}$
$\boldsymbol{v}_{\mathrm{w}}$	Word embedding of w
$\boldsymbol{w}$	One-hot encoding of w
$\boldsymbol{x}_i \in \mathcal{Y}$	Sentence $\boldsymbol{x}_i$ belonging to language X
$\boldsymbol{y}_i \in \mathcal{Y}$	Sentence $\boldsymbol{y}_i$ belonging to language Y
$\mathcal{M}$	A model $\mathcal{M}$
$\mathcal{X}$	Set of sentences belonging to language X
$\mathcal{Y}$	Set of sentences belonging to language Y
w	The word as a symbol
$\sigma(x)$	An activation function with respect to x
N	Number of data points
V	Dictionary; set of all words in vocabulary
GD	Gradient Descent
MCMC	Markov Chain Monte Carlo
MLE	Maximum Likelihood Estimation

NLP	Natural Language Processing
NMT	Neural Machine Translation
SGD	Stochastic Gradient Descent
VI	Variational Inference

$\mathrm{Cov}(\boldsymbol{x}, \boldsymbol{y})$	Multivariate covariance with respect to $\boldsymbol{x}$ and $\boldsymbol{y}$
$\mathrm{Cov}(x, y)$	Covariance with respect to x and y
$\mathbb{E}_x[f(x, y)]$	Expectation of a function $f(x, y)$ with respect to a random variable x. If no ambiguity to which variable we are taking expectation with respect to exists the subscript may be left out
$D_{KL}(p, q\|\|)$	Kullback-Leibler divergence from distribution q to p
$\mathcal{D}$	Set of data points
$\mathcal{N}(\boldsymbol{\mu}, \Sigma)$	Normal distribution with mean $\boldsymbol{\mu}$ and covariance matrix Σ
$p(x)$	Probability distribution with respect to random variable x

$\mathrm{Cov}(\boldsymbol{x}, \boldsymbol{y})$	Multivariate covariance with respect to $\boldsymbol{x}$ and $\boldsymbol{y}$
$\mathrm{Cov}(x, y)$	Covariance with respect to x and y
$\mathbb{E}_x[f(x, y)]$	Expectation of a function $f(x, y)$ with respect to a random variable x. If no ambiguity to which variable we are taking expectation with respect to exists the subscript may be left out
$D_{KL}(p, q\|\|)$	Kullback-Leibler divergence from distribution q to p
$\mathcal{D}$	Set of data points
$\mathcal{N}(\boldsymbol{\mu}, \Sigma)$	Normal distribution with mean $\boldsymbol{\mu}$ and covariance matrix Σ
$p(x)$	Probability distribution with respect to random variable x

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 $\mathcal{X}$ to another language $\mathcal{Y}$. Formally, we have a dataset $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^{N}$, such that $\boldsymbol{x}_i \in \mathcal{X}, \boldsymbol{y}_i \in \mathcal{Y}$ represent the same sentence in two different languages. The goal is then to find a mapping $f : \mathcal{X} \to \mathcal{Y}$ 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 $\mathcal{X}$ and $\mathcal{Y}$ 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

$\mathcal{X}$ to $\mathcal{Y}$.

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 $\mathcal{X}$ to $\mathcal{Q}$ without the model having ever seen a sentence pair of the form $(\boldsymbol{x}, \boldsymbol{q}), \boldsymbol{x} \in \mathcal{X}, \boldsymbol{q} \in \mathcal{Q}$.

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

Probability theory is a natural candidate for doing principled reasoning about uncertainty, in that is isomorphic to believes which follow Cox desiderata. This is one of the reasons it has gained such widespread use in machine learning[1, p. 3-23].

2.1.1 Rules and Theorems

(might just want to dump Maneesh's stuff on probability basics, page 2)

Rules of probability that will be useful to us are the following, if we let $X \in \mathcal{X}$ be a random variable and $p(X)$ the measure with respect to this variable, then

Unit volume

$$\int_{\mathcal{X}} p(X)\, \mathrm{d}X = 1 \tag{2.1}$$

Non-negativity

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

the 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 familiarity with random variables and random vectors, difference between continuous and discrete variables.

Most manipulation of statements about random variables can be stated as a consequence of the two following fundamental rules, given two random variables X, Y defined on the support $\mathcal{X}, \mathcal{Y}$ respectively, then

Sum rule

$$p(X) = \int_{\mathcal{Y}} p(X, Y)\,\mathrm{d}Y \tag{2.3}$$

Product rule

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

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

$$\mathbb{E}_X[f] = \int_{\mathcal{X}} f(X)p(X)\,\mathrm{d}X \tag{2.5}$$

Covariance

$$\mathrm{Cov}(X, Y) = \mathbb{E}_{XY}[(X - \mathbb{E}_X[X])(Y - \mathbb{E}_Y[Y])] \tag{2.6}$$

We then define the variance operator as

$$\mathrm{Var}(X) = \mathrm{Cov}(X, X) \tag{2.7}$$

The generalisation from $f : \mathcal{X} \to \mathbb{R}$ to $f : \mathcal{X} \to \mathbb{R}^n$ is defined in the straightforward manner such that 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 $\mathrm{Cov}(\boldsymbol{f})_{i,j} = \mathrm{Cov}(f_i, f_j)$.[2]

2.1.2 The Gaussian Distribution

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

$$\mathcal{N}(\boldsymbol{X}|\boldsymbol{\mu}, \Sigma) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\boldsymbol{X}-\boldsymbol{\mu})^T \Sigma^{-1} (\boldsymbol{X}-\boldsymbol{\mu})\right) \tag{2.8}$$

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 equations (1.5) and (1.6)[].

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 $\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{x}|\boldsymbol{0}_D, \boldsymbol{I}_{D\times D})$. If we let $\boldsymbol{\Lambda}\boldsymbol{\Lambda}^\top = \Sigma$ be the Cholesky decomposition[3, p. 100-102] of Σ, then we also have that

$$\boldsymbol{x} = \boldsymbol{\mu} + \boldsymbol{\Lambda}\boldsymbol{z} \tag{2.9}$$

, 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 $\Sigma = \boldsymbol{\sigma} \odot \boldsymbol{I}_{D\times D}$. 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 $\boldsymbol{x}_i \sim \mathcal{N}(\boldsymbol{x}_i|\boldsymbol{\mu}_i, \Sigma_i)$, then the joint can be found to be Gaussian distributed as

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

, where

$$\Sigma = \left(\sum_i^n \Sigma_i^{-1}\right)^{-1} \tag{2.11}$$

$$\boldsymbol{\mu} = \Sigma\left(\sum_i^n \Sigma^{-1}\boldsymbol{\mu}_i\right) \tag{2.12}$$

[2, p. 78-84].

In the case of two random variables distributed according to the form as laid out in (1.9), $\boldsymbol{x} \sim \mathcal{N}(\mathcal{N}(\boldsymbol{\mu_x}, \boldsymbol{\sigma_x} \odot \boldsymbol{I}))$ and $\boldsymbol{y} \sim \mathcal{N}(\mathcal{N}(\boldsymbol{\mu_y}, \boldsymbol{\sigma_y} \odot \boldsymbol{I}))$ we have that the resulting distribution $p(\boldsymbol{x}, \boldsymbol{y}) = p(\boldsymbol{x})p(\boldsymbol{y})$ is distributed such that

$$p(\boldsymbol{x}, \boldsymbol{y}) = \mathcal{N}(\boldsymbol{\mu_{x,y}}, \boldsymbol{\sigma_{x,y}})$$

where

$$\boldsymbol{\sigma_{x,y}} = \frac{1}{\boldsymbol{\sigma_x}^{-1} + \boldsymbol{\sigma_y}^{-1}} \tag{2.13}$$

$$\boldsymbol{\mu_{x,y}} = \frac{\boldsymbol{\sigma_x}^{-1}\boldsymbol{\mu_x} + \boldsymbol{\sigma_y}^{-1}\boldsymbol{\mu_y}}{\boldsymbol{\sigma_x}^{-1} + \boldsymbol{\sigma_y}^{-1}} \tag{2.14}$$

.

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 $\mathcal{D}$ we want to be able to fit the parameters $\boldsymbol{\theta}$ such that these generalise to unseen data. The MLE of of the parameters of the model is defined to be

$$\hat{\boldsymbol{\theta}}_{ML} = \underset{\boldsymbol{\theta}\in\boldsymbol{\Theta}}{\operatorname{argmax}}\, \mathcal{L}(\boldsymbol{\theta}; \mathcal{D}) \tag{2.15}$$

.

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 common assumption of i.i.d datapoints, the joint distribution becomes a product of individual probabilities for each datapoint,

$$\mathcal{L}(\boldsymbol{\theta}|\mathcal{D}) = p(\boldsymbol{x}_1,\dots,\boldsymbol{x}_n|\boldsymbol{\theta}) = \prod_i^n p(\boldsymbol{x}_i|\boldsymbol{\theta}) \tag{2.16}$$

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

$$\ell(\boldsymbol{\theta}|\mathcal{D}) = \log \mathcal{L}(\boldsymbol{\theta}|\mathcal{D}) = \sum_i^n \log p(\boldsymbol{x}_i) \tag{2.17}$$

. 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].

MLE estimators have a number of nice properties such as consistency and asymptotic normality, however, the optimization problem itself is often non-convex, making it hard to find the actual estimator[4].

2.1.4 Graphical models

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*, fol-

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

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

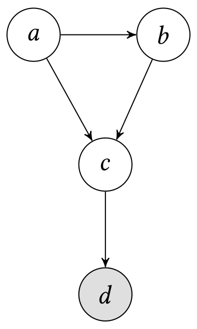
this problem can be reduced even further lowing the notation and nomenclature of graph theory as used in mathematics. While there are many different kinds of graphical models depending on the type of graph structure used (directed graphical models, undirected graphical models, factor graphs, etc.), we 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 $\mathcal{X}$ factors in a conditional manner. In general, the relationship between a given directed graph and the corresponding distribution 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, 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

$$p(\mathcal{X}) = \prod_{k=1}^{K} p(x_k|\mathrm{pa}(x_k)) \tag{2.18}$$

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 (1.18),

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

.

2.1.5 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 $\mathcal{L}$, 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.

The most common setting is in latent variable models, where a latent variable, often denoted by 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[2, 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].

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 z such that

$$p(z, \boldsymbol{x}) = p(z)p(\boldsymbol{x}|z) \tag{2.19}$$

. This also happens to cover the case of our generative model.

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

$$q^*(z) = \operatorname*{argmin}_{q(z) \in Q} KL(q(z||p(z|\boldsymbol{x}))) \tag{2.20}$$

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

2.2 Deep Learning

Until recently the field of NLP were 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 instead using embeddings[12, p. 1 - 2].

What all neural networks have in common is that they are trying to find a functional relationship for the data, with the specific form of the function depending on the task. For NMT this reduces to finding the function $f : \boldsymbol{x} \in \mathcal{X} \to \boldsymbol{y} \in \mathcal{Y}$ such that this f maximizes the likelihood $P(\boldsymbol{x}|\boldsymbol{y})$. Indeed many of the neural networks in existence has been shown to be universal approximators, theoretically being able to simulate a big set of nice functions[13].

2.2.1 Multilayer Perceptron

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 $\boldsymbol{y} \in \mathbb{R}^{d_{out}}$, then an MLP with L layers have the functional form of

$$f(\boldsymbol{x}|\boldsymbol{\theta}) = \sigma_L(\boldsymbol{W}_L \boldsymbol{z}_{L-1} + \boldsymbol{b}_L) \tag{2.21}$$

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

$$\boldsymbol{z}_l = \sigma_l(\boldsymbol{W}_l \boldsymbol{z}_{l-1} + \boldsymbol{b}_l) \tag{2.22}$$

and with the base case

$$\boldsymbol{z}_1 = \sigma_1(\boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1) \tag{2.23}$$

.

$\boldsymbol{W}_l$ and $\boldsymbol{b}_l$ may be of any dimension as long as it is dimensionally consistent with the input and output of the layer 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, $\boldsymbol{\theta} = \{(\boldsymbol{W}_l, \boldsymbol{b}_l)_{l=1}^{L}\}$.

2.2.2 Recurrent Neural Networks

A recurrent neural network acts upon sequences to produce an output. It is a special kind of technique called parameter tying where parameters are shared over many time-steps.

2.2.3 Convolutional Neural Networks

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[14, 15].

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[15].

2.3.1 Language model

We define a sentence to be a vector of words $\boldsymbol{w}_{1:L} = (\mathrm{w}_1, \ldots, \mathrm{w}_l)^\top$ such that each word is an atomic element $\mathrm{w}_i \in V$, where V is the dictionary of words in our language. Then the joint distribution of a word with respect to the underlying probability measure can be rolled out using the probabilistic chain rule which is just repeated application of the original product rule in equation (1.4)

$$P(\boldsymbol{w}_{1:L}) = \prod_{l=1}^{L} P(\mathrm{w}_l | \mathrm{w}_1, \ldots, \mathrm{w}_{l-1}) \tag{2.24}$$

where w_l is the l'th word of the sentence $\boldsymbol{w}_{1:L}$[16].

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, leading to algorithms such as word2vec and Glove[17, 18, 19]. However, these techniques work less well in a neural network setting where instead finding the best embedding jointly with the parameters of the model is peqreferred [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 $\mathrm{w} \in V$, such that 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

$$\boldsymbol{w} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{2.25}$$

[12, p. 6] such that $\boldsymbol{w}_j = \delta_{ij}$.

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

$$\cos_{similarity}(\boldsymbol{v}_1, \boldsymbol{v}_2) = \delta_{\mathrm{v}_1\mathrm{v}_2} \tag{2.26}$$

. 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 $\mathbb{R}^m$. Express the joint probability function (1.24) of word sequences of a sentence in terms of the feature vectors of these words in the sequence and simultaneously learn the word feature vectors and the parameters of the model which dictates the form of the probability function, $\boldsymbol{\theta}$. After this is done words which share similarities in some sense such as `Dog, Puppy` would have a higher similarity score than unrelated concepts such as `Dog, Bulwark`[16].

We may represent this in a mathematical form by trying to find a linear map C from any element $\mathrm{w} \in V$ such that $C(\mathbf{w}) \in \mathbb{R}^m$. Using the usual canonical basis of an 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 $\boldsymbol{C}\boldsymbol{w}$.

2.3.3 Neural machine translation

For a long time the dominant paradigm within machine translation was to use phrase based machine translation systems[20, 21], however since a couple of year back, modelling the word or character level directly with neural networks, so called NMT has become the best performing method[22, 23].

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[24].

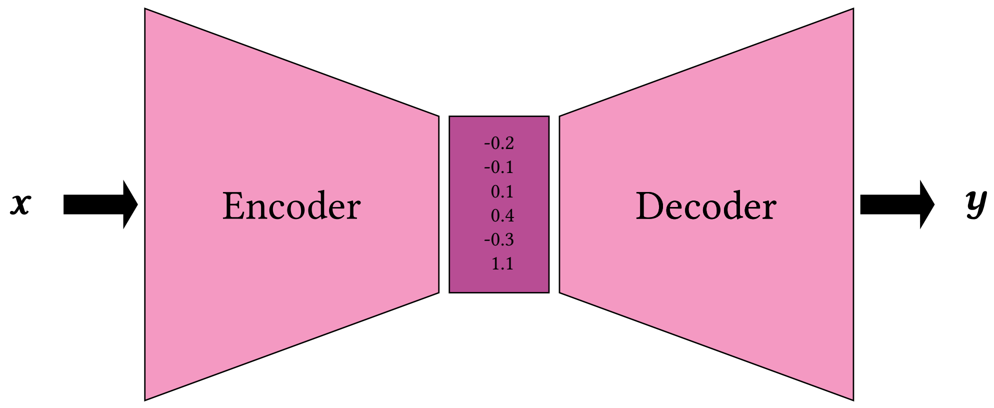


Figure 2.1: Encoder decoder schematic

2.4 Optimization

2.4.1 Problem formulation

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[25].

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[26].

2.4.2 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 can in the most general case be recast as an effort to find the parameters $\boldsymbol{\theta}_{ML}$ that maximizes the log-likelihood function (1.17).

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 $\mathcal{D} = \{z_i\}_{i=1}^{n}$ such that $\boldsymbol{z_i}$ is any general datapoint, Q an objective function that we try to maximize, such that $Q : \Theta \times \mathcal{Z} \to \mathbb{R}$, then the update using GD looks as follows

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \gamma_t \frac{1}{n} \sum_{i=1}^{n} \nabla_{\boldsymbol{\theta}} Q(z_i, \boldsymbol{\theta}_t) \tag{2.27}$$

. 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 V of size

m then SGD does the following update

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \gamma_t \frac{1}{m} \sum_{i \in I_t} \nabla_{\boldsymbol{\theta}} Q(z_i, \boldsymbol{\theta}_t) \tag{2.28}$$

[27][2, 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 [28, 29].

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[30] and AdaGrad[31].

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.

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[32].

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

Chapter 3

Methods and Theory

3.1 ELBO

Ideally we would like to optimize the log-likelihood of the data, $\ell(\boldsymbol{\theta};\boldsymbol{\theta})$. For many models, it is not possible to evaluate this quantity directly due to it being computationally and/or analytically intractable.

Instead we make the following observation

$$\log p_\theta(\mathcal{D}) = \sum_{i=1}^{N} \log p_\theta(\boldsymbol{x}_i) \tag{3.1}$$

and looking at each term in the sum we can see that we may rewrite this as (Let $\boldsymbol{x}$ be known from context)

$$\begin{aligned}
\log p_\theta(\boldsymbol{x}) &= \log \int_{\mathcal{Z}} p_\theta(z, \boldsymbol{x}) \\
&= \log \int_{\mathcal{Z}} q_\phi(z|\boldsymbol{x}) \frac{p_\theta(\boldsymbol{x}, z)}{q_\phi(z|\boldsymbol{x})} \\
&= \log \mathbb{E}_{q_\phi(z|\boldsymbol{x})[\frac{p_\theta(x,z)}{q_\phi(z|x)}]} \\
&\geq \mathbb{E}_{q_\phi(z|\boldsymbol{x})}[\log \frac{p_\theta(\boldsymbol{x}, z)}{q_\phi(z|\boldsymbol{x})}] \\
&= \mathbb{E}_{q_\phi(z|\boldsymbol{x})}[\frac{p_\theta(\boldsymbol{x}|z) p_\theta(z)}{q_{\phi(z|\boldsymbol{x})}}] \\
&= \mathbb{E}_{q_\phi(z|\boldsymbol{x})}[\log p_\theta(\boldsymbol{x}|z)] - D_{KL}(q_\phi(z|\boldsymbol{x})||p_\theta(z))
\end{aligned}$$

Which leads to the following observation, if we call the lower bound $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x})$, that

$$\log p_{\boldsymbol{\theta}}(\boldsymbol{x}) \geq \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(z|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|z)] - D_{KL}(q_{\boldsymbol{\phi}}(z|\boldsymbol{x})||p_{\boldsymbol{\theta}}(z)) \tag{3.2}$$

.

In general this lower bound is still not tractable, however appealing to the LLN we can sample $\boldsymbol{z}^l$ and use the reparametrisation trick to approximate the lower bound by

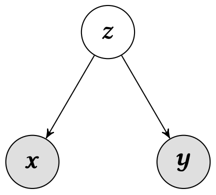
$$\tilde{\mathcal{L}}^A(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}) = \frac{1}{L}\sum_{l=1}^{L} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}, z) - \log q_{\boldsymbol{\phi}}(z|\boldsymbol{x}) \tag{3.3}$$

However, if we use good priors and likelihoods, like categorical and/or normals, we can evaluate the KL analytically and we don't need to sample, giving use the form

$$\tilde{\mathcal{L}}^A(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}) = (\frac{1}{L}\sum_{l=1}^{L} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}|z)) - D_{KL}(q_{\boldsymbol{\phi}}(z|\boldsymbol{x})||p_{\boldsymbol{\theta}}(z)) \tag{3.4}$$

3.1.1 Translation case

In our case we have due to the graphical model



where $\boldsymbol{x}$ is a sentence in language $\mathcal{X}$ and $\boldsymbol{y}$ is a sentence in language $\mathcal{Y}$. From the graphical model we see that the joint distribution factorises such that

$$p(z, x, y) = p(z)p(x|z)p(y|z) = N(0, I)N(\mu_x(z), \sigma_x(z))N(\mu_y(z), \sigma_y(z)) \tag{3.5}$$

where the normal quantities are parametrised by neural networks.

The recognition model is parametrised by the probability distribution $q_\phi(z|x, y)$. We are free to choose this however we like, but the closer this is to the actual conditional distribution over the latent, $p(z|x, y)$ the tighter the ELBO inequality $\ell(x, y) \geq \mathcal{L}(x, y)$ will be. We let the distribution all be gaussians to have analytical tractability

$$\begin{aligned} q(z|x, y) &= q(z|x)q(z|y) \\ &= N(\mu(x), \sigma(x)I)N(\mu(y), \sigma(y)I) \end{aligned}$$

where both normals defined by the output of neural networks.

Since the product of Guassian pdf's are gaussian itself, we have that may write that

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

. Pulling all of these statements together, we get that the SGVB of the translation case reduces to the equation

$$\tilde{\mathcal{L}}^A(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}) = (\frac{1}{L}\sum_{l=1}^{L} \log(\mathcal{N}(\boldsymbol{x}|\mu_{\theta_y}(\boldsymbol{z}), \sigma_{\theta_y}(\boldsymbol{z}))) + \log(\mathcal{N}(\boldsymbol{y}|\mu_{\theta_y}(\boldsymbol{z}), \sigma_{\theta_y}(\boldsymbol{z})))) + \frac{1}{2}\sum_{j=1}^{J}(1 + \log((\sigma_j^{(i)})^2) - (\mu_j^{(i)} \tag{3.6}$$

where the μ, σ is from the normal $\mathcal{N}(\boldsymbol{z}|\mu(\boldsymbol{x}, \boldsymbol{y}), \sigma(\boldsymbol{x}, \boldsymbol{y}))$.

Bibliography

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

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

[3] 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.

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

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

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

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

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

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

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

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

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