DEGREE PROJECT IN THE FIELD OF TECHNOLOGY
ENGINEERING PHYSICS
AND THE MAIN FIELD OF STUDY
COMPUTER SCIENCE AND ENGINEERING,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2018

Cooperative versus Adversarial Learning: Generating Political Text

JACOB JONSSON

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Cooperative versus Adversarial Learning: Generating Political Text

DA233X Project Report

JACOB JONSSON

Master in Machine Learning
Date: December 23, 2018
Supervisor: Johan Boye
Examiner: Olov Engwall
School of Computer Science and Communication

Abstract

This thesis aims to evaluate the current state of the art for unconditional text generation and compare established models with novel approaches in the task of generating texts, after being trained on texts written by political parties from the Swedish Riksdag. First, the progression of language modeling from n-gram models and statistical models to neural network models is presented. This is followed by theoretical arguments for the development of adversarial training methods, where a generator neural network tries to fool a discriminator network, trained to distinguish between real and generated sentences. One of the methods in the research frontier diverges from the adversarial idea and instead uses cooperative training, where a mediator network is trained instead of a discriminator. The mediator is then used to estimate a symmetric divergence measure between the true distribution and the generator's distribution, which is to be minimized in training. A set of experiments evaluates the performance of cooperative training and adversarial training, and finds that they both have advantages and disadvantages. In the experiments, the adversarial training increases the quality of generated texts, while the cooperative training increases the diversity. The findings are in line with the theoretical expectation.

Sammanfattning

Denna uppsats utvärderar några nyligen föreslagna metoder för obetingad textgenerering, baserade på s.k. “Generative Adversarial Networks” (GANs). Den jämför etablerade modeller med nya metoder för att generera text, efter att ha tränats på texter från de svenska Riksdagspartierna. Utvecklingen av språkmodellering från n-grammodeller och statistiska modeller till modeller av neurala nätverk presenteras. Detta följs upp av teoretiska argument för utvecklingen av GANs, för vilka ett generatornätverk försöker överlista ett diskriminatornätverk, som tränas skilja mellan riktiga och genererade meningar. En av de senaste metoderna avviker från detta angreppssätt och introducerar istället kooperativ träning, där ett mediatornätverk tränas istället för en diskriminator. Mediatorn används sedan till att uppskatta ett symmetriskt divergensmått mellan den sanna distributionen och generatorns distribution, vilket träningen syftar till att minimera. En serie experiment utvärderar hur GANs och kooperativ träning presterar i förhållande till varandra, och finner att de båda har för- och nackdelar. I experimenten ökar GANs kvaliteten på texterna som genereras, medan kooperativ träning ökar mångfalden. Resultaten motsvarar vad som kan förväntas teoretiskt.

Acknowledgements

This thesis marks the end of my studies at KTH, and there are many people I'd like to express my gratitude to.

I would like to thank my supervisor Johan Boye for valuable feedback, and for introducing me to the field of natural language processing in his language engineering classes. I'd also like to thank the examiner Olov Engwall for his feedback.

Jakob Weisinger did a great written opposition which improved the overall quality substantially.

Mikael Öhman at C3SE is acknowledged for assistance concerning technical and implementation aspects in making the code run on the C3SE resources.

I would also like to thank Ellen Jonsson who helped me by creating figures.

I am thankful to my friends who had a direct impact on the thesis: Adam, Adam, Baltzar, Olov, Oscar and Patric.

I also owe a lot to the rest of the "fellowship" at KTH: Axel, Fredrik, Harald and Marcus. Your friendship and support has meant a lot to me.

Finally, I want to thank my girlfriend Laurine and my family for always being there for me.

Contents

Chapter 1

Introduction

Machine intelligence takes many shapes, and no definition has been universally accepted. An appealing idea was formed in 1950 by Alan Turing in what he calls *The Imitation Game* [1]. The proposed idea is that one can form an opinion of a machine's cognitive ability by whether it can behave indistinguishable from a human. A machine is said to be able to pass the Turing Test (as it has later been called) if a human interacts with a machine without being able to tell whether the other agent is human or not.

The Turing test is not a fitting universal definition of intelligence, for two reasons above others. Firstly, not all human behavior is intelligent. Secondly, there are intelligent actions which aren't human. Despite this, it provides an intuitive and empirical approach to the field of natural language processing (NLP). There are many goals of NLP, but they all revolve around tasks that involve human languages: human-machine communication, translation tools, or text and speech analysis, to name a few. Being able to understand and express natural language is a key part of machine intelligence.

1.1 Objective

In this thesis, the current state of the art for text generating techniques is evaluated. Starting with a historical progression in the field, the breakthroughs will be highlighted to explain the direction of the progress towards neural network models and specifically adversarial training of neural models. This will be compared to a novel approach:

using a cooperative training system of neural networks, rather than adversaries.

This thesis will present the differences between established and novel techniques, and evaluate whether the models can be used to generate political sentences. To do so, the models are trained on a dataset consisting of sentences from the programs and manifestos of the eight political parties in the Swedish Riksdag. When evaluating language models, there are many aspects to take into account, and there is not yet any definite standard metric. The thesis will also look at how generative language models can be efficiently and fairly compared.

1.2 Research Question

The thesis aims to answer the following questions:

- What is the current state of the art for generative language models?
- How does adversarial training compare to cooperative training?
- Which metrics are most suitable to evaluate language models?
- To what extent can generative neural models be used to create political sentences?

1.3 Scope

Only word-level language models will be evaluated in this study, and character-level models will not be discussed or evaluated. Character-level language models require significantly longer dependencies and typically have lower probabilities of generating the true texts [2].

1.4 Ethics of Natural Language Generation

Any technological advancements should be accompanied by an evaluation of the ethical outlook for the results as well as the procedure. For a general overview of existential risks with Artificial Intelligence (AI),

the Future of Life Institute[1] offers a good starting point. In this section, I will keep the discussion limited to generative natural language models, in a foreseeable future.

The following issues should not be seen as an exhaustive risk, but an illustration of the diversity of the potential issues.

1.4.1 Copyright Infringements

The purpose of copyright is to credit the creator of art, literature and science. While the samples created by the models in this study likely won't make it to any bookshelves, it is easy to imagine very high quality texts being generated by an intelligent model design in combination with a rich source material and computational power. An initial question is who should get credited for the work; should the model trainers be recognized as the authors even though they didn't choose a single word of the text, or can the machine get credited without being human? Who would then receive the earnings?

Generative models, such as GANs, are trying to match an underlying distribution to the extent that the training data allows it. What happens to copyright if a distribution is perfectly matched? It's hard to argue for the right to your own word distributions, but the US copyright law[2] states the following:

> The copyright law requires that for a work to be protected by copyright, it must be original and creative. This means that the work must have been developed independently by its author, and should have some creativity involved in the creation.

A generative model thus would not have the legal rights to a book or poem which was a rewritten version of an existing text, just like a human translation of a book would not be legal to publish. An objection might be that a model would never find the target distribution that perfectly, but imagine a model which simply learns the training data and samples directly from it, and the problem arises.

[1] https://futureoflife.org/background/benefits-risks-of-artificial-intelligence/

[2] https://copyright.uslegal.com/enumerated-categories-of-copyrightable-works/creativity-requirement/

1.4.2 Automation

One of the most common ethical discussions on machine learning and AI is automation, and how it contributes to the loss of human jobs. When it comes to text generation, jobs that could be affected are e.g. copywriters, translators and transcribers. In the 2017 ACL Workshop of Ethics in NLP, a general outlook on the issues and an action plan was presented by Leidner and Plachouras [3]. One of the issues they describe is automation. The first counterargument is that menial jobs are done by machines so that humans can do more interesting and intellectually demanding jobs. They argue that not all people possess the qualifications or intellect for such a transition, or that doing so might induce stress for some people. The second counterargument is that humans would be relieved of work altogether. However, many people self-identify with their profession and feel that working gives them structure and a purpose.

Chapter 2

Background

2.1 Natural Language Processing

Natural language processing is a term used to describe the field of machine learning addressing the interaction between computers and human languages. While computers are superior to humans in many tasks, such as remembering facts or performing numeric calculations, our natural language usage is unrivaled. There are several reasons why teaching language to computers is challenging [4]. Our languages are ambiguous. Consider for example the difference in meaning between the sentence "I eat pasta with my brother" compared to the sentence "I eat pasta with chicken". Moreover, languages are highly variable, meaning that we can express the same meaning by constructing completely different sentences. Further, humans are great at using languages, but worse at understanding and expressing why certain sentences are valid and others seem absurd.

Apart from being ambiguous, variable, and hard to express in strict rules, natural language processing is difficult for computational reasons. Languages are discrete, meaning that words can't be related to each other simply by the characters they are made up of. Instead, their meaning has to be inferred from their surrounding context. Languages are also compositional. As we form sentences by combining words, a sequence of words can carry more meaning than the words themselves. The fact that languages are discrete and compositional leads to data sparsity. Every day, we form sentences we have never heard before without hesitation. For these reasons, we must teach the com-

puters how to interpret and express languages in ways which suit their strengths.

2.1.1 Language Modeling

A human regards language as natural unconsciously, if the text they read or the words they hear sound plausible, in the context they are read or heard. Language models are models which try to calculate the probability of a sentence, i.e. the probability of a sequence of words to appear in the order they do in the evaluated sentence. Formally, given a sequence of words $w_{1:N}$, a language model tries to estimate $P(w_{1:N})$ using the chain rule of probability:

$$P(w_{1:N}) = \prod_{i=1}^{N} P(w_i|w_{1:i-1}).$$

The conditional probabilities are then estimated by evaluating a set of training data and counting the number of appearances each combination has. As a sentence becomes longer, the number of words grows and the conditional probability approaches zero. To avoid zero-probabilities and to make the estimation more tractable, a common assumption to make is that the probability of each word is only depending on the *k* words which precede it. This is known as a Markov assumption.

The task of calculating the probability of a given sentence is equivalent to the task of calculating the most probable upcoming word, given a history of words. This follows since if we are able to calculate the probability of any sequence, we can also calculate the probability of a given word conditional to the sequence that precedes it as

$$P(w_i|w_{1:i-1}) = \frac{P(w_{1:i})}{P(w_{1:i-1})}.$$

This means that languages models can also be used for generative purposes.

The probabilities of words and sequences are typically estimated by counting how many times a word or a sequence appear in a corpus, i.e. a collection of texts easily readable to a computer. The counts are normalized by the total number of words, to lie between 0 and 1. This estimation method is called maximum likelihood estimation (MLE).

To evaluate language models, it is common to evaluate a model's perplexity [5]. The perplexity measures a variant of the probability of generating a sequence. The perplexity of a sequence $w_{1:N}$ is defined as

$$\text{PP}(w_{1:N}) = P(w_{1:N})^{-\frac{1}{N}} = \sqrt[N]{\frac{1}{P(w_{1:N})}}.$$

Using the chain rule, we can expand this to

$$\text{PP}(w_{1:N}) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{1:i-1})}}.$$

2.1.2 N-gram Models

A set of models utilizing the Markov assumption is the n-gram models, where an n-gram represents a set of n contiguous words. This reduces the sequence probability to the following form:

$$P(w_{1:N}) = \prod_{i=1}^{N} P(w_i|w_{i-n:i-1}).$$

The related perplexity would thus be

$$\text{PP}(w_{1:N}) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-n:i-1})}}.$$

The conditional probability of an n-gram $w_{i-n:i}$ is estimated by counting all appearances of the n-gram in the training set, and dividing by all n-grams starting with the *n-1*-gram $w_{i-n:i-1}$.

Ultimately, the Markov assumption only gives an estimation of the true state. In practice, a test set might very well contain n-gram occurrences that never appear in the training data, which is a symptom of the curse of dimensionality. Assigning a zero probability to this sentence would be detrimental to the model. For that reason, there are techniques such as smoothing by assigning a small base probability to every possible n-gram.

2.1.3 Development of Statistical Language Modeling

The statistical language modeling approach has not always been praised. Noam Chomsky published several influential papers in the 1950s and 1960s arguing that the finite-state Markov processes, such as n-grams, can never fully model the complexity of grammar in natural languages [5]. Following this, many linguists and computational linguists decided to explore other methods. With increasing computational power and text data available in extreme quantities, n-gram models gained popularity again in the 80s.

N-gram models, in all their simplicity, have been successful in the past, and Goodman [6] showed in 2001 that the advantages of sophisticated models vanished compared to n-gram models using smoothing and a cache memory, if they are given enough training data.

While the n-gram models in theory can improve indefinitely by increasing the amount of training data and computational power, the Markov assumption means that they will never be able to truly capture long term information dependencies. Further, as the n-gram order increases, the number of parameters needed to describe a model increases exponentially while the model needs even more training data, which is why models larger than 5-gram models are typically not used [5][4].

To conquer the sparsity that arises when modeling natural languages, Bengio et al. [7] suggested in 2003 a neural network approach based on distributed word representations, with remarkable results.

2.2 Neural Networks

A neural network is a mathematical model which consists of small computing units taking an input signal, performing a mathematical operation on them and passing the result as an output signal. The simplest neural network, also known as a perceptron, operating on an input vector $\mathbf{x} \in \mathbb{R}^{d_{in}}$, is simply a linear model:

$$\Sigma_{perceptron}(\mathbf{x}) = \mathbf{x}\mathbf{W} + \mathbf{b}.$$

$\mathbf{W} \in \mathbb{R}^{d_{in} \times d_{out}}$ is called a weight matrix and $\mathbf{b} \in \mathbb{R}^{d_{out}}$ is known as a bias term. Since this system is linear, it can not represent non-linear functions. The output of this system, or layer, can be fed into

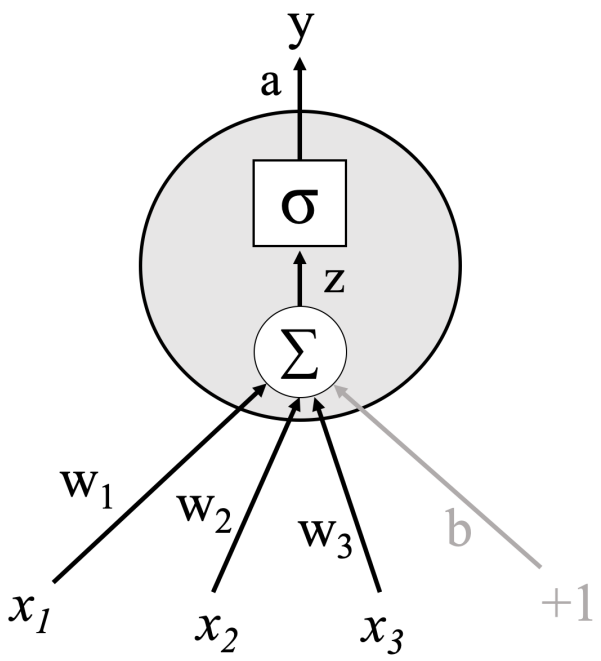

Figure 2.1: Visual representation of a single-layer neural network with inputs x_i and weights w_i which are multiplied and passed through a non-linear function σ to create the output y. Figure by Ellen Jonsson, based on [5].

another layer activated by a non-linear function σ. In the same fashion, a linear layer can be designed by combining different layers of linear equations whose input from previous layers are activated by non-linear functions. Layers which are in between the input layer and the output layer are called hidden layers. In 1989, it was proved that a neural network with one hidden layer is a universal approximator - it can approximate all continuous functions to any desired non-zero error degree [4]. A visualization of a single-layer neural network is shown in Figure 2.1.

Being a universal approximator doesn't tell anything about how to reach good approximations or the effort it takes, but merely that such a solution exists in theory. Nevertheless, the findings spurred a great development in the science of neural networks. Along with radical improvements in computer hardware, optimization techniques and training methods, front figures such as Yann LeCun, Geoffrey Hinton and Yoshua Bengio developed efficient methods of training deeper

and more sophisticated neural networks, proving ground-breaking results across different fields of machine learning such as image and speech recognition [5]. The use of deeper networks is what came to be known as deep learning.

2.2.1 Training a Neural Network

Knowing that a neural network can approximate any continuous function arbitrarily well, the actual performance of a neural network comes down to how well it is trained to assign its parameters. To train a neural network, a loss function L is designed to model the distance between the networks output and the target. The training then consists of minimizing the loss function through gradient-based optimization. The loss function is not necessarily convex, and a solution found is thus not guaranteed to be a global solution [4]. Despite this, gradient-based optimization produces good results in practice.

The gradient informs in which direction each parameter should be changed to approach the local optimum. Typically, the parameter update also includes a regularization parameter λ to avoid overfitting. Denoting the set of all weight and bias parameters $\mathbf{\Theta}$, the parameter update can then be described as

$$\begin{aligned}\mathbf{\Theta}^* &= \arg\min_{\mathbf{\Theta}} \mathbb{L}(\mathbf{\Theta}) + \lambda * \Phi(\mathbf{\Theta}) \\ &= \arg\min_{\mathbf{\Theta}} \frac{1}{n} \sum_{i=1}^{n} L(y^{(i)}, \hat{y}^{(i)}) + \lambda * \Phi(\mathbf{\Theta}) \\ &= \arg\min_{\mathbf{\Theta}} \frac{1}{n} \sum_{i=1}^{n} L(y^{(i)}, f(\mathbf{x}^{(i)}, \mathbf{\Theta})) + \lambda * \Phi(\mathbf{\Theta}).\end{aligned}$$

where f denotes a parametrized function which infers a sample $\hat{y}^{(i)}$ given the input vector and the parameters to be decided, and Φ is a regularization function penalizing large terms.

The gradient calculation is the key in training a neural network. In essence, the gradient equations are the same as in any linear system, and can be found by direct computation using the chain rule of differentiation. However, since a neural network might have millions of differential parameters, computing the derivatives directly is not feasible. In 1986, Rumelhart, Hinton, and Williams [8] proposed a method called backpropagation to overcome this. The method caches intermediary results in gradient calculations so that each gradient can be

calculated relative to the gradients in the layer after. The weights and biases of the network are then updated according to the gradients.

To this day, no other method has taken the place of backpropagation when it comes to calculating the gradients in a neural network. Different variations of training instead typically consist of defining the loss function in a way that suits the purpose, along with choosing an optimization function to efficiently update the network parameters based on the gradients.

2.2.2 Recurrent Neural Networks

In many applications for neural networks, the data is of a sequential nature. This is the case when working with languages; sentences are sequences of words and words are sequences of letters. A neural network as described previously can handle sequences by accepting a fixed-size window of tokens as input to the network, which slides across the full sequence one stride at a time. This is in itself a Markov assumption since the network learns only on the number of previous items the window-size defines. As such, anything outside of the context window is forgotten.

In 1990, Elman [9] proposed to add a cyclical connection to neural networks to combat this, in network models called Recurrent Neural Networks (RNNs). Doing so would give the network a dynamic memory by feeding back hidden unit patterns to themselves. In that way, time is represented in the order tokens are processed rather than as a separate spatial dimension, since the later demands a fixed size of patterns as well as an upper limit of their size. Further, treating time as a spatial dimension complicates distinguishing absolute temporal position from relative temporal position.

An RNN can be defined recursively. A function R computes the current state $\mathbf{s_i}$, being fed the previous state $\mathbf{s_{i-1}}$ and an input vector $\mathbf{x_i}$. The state vector is then mapped to an output vector $\mathbf{y_i}$ by a deterministic function O [4]. Since this is recursively defined, there must be an initial state $\mathbf{s_0}$. This is a hyper-parameter input to the RNN, which is commonly assumed to be the zero vector.

$$\begin{aligned} \text{RNN*}(\mathbf{x_{1:n}}; \mathbf{s_o}) &= \mathbf{y_{1:n}} \\ \mathbf{y_i} &= O(\mathbf{s_i}) \\ \mathbf{s_i} &= R(\mathbf{s_{i-1}}, \mathbf{x_i}) \end{aligned}$$

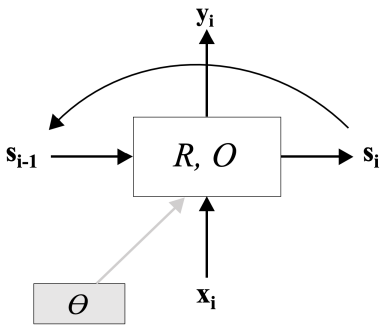

Figure 2.2: Abstract representation of an RNN. Figure by Ellen Jonsson, based on [4].

$$\mathbf{x_i} \in \mathbb{R}^{d_{in}}, \quad \mathbf{y_i} \in \mathbb{R}^{d_{out}}, \quad \mathbf{s_i} \in \mathbb{R}^{f(d_{out})}$$

This abstracted representation can be seen visually in Figure 2.2. The functions R and O are the same across the sequence positions, but can be instantiated in different ways, resulting in different architectures. A popular architecture for language modeling is the Long Short-Term Memory (LSTM) [10]. The LSTM is a so-called gated architecture which lets the system access the full memory in an efficient way by controlling access to certain parts of the state vector.

2.2.3 Teacher Forcing

Training an RNN is not as straight-forward as training a feed-forward network, since the network's size, if unfolded, would grow with one layer at every time-step, which is known as backpropagation through time. This procedure is general, but computationally expensive [11]. Another general method which is less expensive and more stable is called teacher forcing. The idea of teacher forcing is to use the ground truth as the input for the next time-step, rather than using the model output.

Williams and Zipser [11] defined the basic algorithm for teacher forcing by defining an error term between the ground truth and the generated output at every time-step of a training sequence, rather than after the full sequence. The network tries to match certain values at specific time-steps, a temporal supervised learning task, to minimize the total error which is the sum of all time-specific errors. To do this, the parameter update follows the gradient along the total error.

Note that this differs from the general approach to training a neural network because it is not a true gradient-following method, since the true gradient of the full sequence would change when one updates the parameters in each time-step. Williams and Zipser [11] argue that this issue is not severe as long as the weight changes much slower than the time scale of the network operation, meaning that the learning rate has to be sufficiently small. In that scenario, the algorithm is nearly identical to a true gradient-following.

Teacher forcing is a maximum likelihood method in essence, since it updates the model parameters to maximize the likelihood of generating the training data.

2.3 Generative Models

A generative model is a learned representation which is used to generate data in one form or another. Generative modeling is an unsupervised machine learning branch which learns from samples drawn from a distribution p_{data}, and tries to represent an estimate p_G of the distribution. This can be learned either explicitly, or implicitly, meaning that there is no actual distribution to analyze but the model may draw samples from it. Before going into different types of generative models, we should motivate the study of generative models. Goodfellow [12] lists the major reasons to include:

- Training and sampling from generative models is a good way to test our performance in representing high-dimensionality probability distributions, which are important objects in a variety of engineering domains,
- Generative models can be used in reinforcement learning to simulate possible futures,
- Generative models can be trained with missing data, inferring missing properties, and can be used e.g. in semi-supervised learning, where labels are missing from the training data set,
- Many other models work by minimizing an error, giving only one solution from a training sequence. Generative models might instead work with multi-modal outputs, when several answers might be correct.

In the NLP domain, generative models have many use cases. Translation tools need a generative model to sample the intended output. Wu et al. [13] drastically improved Google Translate in 2016 by switching to a neural generative model, yielding 60% fewer translation errors compared to their phrase-based production system. Other applications are e.g. generating dialogue for chat bots or virtual assistants, or automatically captioning images.

2.3.1 Explicit versus Implicit Density Models

The goal of a generative model is to estimate the distribution p_G from training data. This can be done either explicitly or implicitly.

An explicit density model is a model G which specifies a function describing the distribution of an observed random variable, with a set of parameters [14]. Alternatively, there are implicit density models which do not specify an actual distribution. Instead, the implicit density models define stochastic processes (which might also include parametrized models) which directly generate data.

To compare different approaches, it is important to understand the motivation of using one or another.

Not every generative model are based on maximum likelihood, but most are. In fact, most models can be made to use maximum likelihood [12]. For that reason, when comparing major differences between generative models, only maximum likelihood-based models will be compared to eliminate distracting differences. Instead, the major differences between the models is how they compute the likelihood and the gradients, and whether they are exact or approximated.

The models may be divided into nodes of a taxonomy tree, as seen in Figure 2.3. On the left branch of the tree are the explicit density models. For these models, the likelihood maximization is straightforward. The model's generated distribution is plugged in to the likelihood function, and the gradient is followed. However, maintaining computational tractability is hard whilst capturing all the complexity of the data. The computations can't be parallelized in general. Even with a strategical model design, using an explicit model to generate sound might take several minutes for a second of generated sound [12]. For this reason, these models cannot currently work in e.g. dialogue systems, where the user expects an answer in real time.

Another approach creating an explicit density is to approximate the

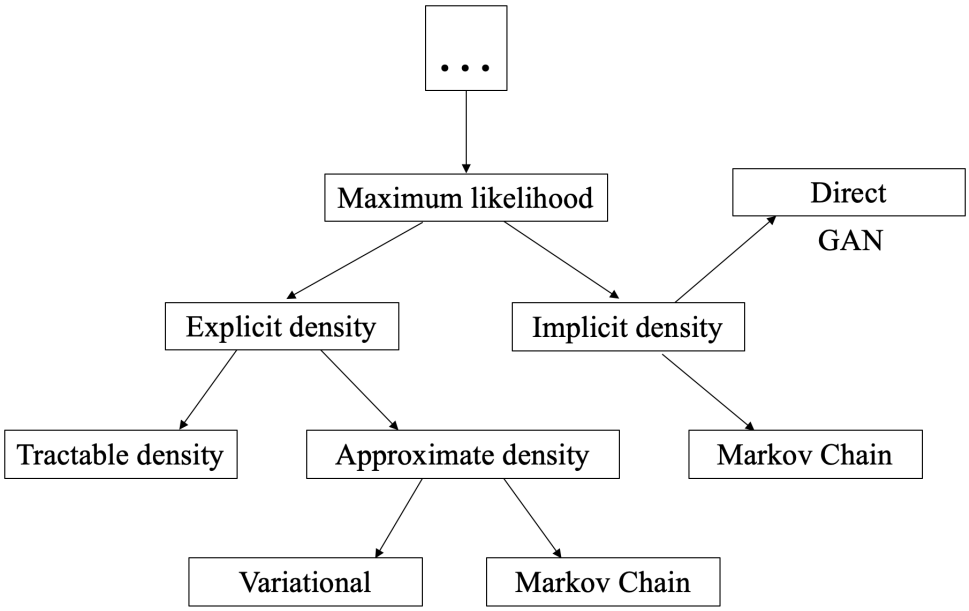

Figure 2.3: Taxonomic tree of generative models. Figure by Ellen Jonsson, based on [12].

density. This can be done either in a deterministic form, using variational methods, or a stochastic form, using Markov chain Monte Carlo methods. Under this category falls the popular variational autoencoder, which in practice tends to produce high likelihood densities. The main issue with variational approaches is that if a too weak posterior or prior distribution is used, the gap between the lower bound and the true likelihood might be too large. Further, lacking a formal quantitative measurement of quality, they are subjectively regarded as producing low quality samples [12]. Markov chain approximations, on the other hand, suffer mainly from expensive calculations. The computations of Markov chains have not scaled to the extent that many deep learning applications demand. Further, even if they were improved, the computational costs will far exceed those of methods sampling in a single step.

Markov chain approximations can be used for implicit density approaches too. In 2014, Bengio et al. [15] presented a framework called Generative Stochastic Networks (GSNs) which define a Markov chain transition operator whose stationary distribution estimates the training data distribution, which can be learned by backpropagation. The framework still suffers from the trouble of scaling Markov chains to

higher dimensional spaces, while the transition operator must be ran several times to generate a single sample. This makes computations expensive.

To tackle the issues with non-parallel computations, gaps between lower variational bounds and true likelihood, expense of Markov chain computations and low quality samples, a novel generative approach was designed.

2.3.2 Generative Adversarial Networks

Generative Adversarial Networks (GANs) are systems of two competing neural networks. One network, known as the generator, tries to generate samples from a distribution p_G resembling samples from the training data distribution p_{data}. The other network, known as the discriminator, simultaneously attempts to learn to distinguish true samples from generated ones. The hypothesis is that as the game between the two players reaches an equilibrium, the generator is able to create realistic samples.

The discriminator is defined by a function D, taking the observed variables $\mathbf{x} \sim p_{\text{data}}$ as input, and the parameters $\boldsymbol{\Theta}^{(\mathbf{D})}$. D outputs a single scalar, representing the probability that $\mathbf{x}$ came from that data. The generator is defined by a function G, but the input consists of the latent variable $\mathbf{z} \sim p_{\mathbf{z}}$. $\mathbf{z}$ is inferred from the input data with added noise, to make the model more robust, and $p_{\mathbf{z}}$ defines a prior probability distribution [12]. G is parametrized by $\boldsymbol{\Theta}^{(\mathbf{G})}$. The networks are trained, as any network, by minimizing a loss function. $J^{(D)}$ and $J^{(G)}$ are both functions of $\boldsymbol{\Theta}^{(\mathbf{D})}$ and $\boldsymbol{\Theta}^{(\mathbf{G})}$, but $J^{(D)}$ can only be minimized by controlling $\boldsymbol{\Theta}^{(\mathbf{D})}$, and vice versa for $J^{(G)}$.

Since the losses depend on the other player's parameters, which cannot be controlled, this scenario is not an optimization problem but a game. This is the motivation for calling the networks "adversarial". The optimization of a loss is to find a local minimum, while the end state of a game is an equilibrium. This game is in equilibrium when $J^{(D)}$ is in a local minimum with respect do $\boldsymbol{\Theta}^{(\mathbf{D})}$ and $J^{(G)}$ is in a local minimum with respect do $\boldsymbol{\Theta}^{(\mathbf{G})}$.

The discriminator network tries to maximize the probability of correctly identifying both samples drawn from the true distribution and samples drawn from the generator. They typically always utilize the

same loss function [16]:

$$J^{(D)}(\boldsymbol{\Theta}^{(\mathbf{D})}, \boldsymbol{\Theta}^{(\mathbf{G})}) = -\frac{1}{2}\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}} \log D(\mathbf{x}, \boldsymbol{\Theta}^{(\mathbf{D})}) - \frac{1}{2}\mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}} \log(1 - D(G(\mathbf{z}, \boldsymbol{\Theta}^{(\mathbf{G})}), \boldsymbol{\Theta}^{(\mathbf{D})})).$$

This is the cross entropy loss where one batch of data is sampled from the true data distribution and the other batch is sampled from the generator. Goodfellow [12] proves that by training the discriminator to minimize this function, the model estimates a ratio

$$\frac{p_{\text{data}}(\mathbf{x})}{p_{\text{G}}(\mathbf{x})},$$

which is used to compute divergences and their gradients.

The loss function for the generator offers more flexibility depending on the purpose of the application. In theory, the simplest design of the game is a zero-sum game, meaning that the sum of all losses are always zero [16]. In other words, $J^{(G)} = -J^{(D)}$, making the solution to finding the optimal parameters a so-called minimax game, maximizing an outer loop after minimizing an inner loop as

$$\begin{aligned}\boldsymbol{\Theta}^{(\mathbf{G})*} &= \arg\min_{\boldsymbol{\Theta}^{(\mathbf{G})}} \max_{\boldsymbol{\Theta}^{(\mathbf{D})}} V(\Theta^{(D)}, \Theta^{(G)}) \\ &= \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}} \log D(\mathbf{x}) + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}} \log(1 - D(G(\mathbf{z}))).\end{aligned}$$

While this loss seems intuitive, there is no theoretical guarantee of its success. Goodfellow et al. [16] proved that updating a distribution p_G according to the criterion

$$\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}} \log D(\mathbf{x}) + \mathbb{E}_{\mathbf{x}\sim p_G} \log(1 - D(\mathbf{x}))$$

guarantees that p_G converges to a good estimate of p_{data} in theory, given enough training data and time. This criterion is also a minimax game. However, the proof is done in a non-parametric setting, meaning that p_G represents a model with infinite capability. In practice, the generative distribution p_G is parametrized by $\boldsymbol{\Theta}^{(\mathbf{G})}$ which define neural networks. Goodfellow et al. [16] explains that due to the non-convexity of the parameter space the guarantees do not hold, but claims that the general performance of neural networks suggests that they are reasonable models to use.

In practice, the zero-sum game loss function proved not to perform especially well [12]. The minimax approach can be disadvantageous for training the generator, because when the discriminator rejects a generated sample with high confidence, the generator's gradient vanishes. This might happen early in the game, before the generator has learned from its mistakes, if the inner loop runs too many times. Instead of designing the loss function as zero-sum game, but still pit the networks against each other, is to flip the objective of the cross entropy loss. The loss function becomes

$$J^{(G)} = -\frac{1}{2}\mathbb{E}_{\mathbf{z}} \log D(G(\mathbf{z})),$$

which means that the generator tries to maximize the probability of the discriminator being wrong rather than minimizing the probability of it being right.

GANs are trained iteratively and numerically using simultaneous stochastic gradient descent (SGD). In each training step, a sample batch $\mathbf{x}$ is drawn from the training set, and another batch $\mathbf{z}$ is drawn from the prior distribution $p_{\mathbf{z}}$. Starting with the inner loop, for k rounds, $\boldsymbol{\Theta}^{(\mathbf{D})}$ is updated by following its stochastic gradient. After this, $\boldsymbol{\Theta}^{(\mathbf{G})}$ is updated along its stochastic gradient on $\mathbf{z}$. Any optimization algorithm can be chosen for both updates.

In Figure 2.4, the different steps of training a GAN is shown in principle. The black, dotted line represents the true data distribution p_{data} to be learned, and the green, solid line represents the generator's distribution p_G. The blue, dashed line represents the discriminative distribution along x, and the lower horizontal line represents the domain from which z is sampled. Goodfellow et al. [16] describes the general steps of the GAN training as follows:

(a) p_{data} and p_G are close to convergence, but not exact yet. D can distinguish fairly successfully between the two distributions.

(b) In the inner loop of the game, D is trained to learn to distinguish the two distributions very well, converging to $\frac{p_{\text{data}}(x)}{p_{\text{data}}(x)+p_G(x)}$. The gradient of D is fed back to G.

(c) G is updated to move to regions more likely to be classified as true data, up the gradient of D.

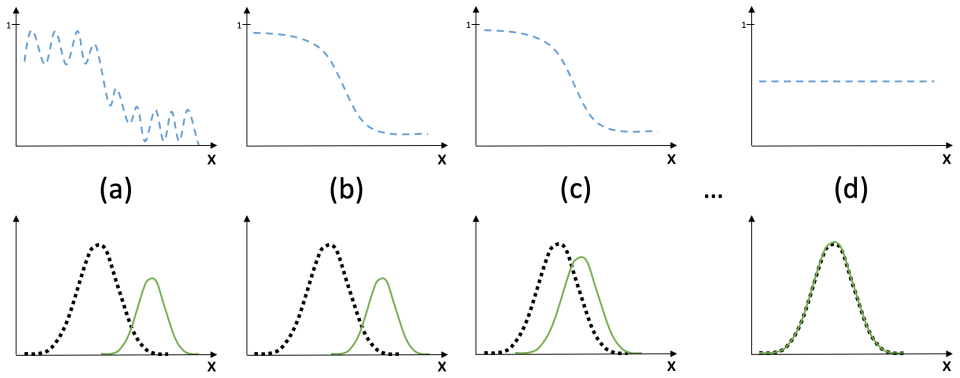

Figure 2.4: Visual representation of the distributions' updates when training a GAN close to convergence. Target distribution in black dots, generator distribution in green solid line and discriminator distribution in blue dashes. Based on [16].

(d) Having repeated steps (a)-(c) enough times, given enough training data, p_G approaches p_{data} exactly. Since they are identical, the discriminator converges to the constant solution $D(x) = \frac{1}{2}$.

The GAN framework was designed to address issues with other approaches as described in subsection 2.3.1. All frameworks have their advantages and disadvantages. For some use cases, when the goal is to understand the distribution fundamentally rather than just being able to sample from it, an explicit representation of p_{data} might be needed, and GANs cannot be used. Further, one of the issues GANs want to address is the ability to generate multimodal output distributions, which means that the generator is likely to generate similar samples over and over again. However, if the training is not balanced and G progresses too fast for D, G might collapse many different values of $\mathbf{z}$ to the same $\mathbf{x}$. In a sense, if G learns one very successful way to fool D, it stops searching for another one.

Aside from the computational advantages, GANs can resist overfitting well in general [16]. The reason is that G does not learn from the true samples directly, but only through the gradients flowing from D. Lastly, any Markov chain method requires a somewhat smooth distribution in order to draw samples from different peaks in the distribution, while a GAN can represent an arbitrarily sharp underlying distribution.

GANs offer solutions to many problems with other types of gener-

ative models. However, the networks are learning through differentiation in continuous space, while natural languages reside in discrete spaces. To understand whether they can still be used for natural languages, we must first understand how neural language models work.

2.4 Neural Network Language Modeling

The idea to utilize the power of neural networks for language modeling was first proposed by Bengio et al. [7] in 2003. Modeling natural language is fundamentally difficult due to the curse of dimensionality, since there are a great number of free parameters. To model the joint distribution of 10 words in a vocabulary of 100 000 words, there would be almost 10^{50} free parameters, and no dataset in the world would be able to fully capture all possibilities. This means that in testing, the model might come across sequences which have never occurred in training. In continuous spaces, generalization is possible with local smoothing. However, in discrete spaces, any change of the variables might have drastic effects on the results. In a vocabulary, there is no necessary resemblance between any word and its neighbours.

Bengio et al. [7] proposed to represent each word with a distributed word feature vector, an embedding in $\mathbb{R}^m$ where m is far smaller than the size of the vocabulary. At the same time, a joint probability function of word sequences was learned. This would result in similar embeddings for words which are similar semantically and syntactically, so that the low-dimensional vector for 'cat' might be more similar to the vector for 'dog' than the vector for 'cast'. This was done by implementing a feed-forward neural network. Testing for a range of network depths, the models' performances constantly exceeded the performance of n-gram models with smoothing, which were considered state-of-the-art at the time.

The major issue with the feed-forward network is that it uses a fixed-length context. As discussed in subsection 2.2.2, this means that anything outside of the context window is forgotten, while natural languages often carry long-term dependencies.

2.4.1 Supervised Recurrent Neural Network Language Models

To be able to store information for an arbitrarily long time, recurrent neural network language models (RNNLMs) were developed in 2010 by Mikolov et al. [17] using a simple recurrent neural network design where the input vector $x(t)$ is concatenated with the state vector $s(t-1)$. The state function R, as described in subsection 2.2.2, is the sigmoid activation function, while the output activation O is the softmax function. The network is described mathematically as

$$
\begin{aligned}
x(t) &= w(t) + s(t-1) \\
s_j(t) &= R\left(\sum_i x_i(t) u_{ji}\right) \\
y_k(t) &= O\left(\sum_{s_j} (t) v_{kj}\right)
\end{aligned}
$$

$$
R(z) = \frac{1}{1+e^{-z}}, \quad O(z_m) = \frac{e^{z_m}}{\sum_k e^{z_k}}.
$$

In statistical language modeling, the models typically do not get updated in the testing phase as the data is processed. This is problematic since texts on a certain topic tend to reuse words, especially such as person names, more frequently than the training data. To address this, Mikolov et al. [17] also tested a dynamic model where the parameters are updated in the testing phase. They call such a model dynamic, while a model which does not learn after the training phase is referred to as static. Since the neural model is working in continuous space, similar words will also have a higher probability. To exemplify, a model reading a text about cats might increase its belief that the word 'dog' might appear in the upcoming text. Testing both static and dynamic models, they were able to significantly outperform state-of-the-art backoff models, thus breaking the myth that improving language models is simply a matter of training n-gram models on more data. The dynamic models improved perplexity drastically, suggesting that a closer examination of on-line learning is important.

It should be noted that the purpose of the RNNLM was to enable capturing long-term dependencies. They concluded that this was not significantly improved. Bengio, Simard, and Frasconi [18] discuss how

SGD is not a suitable method for learning long-term dependencies. Using LSTM architectures proved more suitable for this task [19].

2.4.2 Exposure Bias

While text generation is an unsupervised task, the above described RNNLMs are trained by optimizing a supervised metric, namely the maximum likelihood of the training data, using teacher forcing as described in 2.2.3. The loss function for a model G_Θ can be described as:

$$J_\Theta(y_n) = -\sum_{t=0}^{n-1} \log \hat{P}(x_t|y_t)$$

where the empty string is used as a starting token.

This leads to a problem at the inference step known as exposure bias. At training, it learns to find the most likely upcoming word given a true context. When a text is generated, the upcoming word is therefore found by assuming the context is a valid sequence, while it is actually self-generated. Thus, errors might accumulate quickly. There are four main approaches to address exposure bias: scheduled sampling, reinforcement learning, re-parametrization and adversarial training.

Scheduled Sampling

When humans learn, we are presented gradually more advanced concepts and more information in a structured order. This is known as curriculum learning. In 2009, Bengio et al. [20] explored whether machine learning could also be improved by starting with simple tasks and gradually increasing the difficulty level. Their findings showed that this can have an effect on generalization ability as well as convergence speed.

In 2015, Bengio et al. [19] suggested to change the RNNLM training strategy to a curriculum learning approach in order to remedy the exposure bias. They tried to simulate the inference, meaning that the model is trained at first only on ground truth samples, but gradually is fed more and more generated tokens as input. The method is called scheduled sampling, and proved to work well empirically, with increased robustness in generated sequences.

Scheduled sampling was soon deemed inconsistent, and a breakdown of the algorithm suggests that the hidden states do not contain decisive information about the contents of the context leading up to a word, but merely acts as a trivial counter of word position in a sequence [21]. Even with infinite data, the models might not converge to the correct distribution.

RNNLMs with Reinforcement Learning

Another possible approach to training is to use a reinforcement learning policy-gradient approach, which would train the model directly by optimizing a proxy metric. The problem lies in choosing a suitable metric to optimize. A popular choice is to optimize the BLEU[1] score [22]. BLEU is used to evaluate the quality of translated texts by comparing the generated translation with translations made by humans. The score is calculated as the quotient of the n-grams generated by the model which also appear in the reference documents over the total number of generated n-grams.

The choice of optimizing the BLEU score directly is not easily motivated. For a general text generation, there are no "true" baselines to compare with. Optimizing the BLEU score could also result in a generated text which just repeats a single successful pattern over and over again, which signifies a mode collapse. Further, there are many human written texts that would have bad BLEU scores simply because the sentences they write don't appear in the reference texts. Lastly, BLEU is an expensive metric to calculate [23].

Re-parametrization

The idea of the re-parametrization trick is to rewrite the expected values in the loss functions to be independent on the parameters by separating the parameter from a stochastic element. This approach has not been evaluated much, but initial empirical results are not impressive [24][25]. The generated texts suffer from severe mode collapse and are not readable in general.

A more promising approach is to train a generator using adversarial methods.

[1] Bilingual Evaluation Understudy

2.5 Generative Adversarial Networks for Natural Language Generation

2.5.1 The Discrete Output Issue

GANs were shown to be capable of minimizing the exposure bias from the discrepancy between inference and training in a number of applications [12]. Unfortunately, they cannot be directly applied to text-based generation. Natural language sentences are sequences of discrete tokens while GANs were designed to generate continuous output data. In essence, the GAN generator G draws samples randomly, which are fed to the discriminator D. The gradient of the loss function of D with respect to the outputs of G guides G to make small changes that would make the sample more likely to fool the discriminator. With discrete tokens, there are no small changes [26]. Even using word embeddings, there are likely no small changes that would represent another token due to the sparsity in the word space. Still, there are workarounds which allow adversarial training.

2.5.2 Adversarial Training

Professor Forcing

The issue with exposure bias in generated text sequences is not easy to overcome. Scheduled sampling had proven to give good results in some aspects, but could not guarantee stable performances. Still, the main idea of scheduled sampling sounds intuitively correct; a model should behave similarly whether it is fed ground truth, i.e. teacher forcing, or in the free-running inference mode. To draw on this idea, but avoid the inconsistencies Huszár [21] explained with scheduled sampling, Lamb et al. [27] created a novel training approach called professor forcing. To compare the network's input sequences and to try to make them indistinguishable, their distributions are compared based on the GAN framework.

Lamb et al. [27] created one generative network and one discriminative, and defined a function $B_{\boldsymbol{\Theta}^{(\mathbf{G})}}(x_t, y_t)$ which outputs the system's output and hidden states, where x_t is drawn from the true data and y_t can either be drawn from the data or generated using the parameters $\boldsymbol{\Theta}^{(\mathbf{G})}$. Using the same notation as in subsection 2.3.2, the discriminator

then optimizes the loss function

$$J^{(D)}(\boldsymbol{\Theta}^{(\mathbf{D})}, \boldsymbol{\Theta}^{(\mathbf{G})}) = - \mathbb{E}_{(x_t,y_t)\sim p_{\text{data}}} \log D(B_{\boldsymbol{\Theta}^{(\mathbf{G})}}(x_t, y_t), \boldsymbol{\Theta}^{(\mathbf{D})}) \\ - \mathbb{E}_{x_t\sim p_{\text{data}},\hat{y}_t\sim p_G} \log(1 - D(B_{\boldsymbol{\Theta}^{(\mathbf{G})}}(x_t, \hat{y}_t), \boldsymbol{\Theta}^{(\mathbf{D})})).$$

For the generator, there are two objectives: maximizing the likelihood of the training data, and fooling the discriminator. For the first objective, the standard negative log likelihood (NLL) is used. For the second objective, two different objectives are considered. The first, C_f, tries to change the inference behavior to resemble the teacher-forced behavior, considering it as fixed. The second, C_t, is an optional objective which tries to match the teacher-forced behavior to the inference behavior. All in all, the generator's loss can be described as

$$J^{(G)}(\boldsymbol{\Theta}^{(\mathbf{D})}, \boldsymbol{\Theta}^{(\mathbf{G})}) = NLL(\boldsymbol{\Theta}^{(\mathbf{G})}) + C_f(\boldsymbol{\Theta}^{(\mathbf{G})}|\boldsymbol{\Theta}^{(\mathbf{D})}) +^* C_t(\boldsymbol{\Theta}^{(\mathbf{G})}|\boldsymbol{\Theta}^{(\mathbf{D})}),$$

$$\begin{aligned} NLL(\boldsymbol{\Theta}^{(\mathbf{G})}) &= -\mathbb{E}_{(x_t,y_t)\sim p_{\text{data}}} \log \hat{P}(y_t|x_t), \\ C_f(\boldsymbol{\Theta}^{(\mathbf{G})}|\boldsymbol{\Theta}^{(\mathbf{D})}) &= -\mathbb{E}_{x_t\sim p_{\text{data}},\hat{y}_t\sim p_G} \log D(B_{\boldsymbol{\Theta}^{(\mathbf{G})}}(x_t, \hat{y}_t), \boldsymbol{\Theta}^{(\mathbf{D})}), \\ C_t(\boldsymbol{\Theta}^{(\mathbf{G})}|\boldsymbol{\Theta}^{(\mathbf{D})}) &= -\mathbb{E}_{(x_t,y_t)\sim p_{\text{data}}} \log(1 - D(B_{\boldsymbol{\Theta}^{(\mathbf{G})}}(x_t, y_t), \boldsymbol{\Theta}^{(\mathbf{D})})). \end{aligned}$$

Note that the network described above have a striking resemblance to a GAN. However, this network matches the hidden states as opposed to the output tokens. The hidden states are continuous, so the gradients to update can be found from straight-forward differentiation. Thus, professor forcing is an adversarial training method but not a GAN since it doesn't try to make the distribution of the outputs indistinguishable from the distribution of the observed data. For this reason, it does not guarantee high sample quality [28].

SeqGAN - Policy Gradient

A breakthrough in the step towards using GANs for text generation was the idea to treat the task as a sequential decision making process, rather than as sampling from a distribution. Bachman and Precup [29] described how this can be done efficiently for general data generation by formulating the problem as a policy search in reinforcement learning. In that sense, the generated tokens represent the current state while the upcoming token represents the action. However, this approach requires a task-specific sequence score to give the reward,

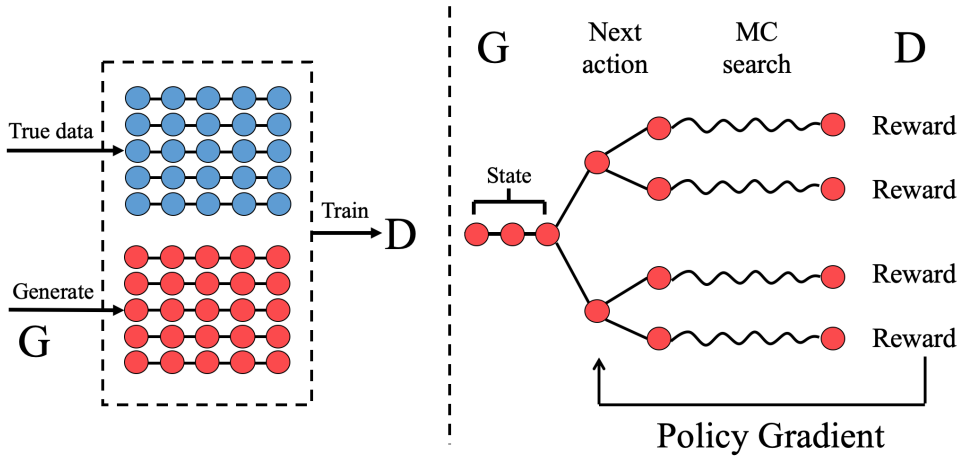

Figure 2.5: SeqGAN is trained by passing a policy gradient found through Monte Carlo search. The reward is provided by D, which have been trained on a mixture of true samples and generated samples. Figure by Ellen Jonsson, based on [26].

upon which the action is based. As discussed in 2.4.2, finding a score for general text generation problem is troublesome. Further, this reinforces another issue. To generate text efficiently, we want the model to know how it's performing while generating partial sequences. A score such as BLEU can only be evaluated for full sequences.

Arguably the most influential step towards using GANs for text generation was presented in 2016 by Yu et al. [26], building upon the idea from Bachman and Precup [29]. They also decided to model a generative network as a stochastic policy in reinforcement learning, using the REINFORCE algorithm [30]. However, their framework based the reward function on a discriminative network - which means that the system is a GAN. The generative model is then regarded as a stochastic parametrized policy, and the state-action value can be approximated through Monte Carlo simulations. The policy is then updated via policy gradient [31], thus bypassing the issue conventional GANs have with gradient updates for sequences of discrete tokens. This framework was called SeqGAN.

The discriminator in SeqGAN is updated in a similar manner to the one in a general GAN, but the network should also give intermediary feedback to give long-term benefits. A parallel can be drawn to a chess game where even if a certain move guarantees taking out the

opponents queen, a player should avoid it if they lose the game two moves later. In SeqGAN, the action-state function takes this into account by introducing a Monte Carlo search for future rewards using a roll-out policy.

Implementing a SeqGAN Architecture

SeqGANs are, as most GAN architectures presented in this study, implemented with an LSTM generator network and a convolutional neural network (CNN) as the discriminator. SeqGAN is a high variance model, and to reduce the variance the models are typically pre-trained using teacher forcing, i.e. maximum likelihood estimation of the training data. This is common for many architectures, and when the term "MLE pre-training" is used henceforth, it is referring to teacher forcing of the LSTM network.

Drawbacks of SeqGAN

SeqGAN proved impressive results in the authors' synthetic data evaluations. However, Lu et al. [23] identified two obstacles which led to the development of improved GAN architectures for generating text. The first one is vanishing gradients. When the discriminator becomes very accurate, almost all outputs will have a zero score, thus no changes are made. This might happen before the game converges, or reaches an equilibrium. Secondly, the REINFORCE algorithm tries to reward tokens earning a high evaluation from the discriminator, and a side-effect might be that the same tokens are generated over and over again - a mode collapse. This motivates the search for improved variants of SeqGAN.

2.5.3 Recent GAN Architectures

If not otherwise specified, the following architectures utilize a policy gradient approach similar to the one used by SeqGAN.

MaLiGAN

To tackle the issue with vanishing gradients, Che et al. [32] introduced a novel training objective based on a normalized maximum

likelihood optimization, which is fully optimized with a range of variance reduction techniques such as importance sampling. The method is called Maximum Likelihood Augmented Discrete GANs (MaLiGANs). Apart from reaching convergence faster, they claim that the model produces more stable results by minimizing the likelihood of bad results simultaneously as maximizing the likelihood of good results.

The optimal discriminator in a GAN approaches the form $D(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x)+p_G(x)}$ when it converges in the inner loop, as explained in 2.3.2. This expression can be rewritten as $p_{\text{data}}(x) = \frac{D(x)}{1-D(x)}p_G(x)$. This can be used to define a reward function based on the renormalized score as $R(y) = \frac{D(y)}{1-D(y)}$. Che et al. [32] prove that this reward function can provide a better learning signal, resulting in low-perplexity results.

RankGAN

Modifying the reward like in MaLiGAN is one possible approach to remedy the vanishing gradients. Another is to focus on rankings rather than a binary discriminator. The discriminator is asked to rank a set of sentences and the goal for the generator is to produce sentences ranked higher than true sentences. Lin et al. [33] created an architecture called RankGAN based on this idea. To do this, they use a reference U consisting only of true samples. The rankings are calculated using embeddings of feature vectors, defined by a series of nonlinear functions F as $v_y = F(y)$.

Using Information Retrieval (IR) techniques to rank texts relatively, as described by Liu [34], they formulate a relevance score

$$\alpha(y|u) = \cos(v_y, v_u)$$

where y is an input sentence, $u \sim U$ and v_y, v_u are the embedded feature vectors of y and u. Letting C^+ denote the training dataset and C^- the set of generated sequences, they define a ranking score using a soft-max-like formula as

$$P(y|u, C) = \frac{\exp(\gamma\alpha(y|u)))}{\sum_{y' \in C} \exp(\gamma\alpha(y'|u))}$$

where γ is the inverse Boltzmann temperature factor, a hyperparameter for ensuring finite sum of expected rewards. The reward

function becomes the expected value of the ranking score:

$$R(y|U, C) = \mathbb{E}_{u \in U}[P(y|u, C)].$$

The loss function which the ranking "discriminator" tries to optimize becomes

$$J^{(R)}(\mathbf{\Theta}^{(\mathbf{R})}) = \mathbb{E}_{y \sim p_{\text{data}}}[\log R(y|U, C^-)] - \mathbb{E}_{y \sim p_G}[\log R(y|U, C^+)],$$

and the full game can be described as

$$\arg\min_{\mathbf{\Theta}^{(\mathbf{G})}} \max_{\mathbf{\Theta}^{(\mathbf{R})}} \mathbb{L} = \mathbb{E}_{y \sim p_{\text{data}}}[\log R(y|U, C^-)] + \mathbb{E}_{y \sim p_G}[\log(1 - R(y|U, C^+))].$$

In a study by Lu et al. [23], it is shown that a more accurate discriminator is not always beneficial. As long as the relative ranking between documents is correct, it is in fact beneficial for the convergence speed to have larger scores as long as the relative ranking is kept intact. For this reason, RankGAN typically has at least comparable performance with SeqGAN while the convergence is improved [33]. A drawback is that the required increased sampling leads to an increased computational cost.

LeakGAN

In some cases, having a scalar signal as the feedback from a discriminator might not be informative enough. A scalar output signal, as hitherto described, is only available after the entire text has been generated. For this reason, SeqGAN and its early variations did not prove remarkable results for sentences longer than 20 words. Guo et al. [35] aimed to mitigate this with hierarchical reinforcement learning concepts [36].

Introducing a hierarchy into text generation can be motivated as real text typically has some form of hierarchical structures such as semantic structures and part-of-speech. The hierarchical reinforcement learning objective is to assign different parts of the learning to different units. Guo et al. [35] refers to some limited earlier success using hierarchy in text generation, but claims to be the first ones to do so without pre-defined subtasks, thus making their framework LeakGAN the first to do so for arbitrary sequence generation tasks.

LeakGAN differs from a typical GAN in how the generator is split into a Manager module and a Worker module, which are built as LSTM

networks. Given a current sequence, the discriminator extracts features which it leaks to the manager. The features are projected into action space through a linear projection, which are combined with a direct embedding of the transition probability distribution based on the previous word to create the final distribution for the next word. Note that this leak is not the adversarial training, but only acts as another input source to the generator.

In their real-life experiments, Guo et al. [35] show that LeakGAN has a significant advantage to SeqGAN in Turing tests.

MaskGAN

Many of the described models have been evaluated and compared by their authors using the BLEU score. The soundness of using BLEU for general text generation can be debated, and is further discussed in the Method and Discussion chapters. However, most of the methods have not evaluated how significant the mode collapse is. In their 2018 study, Lu et al. [23] evaluated mode collapse of the methods above, and a few others, by measuring their self-BLEU [25]. To measure self-BLEU, one takes sampled sentences from the generated collection and calculates their BLEU score on the rest of the collection. The average self-BLUE becomes the self-BLUE of the collection, and a high self-BLUE indicates that most sentences are similar to each other, which indicates a mode collapse. Most of the frameworks suffer from serious mode collapse, but a framework called MaskGAN stands out from the rest with a significantly lower self-BLEU.

The idea of MaskGAN is to train a GAN by feeding sentences are partially masked, so that one or more words are blank entries which the generator tries to fill in with suitable words. The MaskGAN generator uses a seq2seq [37] architecture, which means that the network reads a full sequence and encodes it into a hidden layer and outputs another full sequence. Introduced in 2014, the seq2seq architecture proved to be impressive for translation purposes, but can also be applied to general language modeling. Fedus, Goodfellow, and Dai [28] designed MaskGAN as follows.

Let x_t denote an input token and m_t a masked token, i.e. a token where the token has been replaced with a hidden token. $\hat{x}_t$ denotes the filled-in token, while $\tilde{x}_t$ is a filled-in token passed to the discriminator which might be either real or fake. $\mathbf{x}$ is an input sequence $(x_1, ..., x_T)$

and $\mathbf{m} = (m_1, ..., m_T)$ is a binary mask of the same length. The generator and discriminator are then

$$G(x_t, \mathbf{m}(\mathbf{x}), \mathbf{\Theta}^{(\mathbf{G})}) = P(\hat{x}_t | \hat{x}_1, ..., \hat{x}_{t-1}, \mathbf{m}(\mathbf{x}))$$
$$D(\tilde{x}_t | \tilde{x}_{0:T}, \mathbf{m}(\mathbf{x}), \mathbf{\Theta}^{(\mathbf{D})}) = P(\tilde{x}_t = x_t^{real} | \tilde{x}_{0:T}, \mathbf{m}(\mathbf{x})).$$

The discriminator thus has the same architecture as the generator, but instead of outputting a distribution over the current token, it outputs a scalar probability. It is important to include the masked input to the discriminator rather than just identifying errors in the input texts. For example, without it, if the discriminator is fed a filled-in sequence such as "*the director director guided the series*", it will ultimately fail to recognize that "*director director*" is an error. The reason is because the error is ambiguous, since it is possible that either of the "*director*" words are faulty, as both "*the *associate* director guided the series*" and "*the director *expertly* guided the series*" are potentially valid sequences.

MaskGAN is trained, like SeqGAN and others, using policy gradient and the REINFORCE algorithm. The reward is defined by the discriminator output as

$$r_t = \log D(\tilde{x}_t | \tilde{x}_{0:T}, \mathbf{m}(\mathbf{x}), \mathbf{\Theta}^{(\mathbf{D})}).$$

2.5.4 Comparing Architectures

Ultimately, MaskGAN produces more diverse results than many other GAN frameworks for natural language generation. This does not automatically mean that the generated texts have a higher quality, and there is to this date no metric that can fully translate human judgment of texts. It is not usually sound to claim any framework as superior to others based on a single metric, and evaluation is typically done using combinations of metrics and human judges. This is further discussed in the Method chapter.

2.6 Cooperative Training for Generative Modeling

2.6.1 The Shortcomings of GANs for Natural Language Generation

Despite the many variations of the SeqGAN architecture, there hasn't been one design which shows reliable improvements for all scenarios [23]. There are three major problems: the MLE pre-training, vanishing gradients and mode collapse.

MLE Training

SeqGAN is a high variance algorithm, which relies on MLE pre-training as a variance reduction technique. MLE training in effect optimizes a measure called the Kullback-Leibler (KL) divergence, as shown by Arjovsky and Bottou [38]. The KL divergence between two distributions p_{data} and p_G is defined as

$$KL(p_{data}||p_G) = \sum_{\mathbf{x}} p_{data}(\mathbf{x}) \log \frac{p_{data}(\mathbf{x})}{p_G(\mathbf{x})},$$

where $\mathbf{x}$ is a sequence.

It is important to notice how it behaves when either $p_G(\mathbf{x})$ or $p_{data(\mathbf{x})}$ goes toward 0.

When $p_{data}(\mathbf{x}) > p_G(\mathbf{x})$, the sample $\mathbf{x}$ is more likely to come from the true data than the generator. If $p_G(\mathbf{x})$ then goes toward zero, the KL divergence goes quickly towards infinity. This leads to "mode dropping" scenarios, where a major cost is assigned when the generator fails to cover some parts of the data, which promotes smoother density distributions. Thus, when a mode is not covered by the generative distribution, it is said to be dropped.

On the other hand, when $(p_G\mathbf{x}) > p_{data}(\mathbf{x})$, the sample is more likely to be drawn from the generator. For these cases, when $p_{data}(\mathbf{x})$ is close to zero, the cost is still very low. The result is that samples which are clearly fake can be passed.

The KL divergence is clearly not symmetric from its definition, and if one instead would optimize the reverse KL divergence, $KL(p_G||p_{data})$, the generator risks collapsing into a single mode. In other words, if there are multiple modes, the generator distribution

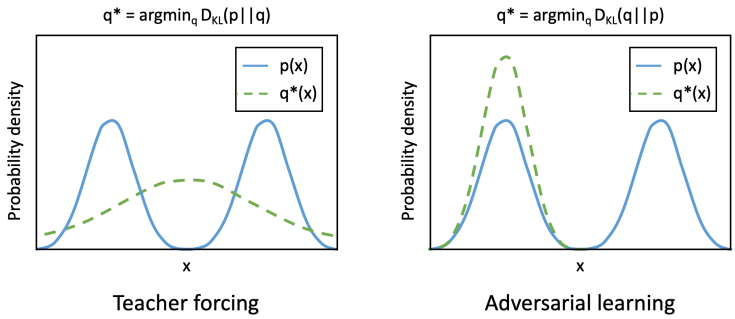

Figure 2.6: KL divergence is not a symmetric measure. In these plots, a single Gaussian is fitted to a target distribution consisting of a mixture of two Gaussians, which it cannot do perfectly. To the left, the MLE criteria tries to make sure that every area which is covered by data is included. Reversing the criteria assigns high cost to the areas where there is no data, so the resulting distribution will have a mode in either of the underlying peaks. This shows how GANs can produce high quality samples, but might suffer from mode collapse. Based on [12].

might make sure to model one of the modes and do it well, as seen in Figure 2.6. This indicates a model with a high quality while lacking diversity.

Vanishing Gradient

SeqGAN has convergence troubles due to being based on model-free RL and the training might get stuck in local minimums, where a larger learning rate is to no use, because of the vanishing gradient as described in subsection 2.5.1 [39]. This can be alleviated by adjusting the learning signal like MaLiGAN or RankGAN does, but this might not always be enough since it is essentially treating the symptom rather than the disease.

Mode Collapse

When SeqGAN variants are trained using REINFORCE, they tend to suffer from mode collapse, meaning that the model finds certain samples producing good results, and reproduces them over and over again instead of exploring other alternatives. According to [39], this is because the learned distribution has optimized the reverse KL divergence. As proven by Lu et al. [23], most SeqGAN variants suffer from

a loss of diversity due to this. The diversity has been traded off from generative quality.

2.6.2 Cooperative Training

Ideally, the target distribution should be represented both in terms of diversity and quality. Lu et al. [39] argues that this can be done by introducing cooperative training (CoT) as an alternative to adversarial training or MLE training.

The idea behind CoT is to train a secondary model as the average of the true distribution p_{data} and the generator's distribution p_G, and try to fit this average as close to both distributions as possible. If the average is able to lie close to the two distributions, our intuition tells us they must also be similar.

MLE training tries to minimize the KL divergence of p_{data} from p_G, while adversarial training minimizes the KL divergence of p_G from p_{data}. Instead of optimizing the KL divergence, Lu et al. [39] optimize a symmetrized and smoothed version of the measure, namely the Jensen-Shannon divergence (JSD) defined as:

$$JSD(p_{data}||p_G) = \frac{1}{2}(KL(p_{data}||M) + KL(p_G||M))$$

where $M = \frac{1}{2}(p_{data} + p_G)$. Arjovsky and Bottou [38] argued that this is a theoretically sound cost function for GANs, but M is typically intractable. Lu et al. [39] propose a design which they prove theoretically optimizes an unbiased estimation of the JSD, and prove that this approaches the true JSD with optimal training. In short, their idea is based on training a so-called mediator $M_{\boldsymbol{\Theta}^{(\mathrm{M})}}$ which is a parametrized model of M. The mediator samples from both p_G and p_{data} to form an estimate of M, and uses MLE to optimize the parameters $\boldsymbol{\Theta}^{(\mathrm{M})}$. An illustration of the process can be seen in Figure 2.7.

The learning objective of the whole process can be described as

$$\max_{\boldsymbol{\Theta}^{(\mathrm{G})}} \max_{\boldsymbol{\Theta}^{(\mathrm{M})}} \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}} \log M_{\boldsymbol{\Theta}^{(\mathrm{M})}}(\mathbf{x}) + \mathbb{E}_{\mathbf{x}\sim p_G} \log M_{\boldsymbol{\Theta}^{(\mathrm{M})}}(\mathbf{x}),$$

which can be compared with the overall objective of GANs

$$\min_{\boldsymbol{\Theta}^{(\mathrm{G})}} \max_{\boldsymbol{\Theta}^{(\mathrm{D})}} \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}} \log D_{\boldsymbol{\Theta}^{(\mathrm{D})}}(\mathbf{x}) + \mathbb{E}_{\mathbf{x}\sim p_G} \log(1 - D_{\boldsymbol{\Theta}^{(\mathrm{D})}}(\mathbf{x})).$$

As such, the mediator can be said to guide the generator in a cooperative sense, while the discriminator of a GAN is an adversary trying to outperform the generator.

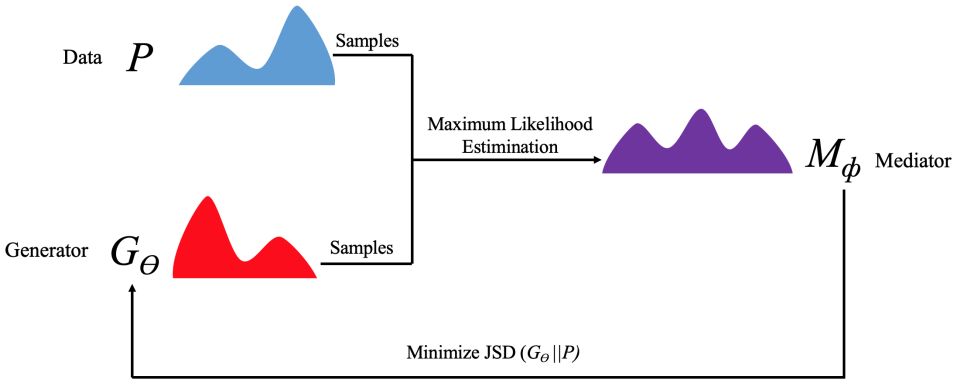

Figure 2.7: The CoT architecture uses a mixture of true and generated samples to train a mediator by minimizing the MLE in between them, using the result to estimate the symmetric measure JSD. Figure by Ellen Jonsson, based on [39].

Lu et al. [39] argues that CoT should not need any pre-training as they claim the algorithm is low variance.

Chapter 3

Method

3.1 Models

In the experiments of this thesis, four models will be compared. One model is a CoT model without any pre-training. Another is a CoT using pre-training, along with a SeqGAN model with equivalent pre-training. Further, an LSTM baseline is trained using MLE, i.e. teacher forcing.

Finally, a fully random model is used as a reference by sampling from the SeqGAN model after random initialization, before any training.

3.1.1 CoT 130

CoT 130 is a model consisting of a generator G and a mediator M. The model is trained for 130 cooperative epochs, without any pre-training.

The generator is an LSTM network with a hidden size of 32, an embedding dimension of 32 and a learning rate of 0.001. A dropout rate of 0.5 is used as a regularization technique. A batch size of 64 is used in training.

The mediator is an LSTM network with a hidden size of 64, an embedding dimension of 64 and a learning rate of 0.001. As for G, a dropout rate of 0.5 and a batch size of 64 is used.

3.1.2 CoT 50+80

CoT 50+80 is the second CoT model, which consists of identical generator and mediator as CoT 130. The difference between the two models lies in the training, where CoT 50+80 is first pre-trained for 50 MLE epochs followed by 80 epochs of cooperative training.

3.1.3 SeqGAN

A SeqGAN model consisting of a generator G and a discriminator D. First, G is pre-trained using MLE for 50 epochs. Afterwards, the full model is trained adversarily for 80 epochs where each generator update is followed by 15 training epochs for the discriminator.

The generator is an LSTM network with a hidden size of 32, an embedding dimension of 32 and a learning rate of 0.01. A batch size of 64 is used in training.

The discriminator is a CNN with two filtered layers with maxpool layers afterwards. The filter sizes are 100 and 200, with strides of 2 and 3 and a 0.2 L2 regularization.

3.1.4 MLE

A standard LSTM with a hidden size of 32, an embedding dimension of 32 and a learning rate of 0.01. A batch size of 64 is used in training. The model is trained for 130 epochs using teacher forcing.

3.2 Accurate Evaluation of Text Generators

In order to evaluate text generators fairly, there has to be a systematic approach to comparison. The best judges of language are still human interpreters, but having people read every sample created by different models is not realistic. Further, a distinguishing feature of GANs (and CoT) is that they are able to represent multimodal distributions. A human judge being presented a few good samples might measure quality well, but not diversity. A good text generator should create samples which are of high quality, i.e. with few errors and reasonable content, while also producing diverse samples.

Ideally, one would find a metric which could numerically determine which model outperforms another, by simply comparing their

scores. Popular metrics are the perplexity, the negative log likelihood, or n-gram-based metrics such as BLEU [23][25]. None of these metrics are able to fully capture the complexity and quality of a natural language.

As previously explained, training models on a likelihood-based objective introduces exposure bias since the models are trained by sampling words following a ground truth sequence, while in inference, the ground truth is provided by the model itself. However, Reinforcement Learning techniques, GANs and CoT address this problem by generating samples which cannot be differentiated from those coming from the true data distribution. These methods cannot be fairly compared with methods which directly optimizes the likelihood using a likelihood measure. N-gram-based metrics on the other hand cannot fully evaluate semantic variations or detect complex syntactic problems, due to the Markov assumption, argues Semeniuta, Severyn, and Gelly [40]. In their 2018 paper, their goal is to design a comparison protocol for unconditional text generation, and their evaluations form the basis of how CoT will be compared to SeqGAN in this study.

3.3 Metrics

Semeniuta, Severyn, and Gelly [40] evaluate several metrics in search for the ones which correlate the most with human judgment. Their ideal metric should both be a) fast to compute and b) able to capture both quality and diversity. It needs to be fast so that the measure can be used to tune the model.

3.3.1 N-Gram-Based Metrics

Three different types of n-gram-based metrics are evaluated. The percentage of unique bigrams to appear in the generated text is one, which is a measure of the diversity. Another measure is BLEU, which will represent the text quality. Lastly, the self-BLEU will be used as a measure of diversity.

These metrics will be calculated for $n = \{2, 3\}$.

3.3.2 Language Model Score

The likelihood of generating the data samples under a pre-trained model will be evaluated, as a measure of text quality.

3.3.3 Reverse Language Model Score

As an alternative to the traditional language model, a reverse language model score can be calculated by training a language model on each of the generated sample sets, and evaluate the score of the true data under these language models. This metric might introduce a bias since the language model trained by the generated samples is not guaranteed to perfectly represent the underlying distribution.

The reverse language model score is sometimes called reverse perplexity. It has been growing in popularity and is used as a diversity measure [41]. Zhao et al. [42] describes the intuition of the metric as follows: "Reverse PPL: measuring the extent to which the generations are representative of the underlying data distribution".

3.3.4 Human Evaluation

The human evaluation can be set up in many forms. In this study, the inspiration was a combination of the evaluations by Semeniuta, Severyn, and Gelly [40] and Fedus, Goodfellow, and Dai [28].

From each model, 100 samples are uniformly selected and rated by 3 human raters. The raters are asked to score each sample on a scale from 1 to 5 on the samples, where 5 represents a text that could have been written by a human, and 1 represents a completely random sequence of words. However, the judges were informed to set grades based on comparative differences, meaning that all scores below 5 would still represent something they judged as not being human written.

The detailed instructions given to the human judges can be found in Appendix A.

3.4 Data

The data chosen to analyze this problem consists of political programs and manifestos of the eight parties represented in the Swedish Riks-

Table 3.1: Examples of data samples used for training.

teknik och etik får inte separeras
med globaliseringen sprids ekonomiska friheter till fler än någonsin tidigare
kunnande, erfarenhet och yrkesetik ska tas till vara
grön skatteväxling betyder höjda miljö- och klimatskatter samtidigt som skatten på jobb sänks
pensionssystemet måste förändras

dag. The text content of each program was pasted into a file which was then preprocessed according to subsection 3.4.1. After the preprocessing, the total dataset consisted of 14 405 unique sentences. This was randomly split up into a training set of 10 000 sentences and a validation set of 4 405 sentences. The average sentence length was 15 words, while the longest was 94 words before preprocessing, 44 words after the preprocessing, and the shortest was 3 words. A random set of samples from the full dataset can be seen in Table 3.1.

A complete list of all data sources can be found in Appendix B.

3.4.1 Preprocessing

Sentence Length

Both SeqGAN and CoT were evaluated on shorter sentences than the original max length in the used dataset. Their maximum sentence lengths were 50 while the raw dataset had sentences as long as 94 words [26][23]. However, sentences this long were outliers with a mean at just 15 words. This means that the generated sequences would also be 94 tokens, although usually with a number of stop signs at the end. To improve the readability of the generated samples and minimize the guaranteed grammatical and syntactical errors that long sequences would generate, and to be closer to the original evaluation conditions, long sentences were manually split up into two or three at the most suitable places. In some cases, this meant that words such as “och” (“and”) were removed, which made both clauses into valid sentences. After adjusting 350 sentences, the maximum sentence length was 47.

Regex Adjustments

All special characters, apart from alphanumeric characters (including the Swedish åääöÅÄÖ), commas and dashes, were removed from the dataset. Page numbers were removed, as well as leading numbers since these typically represented a list of actions or similar. Multiple whitespace characters were removed, along with any trailing or leading whitespace. Every sentence was split at a dot, exclamation mark or question mark.

3.5 Hardware Setup

The computations were performed on resources at Chalmer's Centre for Computational Science and Engineering (C3SE) provided by the Swedish National Infrastructure for Computing (SNIC).

The allocated resources were 20 2.3 GHz Intel 2650v3 CPU cores with 64 GB of RAM. For the most intense calculations, during the mediator training step of CoT, an NVIDIA Tesla K40 GPU core was utilized. The GPU usage was limited by queuing times, as described in subsection 3.7.1.

3.6 Software

All code used for the models was written in Python and ran in Python 3.6.4 with Tensorflow 1.6.0. The analysis and reference language models also used Keras 2.2.4.

In 2017, a UK-China joint research group started an initiative called Geek.AI.[1] Among the members are Weinan Zhang, Jun Wang and Yong Yu, all co-creators of SeqGAN along with Lantao Yu. In 2018, they released an open-source tool for benchmarking text generation models called Texygen [25]. The open-source code can be found at their GitHub repository.[2] Their implementation of SeqGAN formed the basis of one part of the study.

To generate sequences using CoT, the authors accompanied the paper with an open GitHub repository of their algorithm implementa-

[1] http://geek.ai/
[2] https://github.com/geek-ai/Texygen

tion.[3] As of November 2018, the code was not fully tested and could only handle oracle-generated sequences of numbers instead of text data, i.e. sequences created by a parametrized model with known parameters. Further, the evaluation metrics were based on comparing the underlying LSTM that generated the oracle data with the one that was fitted by the data, which was not applicable in this study since the text data was real and not generated. These obstacles were overcome mainly by retrofitting methods from Texygen.

For analysis, data pre-processing and plotting, Python scripts were created.

3.7 Limitations

3.7.1 Computational Resources

The major limiting factor for this project was the computational power. Early versions of the provided algorithm code had no CuDNN support which meant that it had to be ran on CPU cores instead of utilizing the GPU cores. As such, running CoT had a cost of around 25 core-hours (running a script for 1 hour on one core) per epoch. Being allocated a budget of 2 000 core-hours at SNIC, this meant that any extensive hyperparameter search had to be omitted.

Further, the SNIC cluster Hebbe consists of 280 CPU cores and 2 GPU cores. As such, there is typically a long waiting time for running scripts on a GPU, while CPU cores are available within hours after scripts are submitted.

3.7.2 Embedding Metrics

In their study of evaluation metrics, Semeniuta, Severyn, and Gelly [40] include one more metric which was not used for this study. This was the Frechet Distance, which is a measure of document-level embeddings. They claim that even though it is biased, it is widely accepted in the computer vision community. The embeddings were created using the InferSent [43] model. Since there are no pre-trained InferSent model for Swedish, this metric was excluded from the study.

[3] https://github.com/desire2020/CoT

Further, Semeniuta, Severyn, and Gelly [40] calculated the number of unique n-grams as a diversity metric. This was replaced to the percentage of unique n-grams in this study, since models might systematically generate different sentence lengths.

Chapter 4

Results

This section contains the results of the experiments described in chapter 3 followed by an objective analysis of the model results.

4.1 N-Gram-Based Metrics

The n-gram-based metrics BLEU and Self-BLEU were calculated, along with the number of unique n-grams. This was done for $n = 2$ and $n = 3$. No smoothing function was used, and no weighting between different BLEU levels. The reported BLEU scores are corpus level BLEU, since there are no specifically assigned reference sentences, as there would be e.g. if BLEU was calculated for a set of translated sentences.

The resulting scores can be found in Table 4.1, with reference result for the random model in Table 4.2. In Figure 4.1-4.4, the same data is visually represented.

Table 4.1: Results for n-gram-based metrics.

	CoT 130	**CoT 50+80**	**MLE**	**SeqGAN**
BLEU2	0.01596	0.09198	0.1609	0.1816
BLEU3	0.001477	0.008992	0.03675	0.03491
Self-BLEU2	0.09770	0.5302	0.6578	0.8056
Self-BLEU3	0.003778	0.07301	0.1908	0.3705
Unique 2-grams	94.20%	56.23%	44.97%	31.15%
Unique 3-grams	99.79%	96.76%	84.14%	73.80%

Table 4.2: Results for n-gram-based metrics for a random model.

	Random
BLEU2	0.0001013
BLEU3	0.000
Self-BLEU2	0.003776
Self-BLEU3	0.000
Unique 2-grams	99.86%
Unique 3-grams	100.00%

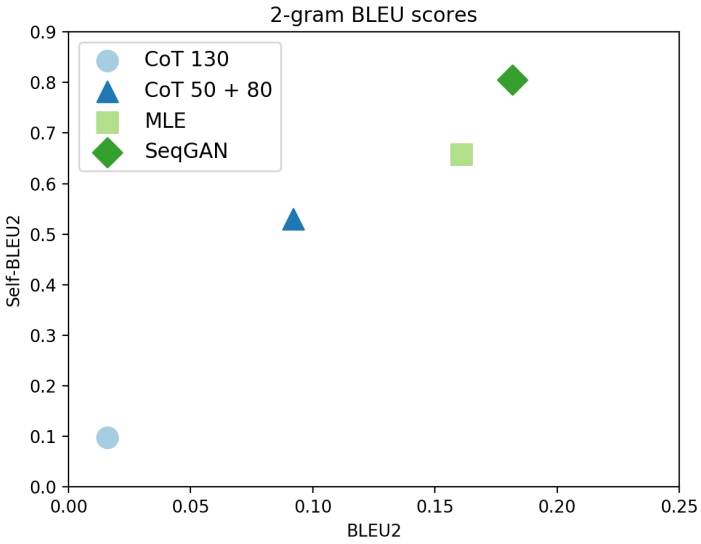

Figure 4.1: BLEU2 and self-BLEU2 results.

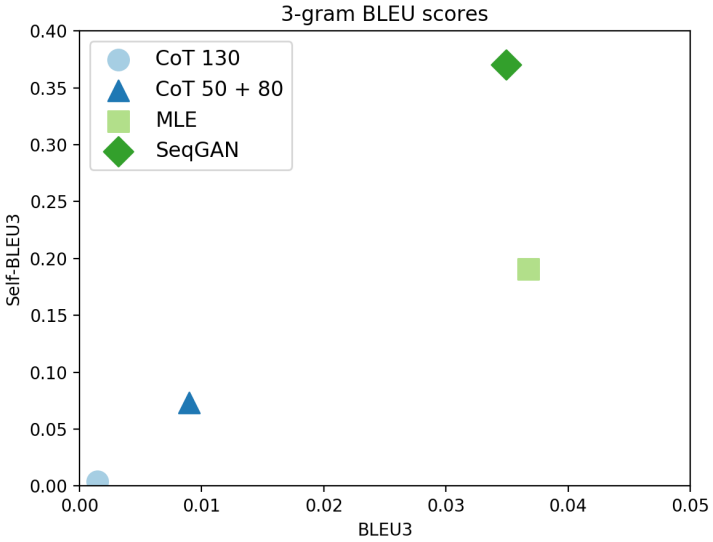

Figure 4.2: BLEU3 and self-BLEU3 results.

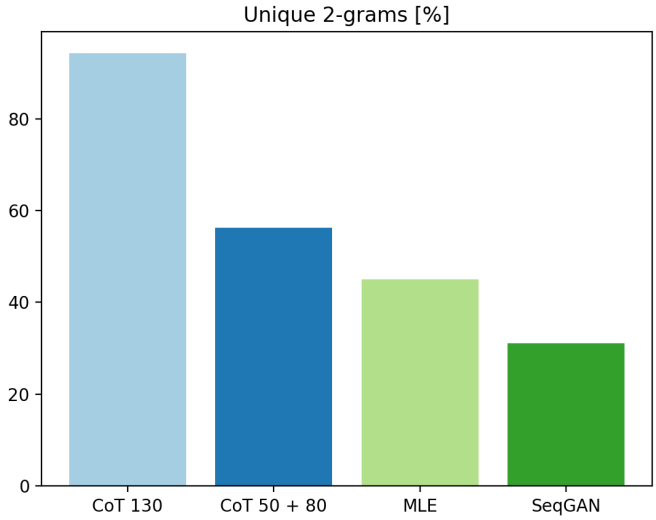

Figure 4.3: The number of unique 2-grams for each model.

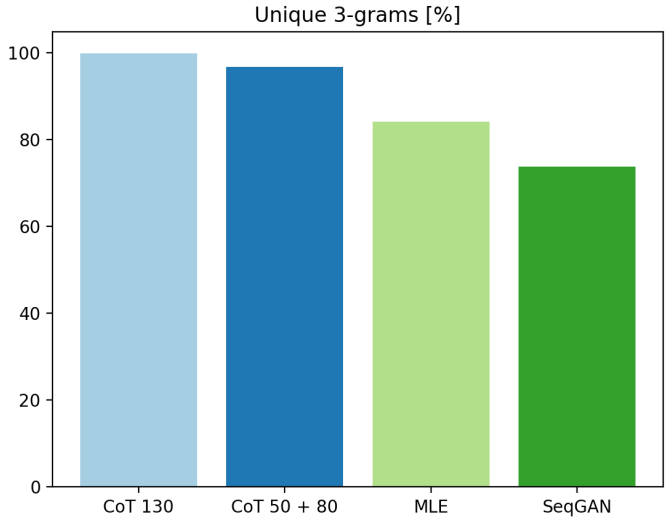

Figure 4.4: The number of unique 3-grams for each model.

Table 4.3: Results from language modeling evaluations.

	CoT 130	**CoT 50+80**	**MLE**	**SeqGAN**
LM Score	16.25	9.78	9.15	8.12
Reverse LM Score	8.44	8.76	8.83	9.34

4.2 Language Model-Based Metrics

First, a language model was trained on the validation set using teacher forcing, as described in subsection 2.2.3 and subsection 2.4.1. The model was an LSTM neural network with a hidden layer size of 32. For each set of texts generated by the evaluated models, the text was passed through the trained language model. The output negative log likelihood per word from every model is reported as the Language Model (LM) score in Table 4.3.

Secondly, as described in subsection 3.3.3, a similarly designed language model was trained on the output data from each of the models. The trained model was then evaluated by passing the validation set sentences through it. The result is reported as the Reverse LM score in Table 4.3.

A visual representation of the LM scores can be seen in Figure 4.5.

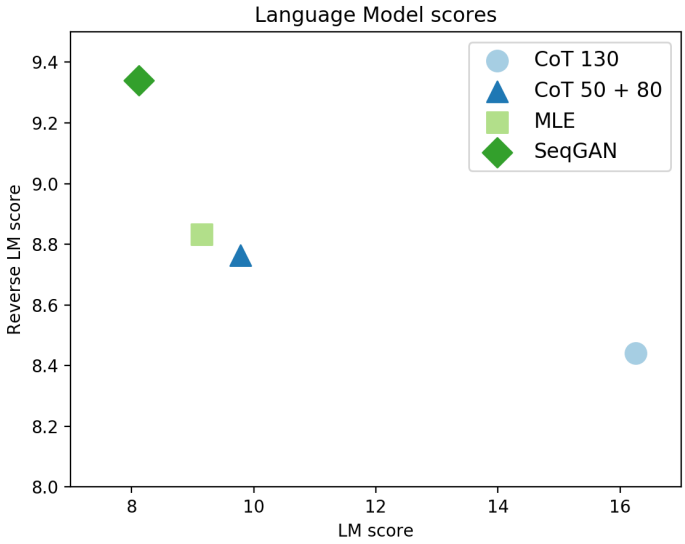

Figure 4.5: Language model scores for each model.

Table 4.4: Average scores assigned by human judges.

	CoT 130	CoT 50+80	MLE	SeqGAN
Human Evaluation Score	2.02	2.71	3.38	3.41

4.3 Human Evaluation

The human judges were given the same set of samples, 100 from each model along with 100 sentences from the validation set, with a randomized order. The judges were asked to assign a score between 1 and 5 as described in subsection 3.3.4. The average score for each set is reported in Table 4.4. For reference, the validation set received a score of 4.95.

4.4 Result analysis

Looking at the overview of all results, as can be seen in Table 4.5, there are great differences between the models. For each metric, the metric name is followed by either an ↑ symbol, meaning that a higher value of the metric is better, or a ↓ symbol which indicates the opposite. For

Table 4.5: Overview of results per model.

Metric	CoT 130	CoT 50+80
BLEU3 ↑	0.001477	0.008992
Self-BLEU3 ↓	0.003778	0.07301
Unique 3-grams ↑	99.79%	96.76%
LM Score ↓	16.25	9.78
Reverse LM Score ↓	8.44	8.76
Human Evaluation ↑	2.02	2.71
	MLE	**SeqGAN**
BLEU3 ↑	0.03675	0.03491
Self-BLEU3 ↓	0.1908	0.3705
Unique 3-grams ↑	84.14%	73.80%
LM Score ↓	9.15	8.12
Reverse LM Score ↓	8.83	9.34
Human Evaluation ↑	3.38	3.41

Table 4.6: Reference scores from a fully randomly generated text.

Metric	Random model
BLEU3 ↑	0.000
Self-BLEU3 ↓	0.000
Unique 3-grams ↑	100.00%
LM Score ↓	18.66
Reverse LM Score ↓	24.44
Human Evaluation ↑	-

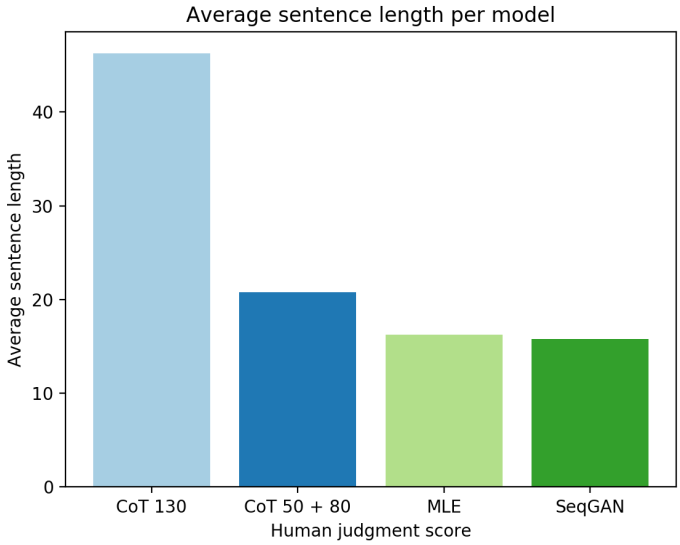

Figure 4.6: The models creating more diverse samples also generated longer sentences.

reference, in Table 4.6 the scores for the fully random model is presented.

While MLE and SeqGAN has a comparable BLEU3 score, the pre-trained CoT has a far lower score, and CoT without pre-training has a score less than a sixth of the pre-trained model in turn. This means that the word subsequences which CoT without pre-training generate appear very rarely in the test set compared to the texts generated by the other models.

Looking at the average sentence length per model in Figure 4.6, it is apparent that the CoT without pre-training tends to generate very long sentences and that the words often do not relate to the already generated. The model generates long sentences because it has not learned how likely it should be to reach the end state. Still, the model does reach the end state at some occasions and never returns from it, which suggests that the learning might be improved by longer training times and more training data. In Table 4.7, selected samples from each model is presented. The samples from the pre-trained CoT have been cut to fit the page width. Note that all samples in Table 4.7 are manually chosen for their quality or absurdity.

The two CoT models have a self-BLEU much lower than the MLE

Table 4.7: Selected samples from all models.

CoT 130
menar den äganderätten som som statsskick vår världsklass kortsiktiga grunder utbildning kvinnor präglar det möjligheter hyrläkare minskande effektiva företagen mått samhällsförändringar [...]
för högskolan ska vara bättre och riktigt det bakom område samhälle förstärks system samhällets egna förutom regleringar riksdagen politik ett bygga anställning radikalt frihetens väl [...]
CoT 50+80
vi använder också grunden för ökad kommunikationer, bra bär vårt inslag, åt jämställt, högre ett sin staten och neutral, och respekt som rätt att säkerhet
etiken erkänner bättre fängelseritningar
MLE
gemensamma angelägenheter berör alla med arbete och bilda opinion, utbildning och värderingar, och därmed producera 000 olika perspektiv och gör att göra det lite undervisningstid
om de myndigheter och förskolan ska få tillgång till sin egen existens dialog
SeqGAN
möjlighet till exempel lokala och rättigheter samt valfrihet och moderaterna måste behandlas med rätt till trygghet och goda möjligheter till återhämtning och populism med utsläppsrätter
statens behov av utbyte mot andra inslag drabbar vapenaffärer till självbestämmande

Table 4.8: Generated samples which fooled the human evaluations.

sverige ska utgå från utmaningar
ingen ekonomi
även fortsättningsvis sett ökar behovet av den exploaterande rasismen i hela landet
svensk grundläggande kärnkonventioner och globala nätverk mot folkstyre
arbetsmarknadspolitiken ska mötas kring ett gemensamt ansvar för barnen
varje människa är förpliktigad att utvecklas till framtida livsval

model and the SeqGAN model. This suggests that they are able to generate significantly more diverse texts, but a purely random model would have a good self-BLEU score as well. In fact, the fully randomized model seen in Table 4.6 gives a perfect self-BLEU3. However, the quality measures are all worse for the random model compared to the CoT 130, and the reverse language model score is far lower. The samples from CoT without training are diverse, but with a too low quality, a higher diversity doesn't really add anything to the model.

Of more interest is the self-BLEU of the pre-trained CoT, particularly in comparison with the SeqGAN model. Both models had 50 epochs of MLE pre-training, but the cooperative training produced remarkably more diverse results than the adversarial training, which had almost twice as high self-BLEU3 as the pure MLE model. Another example of this is the percentage of unique 3-grams. This suggests that CoT has managed to avoid the mode collapse which adversarial training might induce, as described in subsection 2.6.1.

Typically, the human judges were able to pick out close to all true samples, with an average score of 4.95. However, the models were able to generate some samples which fooled the human judges. 6% of the evaluated sentences passed as potentially human written. It is notable that almost all such sentences are short. Some examples can be found in Table 4.8.

SeqGAN had a slightly better score than MLE, while the pre-trained CoT got a lower score, which reinforces that the cooperative training decreased the text quality.

SeqGAN had the best LM score along with the best human evaluation score, which both model text quality. The pre-trained CoT has a comparable but worse result, while the CoT without pre-training has a very high LM score and significantly lower human evaluation score, reinforcing that the sentences generated by pure CoT were close to random words.

An interesting result is the reverse LM scores for all models. SeqGAN, which has good quality performance in general, is once again shown to score worse than pure MLE on a diversity metric. The pure CoT model is performing well, possibly due to the fact that it covers a larger part of the vocabulary than the other models, but to less extent than other diversity metrics. In the reverse LM metric, the pre-trained CoT once again outperforms the pure MLE training in terms of diversity.

Overall, the CoT without pre-training did not produce results well enough to compare its results with the other models. The pre-trained CoT outperformed the identically pre-trained SeqGAN when it comes to diversity, at the trade-off for a lower general text quality.

It should be noted that the human evaluation, as conducted in this study, is not a ground truth to ultimately compare models since it measures quality but not diversity.

Chapter 5

Discussion

5.1 Experiment Analysis

This section reflects upon the approach described in chapter 3 and the findings presented in chapter 4.

5.1.1 Metric Evaluation

Evaluating the metrics goes hand in hand with evaluating the choice of dataset and the methodology. If the dataset is not suitable, this might be apparent for some metrics more than others.

The results of this study reinforce the findings of Semeniuta, Severyn, and Gelly [40] regarding the insufficiency of BLEU as a metric for evaluating text generating models. The BLEU scores match the quality assessment by the human judges fairly well, but underestimate the relative quality of the pre-trained CoT compared to the baseline MLE model.

The reverse LM score is an interesting metric and can detect many forms of mode collapse, as described by Semeniuta, Severyn, and Gelly [40]. Their main issue with the metric is that it is very computationally expensive, and thus cannot be efficiently used for tuning. Further, they notice that the metric can be sensitive to sample quality close to random samples. The results in this study show a discrepancy between the self-BLEU measure of mode collapse and the reverse LM. This suggests that there are mode collapses in the pre-trained CoT which the self-BLEU metric has not detected. The high quotient of unique n-grams in samples from the pre-trained CoT also

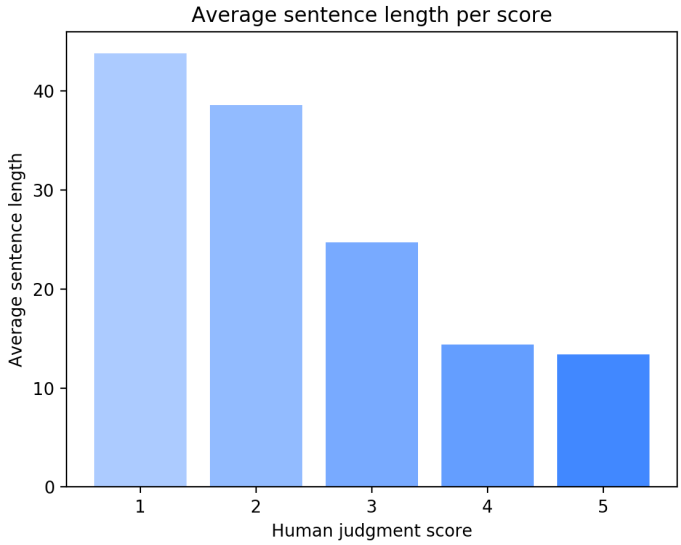

Figure 5.1: The perceived quality was higher the shorter the sentences were.

suggests that mode collapse is rare, but one possibility could be that longer dependencies are reoccurring rather than n-grams.

None of the metrics can capture both quality and diversity, which is why at least two measures should be used when comparing models. While these metrics can accurately capture text quality or diversity, there is one major problem. As found by Semeniuta, Severyn, and Gelly [40], none of the proposed metrics can detect overfitting. This means that a model which simply remembers all samples and recreates them would score perfectly in theory, given large enough training and test datasets.

5.1.2 Choice of Dataset

It was apparent from the human judgment that the longer the generated sentences where, the lower the perceived quality was. In fact, the average sentence length for the samples which received the lowest score was close to 44, meaning that almost all the worst samples did not find the final state. This can be seen in Figure 5.1, along with the general trend that shorter sentence lengths leads to higher perceived quality.

This suggests that all models weren't suited for generating long

sequences. Naturally, sentences as long as 44 words require a lot more learning than a sentence consisting of 5 words. In hindsight, a more fair judgment could have been done with either shorter sentence lengths or more training data. Some similar studies, such as the comparative study by Lu et al. [23], have used the COCO Image Caption dataset[1] which consists of 10 000 training data samples, the same number as the dataset used for this study. However, the average sentence length of the COCO dataset is 10.47 words, while the used dataset has an average length of 14.97 words. Others, such as Semeniuta, Severyn, and Gelly [40], have used datasets with longer sentences, but up to 600 000 unique sentences.

Further, the dataset consists of political text. Many bureaucratic terms are used, and a wide vocabulary. Some words appear very few times, and reoccurring combinations are even more sparse.

5.1.3 Methodology

Using a dataset with a large vocabulary does not have to be a problem. However, in this study, this led to two complications. As discussed in the previous segment, the model might not learn how to correctly use a term if it is very rare. Given a large enough dataset the system could still learn, but if a word only appears once in the training data, how to use the word in different contexts cannot be learned. Secondly, a larger vocabulary increases the training times and the resources needed. The original models by Zhu et al. [25] and Lu et al. [39] did not limit the vocabulary size, but since the resources turned out to be a limiting factor in this project it could have been done. Better trained models could arguably have been worth the trade-off to diverge from the original sources.

Cheaper computations would also have allowed extensive hyperparameter searches. Semeniuta, Severyn, and Gelly [40] shows that the generative adversarial text models can be very sensitive to random initialization and hyperparameter tuning. Pre-training reduces variance, but it has not been formally evaluated to what extent. For each model, they conduct 100 trials to find the optimal hyperparameters via random search, and train the models 7 times with the optimal hyperparameters. The results are then averaged over all trials. The general trend that Semeniuta, Severyn, and Gelly [40] found was that

[1] http://cocodataset.org/

lower learning rates were profitable, which suggests that the adversarial learning is actually detrimental. However, in general there is a lack of insight into most papers in the field when it comes to hyperparameter tuning and robustness. Most papers report a single score such as BLEU or perplexity as the model evaluation, which does not convey the full story, without mentioning any hyperparameter tuning [23][26][32]. This decreases the reproducibility as well as the validity of the results, due to the random initializations. An improvement of this study would need to find a cost-efficient work-around for this.

5.2 Conclusions

The latest advances in generative language modeling have focused on adversarial learning of neural networks. One of the advantages of adversarial learning is that it might relieve exposure bias. However, due to the discrete outputs of natural languages, there is no well-defined gradient to guide a GAN in the training. There have been different proposed workarounds, and one of the most influential models is the SeqGAN model by Yu et al. [26] which uses a reinforcement learning policy gradient for updating weights in the system. Many models based on the same idea have emerged, but none have produced results which are better across all categories. Typically, the GANs can produce good results but suffer from mode dropping, where the output is not diverse enough.

Recently, a novel approach called CoT was proposed by Lu et al. [39], which instead of training an adversary trains a cooperative mediator. This mediator is used to estimate the Jensen-Shannon divergence. The authors findings are remarkable, but a consistent issue with generative language models is how to evaluate and compare models. Most metrics can measure either text quality or diversity. To answer how to do a fair comparison between different models, Semeniuta, Severyn, and Gelly [40] analyzed the robustness of different metrics and their correlation to human judgment, to create a standard protocol for comparing models.

In this study, a SeqGAN model was compared to a pre-trained CoT model and a CoT model without pre-training. A baseline LSTM model trained only with MLE was also compared. The models were trained

on a dataset consisting of texts written by the eight political parties of the Swedish Riksdag.

The CoT model without pre-training seemed not to learn non-random behavior to a significant degree. The model output was diverse, but the quality metric results were poor. The CoT with pre-training and the SeqGAN had superior performances. The SeqGAN created outputs of high quality, but had a lacking diversity. The CoT instead had very diverse results, but could be more easily distinguished from true texts by the human judges. The MLE baseline performance was typically in between the two models, but for n-gram quality measures the SeqGAN did not actually outperform the MLE. However, the more reliable measures of quality, human evaluation and LM score, supported that the adversarial training actually improved the quality.

In general, the human judges were able to pick out the true samples from the fake. However, in rare cases, the models were able to fool the humans by generating political sentences which seemed to have come from a human writer. The models were typically only able to do so for short sequences, and there was a correlation between sequence length and perceived quality.

The protocol proposed by Semeniuta, Severyn, and Gelly [40] contained a metric of the reversed language model perplexity, where a model was trained on the output of a network and held-out sentences from the true dataset where evaluated by the model. In this study, the metric assigned a good score to the CoT without pre-training, which created poor quality samples. This bias of the metric can occur due the lack of training data, but also suggests that the metric is more useful for comparing models of sufficient quality.

The findings of Semeniuta, Severyn, and Gelly [40] was that lower adversarial learning rates were profitable, which means that the adversarial training did not improve the models. The findings of this study suggest that depending on the application, adversarial or cooperative training can be beneficial. Adversarial training after pre-training gave higher quality samples, at the cost of diversity. Cooperative training gave more diverse results, at the cost of text quality.

5.3 Future Work

CoT is a novel approach which has not yet been thoroughly peer reviewed. To build upon the concept, the findings should be fully reproduced, independent from the original authors. The article does not explain hyperparameter tuning procedures or how the inherent instability has been handled. A starting point is to use the same dataset as the original article, an extensive search for hyperparameters followed by several trials, and results averaged over those trials. This should be done for both oracle data and real datasets.

Further, the free-running mode of text generation serves primarily academic or scientific purposes. The real-world applications are typically conditional models, where the model creates text based on some input, such as a sentence to translate, a topic or an image.

The reverse language model metric is an interesting measure which can detect several forms of mode collapse. However, to validate the use as one of the primary language model evaluation metrics, further studies should be conducted to theoretically understand what it shows and empirically support that it is suitable to optimize.

Finally, while Semeniuta, Severyn, and Gelly [40] conducted an extensive study and comparison of current metrics, the search continues for tractable metrics which capture quality, diversity and detect overfitting.

Bibliography

[1] A. M. Turing. "Computing Machinery and Intelligence". English. In: *Mind*. New Series 59.236 (1950), pp. 433–460. ISSN: 00264423. URL: http://www.jstor.org/stable/2251299 (cit. on p. 1).

[2] Alex Graves. "Generating Sequences With Recurrent Neural Networks." In: *CoRR* abs/1308.0850 (2013). URL: http://dblp.uni-trier.de/db/journals/corr/corr1308.html#Graves13 (cit. on p. 2).

[3] Jochen L. Leidner and Vassilis Plachouras. "Ethical by Design: Ethics Best Practices for Natural Language Processing". In: *Proceedings of the First ACL Workshop on Ethics in Natural Language Processing*. Valencia, Spain: Association for Computational Linguistics, Apr. 2017, pp. 8–18. URL: http://www.aclweb.org/anthology/W/W17/W17-1602 (cit. on p. 4).

[4] Yoav Goldberg and Graeme Hirst. *Neural Network Methods in Natural Language Processing*. Morgan & Claypool Publishers, 2017. ISBN: 1627052984 (cit. on pp. 5, 8–12).

[5] Daniel Jurafsky and James H. Martin. *Speech and Language Processing (2Nd Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2009. ISBN: 0131873210 (cit. on pp. 7–10).

[6] Joshua Goodman. "A Bit of Progress in Language Modeling". In: *CoRR* cs.CL/0108005 (2001). URL: http://arxiv.org/abs/cs.CL/0108005 (cit. on p. 8).

[7] Yoshua Bengio et al. "A Neural Probabilistic Language Model". In: *JOURNAL OF MACHINE LEARNING RESEARCH* 3 (2003), pp. 1137–1155 (cit. on pp. 8, 20).

[8] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors". In: *Nature* 323 (Apr. 1986), pp. 533–. URL: http://dx.doi.org/10.1038/323533a0 (cit. on p. 10).

[9] Jeffrey L. Elman. "Finding structure in time". In: *COGNITIVE SCIENCE* 14.2 (1990), pp. 179–211 (cit. on p. 11).

[10] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Comput.* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. URL: http://dx.doi.org/10.1162/neco.1997.9.8.1735 (cit. on p. 12).

[11] Ronald J. Williams and David Zipser. "A Learning Algorithm for Continually Running Fully Recurrent Neural Networks". In: *Neural Comput.* 1.2 (June 1989), pp. 270–280. ISSN: 0899-7667. DOI: 10.1162/neco.1989.1.2.270. URL: http://dx.doi.org/10.1162/neco.1989.1.2.270 (cit. on pp. 12, 13).

[12] Ian J. Goodfellow. "NIPS 2016 Tutorial: Generative Adversarial Networks". In: *CoRR* abs/1701.00160 (2016) (cit. on pp. 13–18, 24, 33).

[13] Yonghui Wu et al. "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation". In: *CoRR* abs/1609.08144 (2016). URL: http://arxiv.org/abs/1609.08144 (cit. on p. 14).

[14] Shakir Mohamed and Balaji Lakshminarayanan. "Learning in Implicit Generative Models". In: *ArXiv e-prints*, arXiv:1610.03483 (Oct. 2016), arXiv:1610.03483. arXiv: 1610.03483 [stat.ML] (cit. on p. 14).

[15] Yoshua Bengio et al. "Deep Generative Stochastic Networks Trainable by Backprop". In: *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*. ICML'14. Beijing, China: JMLR.org, 2014, pp. II-226–II-234. URL: http://dl.acm.org/citation.cfm?id=3044805.3044918 (cit. on p. 15).

[16] Ian Goodfellow et al. “Generative Adversarial Nets”. In: *Advances in Neural Information Processing Systems 27*. Ed. by Z. Ghahramani et al. Curran Associates, Inc., 2014, pp. 2672–2680. URL: http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf (cit. on pp. 17–19).

[17] Tomas Mikolov et al. “Recurrent neural network based language model.” In: *INTERSPEECH*. Ed. by Takao Kobayashi, Keikichi Hirose, and Satoshi Nakamura. ISCA, 2010, pp. 1045–1048. URL: http://dblp.uni-trier.de/db/conf/interspeech/interspeech2010.html#MikolovKBCK10 (cit. on p. 21).

[18] Y. Bengio, P. Simard, and P. Frasconi. “Learning Long-term Dependencies with Gradient Descent is Difficult”. In: *Trans. Neur. Netw.* 5.2 (Mar. 1994), pp. 157–166. ISSN: 1045-9227. DOI: 10.1109/72.279181. URL: http://dx.doi.org/10.1109/72.279181 (cit. on p. 21).

[19] Samy Bengio et al. “Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks”. In: *CoRR* abs/1506.03099 (2015). arXiv: 1506.03099. URL: http://arxiv.org/abs/1506.03099 (cit. on p. 22).

[20] Yoshua Bengio et al. “Curriculum Learning”. In: *Proceedings of the 26th Annual International Conference on Machine Learning*. ICML ’09. Montreal, Quebec, Canada: ACM, 2009, pp. 41–48. ISBN: 978-1-60558-516-1. DOI: 10.1145/1553374.1553380. URL: http://doi.acm.org/10.1145/1553374.1553380 (cit. on p. 22).

[21] F. Huszár. “How (not) to Train your Generative Model: Scheduled Sampling, Likelihood, Adversary?” In: *ArXiv e-prints* (Nov. 2015). arXiv: 1511.05101 [stat.ML] (cit. on pp. 23, 24).

[22] Kishore Papineni et al. “BLEU: A Method for Automatic Evaluation of Machine Translation”. In: *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*. ACL ’02. Philadelphia, Pennsylvania: Association for Computational Linguistics, 2002, pp. 311–318. DOI: 10.3115/1073083.1073135. URL: https://doi.org/10.3115/1073083.1073135 (cit. on p. 23).

[23] Sidi Lu et al. “Neural Text Generation: Past, Present and Beyond”. In: *CoRR* abs/1803.07133 (2018). arXiv: 1803.07133. URL: http://arxiv.org/abs/1803.07133 (cit. on pp. 23, 27, 29, 30, 32, 33, 38, 40, 56, 57).

[24] Sai Rajeswar et al. “Adversarial Generation of Natural Language”. In: *CoRR* abs/1705.10929 (2017). arXiv: 1705.10929. URL: http://arxiv.org/abs/1705.10929 (cit. on p. 23).

[25] Yaoming Zhu et al. “Texygen: A Benchmarking Platform for Text Generation Models.” In: *SIGIR*. Ed. by Kevyn Collins-Thompson et al. ACM, 2018, pp. 1097–1100. URL: http://dblp.uni-trier.de/db/conf/sigir/sigir2018.html#ZhuLZGZWY18 (cit. on pp. 23, 30, 38, 41, 56).

[26] Lantao Yu et al. “SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient”. In: *CoRR* abs/1609.05473 (2016). arXiv: 1609.05473. URL: http://arxiv.org/abs/1609.05473 (cit. on pp. 24, 26, 40, 57).

[27] Alex M Lamb et al. “Professor Forcing: A New Algorithm for Training Recurrent Networks”. In: *Advances in Neural Information Processing Systems 29*. Ed. by D. D. Lee et al. Curran Associates, Inc., 2016, pp. 4601–4609. URL: http://papers.nips.cc/paper/6099-professor-forcing-a-new-algorithm-for-training-recurrent-networks.pdf (cit. on p. 24).

[28] William Fedus, Ian Goodfellow, and Andrew Dai. “MaskGAN: Better Text Generation via Filling in the ____”. In: 2018. URL: https://openreview.net/pdf?id=ByOExmWAb (cit. on pp. 25, 30, 39).

[29] Philip Bachman and Doina Precup. “Data Generation as Sequential Decision Making”. In: *CoRR* abs/1506.03504 (2015). arXiv: 1506.03504. URL: http://arxiv.org/abs/1506.03504 (cit. on pp. 25, 26).

[30] Ronald J. Williams. “Simple statistical gradient-following algorithms for connectionist reinforcement learning”. In: *Machine Learning*. 1992, pp. 229–256 (cit. on p. 26).

[31] Richard S Sutton et al. "Policy Gradient Methods for Reinforcement Learning with Function Approximation". In: *Advances in Neural Information Processing Systems 12*. Ed. by S. A. Solla, T. K. Leen, and K. Müller. MIT Press, 2000, pp. 1057–1063. URL: http://papers.nips.cc/paper/1713-policy-gradient-methods-for-reinforcement-learning-with-function-approximation.pdf (cit. on p. 26).

[32] Tong Che et al. "Maximum-Likelihood Augmented Discrete Generative Adversarial Networks". In: *CoRR* abs/1702.07983 (2017). arXiv: 1702.07983. URL: http://arxiv.org/abs/1702.07983 (cit. on pp. 27, 28, 57).

[33] Kevin Lin et al. *Adversarial Ranking for Language Generation*. 2017. arXiv: 1705.11001 [cs.CL] (cit. on pp. 28, 29).

[34] Tie-Yan Liu. "Learning to Rank for Information Retrieval". In: *Found. Trends Inf. Retr.* 3.3 (Mar. 2009), pp. 225–331. ISSN: 1554-0669. DOI: 10.1561/1500000016. URL: http://dx.doi.org/10.1561/1500000016 (cit. on p. 28).

[35] Jiaxian Guo et al. *Long Text Generation via Adversarial Training with Leaked Information*. 2017. arXiv: 1709.08624 [cs.CL] (cit. on pp. 29, 30).

[36] Alexander Sasha Vezhnevets et al. *FeUdal Networks for Hierarchical Reinforcement Learning*. 2017. arXiv: 1703.01161 [cs.AI] (cit. on p. 29).

[37] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. *Sequence to Sequence Learning with Neural Networks*. 2014. arXiv: 1409.3215 [cs.CL] (cit. on p. 30).

[38] Martín Arjovsky and Léon Bottou. "Towards Principled Methods for Training Generative Adversarial Networks". In: *CoRR* abs/1701.04862 (2017). URL: http://arxiv.org/abs/1702.07983 (cit. on pp. 32, 34).

[39] Sidi Lu et al. *CoT: Cooperative Training for Generative Modeling of Discrete Data*. 2018. arXiv: 1804.03782 [cs.LG] (cit. on pp. 33–35, 56, 57).

[40] Stanislau Semeniuta, Aliaksei Severyn, and Sylvain Gelly. *On Accurate Evaluation of GANs for Language Generation*. 2018. arXiv: 1806.04936 [cs.CL] (cit. on pp. 38, 39, 42, 43, 54–59).

[41] Jules Gagnon-Marchand et al. *SALSA-TEXT : self attentive latent space based adversarial text generation*. 2018. arXiv: 1809.11155 [cs.CL] (cit. on p. 39).

[42] Junbo Jake Zhao et al. "Adversarially Regularized Autoencoders for Generating Discrete Structures". In: *CoRR* abs/1706.04223 (2017). arXiv: 1706.04223. URL: http://arxiv.org/abs/1706.04223 (cit. on p. 39).

[43] Alexis Conneau et al. "Supervised Learning of Universal Sentence Representations from Natural Language Inference Data". In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark: Association for Computational Linguistics, 2017, pp. 670–680. DOI: 10.18653/v1/D17-1070. URL: http://aclweb.org/anthology/D17-1070 (cit. on p. 42).

Appendix A

Human Judge Instructions

The following instructions were given to the human judges together with the 500 sentences to evaluate.

I filen "Human Evaluation" finns 100 samples vardera från de fyra olika modellerna som jämförs, och 100 samples från testdatan som kommer från partiernas politiska manifest. De har slumpmässig ordning.

Varje sample ska betygsättas från 1 till 5 där 5 innebär att det skulle kunna vara en människa som skrivit. 1 bör motsvara en helt slumpmässig ordföljd, till exempel denna som genererats efter bara en träningsepok: "informationstekniken bostadsbubbla valets fiske rätttvisare bränsle instegsjobben vågar förvaltning tesen farlig"

Eftersom modellerna är tränade på ganska långa meningar med ett stort ordförråd kommer de flesta inte att vara helt vettiga innehållsmässigt. Därför borde skalan 1-4 säga ungefär "hur orimlig är meningen", och en 4 behöver alltså inte betyda att ni tror att en människa skrivit meningen.

Utöver betyget har jag även med en kolumn där ni kan markera om det är en mening ni tycker är särskilt bra, rolig etc. Jag kommer välja några sådana att ha med som exempel på output i rapporten. Ni behöver inte välja någon alls om ni inte vill.

Appendix B

Data Sources

All documents are official documents available from the respective parties websites, as of 2018-12-03.

The full set of data sources was as follows:

- Centerpartiet - Idéprogram 2013, as of 2018-12-03 available at: https://www.centerpartiet.se/download/18.145b416915819de0fb62e10f/1478006852784/H%C3%A4r-kan-du-l%C3%A4sa-hela-id%C3%A9programmet.pdf
- Centerpartiet - Valplattform 2018, as of 2018-12-03 available at: https://www.centerpartiet.se/val-2018/valplattform-2018
- Kristdemokraterna - Principprogram 2015, as of 2018-12-03 available at: https://kristdemokraterna.se/wp-content/uploads/2016/11/Kristdemokraternas-principprogram.pdf
- Kristdemokraterna - Valmanifest 2018, as of 2018-12-03 available at: https://kristdemokraterna.se/wp-content/uploads/2016/11/Valmanifest-2018.pdf
- Liberalerna - Partiprogram 2017, as of 2018-12-03 available at: https://www.liberalerna.se/politik/politikpartiprogram/
- Liberalerna - Valmanifest 2018, as of 2018-12-03 available at: https://www.liberalerna.se/wp-content/uploads/valmanifest.pdf

- Liberalerna - Valplan 2018, as of 2018-12-03 available at: https://www.liberalerna.se/wp-content/uploads/valplan-2018.pdf
- Liberalerna - Vår politik 2017, as of 2018-12-03 available at: https://www.liberalerna.se/politik/politikpartiprogram/
- Miljöpartiet - Partiprogram 2013, as of 2018-12-03 available at: https://www.mp.se/sites/default/files/miljopartiets_partiprogram_lagupplost_2013.pdf
- Miljöpartiet - Valmanifest 2018, as of 2018-12-03 available at: https://www.mp.se/politik/valmanifest2018
- Moderaterna - Handlingsprogram 2017, as of 2018-12-03 available at: https://moderaterna.se/sites/default/files/page_attachments/2017-02/handlingsprogram_-_ett_modernt_arbetarparti_for_hela_sverige_2013_2.pdf
- Moderaterna - Idéprogram 2017, as of 2018-12-03 available at: https://moderaterna.se/sites/default/files/page_attachments/2017-06/ideprogram_-_ansvar_for_hela_sverige_2013_1.pdf
- Moderaterna - Valmanifest 2018, as of 2018-12-03 available at: https://moderaterna.se/sites/default/files/page_attachments/2018-08/Valmanifest%20H%C3%96G%20180820.pdf
- Socialdemokraterna - Partiprogram 2013, as of 2018-12-03 available at: https://www.socialdemokraterna.se/globalassets/var-politik/partiprogram-och-riktlinjer/ett-program-for-forandring_2013.pdf
- Socialdemokraterna - Riktlinjer 2017, as of 2018-12-03 available at: https://www.socialdemokraterna.se/globalassets/var-politik/arkiv/kongress-2017/trygghet-i-en-ny-tid---politiska-riktlinjer.pdf

- Socialdemokraterna - Valmanifest 2018, as of 2018-12-03 available at: https://www.socialdemokraterna.se/globalassets/aktuellt/valmanifest-2018.pdf
- Sverigedemokraterna - Principprogram 2011, as of 2018-12-03 available at: https://sd.se/wp-content/uploads/2013/08/principprogrammet2014_webb.pdf
- Sverigedemokraterna - Valmanifest 2018, as of 2018-12-03 available at: https://sd.se/wp-content/uploads/2018/08/Valmanifest-2018-1.pdf
- Vänsterpartiet - Partiprogram 2016, as of 2018-12-03 available at: https://www.vansterpartiet.se/app/uploads/2017/06/Partiprogram_V-1jun2017.pdf
- Vänsterpartiet - Valplattform 2018, lättläst, as of 2018-12-03 available at: https://www.vansterpartiet.se/app/uploads/2018/06/Valplattform_lattlast_A4.pdf
- Vänsterpartiet - Valplattform 2018, as of 2018-12-03 available at: https://www.vansterpartiet.se/app/uploads/2018/04/Valplattform-2018.pdf

TRITA -EECS-EX