



Norwegian University of
Science and Technology

A Continuous Approach to Controllable Text Generation using Generative Adversarial Networks

Dag Inge Helgøy
Markus Lund

Master of Science in Computer Science
Submission date: June 2017
Supervisor: Massimiliano Ruocco, IDI
Co-supervisor: Helge Langseth, IDI

Norwegian University of Science and Technology
Department of Computer Science

Abstract

The challenges of training generative adversarial network (GAN) to produce discrete tokens, have seen a considerable amount of work in the past year. However, the amount of successful work on applying deep generative models to text generation is limited, when compared to the visual domain. One of the reasons, is the challenge of passing the gradient, while keeping the network differentiable. The known effective models that generate text with GAN, extend the original framework proposed by Goodfellow et al. [2014], using reinforcement learning.

We propose a novel approach that requires no modification to the training process introduced by Goodfellow et al. [2014], and in addition is able to produce meaningful text without any pre-training. Our approach is not limited to the textual domain, and could be able to solve a variety of problems related to generating sequences of discrete tokens. We show a proof of concept for an image captioning model that extends our text generator to perform controllable text generation. This approach provides a straightforward method for controlling text generation using images, or any other vector representing relevant information.

Evaluating results produced by GANs is a common problem. The absence of real data for each corresponding sample generated from the network, makes the appliance of common evaluation methods difficult. As a solution to this challenge, we have developed an automatic evaluation method for text generative systems. This method combines the machine learning evaluation metric, bilingual evaluation underway (BLEU), with a set of interchangeable information retrieval techniques. This permits us to evaluate the semantic quality of our models, as well as comparing them to a baseline.

Sammendrag

Utfordringene med å trenere generative motstridende nettverk til å produsere diskrete tegn, har fått mye oppmerksomhet det siste året. Imidlertid er mengden veldig krevende arbeid med å anvende dype generative modeller til tekstgenerering begrenset, sammenlignet med det visuelle domenet. En av grunnene til dette er utfordringen med å propagere gradienten, samtidig som nettverket er deriverbart. De kjente effektive modellene som produserer tekst med generative motstridende nettverk, utvider det originale rammeverket, introdusert av Goodfellow et al. [2014], ved hjelp av forsterkende læring.

Vi foreslår en ny tilnærming som ikke krever modifikasjon av treningsprosessen introdusert av Goodfellow et al. [2014], og i tillegg er i stand til å produsere meningsfull tekst uten å forhåndstrene nettverkene. Vårt forslag er ikke begrenset til det språklige domene, og kan være i stand til å løse en rekke problemer knyttet til generering av sekvenser av diskrete tegn. Vi viser et konsept for en bildebeskrivelsemodell som utvider vår tekstgenerator til å utføre kontrollerbar tekstgenerering. Denne tilnærmingen gir en enkel metode for å kontrollere tekstuering ved hjelp av bilder, eller en hvilken som helst annen vektor som representerer relevant informasjon.

Evaluering av resultater produsert av generative motstridende nettverk er et kjent problem. Fraværet av ekte data for hvert eksempel generert av nettverket, gjør evaluering vanskelig. Som en løsning på dette problemet har vi utviklet en automatisk evalueringsmetode for tekstueringssystemer. Denne metoden kombinerer en evalueringssmetode for maskinoversettelse med utskiftbare informasjonskjefinningsteknikker. Dette tillater oss å evaluere den semantiske kvaliteten til våre modeller, samt sammenligne dem med en referansemodell.

Preface

This Master's thesis was written by Dag Inge Helgøy and Markus Lund during the spring of 2017. The project is the final delivery of the Master of Science (MSc) degree in Computer Science at the Department of Computer Science (IDI) at the Norwegian University of Science and Technology (NTNU). We would like to thank Massimiliano Ruocco and Helge Langseth for the guidance during this thesis and for providing a good environment for exchanging ideas with them and our fellow students.

Dag Inge Helgøy, Markus Lund
Trondheim, June 11, 2017

List of Figures

2.1	Single artificial neuron	5
2.2	Feedforward neural network	6
2.3	Unfolded recurrent neural network with output for every timestep .	8
2.4	Unfolded recurrent neural network with output only at the last timestep	9
2.5	Long short-term memory cell	10
2.6	Convolutional neural network	10
2.7	Word2Vec vector composition	11
2.8	Word2Vec training process	12
2.9	Word2Vec word embedding visualization	13
2.10	Word2Vec architectures	13
2.11	Principal Component Analysis	15
2.12	Autoencoder	16
2.13	Generative adversarial network overview	17
2.14	Comparison of mean squared error and adversarial techniques	18
2.15	GAN image-to-image translation	19
2.16	Word2VisualVec and DeViSE architectures	22
2.17	Sequence-to-sequence encoder-decoder example	24
4.1	Word2VisualVec image retrieval	33
4.2	GAN for image generation	34
4.3	GAN MNIST comparison	35
4.4	Generated images trained on the custom figures dataset	36
4.5	GAN image generation variety	37
5.1	SeqGAN model	42
5.2	Proposed GAN text generator architecture	43

5.3	Text-to-image GAN architecture	45
5.4	Proposed GAN image captioning architecture	46
6.1	One corresponding Flickr30k caption example	48
6.2	One corresponding Oxford-102 flower caption example	49
6.3	Proposed text generation evaluation	53
6.4	Word mover's distance	53
6.5	Comparison of β score for softmax distribution models	58
6.6	β and loss for reference and discriminator dropout models	61
6.7	β and loss for one-hot transformation and discriminator one-hot transformation/dropout models	62
6.8	Comparison of dropout ration on the word embedding model	64
6.9	Comparison of complexity ration on the word embedding model	66
6.10	Comparison of Word2Vec and GloVe for the word embedding model	68
6.11	Comparison of word embedding model epochs	69
6.12	Comparison of word embedding model and baseline	70
6.13	Example images of three flower species	73
6.14	Validation images for three flower species	74
6.15	Unfolded sequence-to-sequence model	77
6.16	Comparison of sequence-to-sequence normalized latent variables and gaussian distribution	78
6.17	Loss development of Sequence-to-sequence model as GAN initialization	80
A.1	Complete Word2Vec word embedding visualization	103

List of Tables

4.1	Word2VisualVec preliminary results	32
4.2	Character-based inference	37
4.3	Character-based training	38
4.4	Word-based training	38
4.5	Word-based inference	38
6.1	Text pre-processing	50
6.2	Examples of retrieval using a correct Oxford-102 flower sentence . .	54
6.3	Examples of retrieval using an erroneous Oxford-102 flower sentence	55
6.4	Examples of retrieval using an erroneous Flickr30k sentence	55
6.5	Comparison of retrieval methods using a correct Oxford-102 flower sentence	56
6.6	Comparison of retrieval methods using a correct Oxford-102 flower sentence	56
6.7	Comparison of retrieval methods using a correct Flickr30k sentence .	56
6.8	One-hot transformation	58
6.9	Reference model sentences	60
6.10	One-hot/dropout model sentences	60
6.11	One-hot transformation model sentences	60
6.12	One-hot transformation/dropout model sentences	60
6.13	Comparison of dropout ration on the word embedding model	65
6.14	Comparison of complexity ration on the word embedding model . .	67
6.15	Comparison of word embedding methods	68
6.16	Comparison of word embedding model epochs	69
6.17	β score comparison of models and datasets	71
6.18	Comparison of word embedding model and baseline	71
6.19	Sentences generated by the controllable text generation model . . .	74

6.20 Sentences generated by the image captioning model	75
6.21 Sentences generated by the image captioning model using validation data	76
6.22 Comparison of Sequence-to-sequence LSTM hidden units	77
6.23 Comparison of Sequence-to-sequence embedding dimension	78
6.24 Comparison of sequence-to-sequence embedding dimension	79
A.1 Comparison of dropout ratios on word embedding models	99
A.2 Complete examples of retrieval using a correct Oxford-102 Flower sentence	100
A.3 Complete examples of retrieval using an erroneous Oxford-102 flower sentence	101
A.4 Complete examples of retrieval using an erroneous Flickr30k sentence	102

Acronyms

- BGAN** boundary-seeking generative adversarial network. 27
- BLEU** bilingual evaluation understudy. 51–53, 56, 58, 59, 64, 83, 87
- CBOW** continuous bag of words. 12
- CNN** convolutional neural network. 1, 7, 9–11, 21, 23, 26, 42, 43, 45, 89, 90
- DeViSE** deep visual-semantic embedding model. 20, 21
- EM** earth moving. 29
- GAN** generative adversarial network. 1–5, 17–19, 25–29, 31, 32, 34–36, 38, 42, 43, 45–47, 68, 71, 72, 75–77, 79, 83, 84, 86–90
- GloVe** global vectors. 12, 31, 50, 62, 66, 67, 85
- HierSE** hierarchical semantic embedding. 20
- LSTM** long short-term memory. 1, 5, 8, 11, 23, 26, 27, 31, 32, 37, 38, 42–44, 65, 75, 76, 82, 90
- MLE** maximum likelihood estimation. 44
- MSE** mean squared error. 6, 18, 25, 75
- NLP** natural language processing. 19
- PCA** principal component analysis. 14–16, 72

R@K recall at k. 32

RNN recurrent neural network. 3, 7, 8, 11, 23, 89

RTT-GAN recurrent topic-transition generative adversarial network. 26

SeqGAN sequence generative adversarial nets. 26, 41, 42, 44, 69–71, 81, 84–87, 89

SGD stochastic gradient descent. 6, 42

t-SNE t-distributed stochastic neighbor embedding. 14, 79, 99

TF-IDF term frequency-inverse document frequency. 52, 54–56, 100–102

VAE variational autoencoder. 17, 45

VBN virtual batch normalization. 29

WGAN Wasserstein generative adversarial network. 29, 89

WMD word mover’s distance. 52, 54–56, 100–102

CHAPTER 1

Introduction

1.1 Background and Motivation

Initial work by Goodfellow et al. [2014] features an adversarial training process able to produce visually convincing images, and has brought about a myriad of extensions in the field of generative adversarial network (GAN). The proposed framework consists of two networks, one trained to generate realistic data and a second to discriminate between real and generated data. While GANs have been successfully used in computer vision [Radford et al., 2015; Lotter et al., 2016], less work has been dedicated to natural language applications.

The techniques used in these solutions often involve convolutional neural networks (CNNs). However, when the problem is both discrete and sequential, as it is with natural language, other techniques, such as maximum likelihood models often achieve better results [Bengio et al., 2015]. Basing the generator on long short-term memorys (LSTMs) allows us to test the sequence generation capability of GANs.

Work by Reed et al. [2016] shows that it is possible to let text drive content in generative image generation. By combining the latent variable from GANs with a dense representation of a textual input, new unseen images matching the inputs are generated. An interesting challenge based on the techniques introduced by Reed et al. [2016] and Goodfellow et al. [2014], is to generate a textual image description given an input image. The ability to automatically generate accurate descriptions of images is useful for many different applications and use cases. Automatic image captioning could help a vision impaired person to easier understand the content of images, categorize images based on visual features or improve information retrieval by adding the ability to do textual retrieval of unlabeled images.

However, an already significant challenge of using GAN for image captioning is the generation of text. Working with GANs for sequential discrete data generation comes with a series of challenges, and due to the discrete nature of generating text, it is difficult to use backpropagation to update the weights of the network. To enlighten upon this, we will describe two main challenges of generating text with

GANs; working with sequential data and the discrete representation of text.

1.1.1 Sequential Data

Generating long sequences of words often suffer from exponentially accumulating error [Bengio et al., 2015], since generated words are used to predict succeeding words. If the network generates word representations that are far from the representations of real words provided by the training data, this error would propagate to the next timestep of the sequence generation. This error could increase exponentially, and might cause the predictions of the network to reside in a space that has no real words. This issue is significant for GANs, as the discriminator is trained on the inferred sequences produced by the generator.

1.1.2 Discrete Data

Since GANs are designed to be used for continuous values such as images, discrete values such as text has proven to be more challenging to produce. The main challenge of working with discrete data and GANs is the inability to backpropagate the gradient from the discriminator to the generator. This prevents the network's ability to fine-tune its weights, which is dependent on the gradient's ability to provide accurate updates. When the error is simply based on a word not being correct, the granularity of the error is large. If the predicted word in a sequence is "man", when the correct word is "boy", this error would simply state that the word "man" is incorrect. This method does, however, not state to what degree the word "man" is related to the word "boy". There is no word that corresponds to "almost boy", which means that the gradient provides no fine-tuned feedback to the network.

1.2 Goals and Research Questions

Goal Produce descriptive image captions using GANs

The goal of this thesis is to produce text describing the content of an image using the GAN framework. This consists of generating an end-to-end system that is able to produce sequences of words and introducing images to both the generator and discriminator. This would allow the discriminator to provide feedback on whether the generated sequence is meaningful, as well as whether the sequence is descriptive of the given image.

Research question 1 Is there an effective method for dealing with the discrete and sequential structure of natural language when working with GANs?

Research question 2 Is the generation of natural language using GANs able to be controlled using images?

1.3 Research Method

To answer Research Question 1, a text generation model will be implemented and trained using an adversarial approach. This GAN text generator will need to base its training approach on prior studies, utilizing the techniques established in the research of GANs. To train a discrete and sequential GAN, requires a novel approach due to the lack of published work on similar models. In this work, we will present an architecture that represents words as continuous values and uses recurrent neural networks (RNNs) to generate text. We call this approach “The Continuous Approach”. Controlling GANs, being an already difficult task, requires an extension of the previously presented approach. Due to the challenge of evaluating GANs, we will apply a proposed evaluation method, as well as observation.

1.4 Contributions

We propose a solution for generating text by combining standard machine learning techniques with GAN, and an automatic evaluation method for evaluating such models. The text generative model is extended to create a novel image captioning model. The experiments demonstrates the viability of our new approach to generating text with GAN. The currently known effective models that generate text with GAN extends the original framework proposed by Goodfellow et al. [2014] with reinforcement learning. We present a novel solution that requires no modification to the training process proposed by Goodfellow et al. [2014], and is able to produce meaningful text without any pre-training. Experiments on our novel image captioning model indicate the model’s ability control text generation using GANs.

1. *The continuous approach to generating text with GAN is able to expand existing datasets with new meaningful sentences.*
2. *By combining classic machine translation metrics and theory from information retrieval, we propose a model for evaluating generative models in the textual domain.*
3. *We show a proof of concept for generating image captions using the continuous approach.*

1.5 Thesis Structure

The report is organized as follows. Chapter 2 introduces concepts and models central for understanding our proposed models. In Chapter 3, state-of-the-art implementations and related works in the area of GANs are presented. Here, the continuous approach, alongside the alternative approaches, is presented and its challenges are discussed. Chapter 4 presents our preliminary studies where we research concepts related to image captioning, text generation and the training of GANs. In Chapter 5, the architecture of our continuous approach is described

in detail, as well as the baseline used for comparison. Chapter 6 introduces the datasets used and presents our proposed models and corresponding results. In Chapter 7, a summary of our work and a description of the challenges we faced during the project lifetime is provided. The chapter also sums up ideas for dealing with the shortcomings of our approach and possible further extensions of the model.

CHAPTER 2

Background Theory

In the following chapter, we introduce multiple subjects related to machine learning and information retrieval which are encountered in this thesis. This serves as an introduction to the techniques and terminologies used in the scope of this project.

Deep learning architectures such as long short-term memory (LSTM), form the basis for our proposed models and is introduced together with the terminology necessary to explain the benefits of our models. State-of-the-art approaches to continuous representations of words are then discussed together with theory related to representation of images. Dimensionality reduction techniques used in a variety of visualization and information extraction tasks throughout our work is also presented. Following, the generative adversarial network (GAN) architecture which we base our work on is introduced, alongside theory related to generative models. The chapter concludes with a brief overview of the image captioning architectures leading up to the current state-of-the-art is also presented.

2.1 Artificial Neural Network

In its simplest form an artificial neural network [McCulloch and Pitts, 1943] is a model which takes input values and returns output values based on this input. The input values are manipulated by a set of weights which are adjusted to receive

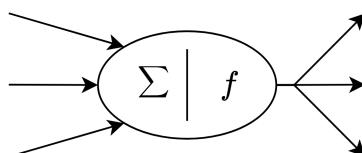


Figure 2.1: A single artificial neuron inspired by the neuron in the human brain. Σ denotes the summation of the input weights and f is the activation function.

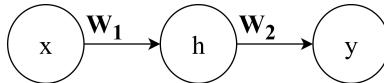


Figure 2.2: A feedforward neural network where W_1 and W_2 denote the weights, x the input, y the output and h the single hidden layer.

the desired output values. The concept of artificial neurons is inspired by the neurons in the human brain, where each neural unit is connected to others through synapses. The synapses transmit electrical impulses that, if in excess of a certain threshold, fires the signal through to other neurons connected to it. This network of interconnected signals is highly complex when in greater numbers. In a similar fashion, an artificial neural network, made up of nodes with simple summation functions, can represent complex functions when put together in greater numbers. The weights of the network are multiplied by the edges going in and out of the neurons and are adjusted as the network is trained. Each neuron consists of multiple inputs, that are summed together and activates all the output weights if the sum exceeds the threshold determined by the activation function. An illustration of this can be seen in Figure 2.1.

Feedforward networks is a category of neural networks where the connections of the network do not form a cycle. The characteristics of these networks are the sequential propagation of values through the network, as indicated by the arrows in Figure 2.2. These networks are used to approximate some function f using an input x and a target value y . While feedforward network is used interchangeably for single and multi-layer networks, deep neural networks specify that there must be more than one hidden layer in the network. These layers of nodes can be seen as a chain of functions that make up the approximation function of the entire network and is the contributing part to the term “deep” in the model’s name. For example, in a deep neural network with three layers, this function could be defined as $f(x) = f^{(1)}(f^{(2)}(f^{(3)}(x)))$. Since the output of each layer is not shown, we refer to these layers as hidden layers.

A common way of adjusting the weights, \mathbf{W} , to be able to express complex functions, is to use backpropagation [Rumelhart et al., 1986]. When data is fed through the network, each sample, \hat{y} , is compared to the desired output, y , and an error value is calculated using a loss function, such as mean squared error (MSE), see Equation 2.1. Each node in the network has a corresponding value that describes its contribution to the error. This is found by calculating the partial derivative of the loss with respect to the weights, which is referred to as the gradient. The weights of the network are updated so that they minimize the error in a process called stochastic gradient descent (SGD) [Goodfellow et al., 2016, p. 286-288].

$$MSE = \frac{1}{n} \sum_{i=1}^n (\hat{y}_i - y_i)^2 \quad (2.1)$$

2.1.1 Dropout

Overfitting refers to the networks ability to conform too well to the training data, which in turn limits the networks ability to generalize. Dropout [Srivastava et al., 2014] is a simple and effective method for preventing overfitting in neural networks. By removing a random selection of nodes in a single layer, alongside its incoming and outgoing connections, nodes are forced to be less dependent on its neighboring nodes. This also reduces the ability of certain nodes to dominate the network and, as a consequence, the network will be better at generalizing and learning.

2.1.2 Softmax

The softmax distribution, see Equation 2.2, is used when working with discrete data and is a categorical distribution used to represent a probability distribution over K possible classes, so that the sum of the distribution equals to 1. In the context of neural networks, it is often used as the last layer of a classifier, to determine the probability of an output, \mathbf{z} , representing a specific class.

$$\text{softmax}(\mathbf{z})_j = \frac{\exp^{z_j}}{\sum_{k=1}^K \exp^{z_k}} \quad \text{for } j = 1, 2, \dots, K \quad (2.2)$$

2.2 Recurrent Neural Network

Recurrent neural network (RNN) [Goodfellow et al., 2016, p. 368-383] is an extension of feedforward networks which can handle input sequences with variable length and is useful when working with sequences. It is possible to illustrate the set of computations done in a RNN by unfolding the recurrent computations into a computational graph. This results in a graph that can illustrate several timesteps of the network and the relation between these steps, as seen in Figure 2.3. The main advantage of RNNs over regular feedforward networks is the ability to retain information from previous timesteps, t . By using the same set of weights, W , it is possible to apply the model to examples of different lengths and generalize over them.

While there are many ways of structuring an RNN, any network using a recurrent connection can be considered a RNN. One example structure, Figure 2.3, has connections between the hidden units, h , of the network and produces an output, \hat{y}^t , for each value of the input sequence, x^t . Another structure, seen in Figure 2.4, only outputs a value \hat{y}^τ in the last timestep τ .

RNNs are able to predict the likelihood of a word given previously seen words, which in turn makes it possible to use them to generate natural language [Sutskever et al., 2011; Mikolov et al., 2010], and together with convolutional neural networks (CNNs) to produce image captions, which will be further explored in Section 2.7. In addition, RNNs have been used in the domain of audio and video, e.g. to translate speech to text [Graves and Jaitly, 2014] or to classify human actions in video [Baccouche et al., 2011].

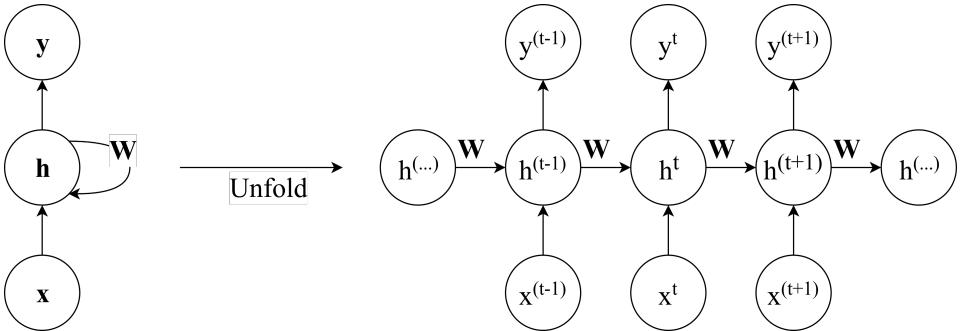


Figure 2.3: The unfolding of a recurrent neural network with a single hidden layer \mathbf{h} , inputs \mathbf{x} , and outputs \mathbf{y} . Every node in the time-unfolded graph (right) is associated with a timestep, t . The weights, \mathbf{W} , are shared across the unfolded layer. Adapted from [Goodfellow et al., 2016, 369].

2.2.1 Long Short-Term Memory

One of the major challenges of RNNs, is the learning of long-term dependencies, explored by [Bengio, 1994]. The impact of the gradient successively decreases when backpropagation is used through many timesteps, causing a concept called vanishing gradient. In some cases it is also possible for the gradient to explode. This basic problem is apparent when working with English sentences, since the meaning of a sentence is often determined by words that are not in the vicinity of each other. In the sentence “The man that works as the local doctor went inside”, the information that the man went inside is separated by many words that do not contribute to this conclusion.

One of the most effective approaches to the vanishing/exploding gradient problem has been the introduction of gated RNNs. LSTMs [Hochreiter, 1997] is a commonly used gated RNN that takes advantage of gates in a way that selectively stores or forgets information. The idea is that there are internal paths in the RNN-cell that allows it to store a state and learn what information is important. This means that information deemed irrelevant is not allowed to alter the state of the cell.

A long short-term memory-cell, showed in Figure 2.5, replaces the hidden units in a simple RNN and has internal recurrent connections, in addition to the outer recurrent connections used in simple RNNs. It consists of three gates, the first being the forget gate, f , which looks at the previous state, $h^{(t-1)}$ and the input, x^t , to decide how much of it should be kept for each element in the previous state. In the next step, the input gate, g , finds which values that should be used in updating the state, and performs an update of the state based on the proposal given by the input neuron, i . The output gate, o , controls what is outputted by the cell and what should only be stored in the cell.

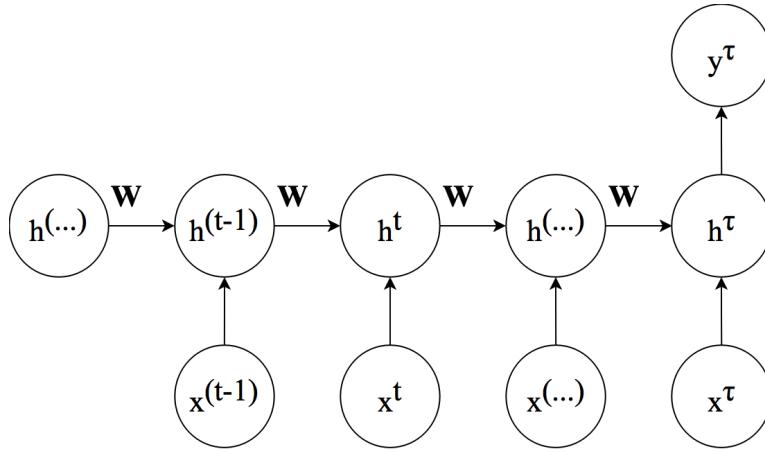


Figure 2.4: A unfolded recurrent neural network with hidden layer h , input x , and output y . The output value is only outputted at the last timestep, τ . The weights, \mathbf{W} , are shared across the unfolded layer. Adapted from [Goodfellow et al., 2016, 371].

2.3 Convolutional Neural Network

Convolutional neural networks (CNNs) [Le Cun et al., 1989] are feedforward neural networks specialized in recognizing patterns and are often used on problems regarding images. Traditional methods for classification in computer vision involved hand-crafting features to be used in a classifier. The idea behind CNNs is to combine the extraction of features and classification in one trained model. CNNs have been successful in tasks like image classification and face recognition, and contains operations that mimic how animals visually perceive images and patterns. Figure 2.6 shows a typical structure of a CNN and is illustrated with convolutional layers that can extract features from an image. These features are subsequently processed using subsampling, which reduces the dimensionality of the input. In the end, there is a fully connected layer which outputs a high-dimensional vector representing the visual features of the image.

2.3.1 Image Embedding

We define image embedding as the method of transforming an image into a dense vector of real values, which represents the visual features of the image. One way of creating such embeddings is to train a CNN with labelled images and letting the network learn to classify based on the visual features of the images. Instead of designing and training such a network, it is possible to use existing pre-trained image classifiers, such as the Inception v3 architecture introduced by Szegedy et al. [2016], to extract features from images, in a process called transfer learning. This can be done either by only training the last fully-connected layers or fine-tuning the entire model.

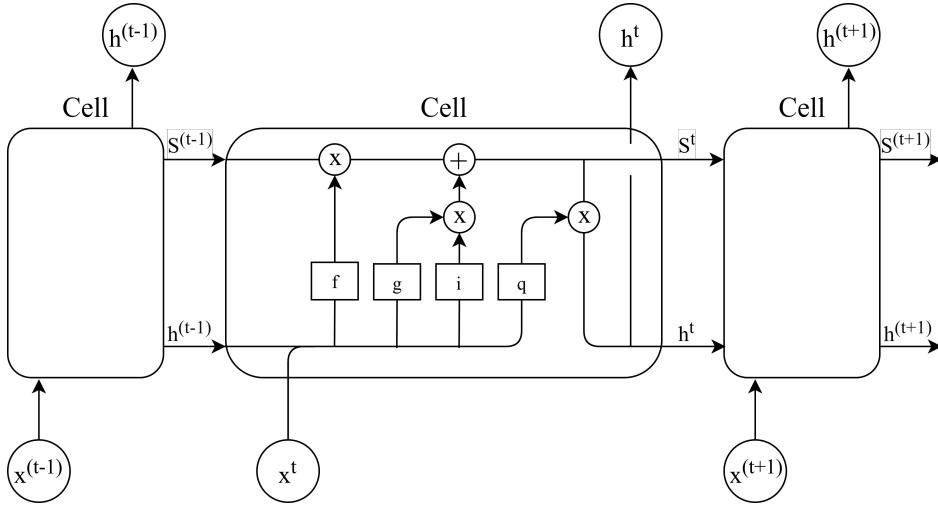


Figure 2.5: An unfolded block diagram of an LSTM cell. x and $+$ denotes pointwise multiplication and concatenation respectively. The state, S , is multiplied with the output of the forget gate, f . The state is then concatenated with the multiplication of the result from the input gate, g , and the input neuron, i . Then, the update to the hidden state, h , is controlled by the output gate, q . Adapted from Olah [2015]

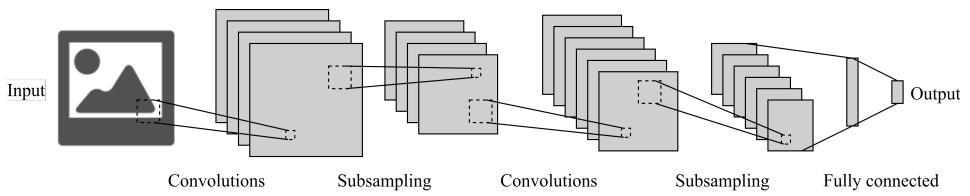


Figure 2.6: Structure of typical CNN. Convolutional layers use different filters to extract features from the image. Subsampling [LeCun et al., 1998] makes the image smoother by, for example, averaging the pixels in the image. This might also cause an reduction in dimensionality.

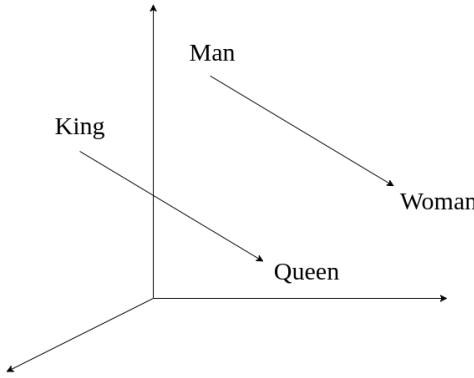


Figure 2.7: The relationship between the vector representations of man and Woman is the same as between King and Queen. This makes it possible to calculate the following: $King - Man + Woman = Queen$.

2.3.2 Text Classification

Johnson and Zhang [2015] have shown interesting results for the task of text classification, which was first introduced by Collobert and Weston [2008]. The use of RNNs for text classification suffer from long-term dependencies as discussed in Section 2.2.1. While the most successful sequence processing model currently is the LSTM, Conneau et al. [2017] argue that they lack task-specific structure and is unable to capture the hierarchical structure of sentences in the form of characters, words and sentences. They follow up by arguing that this can be captured by using deep CNNs on the lowest atomic representation of text, characters. Using convolutions and max-pooling layers allows the CNN to learn a hierarchical structure of sentences.

2.4 Word Embedding

Word embedding is a feature learning technique used to represent a word as a vector of real values. It can harbor information otherwise hidden from representations like one-hot or hashing. Word embeddings provides a dense and rich word representation, which can be used in pre-processing for deep learning. It also has the ability to deal with the curse of dimensionality, since words can be represented using a specified dimensionality rather than the size of the vocabulary. It spreads words in a multi-dimensional embedding space allowing concepts to be calculated using algebraic vector calculations like in Figure 2.7.

2.4.1 Word2Vec

Methods that captures the semantic relationship between words like Word2Vec [Mikolov et al., 2000] has gained a lot of interest over the past few years be-

Source data	Samples
The man that works as the local doctor went inside	The, man The, that
The man that works as the local doctor went inside	man, The man, that man, works
The man that works as the local doctor went inside	that, The that, man that, works that, as
The man that works as the local doctor went inside	works, man works, that works, as works, the
The man that works as the local doctor went inside	as, that as, works as, the as, local

Figure 2.8: Sliding window of 5 with target word (stippled blue) and its context (stippled white). Training samples (right) is shown as pairs of target word and its context.

cause of its speed of creating semantically accurate embeddings. Figure 2.10 shows how semantically similar words are clustered together in a two dimensional plot. Word2Vec uses a shallow, two-layer neural network to predict words based on the context of the words surrounding it. This is done using a neural network with one hidden layer, whose size determines the dimensionality of the resulting word embeddings. After training the model on words and contexts as training data, as seen in Figure 2.8, the weights of the hidden layer are used as the word embeddings.

It can be structured in two ways; continuous bag of words (CBOW) or Skip-gram. The former uses the context as input and the predicted word as output, which can be seen on the left in Figure 2.10. The latter reverses these positions as seen on the right in Figure 2.10. By training the models on a corpus of text, the error calculated between the context and the predicted word is used to adjust the weights in the network. According to Mikolov et al. [2000], the Skip-gram architecture emphasizes words in close vicinity, when compared to the CBOW architecture. However, CBOW is faster to train.

2.4.2 GloVe

Global vectors (GloVe) [Pennington et al., 2014] bases its vector representation on the co-occurrence of words such as Word2Vec, but differ in the way it uses this information. GloVe is a count-based model and starts by creating a frequency matrix for the occurrences of words in different contexts. This high-dimensional matrix is then factorized to lower the dimensionality and produces a matrix which has a dense vector representation for each word.

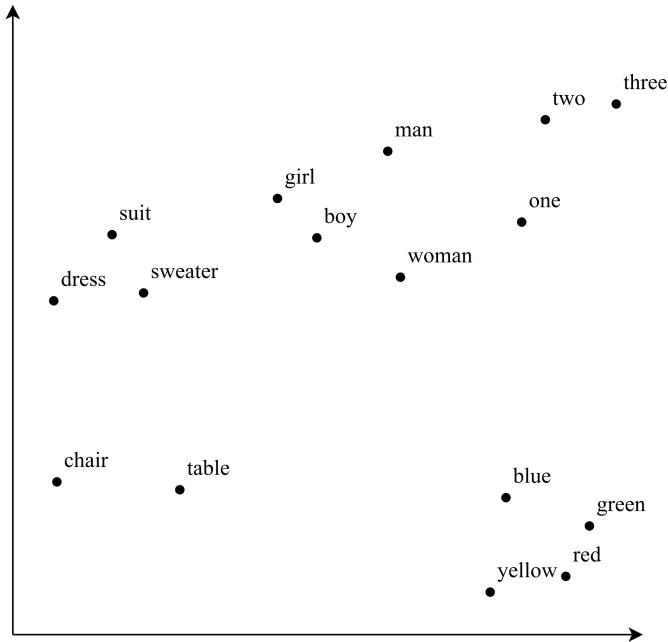


Figure 2.9: Selected words from Word2Vec trained on Flickr30k Young et al. [2014] captions. The plot is adapted from the 2-dimensional reduction of the embedded data which can be found in Appendix A.

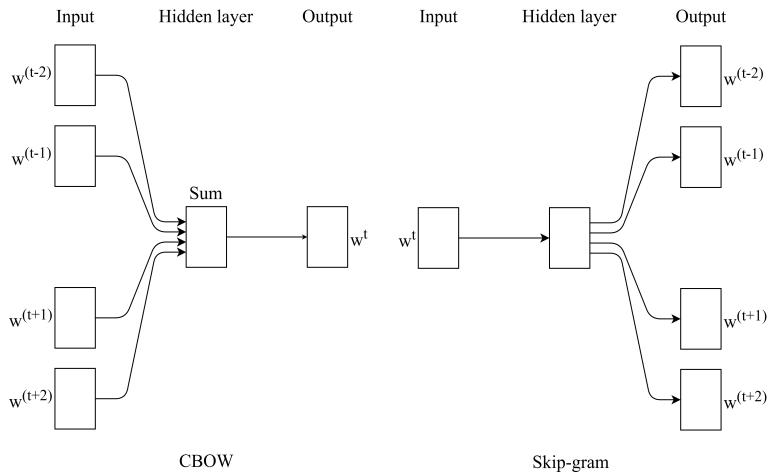


Figure 2.10: The two Word2Vec architectures, CBOW (left) and Skip-gram (right). $w(t)$ denotes the predicted word.

2.5 Dimensionality Reduction

The ability to compress high-dimensional vectors into smaller representations, could both save space and reduce the computational time needed to process the vectors. High-dimensional data being difficult to interpret or visualize are also problems that can be alleviated by dimensionality reduction. For dimensionality reduction techniques, local structure refers to maintaining close points together and global structure means maintaining the geometry at all scales. In this section we look at two different methods used to reduce the dimensionality of vectors, while trying to maintain as much information as possible.

2.5.1 Principal Component Analysis

Principal component analysis (PCA) [Hotelling, 1933] is a widely used technique for dimensionality reduction. It works by defining an orthogonal coordinate system based on the principal components, which are defined as being linearly uncorrelated variables. The first principal component is chosen to have the highest possible variance. Subsequently, the principal component with the highest variance, which also is orthogonal to the preceding principal components, is chosen until the specified dimensionality is reached, as shown in Figure 2.11. The data can then be represented with the new set of principal components. While original axes might be well defined, such as being frequencies of events occurring, the newly defined principal components are not, since they are unambiguously determined by the variance of the data.

Being a linear approach, PCA struggles to interpret complex polynomial relationships between features and is not good at maintaining the local structure of the data points.

2.5.2 t-Distributed Stochastic Neighbor Embedding

T-distributed stochastic neighbor embedding (t-SNE) [Van Der Maaten et al., 2008] is a non-linear approach to the dimensionality reduction problem and works by calculating a probability distribution that represents pairs of data points based on their similarity. By creating a similar distribution in only two or three dimensions and minimizing the divergence between these distributions, it is able to represent high-dimensional data whilst maintaining both their local and global structure.

2.5.3 Autoencoder

An autoencoder [Rumelhart et al., 1985] is a neural network that is trained to minimize the difference between the input and the output. The autoencoder model, as seen in Figure 2.12, can be described as two functions. The encoder function $h = f(x)$ that produces a hidden latent variable representation h and the decoder function $r = g(x)$ that tries to reconstruct x . The goal of the autoencoder is not to perfectly map between input and output, as this not necessarily means that the

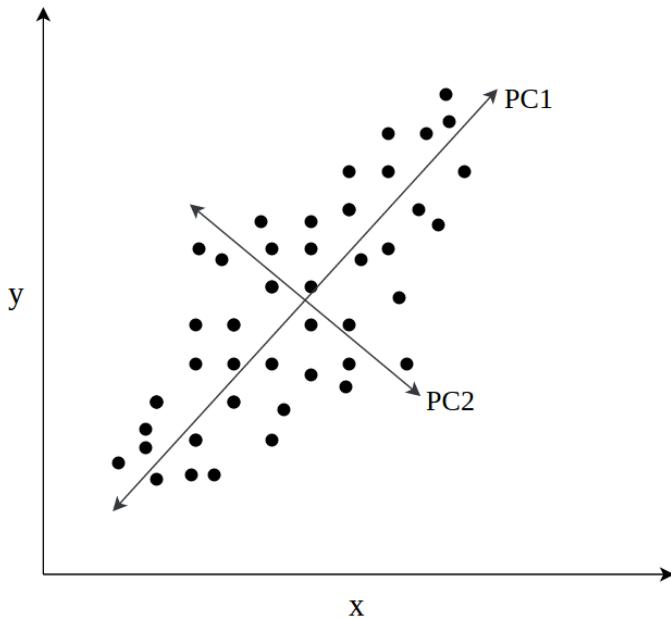


Figure 2.11: Illustration of the first and second chosen principal component chosen with the PCA algorithm. The black points denote the data points in a two-dimensional space with axes x and y. Principal components are the dimensions that retain as much variance as possible. All subsequently chosen components needs to be perpendicular to the already chosen components, which in this example would be that “PC2” component needs to be perpendicular to “PC1”.

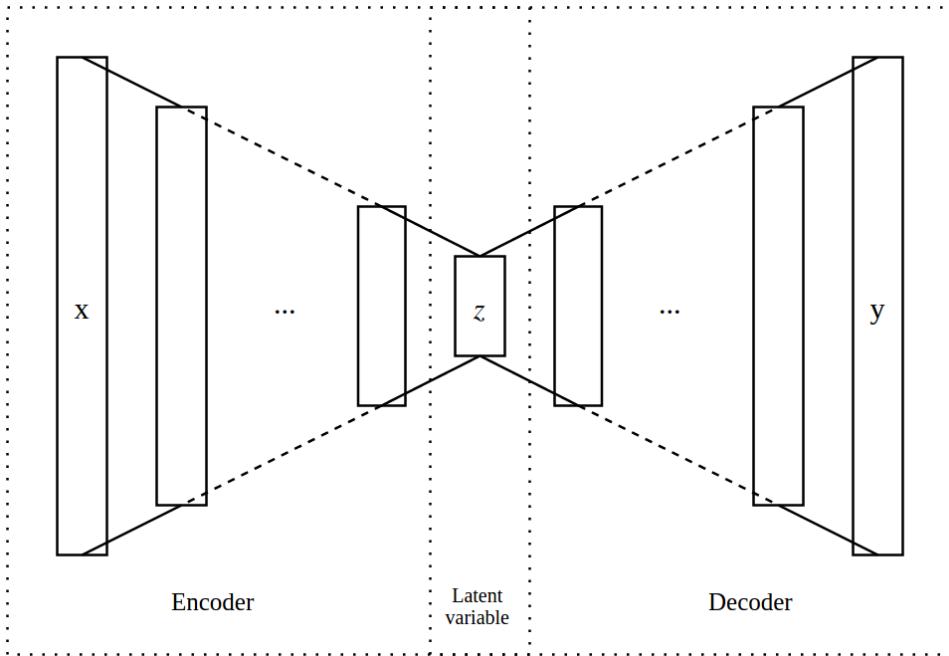


Figure 2.12: A deep autoencoder with input x and output y . The network is trained to approximate y as close to x as possible. z denotes the compressed latent variable.

model is learning anything, but to learn a dense representation of the input and reconstruct it without losing information.

One approach that requires the model to learn is restricting the size of the latent variable h , making it smaller than x . This leads to a more space-efficient representation of the input that consists of what the network determines to be the most salient features. This is referred to as an undercomplete autoencoder.

This learned representation has many useful properties, with dimensionality reduction and domain adaptation being among the most popular. G.E. Hinton and Salakhutdinov [2006] used a stacked autoencoder to obtain an error rate lower than PCA on a 30-dimensional representation. Glorot et al. [2011] used a trained autoencoder on a problem in the domain of sentiment analysis and showed results that outperformed state-of-the-art methods at the time.

2.6 Generative Models

There are two main classes of models used in machine learning, discriminative and generative. While there has traditionally been a belief that discriminative models are to be preferred, work on generative models has seen significant work and results in the recent years. Discriminative models learn to map an input x to an output y , or $P(y|x)$. Generative models, however, learn both the input and the output in

a joint probability distribution, $P(x, y)$, which means that it is able to learn the structure of data which has no labels.

The general definition of generative models only specifies that the models are used to generate random observable data. This allows for a variety of approaches such as Naive Bayes [Hand and Yu, 2001], Restricted Boltzmann machines [Smolen-sky, 1986] and Hidden Markov models [Baum and Petrie, 1966], in addition to the newer approaches such as GANs [Goodfellow et al., 2014] and variational autoen-coders (VAEs) [Kingma and Welling, 2013] which will be more thoroughly described in the following sections. Generative models learn a joint probability distribution over the entire variable space, in contrast to discriminative models, which learn a conditional probability distribution. This allows generative models to produce data without having seen the given variable, while discriminative models need to be modeled after observed variables.

2.6.1 Variational Autoencoders

In addition to GANs, variational autoencoder [Kingma and Welling, 2013] has recently shown great results in the field of generative modeling. It consists of an encoder and decoder network, similar to that of an autoencoder network. VAEs works by restricting the latent variables generated by the encoder part of the net-work to follow the gaussian distribution. Generating a new sample consists of sampling such a latent vector and passing it into the decoder.

2.6.2 Generative Adversarial Networks

Generative adversarial network is a framework proposed by Goodfellow et al. [2014] that uses an adversarial processes to estimate generative models. The goal of the framework is a model which estimates an existing probability distribution. Such a model could further be used to help semi-supervised learning [Odena, 2016], expand existing distributions [Goodfellow et al., 2014] or do multimodal translation [Reed et al., 2016]. The framework consists of two adversarial networks, a generative model and an discriminative model, as seen in Figure 2.13. The generative model is used to synthesize data in a convincing manner, while the goal of the discriminative model is to determine whether a sample is from the real data distribution or the generative model distribution.

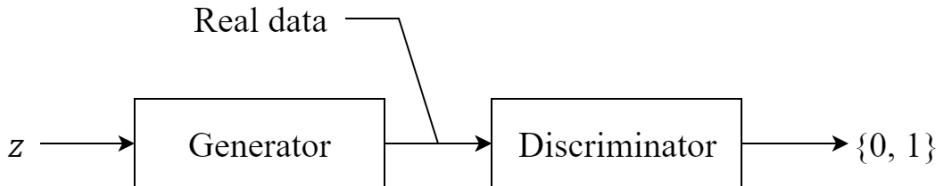


Figure 2.13: An overview of a GAN model. Noise, z , is drawn from the gaussian distribution, while the real data is what the generator is trying to imitate. The goal of the discriminator is to label the real data true, 1, and the generated data false, 0.

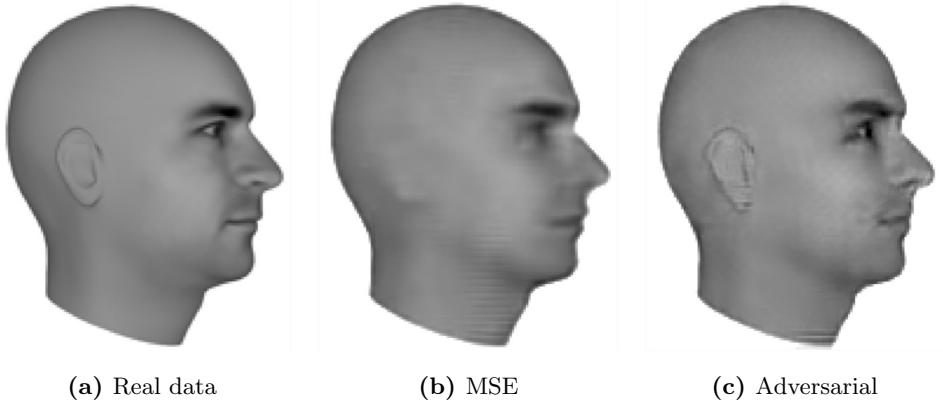


Figure 2.14: Comparison between a traditional model which minimize the MSE and a model using adversarial techniques. Taken with permission from [Lotter et al., 2016].

GANs have been used on a variety of problems in the image domain. For many problems, a single input may correspond to many different equally correct outputs. Traditional machine learning methods, where a model is learned to minimize the MSE between the predicted output and the desired output, are trained to only output a single value given an input. This will cause traditional models to output an average of the possible outputs, possibly resulting in an unrealistic output. Generative models in general, and specifically GANs produce acceptable results in these situations. In Figure 2.14, a comparison between an traditional model which minimize the MSE and an adversarial model is shown. The prediction produced by the MSE model in Figure 2.14b is more blurred than the ground truth, see Figure 2.14a and the prediction produced by the adversarial model in Figure 2.14c.

Translation from one image to another is a very interesting approach to GANs. The possibility to convert a sketch of an image into an image, as seen in Figure 2.15a, or converting aerial imagery into maps, as seen in Figure 2.15 are examples of uses for image-to-image translation shown by Isola et al. [2016].

The training procedure pits two models together in a minimax two-player game, where a generative model G learns a data distribution, and a discriminative model D estimates the probability of a sample coming from the training data rather than G , $D(x)$. G is trained to maximize the probability of D making a mistake, i. e. playing a minimax game following the value function in Equation 2.3. An optimal solution is in the form of a Nash equilibrium [Nash, 1950], and is found when the likelihood of D classifying either true or false reaches $1/2$, i. e. when $p_g = p_{data}$. This means that D is unable to differentiate between the generated data and the real training data.

$$\min_G \max_D V(D, G) = \mathbf{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbf{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))] \quad (2.3)$$



Figure 2.15: Isola et al. [2016] introduced the concept of image-to-image translation using GAN. Due to the multiple correct outputs for each input a generative model is suitable for training the model. Figure 2.15a shows examples of inference when the model is trained to colorize and fill out an outline of an object, while Figure 2.15b shows a model trained to convert from aerial photos to maps. Taken with permission from [Isola et al., 2016].

GANs are usually trained by backpropagating the loss over both G and D together. One epoch of training the network is completed by training the G and D alternately. In the beginning of an epoch the generator generates N samples using its current network weights. These samples are combined with M samples from the training data which are used to train the discriminator as a binary classifier using backpropagation. The M samples from the training set should be classified as 1 while the N samples generated by G should be classified as 0. This allows the discriminator to learn the differences between the samples from the training set and the generated samples. The second stage of the epoch is to train the generator. This time the generator creates a new set of samples which are propagated through the complete network, both the generator and the discriminator. The generator is trained to trick the discriminator by minimizing $\log(1 - D(G(z)))$, with the goal of adjusting its weight such that the discriminator would classify the generated samples as genuine samples from the training data. This is accomplished by locking the weights of the discriminator while only adjusting the generators weights.

In practice, Goodfellow et al. [2014] suggests that making the generator maximize $\log(D(G(z)))$, instead of minimizing $\log(1 - D(G(z)))$, counteracts vanishing gradients early in the training process.

2.7 Image Captioning

The task of providing a textual description for visual data has been a common research topic in the field of computer vision and natural language processing (NLP), combining breakthroughs from both fields [Vinyals et al., 2015]. A lot of early work in computer vision has been based on video, and often consisted of complex systems such as the visual recognizers together with rule-based logic systems. NLP has long been dominated by hand-written rules, which evolved into probabilistic

decision models more robust to unseen changes. The increase of computational power and availability of data allows end-to-end systems to generate natural language captions based on a large corpus of data without any prior knowledge about natural language.

2.7.1 Retrieval Based Methods

The task of image captioning has mostly been solved by learning to produce a mapping between both images and captions. This mapping can then be used in a retrieval process for finding the most suitable image caption.

Work on image captioning has recently gained a lot of interest. Farhadi et al. [2010] builds a system that scores the similarity between sentences and images using a common space, called the meaning space. This common space is utilized by mapping both images and sentences together.

Representing the meaning of an image in the form of a caption necessitates some understanding of what is important in the image. To represent entities in this space, a triplet of $\langle object, action, scene \rangle$ is defined. This should represent what a human would mention first when describing the image. The problem of mapping sentences to images is reduced to the prediction of a set of discrete triplets from the meaning-space. Sentences are mapped to triplets using series of language analysis methods. While this method could describe many common visual features, the single triplet meaning space has significant problems when describing complex images with multiple visual focal points.

More recent retrieval work has focused on embedding both images and sentences in a common space.

2.8 Multi-Modal Comparison

Deep visual-semantic embedding model (DeViSE) [Frome et al., 2013] combines two pre-processing neural networks for text and image into a neural network. It trains on the similarity of word embeddings and a transformed visual model. The pre-processing of words is done by learning a Skip-gram model, see Section 2.4.1, to represent words as embedding vectors using Word2Vec [Mikolov et al., 2000]. This model is trained on a corpus of 5.7 million documents from Wikipedia¹ and transforms words into vectors of 500 or 1000 dimensions. The creation of vector representations of images is done using a model pre-trained on the ImageNet dataset². To allow comparison, the images are transformed into a dimensionality equal to the size of word embeddings. DeViSE creates this space to be able to perform comparison in a mutual space. An illustration of how DeViSE creates a common textual space can be seen in Figure 2.16b.

Hierarchical semantic embedding (HierSE) builds on work by Norouzi et al. [2015]. This model also uses Word2Vec [Mikolov et al., 2000], however it trains

¹<http://wikipedia.org/>

²<http://image-net.org/>

its Skip-gram model on Flickr³ tags instead of web documents like DeViSE. The argument for this is that tags better represent visual features as the co-occurrences of tags. Since HierSE uses word embeddings made from features, it can match using a single word instead of an averaging over a sentence, which is done in prior works [Frome et al., 2013; Norouzi et al., 2015]. In addition, the proposed method utilizes a hierarchical structuring of labels that improves similarity measures for rare labels.

Images can be used to combine deep visual semantic features with traditional text features to improve web-store search. The authors of the paper “Images Don’t Lie” [Lynch et al., 2015] approach this by concatenating a vector representing the text features with a vector representing the visual features from the corresponding image. By using a pre-trained convolutional neural network, the authors are able to convert an image into a vector representing its visual features. Experiments done using this model show that they were able to improve search results by 1.7% in average ranking quality.

2.8.1 Word2VisualVec

The motivation behind the Word2VisualVec architecture is to embed text into a visual feature space making it is possible to capture both the semantic meaning from a text and the visual features from an image into a common space [Dong et al., 2016].

Dong et al. [2016] manages to create a new deep neural network architecture that learns to predict a visual feature encoding from text. They also show that this model outperforms state-of-the-art models on both text-to-image retrieval and image-to-text retrieval. Their goal is to create a model which learns a visual representation of text and then use this text representation to say how similar it is to an unlabeled image.

Pre-trained deep CNNs are used to extract feature vectors from images. This is done by extracting the values from the last fully-connected layer in the pre-trained models. Representing text is done using Word2Vec, see Section 2.4.1.

The cross-media model consists of a feedforward network learning the visual features from a vector representing textual features. Figure 2.16a shows that features from text are extracted before they are fed into the Word2VisualVec model, which generates a visual encoding of the text. This encoding can then be directly compared to other visual features generated from a CNN.

Retrieval based methods have the basic limitation of not necessarily being able to describe all unseen compositions of objects, since the caption itself is limited to the available captions in the dataset.

2.8.2 Generative Based Methods

In order to produce new captions describing unseen motives, an effective approach to text generation is needed. The methods introduced in this section are progres-

³<https://www.flickr.com>

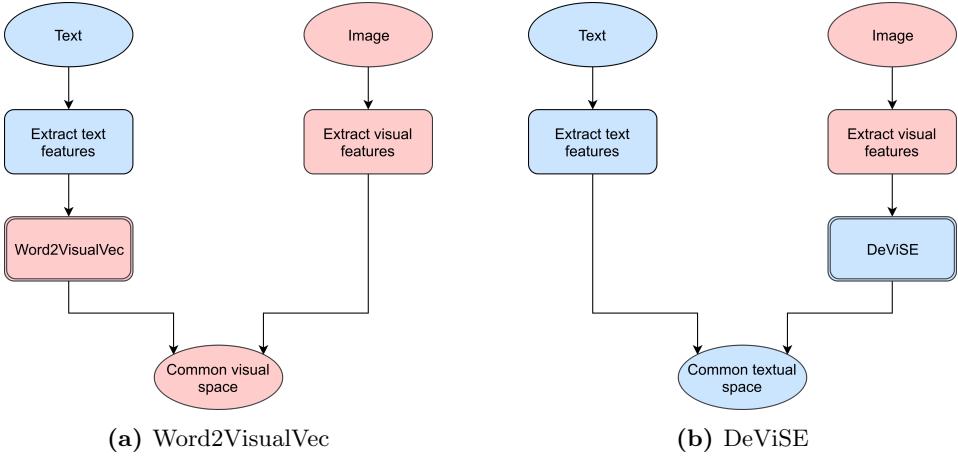


Figure 2.16: Illustrations of the Word2VisualVec and DeViSE architectures.

sively able to generate more natural sentences, and more effective on the problem of image captioning.

Rather than retrieving pre-existing sentences, Li et al. [2011] focuses on generating composed phrases from scratch. First, visual information is extracted from the image and is encoded in a triplet consisting of objects, visual attributes and spatial relationships. Then, a selection of sentences found relevant based on the triplets is selected and fused together, resolving the issue of possible synonyms and re-ordering of words.

Kulkarni et al. [2013] follows the same initial ideas as Farhadi et al. [2010] and Li et al. [2011], but does not limit the retrieved information of the image to a single triplet, but rather allows several of these triplets to influence the generation of text.

Both Li et al. [2011] and Kulkarni et al. [2013] explore the idea of generating text using a N-gram language model and a template-based approach. An N-gram language model is a probabilistic distribution that predicts the probability of the next word in a N-word sequence based on the previous N-1 words. While N-gram language models has been widely used to generate text, a severe limitation when producing text is that it is unable to maintain coherent meaning over a sequence of sentences. In addition, it can be a challenge to produce a grammatically correct language. The template-based approach assumes that there exists a set of syntactic patters that can be used as a structure for the information, which have been gathered in triplets. This is an attempt at dealing with the limitations of language models. While the template-based approach is able to produce high-quality sentences, it is only able to do so in a limited domain and with a limited number of words. Consequently, it is not able to generalize well over new domains.

Mitchell et al. [2012] extends the work done by Li et al. [2011] and Kulkarni et al. [2013], but rather uses syntactic trees to generate sentences. Similar work by Kuznetsova et al. [2012] also use parsing at its core.

Vinyals et al. [2015] created an end-to-end framework able to produce reasonable

descriptions of images using neural-networks. They use a CNN that encodes images to a dense vector representation and use it as the initial input of the LSTM, which then generates a sequence of words. This is possible since the words are embedded into the same high-dimensional space as the images, using a word embedding W_e and CNN respectively. The sentence generating LSTM is trained to predict each word, S_t , of a sentence given the image, I , and all preceding words as defined by:

$$x_{-1} = \text{CNN}(I) \quad (2.4)$$

$$x_t = W_e S_t \quad t \in 0 \dots N - 1 \quad (2.5)$$

$$p_{t+1} = \text{LSTM}(x_t) \quad t \in 0 \dots N - 1 \quad (2.6)$$

Maximum likelihood models are trained to predict the next token of a sequence given the current state and the previously seen tokens. As pointed out by Bengio et al. [2015], a significant drawback of this approach is that tokens not seen in the training data might be predicted during inference. This means that the predicted token, and all future predicted tokens, might be influenced by an erroneous word approximation, which does not exist in the training data. This error might accumulate over the generation of a sequence.

2.9 Sequence-to-Sequence Methods

Inspired by the works done with autoencoders, the sequence-to-sequence approach was initially introduced as the RNN Encoder-Decoder by Cho et al. [2014].

The model is able to propagate an input sequence, $X = x_1, x_2, \dots, x_l$, of length l into a rich fixed-length vector, z , called the latent representation. The model then uses this latent representation to predict the same or another sequence, $Y = y_1, y_2, \dots, y_{l'}$, of length l' as shown in Figure 2.17. These two steps are done by the encoder and decoder, respectively, and are trained together. The encoder can read a variable length sequence and update its hidden state after each time-step. When it reaches the end of the sequence, the hidden state of the RNN represents the latent variable. The decoder part of the network is trained to generate a sequence based on the previous hidden state and the latent variable coming from the encoder. In addition, it is conditioned on the output from the previous time-step of the decoder. The length of this output vector is independent of the length of the input sequence, making the model suitable for a variety of problems.

The latent representation, z , has a useful property in that it is able to represent semantically similar sequences of natural language close to each other. While the model has been successfully used to translate between languages, it has also seen wide use in initializing image captioning models. This is used as the initial starting point for the image captioning framework of Vinyals et al. [2015]. In this work, we try to use a sequence-to-sequence model as the starting point of the generative model.

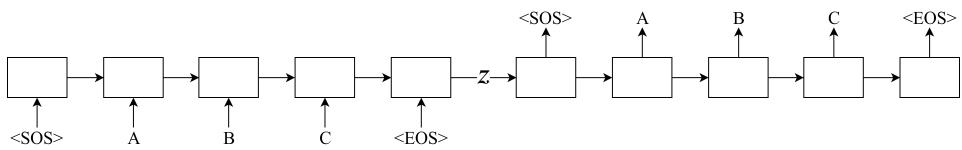


Figure 2.17: A sequence containing the tokens $\langle \text{SOS} \rangle$, A, B, C and $\langle \text{EOS} \rangle$ propagated into the latent representation, z . This representation is then used to reproduce an identical sequence. Adapted with permission from [Sutskever et al., 2014]

CHAPTER 3

Related works

This chapter will describe work that is directly relevant or are alternatives to our proposed approach. Approaches that deal with the challenge of propagating gradients based on discrete data are discussed. In addition, we present state-of-the-art training techniques to alleviate the stability challenges of training generative adversarial networks (GANs), only some of which are implemented in our works.

3.1 Text Generation

In the past year, training GANs to produce discrete data has seen a recent surge of new approaches, with varying results. Training GANs using natural language is a challenging problem due to text being inherently discrete. Approximations to discrete sampling and using the REINFORCE gradient estimator are two of the most popular approaches. Correspondingly, Goodfellow [2016] suggest three approaches to allow working with discrete data using GANs; using the REINFORCE algorithm, Gumbel-Softmax and translating the variables into continuous values. In addition to these approaches, we present a few other recent attempts at producing discrete text using GANs.

3.1.1 Reinforce

A promising approach is to use the REINFORCE algorithm [Williams, 1992], inspired by reinforcement learning, to give the generator a reward for generating data that fools the discriminator. Commonly used loss functions, like mean squared error (MSE) and cross entropy [Shore and Johnson, 1980], will have undefined derivatives when used on discrete values. To deal with this, REINFORCE uses the expected cost, which is a smooth step function that makes the function differentiable.

Work by Bahdanau et al. [2016] proposes a reinforcement learning approach by phrasing the problem of text generation as an action-taking problem. The state is

considered to be the sequence of tokens generated so far, while the action would be which token to generate next.

By combining this with the GAN framework, Yu et al. [2016] are able to bypass the generator differentiation problem. Their model, sequence generative adversarial nets (SeqGAN), uses policy gradient [Sutton et al., 1999] in the domain of poem generation, speech generation and music generation. Policy gradient works by sampling sequences of actions to see what actions receive the best rewards. Then, the weights of the network are updated to make it more probable to perform that action again. The discriminator is used to evaluate the action and to give a reward based on the estimated probability of a generated sentence being real.

Li et al. [2017] takes the inspiration from reinforcement learning one step further by employing a critic to estimate the future rewards of the current state. They extend this model in the domain of dialogue generation by rewarding each intermediary generated word within the sequence. This allows the model to reward words that fit into the context, while not contributing to the correctly summarized meaning. Yang et al. [2017] also work on the SeqGAN model and uses it for the task of neural machine translation. However, the target-sentence generated by the network is conditioned on the source-language sentence. Another difference between Li et al. [2017] and Yang et al. [2017] is that the former uses a long short-term memory (LSTM) to produce sentences, while the former uses a convolutional neural network (CNN). This choice is further discussed in [Yang et al., 2017]. Their experience in the domain of neural machine translation is that using CNNs perform well in contrast to the unstable training of using LSTMs. Their reasoning for this is that the discriminator is too strong for the generator. However, this assertion is not substantiated by any evidence.

In the domain of image captioning, Liang et al. [2017] propose an adversarial framework for producing paragraphs describing images. The recurrent topic-transition generative adversarial network (RTT-GAN) bases its generator on semantic regions extracted from images and generates sentences that are discriminated by both a sentence discriminator and a topic-based discriminator, both of which are LSTMs.

Methods using policy gradient deals with the discrete nature of text by using a stochastic policy gradient, which works well with discrete actions and states. The stochasticity of the policy gradient also provides randomness to the generator, which means that the challenge of keeping the generator differentiable is avoided. However, the approach suffers from high variance, which means that training requires a large number of iterations in order to produce good results.

In addition, using SeqGAN requires supervised pre-training of the model before it is able to learn. Without it, the generator is never able to produce relevant data, indicating that the discriminator is unable to provide sufficient feedback. SeqGAN is used as our baseline for text generation and is further described in Section 5.1.

3.1.2 Gumbel-Softmax

Jang et al. [2016] proposes an approach that re-parametrizes discrete variables, but only prove its effectiveness on simple one layer models using discrete latent

variables. The purpose of the re-parameterization trick is to overcome the inability to backpropagate the gradient by making it independent of the latent variable. Gumbel-Softmax is an attempt at making this possible for discrete variables as well.

While working with a discrete representation of data necessitates sampling one-hot vectors, y , from the softmax distribution and renders the gradient unusable, sampling from the Gumbel-Softmax distribution does not. Instead, they define the smooth Gumbel-Sofxmax distribution, which allows the sampling of continuous variables with a specified degree of similarity to the real one-hot values.

Hjelm et al. [2017] report difficulties with using the Gumbel-Softmax technique when compared to their novel boundary-seeking generative adversarial network (BGAN) model. They provide a table of tested hyperparameters that did not produce any satisfactory results.

In the context of GANs, this is, to our knowledge, the only attempt of using Gumbel-Softmax to train GANs and motivates our choice of abandoning the use of this approach for our work.

3.1.3 Alternative Methods

Zhang et al. [2016] propose a model that learns an embedded representation of the sequence of words and uses an LSTM to sequentially produce words. This is done by letting a LSTM generator produce embedded words until the end of sequence token, at which point, the hidden state of the LSTM represents the entire sentence. While the loss function is based on the original loss function provided by Goodfellow et al. [2014], they approach the discretization problem using an approximation function called soft-argmax. The few examples provided show grammatically correct, but semantically limited, sentences.

BGAN, introduced by Hjelm et al. [2017] proposes another approach by forcing the representation of generated samples to be close to the center of the real samples using an alternative loss function for the discriminator. The idea of this loss function is to minimize the Kullback-Leibler divergence, which is a measure of the difference between two probability distributions, between the generated and real data distributions, trying to force the samples to always lie on the descision-boundry of the discriminator. This results in a more stable training process, which is one of the main challenges for both REINFORCE- and continuous representation-based methods. However, while they are able to stabilize the training of GANs, the character-based examples generated by their model do not show convincing results and is only able to generate a few correctly composed English words.

3.1.4 Continuous Representation

To our knowledge, there has been no successful work that uses dense representations of text as the output of the generator, to allow a differential loss function. A common representation of this is word embeddings, such as Glove or Word2Vec described in Section 2.4. Since word embeddings are continuous, the original loss

function proposed by Goodfellow et al. [2014] is applicable, without any modifications or workarounds. It is this approach of representing the output of the generator as continuous values we base our research on.

3.2 Training Challenges

Consistently training GANs remain an open problem and prove to be one of the major challenges for experimenting with GANs. It involves finding a Nash equilibrium [Nash, 1950] to the two-player adversarial game that is the minimization of losses of the discriminator and generator. While work devoted to finding ways of stabilizing the training of GANs has seen a lot of work on continuous problems [Arjovsky et al., 2017; Salimans et al., 2016; Che et al., 2017], it has only recently been shown attention in the domain of natural language [Hjelm et al., 2017].

3.2.1 Mode Collapse

Another challenge of training GANs is the problem known as mode collapse. This refers to the generator learning to map several noise samples to the same output. In the domain of image synthesis, an example of this would be when the model generates multiple images of motive A. Then the discriminator learns that images of motive A are all fake, including real images. This makes the generator stop producing images of motive A and starts making images with another motive, B. In turn, this makes the discriminator think all images of motive B are fake, which starts the cycle over again. This remains as one of the most important research problems of GANs [Goodfellow, 2016]. An optimal approach of training would be to force the generator to produce images of all motives, and thus improving the creation of all motives simultaneously.

3.3 Training Techniques

A commonly used technique to limit vanishing gradients, which is discussed in Section 2.2.1, is to smooth the labels of real and generated samples from 0 and 1 to x and $1-x$, where x is a value around 0.2 [Szegedy et al., 2016; Pfau and Vinyals, 2016]. This allows the discriminator to always return relevant gradients.

To freeze the weights of either the discriminator or generator during training helps if either of the models is becoming too accurate, stopping the network from converging [Pfau and Vinyals, 2016]. This allows the other model to “catch up” and continue training either for a specified number of iterations or until a similar loss value is reached.

Most of the work done with GANs uses some extension of the techniques proposed in the deep convolutional generative adversarial network architecture [Radford et al., 2015]. The first technique is to use batch normalization [Ioffe and Szegedy, 2015] in both the discriminator and generator. Yang et al. [2017] also argue that batch normalization accelerates training time significantly. While training GAN using batch normalization has been common practice, recent results by

Xiang and Li [2017] and Hjelm et al. [2017] has shown that batch normalization does not improve stability during training.

In addition, batch normalization has been shown to suffer from high correlation between examples inside a batch [Salimans et al., 2016; Goodfellow, 2016]. This means that examples are more dependent on the other examples inside each batch, rather than being independent from each other.

Salimans et al. [2016] extend the batch normalization method by using a constant reference batch during the training of the model, in an approach called virtual batch normalization (VBN). This means that instead of being dependent on the other examples in a batch, examples are normalized on the statistics of the reference batch, which, in turn, is only dependent on its own statistics. Salimans et al. [2016] show that VBN is effective against mode collapse.

Salimans et al. [2016] also introduce the technique of minibatch features, which allows the discriminator to compare each example to a set of real and generated examples. This allows the discriminator to determine if the example is similar to other generated samples and to provide better feedback to the generator.

Arjovsky et al. [2017] proposes an alternative value function that is based on the earth moving (EM) distance [Rubner et al., 2000], called Wasserstein generative adversarial network (WGAN). Additionally, they use weight clipping to accommodate the new value function. GANs are notoriously hard to train and does not provide any feedback on the quality of the model during training, since the loss functions are relative to each other. WGAN has shown advantages in the quality of generated samples and as a countermeasure for mode collapse.

CHAPTER 4

Preliminary studies

In this chapter, we will describe preliminary work done leading up to the proposal of our model. The foundations in retrieval based image captioning and generative adversarial networks (GANs) for image generation provide valuable experience and knowledge necessary for the usage of our model in the image captioning domain. In addition, several state-of-the-art generative models are implemented [Bengio et al., 2015; Vinyals et al., 2015]. The following work on understanding and learning the concepts and techniques of GANs consists of training a GAN model on the MNIST dataset¹, the *hello world* of the machine learning domain.

4.1 Retrieval based Image Captioning

In this section, we present the implementation of the Word2VisualVec architecture introduced by [Dong et al., 2016], which is discussed in Section 2.8.1. In addition, results that compare various text and image representations used in the task of text-to-image retrieval are presented and discussed. In our work, which can be found online², we introduce three easily exchangeable parts that allows comparison using various methods for image and text representation. The three parts consist of word embedding, image embedding and a model to create a mapping between the two. We also compare our model to another baseline solution to measure the effectiveness of the visual embedding approach.

Three different methods for representing the captions, Word2Vec, global vectors (GloVe) and training an embedding using a long short-term memory (LSTM), and two different pre-trained image classifiers, Inception [Szegedy et al., 2015] and VGG-19 [Simonyan and Zisserman, 2015], gives us six different models. Evaluation is done using the Flickr30k dataset [Young et al., 2014], which is further described in Section 6.1, to perform text-to-image retrieval. Each caption from the test

¹<http://yann.lecun.com/exdb/mnist/>

²https://github.com/ruoccoma/master_works/tree/master/helgoy-lund

data, is used to predict a visual feature vector that is compared to the images in the Flickr30k dataset. The comparison is done using the cosine similarity, see Section 6.2.1, between the transformed caption, and all of the images in the dataset. Recall at k (R@K) is the percentage of queries for which the model predicts the correct result within the top k retrieved results and is used as our evaluation metric.

In Table 4.1 three different recall intervals are shown for the six combinations of models and the baseline model. The results are calculated using the captions from the test dataset. Figure 4.1 shows the top 3 retrieved images for the textual query “A group of people are rock climbing on a rock climbing wall” using the best performing model, VGG-19/LSTM. The image representing the query caption is shown in Figure 4.1a.

Image embedding	Word embedding	R@1	R@10	R@100
Baseline				
VGG-19	GRU	0.03	0.25	2.06
Our models				
VGG-19	LSTM	2.39	13.42	49.42
VGG-19	GloVE	0.60	3.66	18.74
VGG-19	Word2Vec	0.49	3.31	16.00
Inception	GloVE	0.10	0.81	5.27
Inception	Word2Vec	0.08	0.64	4.57
Inception	LSTM	0.00	0.03	0.28

Table 4.1: A comparison of Word2VisualVec models using R@K for the task of text-to-image retrieval.

During this work, we create an end-to-end system which compare both images and text in a common visual feature space. While the functionality for doing text-to-image, text-to-text and image-to-text exist in the system, only experiments on the first is performed. We have implemented three different methods for mapping from a textual caption into a visual feature space. We conclude that the best translation from textual to visual features is done using an LSTM based model, which learns a dense representation of an image caption. It is also apparent that the pre-trained VGG-19 network receives, on average, better results than the Inception model. We compare our model to a baseline model which has shown satisfactory results on text-to-image retrieval. Our model performs better than the baseline model when both are trained and tested on the Flickr30k dataset.

4.2 Researching GANs

A common use case of GANs is the ability to increase an existing dataset by generating samples similar to existing data. Our experiments consist of both extending existing datasets, as well as generating images similar to our own custom drawn images. The reasoning behind running these experiments is to familiarize ourselves with the GAN model, both in structure and how to efficiently train such a model.



(a) Query image with caption: *A group of people are rock climbing on a rock climbing wall*



(b) Top 1



(c) Top 2



(d) Top 3

Figure 4.1: Images retrieved when querying with caption corresponding to image in Figure 4.1a using the best performing model, VGG-19/LSTM model

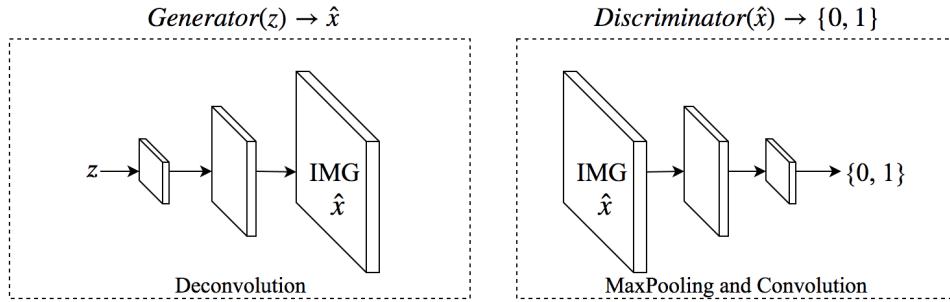


Figure 4.2: Overview of GAN model learning to create images. The generator creates an image, \hat{x} , from a noise, z , drawn from a normal distribution. The discriminator network classifies \hat{x} as a real or generated image.

In the image generation experiments we use the following notation. We denote the generator network as $\text{Generator} : \mathbb{R}^Z \rightarrow \mathbb{R}^I$, the discriminator network as $\text{Discriminator} : \mathbb{R}^I \rightarrow \{0, 1\}$, where Z is the dimension of the noise input to the Generator and I is the dimension of the image. The noise vector, z , is sampled from a normal distribution, $z \in \mathbb{R}^Z \sim \mathcal{N}(0, 1)$. An illustration of the architecture is shown in Figure 4.2.

The generator uses the concept of deconvolution successfully used in Dosovitskiy et al. [2015] to create two-dimensional images from one-dimensional noise vectors. The discriminator implements a set of convolutional and maxpooling layers [Goodfellow et al., 2016, p.330] to learn a representation from two-dimensions into a single floating-point number, classifying an image as either real or fake.

4.2.1 MNIST

Our initial image generation experiment with GAN expands expand the classical image dataset MNIST³. MNIST is a database containing images of handwritten digits. The database contains in total 70,000 black and white images of 28 by 28 pixels with labels. Examples of digits from the dataset can be seen in Section 4.2. We train a generator which created images of single digits from random noise, and a discriminator which classifies the generated images either as an image from the MNIST dataset or as a generated image. This experiment is a classical use case of machine learning in general, and GANs in particular. Figure 4.3b shows a sample of digits produced by the trained generator. Figure 4.3a shows a sample of actual images taken from the MNIST dataset. As the two figures illustrate, the generator trained is able to create images fairly similar to the actual images. The main drawback of this generator is the variety of images created. As seen in Figure 4.3b, the generator tends to favor some digits over others, occasionally causing digits to appear less often than others. This issue is often referred to as mode collapse, and is described in Section 3.2.1.

³<http://yann.lecun.com/exdb/mnist/>

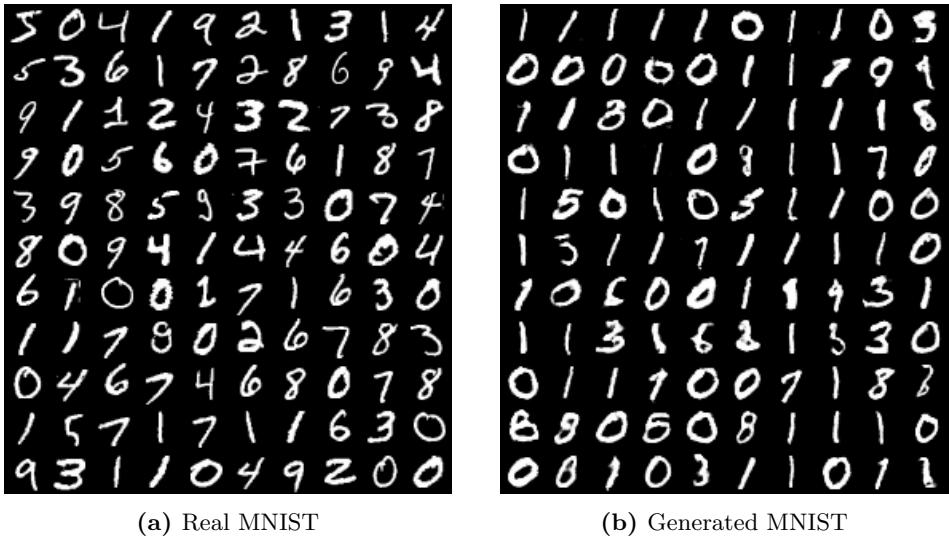


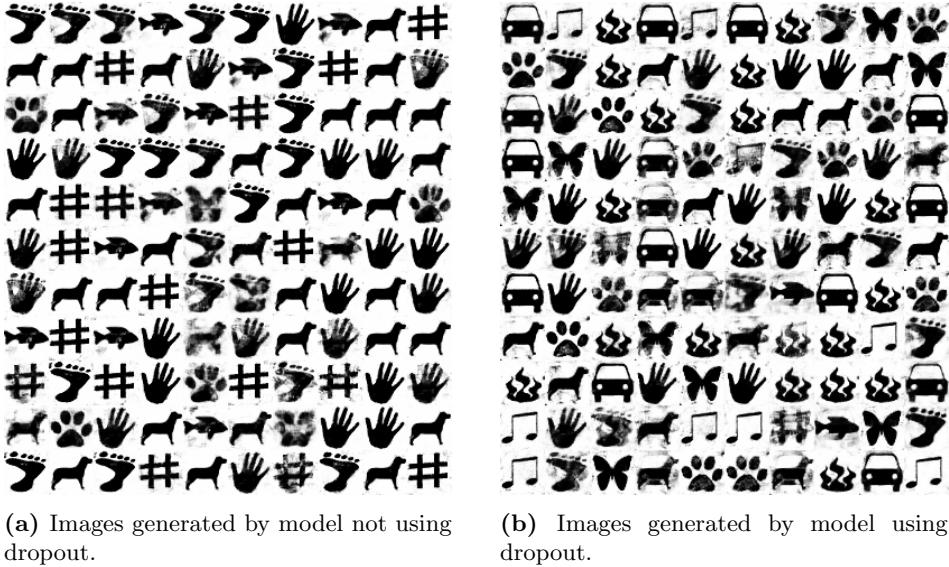
Figure 4.3: Comparison between images drawn from the MNIST dataset and images drawn from the approximated random distribution learned by the generator.

4.2.2 Custom Drawn Images

The MNIST dataset contains images with drawings of 10 different digits. Due to the satisfactory results on the initial experiment, we perform experiments on larger images with more complex figures. We manually create a set of 40x40 pixel images, in contrast to MNIST's 28x28, with a set of figures as motives. We train the GAN using these images for approximately 2 hours and receive the result seen in Figure 4.4. The model is capable of recreating images at a satisfactory level, with some images being indistinguishable from the original image. On the other hand, this model suffers from the same lack of variety as the model trained on the MNIST dataset.

4.2.3 Preventing mode collapse

As seen in the two previous experiments, the generator is capable of convincingly replicating some of the images, while ignoring others. Which image motives the generator creates, changes during the course of training, i.e. after some epochs of training the generator creates the image of a butterfly perfectly, but in others it does not draw a butterfly at all. This could be caused by the discriminator learning to classify all images of, for example, butterflies as false, forcing the generator to change which motive to draw. To tackle this problem of low variety, we perform a similar experiment with nearly the same structure as the previous. In this experiment, we remodel the discriminator to use dropout, a technique which adds noise to data propagating through the model, described in Section 2.1.1. This forces the discriminator to generalize more over the seen training examples, allowing the generator to draw a wider variety of motives.



(a) Images generated by model not using dropout.

(b) Images generated by model using dropout.

Figure 4.4: Generated 40x40 images trained on the custom figures dataset. Each image is generated after 100 epochs of training.

ing the generator to learn smaller adjustments. In other words, the generator is allowed to perform improvements on already explored images, instead of pivoting and completely changing which image to draw. The results of the model trained with dropout is seen in Figure 4.4b. The generator used to create the image in this figure is trained equally long as the model without dropout, seen in Figure 4.4a. There are two major differences in the results from these two models. The most interesting difference is the larger variety of images shown in the results from using dropout. Figure 4.5 shows a comparison of the results from the two different discriminators. As illustrated in the figure, the model which introduces dropout to the data during training, manages to represent all but two images. The model without dropout lack representations from five images.

Adding dropout to the model clearly increase the variety within a random sample of generated images. However, using dropout causes another difference with regards to image quality. While the model without dropout creates a limited variety of images, it is able to do so with convincing image quality. The model with dropout, however, produces more blurry and unfinished images. A probable explanation for this behavior is that the model needs more time to train when using dropout, due to the model being forced to generalize more over the data.

4.3 Maximum Likelihood Text Generation

Text segmentation is the problem of splitting natural language into meaningful units. As we want to use GANs to generate text, this has significant impact on the

Dropout	✗	✓	✓	✓	✓	✗	✓	✓	✓	✓
No dropout	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓

Figure 4.5: Comparison of the variety in generated images from two models after 100 epochs of training.

Seed	Inference
dog	.
dog running	in the sand .
a boy is running	through a field .
surfer	.
man climbing	a street with a stick in the background .
a man riding	a bike .

Table 4.2: Inference from the Character-based LSTM generating text starting with a given seed.

further development of our solution. As a consequence, we implement models using two popular text segmentation techniques, character- and word-segmentation and conduct experiments using both. Maximum likelihood models work by maximizing the probability of each token given the current state and the previously tokens.

4.3.1 Character-level LSTM

Training a character generator based on the Flickr30k dataset shows good results using only a single LSTM with 256 hidden units. We train this model to predict the next character on a sequence of 20 preceding characters, shown in Table 4.3 with only a sequence of 5 preceding characters for illustration purposes. Since the vocabulary is only of 41 characters, we use categorical cross entropy [Shore and Johnson, 1980] as loss function. This model is able to produce some grammatically correct captions, as seen in Table 4.2.

4.3.2 Word-level LSTM

By replacing the characters with words, we are able to start training a similar model to predict the next word in a sentence. However, using the one-hot encoding for words introduces some challenges. The encoding of a vocabulary of, for example, 20 000 words, requires one 20 000-dimensional vector for each word. The sparseness of the data makes the model time-consuming to train.

When comparing to the character-level approach it is apparent that the main difference lies within the number of classes the network has to predict. While the

A		t	a	l	l		m	a	n
A		t	a	l	l		m	a	n
A		t	a	l	l		m	a	n

Table 4.3: The output character (yellow background) is trained with the sequence of 5 preceding characters (grey background).

A	tall	man	is	running	down	the	street
A	tall	man	is	running	down	the	street
A	tall	man	is	running	down	the	street

Table 4.4: The output word (yellow background) is trained with the sequence of 5 preceding words (grey background).

former only choose between 41 classes, the latter needs to be able to differentiate between the size of the vocabulary. This makes the word-level model much larger and potentially slower to provide sufficient results, given a large vocabulary.

Implementing the word-level model leads us to reduce the complexity of the vocabulary to gain sufficient results, see Table 4.5. The model generating the examples uses a vocabulary size of 1000 words. We also limit the size of the vocabulary to reduce the dimensionality of the vectors representing words. The training data was built by beginning at the start of the sentence and predicting the next word. Then, we predict the next word by using the preceding two words, and so forth, similar to the training data in the character-level approach. Inference is stopped when the end of sequence token is reached. This approach is shown in Figure 4.4.

Seed	Inference
dog	in front of a .
dog running	down a street .
a boy is running	on a beach .
surfer	in a blue shirt is a .
man climbing	a rock .
a man riding	a bike in a .

Table 4.5: Inference from the Word-level LSTM generating text starting with a given seed.

4.3.3 Text Segmentation

While the model based on character-segmentation showed good results, see Table 4.2, there is no continuous alternative to the discrete one-hot representation of characters. This would be difficult to work with since discrete tokens inhibits GANs ability to backpropagate gradient, which is talked about in Section 1.1.2. While workarounds could exist, see Section 3.1.2 and Section 3.1.1, working with

the well-recognized word embedding approach provides semantic relationships to the tokens and allows us to transform the data from discrete to continuous. The choice of word segmentation forms a vital part of our proposed model.

CHAPTER 5

Architectures

In this chapter, we present the architecture of our two proposed models. The first is a text generation model, and the second extends the first model by rendering it controllable. These models are described in Section 5.2 and Section 5.3 respectively. The text generation model features two generator implementations; the first generates softmax-distributions, while the second generates approximated word embeddings. Hence, they are named the softmax distribution and word embedding model, respectively. The image captioning model is described as an extension of the word embedding model that uses image embeddings to control the text generation. Before these models are explained further, a text generator baseline is introduced.

5.1 Baseline: SeqGAN

In the following section, we present a text generator baseline architecture, sequence generative adversarial nets (SeqGAN) [Yu et al., 2016], which has already been introduced in Section 3.1.1. SeqGAN is chosen as the baseline due to its recognition in the field and promising results. In addition, the provided implementation is open-source and available online¹. Yu et al. [2016] approach sequence generation as a reinforcement learning problem. This is presented as defining the state as the currently produced sequence, and the action as which token to choose next. The goal of the generator is to maximize the expected end reward. The reward is calculated using the REINFORCE algorithm [Williams, 1992], and is estimated using the discriminators probability of sequence being real. This reward is only produced for complete sentences. However, SeqGAN also produces a reward for intermediary states as well. This is done by randomly sampling the rest of the sequence using Monte Carlo tree search [Coulom, 2006]. Starting at the already predicted sentence, actions are randomly expanded N times, and an average score of these is used as the reward for the state. With this approach, the results of

¹<https://github.com/LantaoYu/SeqGAN>

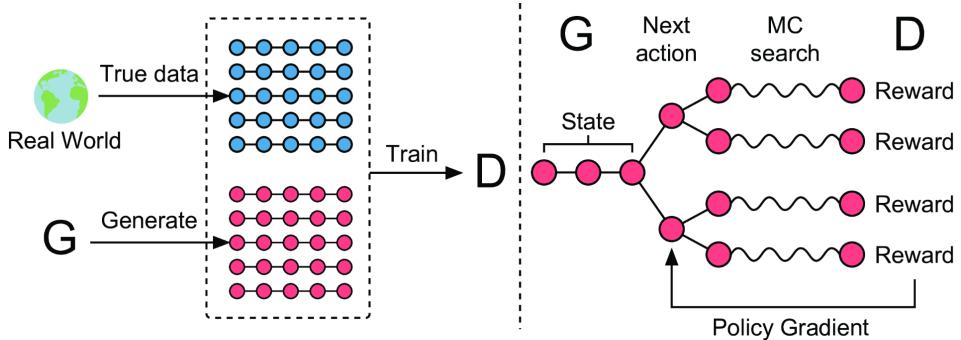


Figure 5.1: The SeqGAN model. The discriminator, D, is trained to differentiate between the true and the generated data (left). The final rewards are passed back to the next action using policy gradient [Sutton et al., 1999], and is used to train the generator, G (right). Figure taken with permission from [Yu et al., 2016].

future possible sequences can also be considered. An illustration of the model is seen in Figure 5.1.

SeqGAN uses an long short-term memory (LSTM) for the generative model and outputs the softmax distribution in the last layer. The discriminative model uses a convolutional neural network (CNN) with a fully connected layer at the end. Similarly to our approach, they use a sigmoid activation function to output the probability of a sentence being real.

5.2 Text Generation

This section covers the structure of our proposed model for using generative adversarial networks (GANs) on the text generation problem.

5.2.1 Training GAN on Discrete Sequential Data

Based on the continuous approach, we train our model by calculating a gradient based on the output of the discriminator with respect to the generator, and update the weights using backpropagation. In other words, we try to minimize the difference between discriminators log-probabilities for the samples being labeled real or false and their correct labels. In comparison to the REINFORCE and Gumbel-Softmax approaches presented in Chapter 3, this way of training works in the same way as the original approach presented by Goodfellow et al. [2014]. While their solution is in the domain of image generation rather than text, our models are trained using the same techniques and loss function, which is described with Equation 2.3.

Adam [Kingma and Ba, 2014] and stochastic gradient descent (SGD) [Goodfellow et al., 2016, p. 286-288] is used as the optimizer for the generator and discriminator respectively. This choice is based on what is common in literature regarding

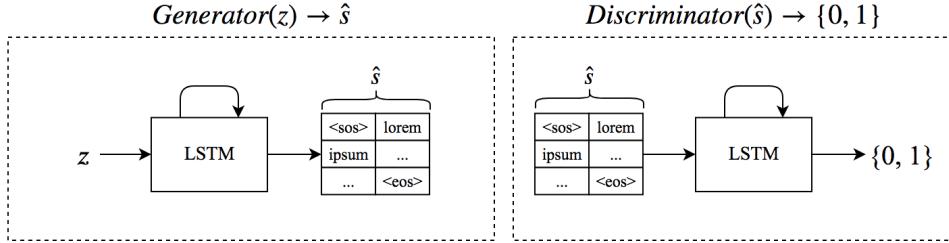


Figure 5.2: Overview of GAN model learning to create sentences. The generator creates a sentence \hat{s} from a noise z drawn from a normal distribution. The discriminator network classifies \hat{s} as a real or generated sentence.

the training of GANs. The activation function of the discriminator is the sigmoid activation function, which is commonly used for binary classification.

Working with sequences of discrete tokens requires a conversion to a continuous representation. We present implementations of these models using either word embeddings or the softmax distribution. The word representations created by the generator are fed directly into the discriminator without doing retrieval of the closest correct word embedding or the argmax operation, depending on which representation is chosen. As the generator relies on feedback from the discriminator during training, it would be convenient to feed the discriminator with as accurate word representations as possible. For the word embedding model, this would be achieved by using the generated word embeddings to find the most similar word embedding, and using these representations as input to the discriminator. A similar approach is used in the softmax distribution approach, but instead of feeding the discriminator with vectors from the softmax distribution, it would be desirable to convert these vectors to one-hot vectors representing single words. However, both of these methods would render the network non-differentiable and would prevent the calculation of the gradient.

As a consequence of not being able to use these approaches, the output from the generator may be inaccurate, when compared to the representation of real data. On the other hand, using the raw output of the generator as input to the discriminator and then propagating the error back through both models allows for directly usage of the known loss function from [Goodfellow et al., 2014].

Goodfellow et al. [2014] used GANs to generate images by basing their generator and discriminator on CNNs. As our goal is to generate text and control text generation using GANs we need models suitable for learning and understanding sequential data. LSTMs have shown great results on problem regarding sequential data in general, and particularly regarding text [Bengio et al., 2015]. Both the generator and the discriminator in our models are, as a result, based on LSTMs.

5.2.2 The Text Generation Architecture

We denote the generator network as $\text{Generator} : \mathbb{R}^Z \rightarrow \mathbb{R}^{L \times T}$, the discriminator network as $\text{Discriminator} : \mathbb{R}^{L \times T} \rightarrow \{0, 1\}$, where Z is the dimension of the

noise input to the generator, L is the number of words in a sentence and T is the dimension of a single word. The noise vector, z , is sampled from a normal distribution, $z \in \mathbb{R}^Z \sim \mathcal{N}(0, 1)$. An illustration of the architecture is shown in Figure 5.2.

In the experiments, we research two different types of word representations. As our preliminary studies show, it is possible to represent words as L -dimensional one-hot vectors, where L is the size of the vocabulary. The model using this representation makes the generator of the model output vectors from a softmax distribution. The second word representation is word embeddings. In this case the dimension of L is the equal to the size of the word embedding vectors.

The main difference between the word embedding model and the softmax model, is the training of the generator. The former outputs estimated word embeddings using the hyperbolic tangent activation function, while the latter calculates an output distribution using the softmax activation function. The reason for this choice is that the softmax distribution model originally works with one-hot vectors. The model based on word embeddings uses the hyperbolic tangent as activation function in its generator. Because of these different word representations, the dimensionality of the words, T , is equal to the vocabulary size in the softmax distribution model. In the word embedding approach, however, T is independent of the vocabulary size approach.

5.2.3 Differences to the Baseline

While our approach introduces noise in the form of the input to the network, the stochastic property of the SeqGAN policy model provides the variety required for a generative model.

The gradient of the SeqGAN model is based on the sampling of a large discrete action space, which is limited if the action space is mainly consisting of poor choices. This means that it is necessary for the SeqGAN model to pre-train both the generator and discriminator before starting adversarial training. Without this supervised pre-training, the discriminator is too strong for the generator and will not be able to guide the generator in the right direction. First, the generator is pre-trained on the training set using maximum likelihood estimation (MLE). This training method predicts the next token given the sequence so far using an LSTM, similar to what is described in Section 4.3.2. This allows the discriminator to be pre-trained to differentiate between these generated sentences and the real data by minimizing the cross entropy [Shore and Johnson, 1980] between the labels and the predicted probability.

5.3 Image Captioning

Reed et al. [2016] propose a way to control the generation of images using text. They are able to control the image generation by translating from an image description into a suitable image. Their framework, as seen in Figure 5.3, works by introducing a dense representation of a sentence to both the generator and dis-

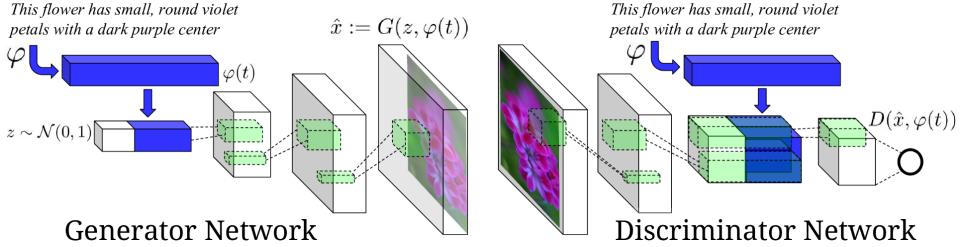


Figure 5.3: The text-to-image GAN architecture. φ denotes the dense embedding, while \hat{x} is the generated image. Taken with permission from [Reed et al., 2016].

criminator. For the generator, the sentence representation is concatenated to the noise drawn from the gaussian distribution, while it is depth concatenated to the four spatial dimensions in the discriminators CNN before it is convoluted down to a single probability.

[Hu et al., 2017] use variational autoencoder (VAE) to generate text in a controlled fashion. To control the text generated by the VAE they employ a discriminator to determine if the sentence corresponds to a given condition. Their model is able to control the generation of “positive” and “negative” sentences, as well as present and past tense. Their extension of the VAE text generator requires a dedicated discriminator for each attribute they want to control. This model is non-trivially extendable to the image captioning problem. It would require a separately trained discriminator for each training image and the lack of discriminators for new images would make them unable to be captioned. In the following section we propose an architecture suitable to do both image captioning, as well as control text generation based on sentiment.

We borrow ideas from Reed et al. [2016] to create an image-to-text generative model. The structure of this image captioning GAN is similar to the word embedding model. We denote the generator network as $Generator : \mathbb{R}^Z \times \mathbb{R}^I \rightarrow \mathbb{R}^{L \times T}$, the discriminator network as $Discriminator : \mathbb{R}^{L \times T} \times \mathbb{R}^I \rightarrow \{0, 1\}$, where Z is the dimension of the noise input to the generator, I is the dimensionality of the image representation, L is the number of words in a sentence and T is the dimension of a single word. The noise vector, z , is sampled from a normal distribution, $z \in \mathbb{R}^Z \sim \mathcal{N}(0, 1)$. An illustration of the architecture is shown in Figure 5.4.

The main extension to the text generation model is the introduction of the image representation, which adds consistency to the noise, z . The intention is to enable the ability to control the text generation, e.g. generating a sequence of words describing an image. In the case of image captioning, images are represented as a control vector, c , and is used to influence the generated sentence, \hat{s} . Images are represented as high-dimensional vectors using an existing model and transfer learning, see Section 2.3.1. To reduce the complexity of these vectors we perform dimensionality reduction, see Section 2.5 for more details.

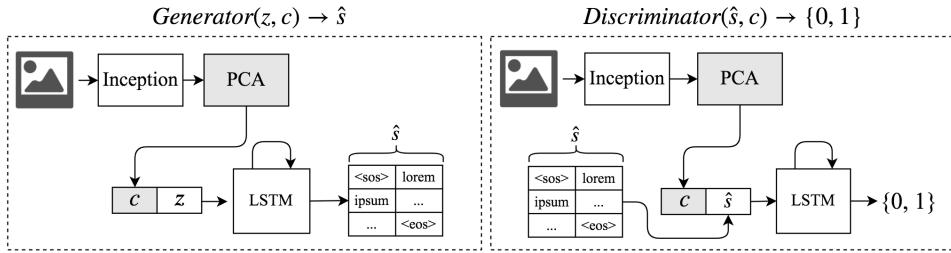


Figure 5.4: Overview of the GAN model learning to create image captions. The generator creates a sentence, \hat{s} , from a noise, z , drawn from a normal distribution which is combined with some representation of an image. The discriminator network classifies \hat{s} as a real or generated sentence by looking at both \hat{s} and the representation of an image, c .

CHAPTER 6

Experiments and Results

This chapter presents our experiments on generating text using generative adversarial networks (GANs). The first sections gives an overview of the data and pre-processing techniques used, as well as a detailed description of our evaluation method. The experiments cover both different approaches to text generation and results from our proposed image captioning architecture.

6.1 Data and Pre-Processing

This section provides a brief overview of different datasets and pre-processing techniques. We train and test our models on two different datasets, which both contain images and their related descriptions. We also give a brief technical overview of how the different experiments are implemented.

6.1.1 Datasets

The Flickr30k dataset [Young et al., 2014] contains an image caption corpus of 158,915 captions describing 31,783 images. The images are gathered from the Flickr website¹ and the captions are generated by humans. Example images from Flickr30k with captions is shown in Figure 6.1.

As the Flickr30k dataset provides a wide variety of images and captions, we want to investigate if our models can perform better using a dataset with less variance. The Oxford-102 flower dataset² contains over 8,000 images belonging to 102 different categories of flowers. Each picture is annotated with 10 different captions describing each flower in detail. Example images with captions is shown in Figure 6.2.

¹<https://www.flickr.com>

²<http://www.robots.ox.ac.uk/~vgg/data/flowers/102/>



A black dog runs into the ocean next to a pile of seaweed



A man sits with his head down on a subway



Several groups of pedestrians on a city street during the day



Bicyclist pull ahead of other racers with the crowd cheering wildly

Figure 6.1: Example images with one corresponding caption from the Flickr30k dataset.



This flower has petals that
are pink and has yellow
stamen



The flower has six oval
shaped white petals and
yellow stamen



This flower has petals that
are purple and has a white
style



This flower has petals that
are yellow and are very
stringy

Figure 6.2: Example images with one corresponding caption from the Oxford-102 flower dataset.

	a	man	is	surfing!			
SOS	a	man	is	surfing	EOS	PAD	PAD
	woman	surfs	on	a	large	wave.	
SOS	woman	surfs	on	a	large	wave	EOS
	the	flower	has	turquoise	petals.		
SOS	the	flower	has	UNK	petals	EOS	PAD

Table 6.1: Three sentences before (above) and after (below) pre-processing. A filter removes punctuation marks and abbreviations for unknown words (UNK), starts of sequences (SOS), ends of sequences (EOS) and paddings (PAD) are added.

6.1.2 Pre-Processing of Data

Both the Flickr30k and the Oxford-102 flower datasets contain images in the JPEG format and captions represented as character strings. While these formats are practical and readable for humans, this is not necessarily the case for computers. During our experiments, we perform different pre-processing techniques on both text and images.

The captions in the Flickr30k and Oxford-102 flower datasets are converted to lowercase and special characters are removed for simplicity. To deal with the use of variable-length sentences for models that only allows fixed-size sequences, special tokens are added for the start and end of the sequence, as well as a padding and an unknown word token. The padding token is used when a sentence is too short, while the unknown token is used if a word is not in the current vocabulary. An example of pre-processing of three different sentences with fixed sequence length can be seen in Table 6.1.

A common method for representing words as vectors is to create a one-hot representation of each word. A one-hot bit vector is a vector containing a single 1, while the rest of the values are 0. The placement of the 1 represents the word from a vocabulary of N words, which causes the one-hot vector to be of length N.

Section 2.4.2 describes a method for creating vector representations from words, also known as word embeddings. In some of the experiments we use a set of pre-trained word vectors created using this algorithm. This dataset of word embeddings was created by training a model on Wikipedia articles, as well as news articles, containing 6 billion different tokens (words, numbers and special characters). The dataset itself contains vectors for 400,000 distinct words each with a dimension of 300.

As well as the pre-trained global vectors (GloVe), we train custom word embeddings using the Word2Vec algorithm, explained in Section 2.4.1. We train multiple sets of word embeddings using either the Flickr30k or Oxford-102 flower dataset. Within these datasets, parameters such as dimensionality, vocabulary size and training duration are experimented with. More details regarding a specific Word2Vec word embedding are given when it is used.

6.1.3 Implementation

All off our experiments and models are implemented using Keras³ and trained on an GeForce GTX Titan X GPU. Keras is a neural network framework written in Python. We use Tensorflow⁴ as the backend for Keras. This framework allows us to implement, train and test a variety of neural network techniques. The baseline model explained in Section 5.1 is implemented in Tensorflow and is available online⁵. Our complete code base is available on GitHub⁶.

6.2 Evaluation

The ability to computationally evaluate generative models is a non-trivial task. In contrast to information retrieval systems and classifications tasks, generative models have no correct solution and multiple answers might be acceptable. Evaluation metrics such as precision, recall or accuracy are therefore not easily applicable to generative models. In the case of text generation, some possible evaluation methods, such as human evaluation and bilingual evaluation understudy (BLEU) [Papineni et al., 2002], exist. However, human evaluation is very time consuming and since there exists infinitely many correctly generated sentences, BLEU is not out-of-the-box suitable for text generation.

We are interested in producing semantically meaningful sentences that make sense in the context of the dataset the model is trained on, and propose an evaluation method based on this. The main goal of our text generation models is to produce sentences that match the distribution of sentences of our datasets. Evaluating syntactically and grammatically correct sentences are not prioritized, but could be interesting to incorporate in our evaluation metric, and is further discussed in the further works in Section 7.5.

6.2.1 Evaluation Method

By combining ideas from information retrieval and one of the most established evaluation metrics used on machine translation, BLEU, we propose a new automatic evaluation method for text generation models. Our motivation for implementing such a metric is to capture whether sentences “make sense” in the context of the dataset. More specifically, if the sentences contain similar semantic content as the sentences from the dataset.

BLEU is a method commonly used for machine translation tasks. The introduction of BLEU provides an inexpensive and automated alternative to the previous human-based evaluation methods. The idea behind BLEU is that the best translation a machine can provide is measured on its similarity to a human translation. When applying BLEU on machine translation problems, the algorithm is given a

³<https://www.keras.io>

⁴<https://www.tensorflow.org/>

⁵<https://github.com/LantaoYu/SeqGAN>

⁶<https://github.com/DayNoone/Master>

machine translated sentence and a small set of correctly human translated sentences, called reference sentences. These reference sentences are used to indicate similarity on a scale of 0 to 1, where 1 would indicate the sentences being identical. The score is based on a set of precision- and frequency-based measures and has been endorsed as the de facto standard of evaluation in the domain of machine translation. The main problem of using BLEU on text generation models is that there exists no correct “translation” for a given generated sentence.

Retrieving Reference Sentences

As the BLEU scoring function depends on relevant reference sentences, a method for retrieving such sentences is required to create a fully automatic evaluation method. In the case of text generation, one approach is to select reference sentences by retrieving the most similar sentences to the generated one. Our proposed model consists of the following steps; given a generated sentence, find the most relevant sentences from the complete dataset and use these reference sentences in the BLEU algorithm. As the retrieved reference sentences has a great impact on the final BLEU score, we implement three different retrieval methods; term frequency-inverse document frequencies, cosine distance and word mover’s distance (WMD). Each of these retrieval techniques use different approaches to retrieval, which adds variation to the sentence retrieval process. This allows us to calculate a BLEU score for each of the retrieval methods, and finally obtain the final score, β , by taking an average over these scores. An illustration of our text generation evaluation method is shown in Figure 6.3.

TF-IDF is the first retrieval method, and is based on the frequency of word occurrences in a collection of documents or, in this case, sentences. There are two parts to TF-IDF. First, appearances of a queried word in a document is counted. Then, the frequency of the words in the entire collection is counted and used to offset the frequency of the words in each document, i.e. common words like “the” and “a” are weighted less. The query sentences are paired with every sentence in the dataset, and the sentences with the TF-IDF score is used as reference sentences.

Accurately quantifying the similarity between two embedded sentences has traditionally been conducted using TF-IDF or similar frequency-based methods. A significant drawback of these methods is that they have no method of compensating for semantically similar, but syntactically different sentences. For instance, the sentences “Obama speaks to the media in Illinois” and “The President greets the press in Chicago” is not represented using the same words, but share a similar meaning. WMD [Kusner et al., 2015], our second retrieval method, is a distance function based on word embeddings like Word2Vec and works by finding the distance between pairs of word embeddings in the query and document. In other words, it calculates the lowest transport cost of moving all the words in the query to the document within the word embedding space, illustrated in Figure 6.4. Thus allowing comparison of semantic similarities.

The third and last retrieval method is based on the cosine similarity between vectors. The cosine similarity between two vectors, \mathbf{A} and \mathbf{B} , with length n , is measured using the cosine angle between them, shown in Equation 6.1. Cosine

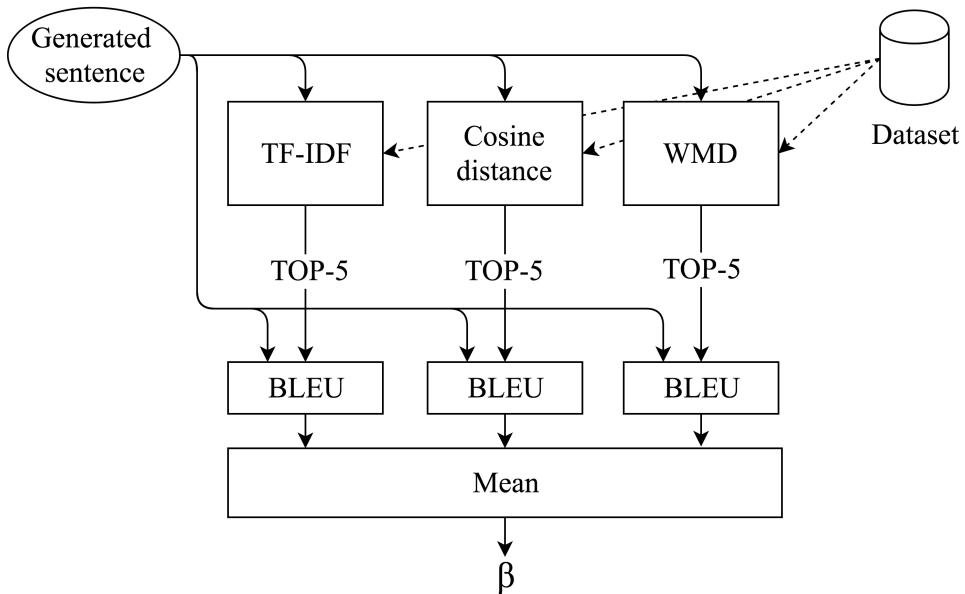


Figure 6.3: Illustration of how the text generative models are evaluated. The three different retrieval methods use the generated sentence to find the 5 most similar sentences from the dataset. The BLEU score is calculated three times using each of these three sets of 5 related sentences. Finally, the average of these three BLEU scores is calculated as the final score, β .

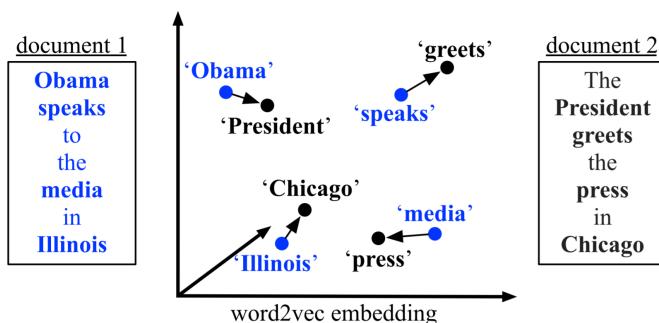


Figure 6.4: Non-stopwords (bold) are paired based on similarity. The calculated distance between the two documents is based on the sum of the distances between the pairs of words. Taken with permission from [Kusner et al., 2015].

similarity can be used as a document similarity metric by representing each document as a single vector [Li and Han, 2013]. In our experiments, each word is represented as an L-dimensional word embedding vector, where every word in a sentence is added together creating one L-dimensional sentence vector. These sentence vectors are used during the retrieval process.

$$\text{similarity}(\mathbf{A}, \mathbf{B}) = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}} \quad (6.1)$$

The three retrieval methods are chosen for their difference in calculating the similarity between sentences. While TF-IDF is a traditional retrieval method based on word frequencies, the WMD and cosine distance apply the use of word embeddings to allow comparison of semantics. The main difference between WMD and cosine similarity is that the former compares sentences on a word level, while the latter compares sentence vectors. WMD also ignore commonly used words, in contrast to cosine similarity where the sentence vectors are represented using all words in the original sentence.

Table 6.2 shows the top-2 retrieved similar sentences using the three retrieval methods. Top-2 is chosen for comparison purposes as differences are apparent from a small subset. The query sentence in this example is an actual sentence chosen from the Oxford-102 flower dataset. We choose this sentence to demonstrate the difference between the retrieval methods. All three methods retrieve the identical sentence among the top-2 most similar sentences. However, only cosine similarity and TF-IDF rank the identical sentence as the most similar. The reason for this is that they both compare every word in a sentence, while the WMD disregards common words. WMD ranks a sentence which have replaced the word “this” with the word “the”, ranking both sentences as identical to the query. An interesting characteristic of the cosine similarity retrieval is that this method allows the retrieval of syntactical equal, but semantically different sentences. As seen in Table 6.2, the second highest rated sentence has a similar structure, but the colors are exchanged. The reason for this is the similarity of colors in the high-dimensional word embedding space, where the distance between colors is short.

Query sentence chosen from dataset
<sos> this flower has petals that are yellow with white edges <eos>
Top-2 sentences retrieved using Cosine similarity
<sos> this flower has petals that are yellow with white edges <eos>
<sos> this flower has petals that are pink with yellow edges <eos>
Top-2 sentences retrieved using TF-IDF
<sos> this flower has petals that are yellow with white edges <eos>
<sos> this flower had petals that are yellow with white edges <eos>
Top-2 sentences retrieved using WMD
<sos> the flower has petals that are yellow with white edges <eos>
<sos> this flower had petals that are yellow with white edges <eos>

Table 6.2: Table showing the top-2 retrieved sentences from Oxford-102 flower dataset for each of the three retrieval methods; cosine similarity, WMD and TF-IDF for a given query sentence chosen from the dataset.

Query sentence containing errors
<sos> this flower has blue petals with with with green stamen <eos>
Top-2 sentences retrieved using Cosine similarity
<sos> this flower has purple and pink petals with with stamen <eos>
<sos> this flower has white petals with green lines and stamen <eos>
Top-2 sentences retrieved using TF-IDF
<sos> a flower with bright blue leaves and a blue stamen <eos>
<sos> this flower has light blue petals with darker blue veins <eos>
Top-2 sentences retrieved using WMD
<sos> this flower has petals that are blue with green stamen <eos>
<sos> a flower with pink petals and green stamen <eos> <pad> <pad>

Table 6.3: Table showing the top-2 retrieved sentences from Oxford-102 flower dataset for each of the three retrieval methods; cosine similarity, WMD and TF-IDF for a given query sentence containing errors.

Query sentence containing errors
<sos> Two soccer players swim on the soccer field <eos><pad><pad>
Top-2 sentences retrieved using Cosine similarity
<sos> two soccer players walk on the soccer field <eos><pad><pad>
<sos> soccer players interact on the field <eos><pad><pad><pad><pad>
Top-2 sentences retrieved using TF-IDF
<sos> a man in swim trunks is juggling on the beach <eos>
<sos> swimmers in the water with swim hats on their heads <eos>
Top-2 sentences retrieved using WMD
<sos> two soccer players walk on the soccer field <eos><pad><pad>
<sos> two soccer players playing soccer on a field <eos><pad><pad>

Table 6.4: Table showing the top-2 retrieved sentences from Flickr30k dataset for each of the three retrieval methods; cosine similarity, WMD and TF-IDF for a given query sentence containing errors.

Table 6.3 shows retrieval of sentences given an erroneous query sentence. The retrieval is done using the Oxford-102 flower dataset. The query sentence in this table contains the word “with” three times sequentially. The cosine similarity method retrieves a sentence from the dataset that contains an error as the top-rated sentence. The WMD method retrieves what looks like the most similar sentence, describing a flower with blue petals and green stamen. The TF-IDF method, however, focuses on the color blue, and, subsequently, retrieves the sentences containing the word blue. A reason for this could be that there are fewer blue flowers in the dataset than other colors, causing the TF-IDF to prioritize this color.

The last retrieval example is done using the Flickr30k dataset. An example from this dataset is shown due to the increased variety of sentences, in comparison to the Oxford-102 flower dataset. Table 6.4 shows the top-2 retrieved sentences for a query sentence, which contains an error. In the query sentence, the word “walk” is exchanged with the word swim, creating a semantically invalid sentence. In this retrieval example both the cosine similarity and WMD retrieves valid and relevant sentences. However, the TF-IDF method does not retrieve the sentences a human would normally choose as the most similar sentence. Instead of comparing semantics, the TF-IDF method uses relative word frequencies, which cause the

method to retrieve sentences describing swimming.

The three different retrieval methods perform comparison differently, creating diversity within the retrieved sentences. These sentences are used as reference sentences for the BLEU algorithm. As the BLEU evaluation metric is not originally applicable on text generation problems, the difference in the scores corresponding to the different retrieval methods can be significant. Table 6.5 - 6.7 show the final score, β , as well as the individual BLEU score for each retrieval method, for three different query sentences. The β for the query drawn from the dataset does, as expected, equal to 1.0. It is also natural that each of the retrieval methods receive a BLEU score equal to 1.0. In Table 6.6 and Table 6.7, the BLEU score and β is presented for two different erroneous query sentences, compared to the Oxford-102 flower dataset and the Flickr30k dataset, respectively. Here, it is apparent that the retrieval methods do not retrieve the same reference sentences, due to the difference in the calculated BLEU scores. Arguably, all the retrieval methods provide good reference sentences, which is the reasoning for not choosing one single retrieval method, but three. Tables showing the top-5 retrieved sentences for each of the three query sentences discussed is found in Table A.2-Table A.4.

Query sentence - Chosen from Oxford-102 flower dataset			
<sos> this flower has petals that are yellow with white edges <eos>			
Cosine similarity	TF-IDF	WMD	β = Mean
1.0	1.0	1.0	1.0

Table 6.5: Evaluation score, β , for each retrieval method compared to mean score for sentence from dataset.

Query sentence - Erroneous from Oxford-102 flower dataset			
<sos> this flower has blue petals with with green stamen <eos>			
Cosine similarity	TF-IDF	WMD	β = Mean
0.8489	0.8014	0.6263	0.7589

Table 6.6: Evaluation score, β , for each retrieval method compared to mean score for erroneous sentences. Compared to Oxford-102 flower dataset.

Query sentence - Erroneous from Flickr30k dataset			
<sos> Two soccer players swim on the soccer field <eos> <pad> <pad>			
Cosine similarity	TF-IDF	WMD	β = Mean
0.7265	0.4724	0.7265	0.6418

Table 6.7: Evaluation score, β , for each retrieval method compared to mean score for erroneous sentence. Compared to Flickr30k dataset.

6.3 Experiment 1 - Text Generation using GAN and Softmax

Initial experiments for generating text is done by converting sentences into sequences of word indices, using a predefined vocabulary size and ignoring rare words. This data structure allows discrete representation of words as one-hot vectors. The one-hot representation also allows the use of the softmax activation function and for fast retrieval of generated words. A single word retrieval consists of a single dictionary look-up. Both the generator and the discriminator are implemented using LSTMs. The generator produces sequences of vectors, and the discriminator uses these vectors, together with real sentences, represented as one-hot vectors, to learn. An overview of the architecture is described in Section 5.2.

6.3.1 Experimental setup

Multiple training sessions of models with different topologies and hyperparameters are performed. The four main experiments using one-hot vectors have some shared features. The size of the vocabulary is 1000 words, which causes each word to be represented as a 1000-dimensional one-hot vector, i.e. the size of T is 1000. The models are trained using a sentence length of 12, $L = 12$, and each model is trained on 11000 captions from the Oxford-102 flower dataset. The dataset consists of all flower descriptions containing 10 or less words. The reason for this choice is to allow both the generator and the discriminator to recognize sentence structures by avoiding to train on cropped sentences. In addition, limiting the length of the sentences simplifies the experiment. The networks are trained on 12 word long sentences, since each flower description is extended with the start-of-sequence, $\langle \text{sos} \rangle$, and the end-of-sequence token, $\langle \text{eos} \rangle$.

The differences between the models lie within the training process of the discriminator. The discriminator is trained on both generated and real data. In the case of real data, words are represented as one-hot vectors, while the generated data are vectors containing a variety of real numbers. This difference in data could allow the discriminator to distinguish between real and generated examples by only looking at the data structure. To prevent this problem, two of the models implement a transformation function. This function converts an L -dimensional one-hot vector, into an L -dimensional vector of real values in the range $[0,1]$, that adds up to 1, see Algorithm 1 and Table 6.8. As the image generation results in Section 4.2 improved significantly by adding dropout in the discriminator network, two models using dropout in the discriminator are also trained. The four different one-hot experiments consist of a reference model, a model implementing only dropout, a model implementing only the transformation function and a model implementing both dropout and one-hot transformation. The reference model is neither implementing dropout nor the one-hot transformation function. Each model is trained for 3000 epochs and we manually choose the best set of weights used for prediction for each model. The reason for this is that some models stops learning early while others need more time.

One-hot	0	0	1	0	0
Example transformation	0.1	0.3	0.5	0.05	0.05

Table 6.8: Translation from a 5-dimensional one-hot vector to a 5-dimensional vector of real values in the range [0,1].

Algorithm 1: One-hot transformation function which converts one-hot vectors into vectors of real values in range [0,1].

Data: L-dimensional One-hot vector, X
Result: L-dimensional vector of real values in the range [0, 1] that add up to 1

```

1  $Y = L$ -dimensional vector with values in the range[0,
0.3];
2  $i = \text{argmax}(X)$ ;
3 Get random number,  $R$ , in range [0.5, 1];
4  $Y[i] = R$ ;
5 return  $Y / \text{sum}(Y)$ 
```

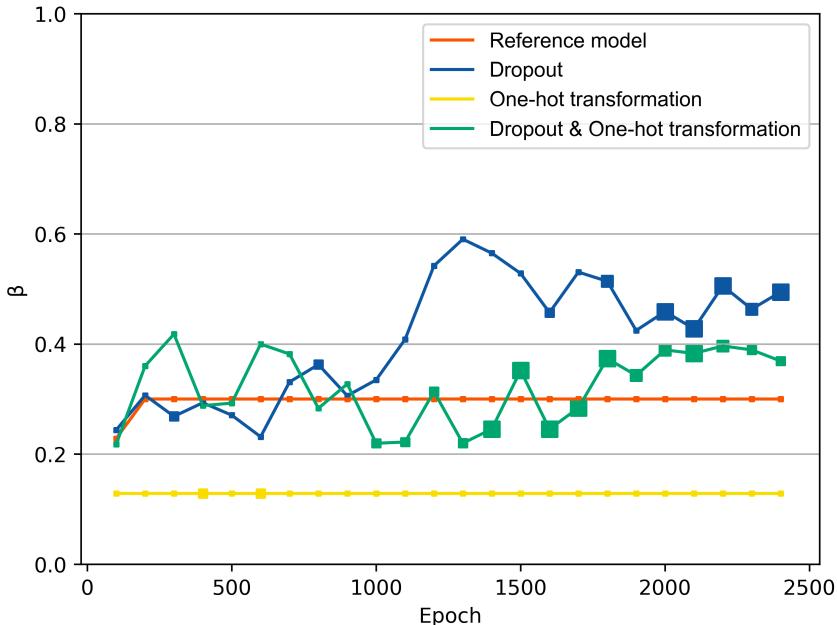


Figure 6.5: Graph comparing our custom BLEU score for 4 models using the softmax distribution. The x-axis displays the epoch, while the y-axis represents the score, β . Each color represents one model while the size of the squares of each point represents the number of distinct sentences created, with larger boxes being many distinct sentences.

6.3.2 Results

Table 6.9-6.12 shows the best observable sentences produced by the four different one-hot models after training. Figure 6.6 and Figure 6.7 display the loss value and the score value, β , during training of the generator. The loss values of the discriminator are omitted in the plots due to its value consistently being close to zero in all models. In Figure 6.5, a comparison of the development of our custom BLEU score, β , is shown for each model. The scoring function is described in detail in Section 6.2.1.

Reference Model

As seen in Table 6.9, this model learns the basic sentence structure, comprised of sentences starting with an $\langle \text{sos} \rangle$ token and ending with an $\langle \text{eos} \rangle$ token. It is not without flaws, as the start and end tokens sometimes appears multiple times and, in other cases, not at all. Syntactically and semantically, the model generally performs poorly. Other than learning that a sentence can start with “the petals” or “the flower”, it fails to generate plausible sequences of words. These sentences are produced after 63 epochs of training, which, as can be seen in Figure 6.6a, is the time the model stops learning. This is further discussed in Section 7.1.1.

Dropout Model

The dropout model produces some human-like sentences. Sentences generated after 2050 epochs of training are shown in Table 6.10. After epoch 2050 the model stops improving, and, to a certain degree, starts preforming worse. Figure 6.6b shows the development of the loss and β during the course of the training. After epoch 2050 the generator loss stops decreasing, and its β starts to fluctuate around 0.5. Only a small amount of the sentences produced by this model could be misjudged as human generated. Compared to the reference model, the dropout model is also able to produce and correctly place sentences containing all three of the special tokens. The start and end tokens appear once and the padding token, $\langle \text{pad} \rangle$, is repeated after the end token.

One-hot Transformation

Table 6.11 shows sentences generated by the one-hot transformation model, after 13 epochs of training. This model has a discriminator that examines sentences outputted by the one-hot transformation function. As can be seen, the only feature the model is able to learn is that each sentence should end with the $\langle \text{eos} \rangle$ token. In Figure 6.7a, it is also apparent that the model stops learning early in the training process.

One-hot Transformation and Dropout

Sentences generated by the final model is shown in Table 6.12. Most of the sentences produced after 80 epochs of training is structured correctly, and some also contain

words ordered in a syntactically correct order. When looking at the loss curve for this model in Figure 6.7b, it is clear that somewhere around epoch 80 the model stops learning, which is also where it stops producing correctly structured sentences.

```
<sos> the petals has has has has with with with with <eos>
<sos> the petals has has has has with with with with with
<sos> the petals has has has has with with with with <eos>
<sos> the flower has has has with with with with <eos> <eos>
<sos> <sos> has has has with with with with <eos> <eos>
```

Table 6.9: Generated sentences from reference model using one-hot vectors after 63 epochs of training.

```
<sos> a flower blue red red stamen <eos> <pad> <pad> <pad>
<sos> the flower has blue petals and dark middle <eos> <pad> <pad>
<sos> this flower has yellow petals and middle stamen <eos> <pad> <pad>
<sos> the purple petals <eos> <eos> <sos> the purple <sos> purple <eos>
<sos> this flower has yellow yellow petals and yellow stamen <eos> <pad>
```

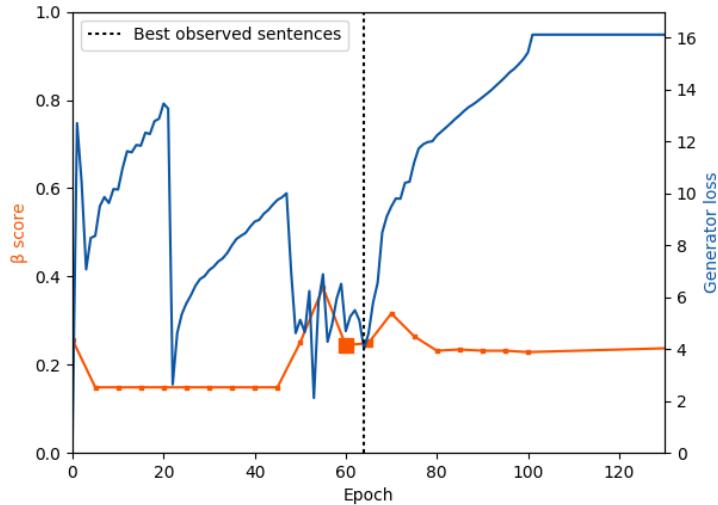
Table 6.10: Generated sentences using one-hot vectors and dropout in discriminator after 2050 epochs of training.

```
large large large large <eos> <eos> <eos> <eos> <eos> <eos>
large large large large <eos> <eos> <eos> <eos> <eos> <eos>
large large large large <eos> <eos> <eos> <eos> <eos> <eos>
large large large large <eos> <eos> <eos> <eos> <eos> <eos>
large large large large <eos> <eos> <eos> <eos> <eos> <eos>
```

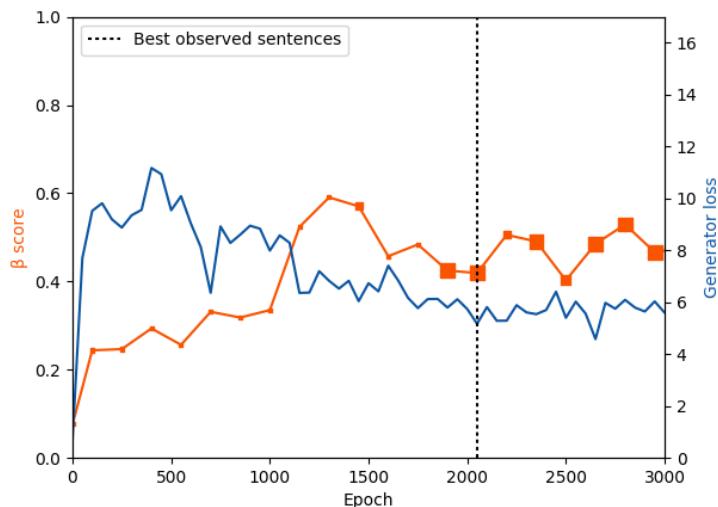
Table 6.11: Generated sentences using one-hot vectors and one-hot transformation in discriminator after 13 epochs of training.

```
<sos> this flower white flower has spiky layered petals to edges <eos>
<sos> this flower white flower has spiky layered petals to edges <eos>
<sos> this flower white flower has spiky layered petals to edges <eos>
<sos> this flower white flower has spiky layered petals to edges <eos>
<sos> <sos> pink pink and and main main main main main main
```

Table 6.12: Generated sentences using one-hot vectors, dropout and one-hot transformation in discriminator after 80 epochs of training.

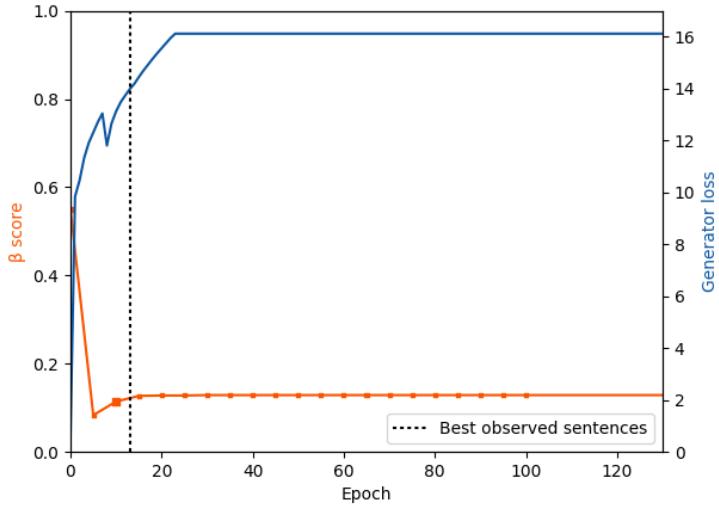


(a) Reference model. Discriminator not using dropout.

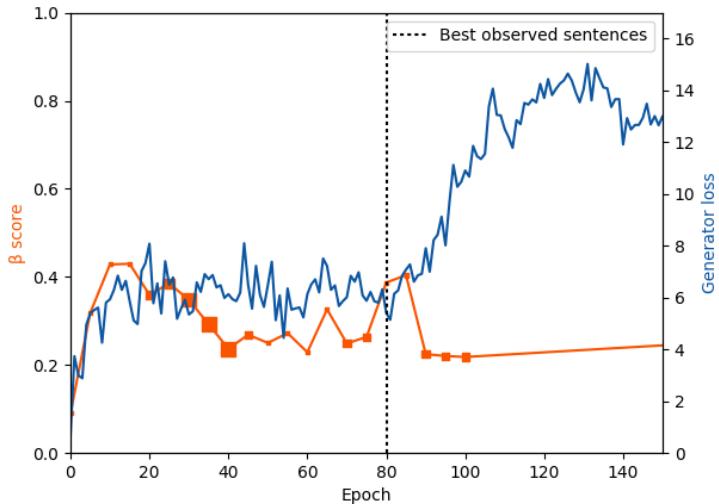


(b) Discriminator is using dropout.

Figure 6.6: Graphs showing how the loss and the score, β , of the generator changes during training for two models. The discriminators of both models are trained using one-hot vector representation of real training data.



(a) Discriminator implements one-hot transformation function.



(b) Discriminator implements one-hot transformation function and use dropout.

Figure 6.7: Graphs showing how the loss and the score, β , of the generator changes during training for two models. The discriminators of both models are trained using the one-hot obscurity function, which transform one-hot vectors into vectors containing a variety of numbers. The size of the squares of each point represents the number of distinct sentences created, with larger boxes being many distinct sentences.

6.4 Experiment 2 - Text Generation using the Word Embedding Model

Using discrete one-hot representations of words do not result in satisfactory sentences. In the following experiment, sparse representations of words are first converted into dense representations. This is accomplished using word embeddings, where words are represented as vectors in a multidimensional space. This will prevent the discriminator from easily distinguishing between real and generated data, just by looking at the data structure. Another positive side effect of this solution is that these vectors are also able to capture the semantic meaning of words. Similar words have shorter distances between them in the high-dimensional space, allowing the generator to approximate a vector that can be used to retrieve an actual word. The major drawback with the word embedding approach is that the generator has a vastly larger search space, potentially making it harder for the generator to recognize patterns and good solutions.

6.4.1 Experimental Setup

To represent words as vectors in an high-dimensional space, either pre- or custom-trained word embeddings are needed. We experiment with both pre-trained word embeddings based on the GloVe algorithm, and word embeddings custom trained on our datasets using Word2Vec, see Section 6.1.2. With regards to the custom-trained word embeddings, two different sets, one trained on captions from the Oxford-102 flower dataset and one trained on captions from the Flickr30K datasets are trained. Both sets of word embeddings are represented using 50-dimensional vectors. The reasoning behind training multiple sets of word embeddings is to capture each dataset’s specific syntax and custom words, e.g. words such as “petals” and “stamen” from the Oxford-102 flower dataset.

The initial experiment compares the impact of different dropout ratios, ranging from 0.25 to 0.99. The following experiment examines how the complexities of the generator and discriminator affects learning and performance. The third experiment compares two models trained on different sets of word embeddings. All three of these experiments use captions from the Oxford-102 flower dataset as training examples. The final experiment is trained using the Flickr30k dataset, which contains sentences with a larger variety in both syntax and semantics. An overview of the architecture for every experiment is described in Section 5.2.

6.4.2 Results

The different models are evaluated using both the evaluation score, β , discussed in Section 6.2.1, and by observing the generated sentences created by the different models. The evaluation score is calculated for each model by taking the average score for 100 generated sentences.

Comparing Dropout

Figure 6.8 compares the evaluation score, β , for four different models. Each of the models implement different values for dropout in the discriminator, while otherwise being identical. As seen in the figure, a dropout ratio around 0.25 consistently gives the highest β value. On the other end, a dropout ratio equal to 0.99 denies the model from learning, since too much of the training data is disregarded by the discriminator. This is also apparent when looking at actual sentences produced by these models, shown in Table 6.13. The most evident difference between the four models is the one implementing a dropout ratio of 0.99. As seen in the table, this model fails to learn. In the other models, the differences are not so apparent. However, when looking at multiple predictions, a trend is that the models with smaller dropout ratio contains sentences with more variation. A larger table with predictions is found in Table A.1.

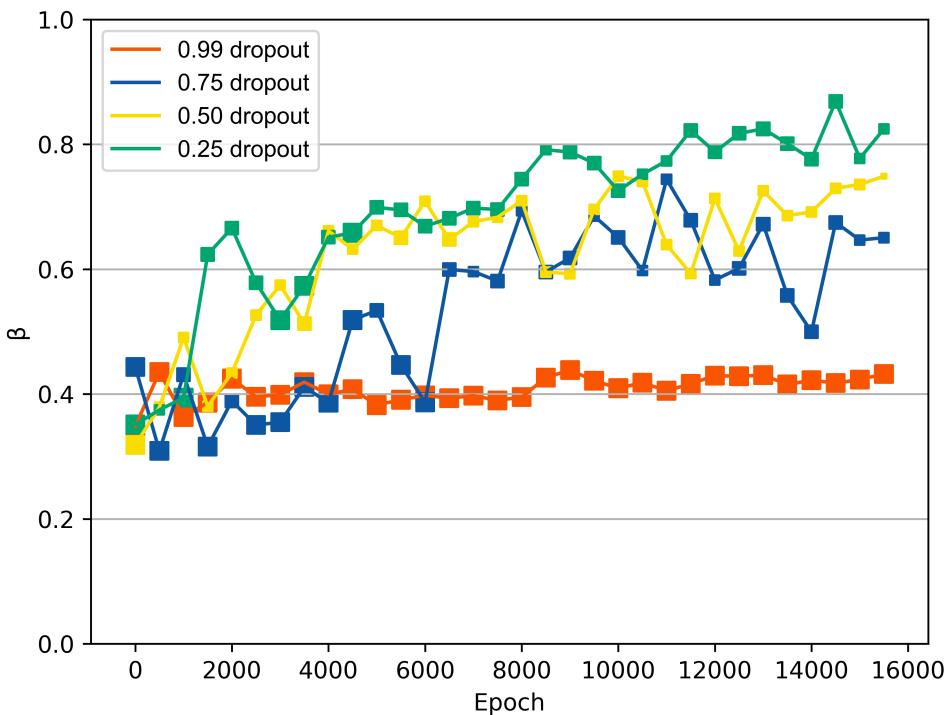


Figure 6.8: Graph comparing our custom BLEU score for 4 models using different values for dropout in the discriminator. The x-axis displays the epoch, while the y-axis represents the score, β . Each color represents one model while the size of the squares of each point represents the number of distinct sentences created, with larger boxes being many distinct sentences.

```

0.25 dropout:
<sos> this flower has petals that are pink with flowery stigma <eos>
<sos> this flower has petals that are white with green stamen <eos>
0.50 dropout:
<sos> this flower has petals that are purple with red stamen <eos>
<sos> this flower has petals that are pink with white stamen <eos>
0.75 dropout:
<sos> this flower has petals that are pink with red stamen <eos>
<sos> a purple petals with a <eos> number with are pistol <eos>
0.99 dropout:
sit lilac curves such hairs whose lilac appear quite lilac bulbous huge
polka reach get lilac cup-shaped curling sits sit closer going sit clam

```

Table 6.13: Randomly chosen generated sentences for four different dropout ratios. Every sentence is generated after 16000 epochs of training.

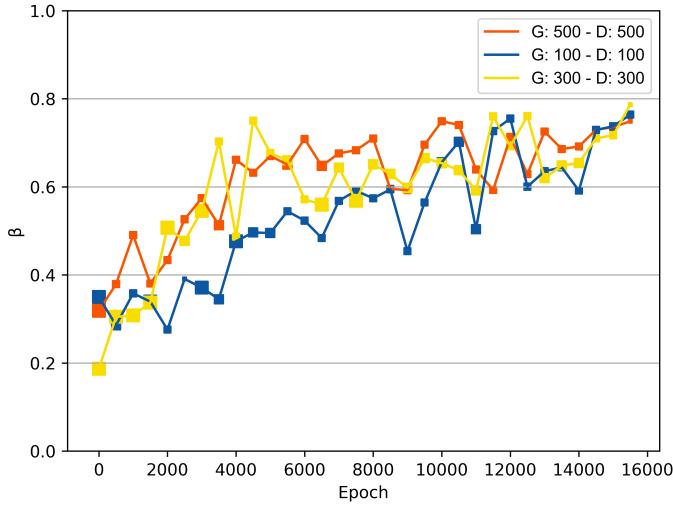
Comparing Complexity

Another interesting comparison is the complexity of the generator and the discriminator. A larger model takes longer time to train, but it is interesting to research if it performs better. Figure 6.9 shows the development of β . The models in these graphs are all trained on the Oxford-102 flower dataset, and is trained for 16000 epochs. Figure 6.9a compares three models where the generator and the discriminator have equal sizes. By looking at the graph in the figure there is no clear evidence that one model performs better than others. This is also the case when we compare models where the generator and the discriminator have unequal sizes. However, when looking at the generated sentences, there is some differences worth noticing. In Table 6.14, 10 generated sentences from two models; one with generator and discriminator with a size of 100, and the other with a size of 500, are compared. There are two noticeable differences in the sentences generated by these models. The first is that the larger model contains less erroneous sentences. The second and more apparent difference is the lack of diversity in the smaller model. The sentences in the table are sorted alphabetically, and it is clear that the smaller model has generated multiple identical sentences. On the other hand, the larger model has some duplicates, but overall contains much more variety.

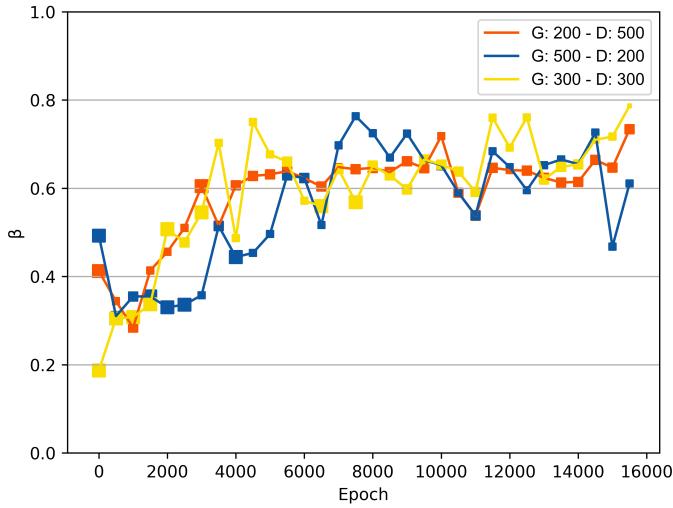
Comparing Vocabulary Size

As our methods rely heavily on good embeddings representing the different words, it is interesting to see how different sets of word embeddings influence performance of the models. As seen in previous experiments, the models using custom made word embeddings, that are trained on the Oxford-102 flower dataset, performs well after a sufficient amount of epochs.

Figure 6.10 compares the β score for a reference model, the best model from previous experiments and an identical model using a pre-trained set of word embeddings. These word embeddings are obtained using the GloVe algorithm and contains a vocabulary of 400 000 words. The usage of GloVe is explained in detail in Section 6.1.2. In contrast, the custom trained word embeddings obtained



(a) Comparison of models where generator and discriminator have equal size.



(b) Comparison of models where generator and discriminator have different amounts of long short-term memory (LSTM) hidden units.

Figure 6.9: Graphs showing how the score, β , of the generator during training for multiple models. Size of generator and discriminator is denoted in the legend by the letter G and D, respectively. The x-axis displays the epoch, while the y-axis represents the score, β . Each color represents one model while the size of the squares of each point represents the number of distinct sentences created, with larger boxes being a large percentage of distinct sentences.

```

Generator: 100 - Discriminator: 100
<sos> this flower has yellow pistil and a stamen <eos> <pad> <pad>
<sos> this flower has <sos> that are gold a stamen <eos> <pad>
<sos> the flower has petals that are pink with red stamen <eos>
<sos> this flower has petals that are pink with red stamen <eos>
<sos> this flower has petals that are pink with red stamen <eos>
<sos> this flower has petals that are pink with red stamen <eos>
<sos> this flower has petals that are pink with red stamen <eos>
<sos> this flower has petals that are pink with red stamen <eos>
<sos> this flower has petals that are pink with red stamen <eos>
<sos> this flower has petals that are pink with red stamen <eos>
<sos> this flower has petals that are pink with red stamen <eos>
Generator: 500 - Discriminator: 500
<sos> a flower with long and thick petals that are pink <eos>
<sos> the petals are delicate and red pedals that are pink <eos>
<sos> the petals on this flower are purple with red stamen <eos>
<sos> this flower has petals that are pink with filament <eos> <pad>
<sos> this flower has petals that are pink with white stamen <eos>
<sos> this flower has petals that are purple with red stamen <eos>
<sos> this flower has petals that are purple with red stamen <eos>
<sos> this flower has petals that are white and folded together <eos>
<sos> this pink flower has flower flat and large white petals <eos>
<sos> this pink flower has flower flat and large white petals <eos>

```

Table 6.14: 10 generated sentences for two different models. Every sentence is generated after 16000 epochs of training. The only difference in the models is the amount of LSTM hidden units of the generator and the discriminator.

using Word2Vec has a vocabulary size of 1000. The figure shows only the first 3500 epochs for the model using word embeddings from GloVe. The reason for this is the lack of improvement in the model, which is apparent when looking at Table 6.15. The table shows the sentences produced by the model applying GloVe word embeddings. As seen in the table, the sentences completely lack any kind of syntactical structure or semantic meaning. The reference model has, on the other hand, learned to produce some good sentences. An interesting result is the high β score the model receives, despite the poor quality of the generated sentences. This is further discussed in Chapter 7.

Sentence variation

During training, the loss of the generator and discriminator starts moving apart from each other. Until this point, the generator has learned what kind of sentences are able to fool the discriminator, and starts to mainly produce these sentences. This can be seen in Table 6.16. After epoch 2500, the generator performs well on some sentence structures, while failing to capture others. After a significant amount of further training, the generator is forced to explore more and is able to find other meaningful sentences to generate. This is also visible in the table, as the structure of the sentences generated is more diverse. Similarly to samples generated using GAN in the visual domain, the generator is able to generate sentences not seen in the training data. The development of the loss for this model is seen in Figure 6.11.

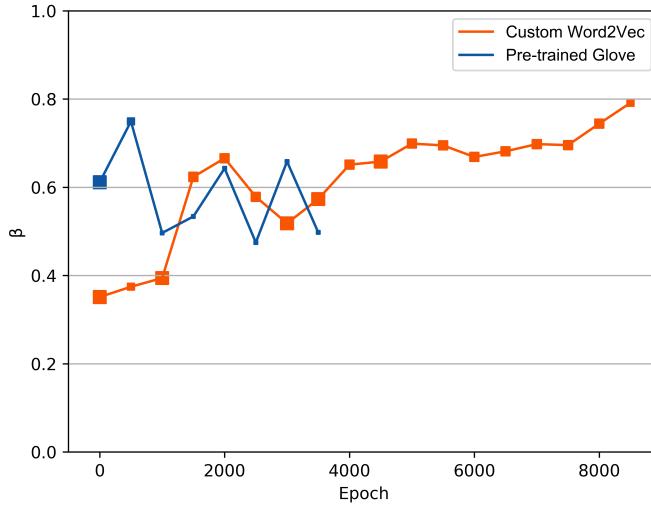


Figure 6.10: A comparison between using custom trained Word2Vec and pre-trained GloVe word embeddings. The size of the squares of each point represents the number of distinct sentences created, with larger boxes being a large percentage of distinct sentences.

```

Model using pre-trained GloVe word embeddings - 400 000 vocabulary size
<sos> plot raincoat petals attractions wielded fuchsia gently nut ol <pad><pad>
<sos> plot raincoat petals attractions wielded fuchsia gently nut ol <pad><pad>
<sos> plot raincoat petals attractions wielded fuchsia gently nut ol <pad><pad>
<sos> plot raincoat petals attractions wielded fuchsia gently nut ol <pad><pad>
<sos> plot raincoat petals attractions wielded fuchsia gently nut ol <pad><pad>
<sos> plot raincoat petals attractions wielded fuchsia gently nut ol <pad><pad>
<sos> plot raincoat petals attractions wielded fuchsia gently nut ol <pad><pad>
<sos> plot raincoat petals attractions wielded fuchsia gently nut ol <pad><pad>
<sos> manhattan shiny petals attractions wielded fuchsia gently nut ol <pad><pad>
<sos> plot raincoat petals attractions wielded fuchsia gently nut ol <pad><pad>
<sos> plot raincoat petals attractions wielded fuchsia gently nut ol <pad><pad>

Model using custom trained Word2Vec word embeddings - 1000 vocabulary size
<sos> this flower has petals that are yellow with yellow stamen <eos>
<sos> this flower has petals that are yellow with yellow stamen <eos>
<sos> this flower has petals that are yellow with yellow stamen <eos>
<sos> this flower has petals that are yellow with yellow stamen <eos>
<sos> this flower has petals that are yellow with yellow stamen <eos>
<sos> the flower has blue dozens flower a purple <pad> petals <eos>
<sos> the flower has petals thin are has purple yellow stamen <eos>
<sos> the petals has blue thick petals <eos> petals a also <eos>
<sos> the petals has blue thin flower a and <pad> petals <pad>
<sos> a flower has and and are petals yellow purple <eos>

```

Table 6.15: 10 generated sentences for two different models. Every sentence is generated after 3000 epochs of training. The only difference is the set of word embeddings used.

2500 epochs of training

```
<sos> this flower has petals that are yellow with flower stamen <eos>
<sos> this flower has petals that are yellow with flower dots <eos>
<sos> this flower has petals that are yellow with yellow trim <eos>
<sos> this flower has petals that are yellow with yellow dots <eos>
<sos> this flower has petals that are yellow with yellow dots <eos>
<sos> this flower has petals that are yellow with yellow dots <eos>
<sos> this flower has petals that are yellow with yellow dots <eos>
<sos> this flower has petals that are yellow with yellow dots <eos>
<sos> this flower has petals that are yellow with yellow stamen <eos>
<sos> this flower has petals that are yellow with yellow stamen <eos>
```

15000 epochs of training

```
<sos> a bright white large red flower with a spiky <pad> <eos>
<sos> a flower with orange petals and a green stamen <eos> <pad>
<sos> a bright white large red flower with a pistil <pad> <eos>
<sos> a flower with orange petals and a green stamen <eos> <pad>
<sos> a flower with orange petals and a green stamen <eos> <pad>
<sos> a flower with orange petals and a green stamen <eos> <pad>
<sos> this flower has orange petals and a green stamen <eos> <pad>
<sos> a bright white large red flower with a green <pad> <eos>
<sos> the petals on this flower are white with white stamen <eos>
<sos> this pink shaped flower has dark petals and pink spots <eos>
```

Table 6.16: 10 generated sentences after two different epochs of training for the word embedding model.

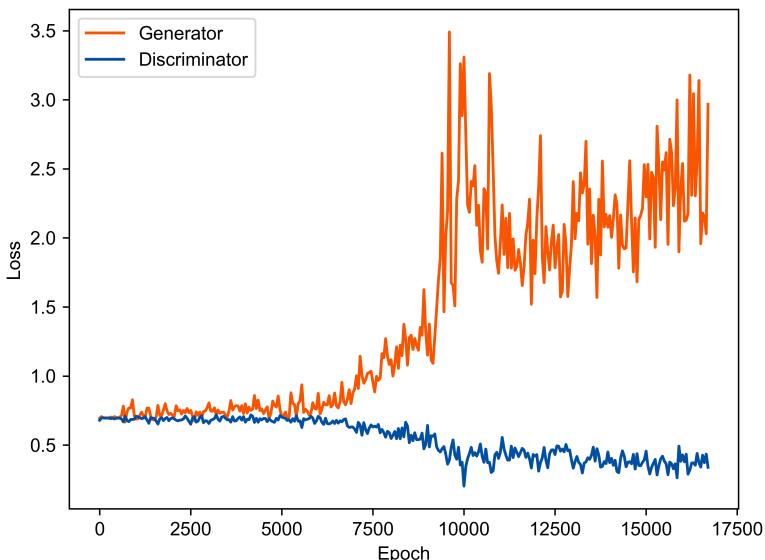


Figure 6.11: The development of loss for generation and discriminator. The model is trained on the Oxford-102 flower dataset using the word embedding approach.

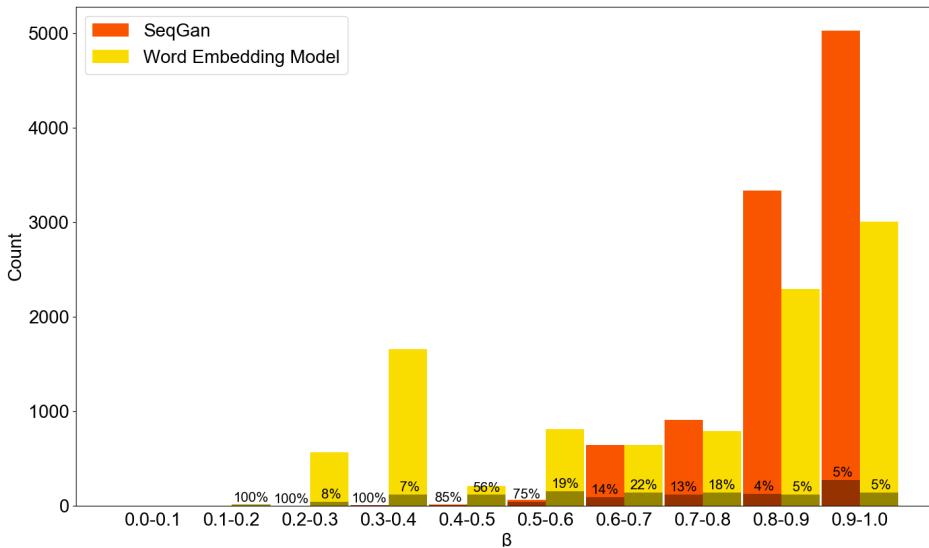


Figure 6.12: A β score comparison between the word embedding model and the baseline sequence generative adversarial nets (SeqGAN) model trained on the Oxford-102 flower dataset. Both have generated 10000 sentences. The sentences are placed in buckets of size 0.1, indicating their β score. The percentage of unique sentences in every interval is indicated with text and a darker shading of the corresponding model’s color.

An interesting comparison between our word embedding model and the baseline is seen in Figure 6.12. This figure shows a β score comparison between these two models having generated 10000 sentences, trained on the Oxford-102 flower dataset. The sentences are placed in buckets of size 0.1, indicating their β score. The percentage of distinct sentences for each bucket is displayed in darker shading. It is clear from the figure that our word embedding model produces sentences with a lower beta-score than the baseline. An interesting observation is that the percentage of distinct sentences is quite similar within each of the buckets. The two buckets representing β values between 0.8-0.9 and 0.9-1.0 have a difference of 1% and 0%, respectively, for the unique sentences ratio. An interpretation of this is that once a model finds a good sentence, it starts producing many identical sentences, which results in the model being awarded with a higher score.

Introducing Flickr30k

The previous results show that models trained on the Oxford-102 flower dataset learn to create new meaningful sentences. The best models apply a combination of custom created word embeddings with a small vocabulary size and a small dropout ratio in the discriminator. The captions from the Oxford-102 flower dataset are, however, quite syntactically and semantically similar. Most of the sentences begin and end with the same structure allowing the models to thoroughly learn this

	Softmax	Word Embedding	Baseline
Oxford-102 flower	0.5254 (76.86%)	0.7917 (9.95%)	0.9460 (6.76%)
Flickr30k		0.6353 (88.41%)	0.8444 (28.91%)

Table 6.17: Score, β , for the best model representing words as one-hot vectors and word embeddings, and the baseline model. The models are evaluated on two different datasets. The score is a calculated average from 10 000 generated sentences for each of the models. In parentheses behind the score is the percentage of how many unique sentences each model generated.

Word embedding model

```
<sos> a newspaper is on a trees with the woods <eos> <pad>
<sos> a man in his blue frisbee in the <eos> <pad><pad>
<sos> two men on a green on the bar <eos> <pad><pad>
<sos> a man of people are in the background log <eos> <pad>
<sos> two men are holding on the room on the bar <eos>
<sos> people are with a room and on the bar <eos> <pad>
<sos> two men dancing in in the background <eos> <pad><pad><pad>
<sos> people walking with leaps is hill <eos> <pad><pad><pad><pad>
<sos> a man wearing white are on the background <eos> <pad><pad>
<sos> a dog is playing at a <eos> <pad><pad><pad><pad>
```

SeqGAN

```
<sos> a performing <eos> <pad><pad><pad><pad><pad><pad><pad>
<sos> a construction worker <eos> <pad><pad><pad><pad><pad><pad><pad>
<sos> a man surfing a wave <eos> <pad><pad><pad><pad><pad><pad>
<sos> two dogs play in the snow <eos> <pad><pad><pad><pad><pad>
<sos> a brown dog is running down the beach <eos> <pad><pad>
<sos> a man sitting on a bench <eos> <pad><pad><pad><pad>
<sos> a man surfing <eos> <pad><pad><pad><pad><pad><pad>
<sos> a dog runs in the snow <eos> <pad> <pad><pad><pad><pad>
<sos> a man his <eos> <pad><pad><pad><pad><pad><pad>
<sos> a bicyclist in the snow <eos> <pad><pad><pad><pad><pad>
<sos> a person <eos> <pad><pad><pad><pad><pad><pad><pad>
```

Table 6.18: 10 generated sentences from SeqGAN and the word embedding approach. Both models are trained on the Flickr30k dataset.

structure. It is therefore interesting to look at a more complex dataset, containing sentences with a much larger variety.

In contrast to the Oxford-102 flower dataset, the Flickr30k dataset contains sentences describing a vast variety of images. Example images with corresponding captions from both datasets are given in Figure 6.2 and Figure 6.1, respectively. As a result of the large variety in sentences describing images, our models need more training to start producing meaningful sentences. Table 6.17 compares the average score from 10000 generated sentences for both our models and the baseline model. In parentheses behind the score is the percentage of how many unique sentences each model generated. The softmax distribution model is not trained on the Flickr30k dataset since the word embedding approach achieves greater success on the Oxford-102 flower dataset. In the table, it is apparent that the baseline model achieves a higher β score than both our models. Another observation is that a lower β score gives a higher variance in the sentences created.

In Table 6.18, inference from both the word embedding model and SeqGAN is shown. Our model creates sentences with correct placement of the start-of-

sequence, end-of-sequence and padding tokens. In addition, the first few words of the sentence usually fit together and the structure of the sentence often looks similar to that of a real sentence, even if the words do not provide any meaning. Starting from the beginning of the sentences, the model starts to learn a promising structure. Additionally, it is apparent that the model has learned certain phrases and combinations of words such as “a dog is”, “two men are” and “in the background”. As seen in Table 6.18, SeqGAN is able to produce more meaningful sentences with correct structure and grammar, than our model.

6.5 Experiment 3 - Image Captioning

The goal of this thesis is to research whether GANs can be used to do controllable text generation and image captioning, e.g. given an image, generate a suitable textual description. During the text generation experiments in Section 6.3, the generator is given random noise to generate sentences, without any possibility of controlling the outputted sentences. The following experiments investigate if it is possible to combine the random noise with some user-defined data, which gives control of the generation of sentences. The architecture used for these experiments is defined in Section 5.3. The following experiments investigate whether this architecture can be used to control text generation, and research if it can be further extended to perform well when generating image captions.

6.5.1 Experimental Setup

As with text generation, the generator’s task is to create sentences similar to a set of given examples. The main difference from previous experiments is the usage of the noise vector, z . The noise vector is combined with a Z -dimensional control vector, c . The task of this control vector is to influence the sentences outputted by the generator. Each caption is paired with a control vector, where captions describing a specific flower are paired with the vector representing the flower. During training, the discriminator is given these pairs of data to learn the relationship between captions and image data. The generator is given the combination of noise and image data to produce sentences. These are then propagated through the discriminator, alongside the control vector representing the image. As a consequence of this architecture, the generator is given feedback on how well the generated sentence fits the control vector, in addition to the feedback on the authenticity of the generated sentences. When generating new image captions after training, the generator is given a control vector representing a new image. Thus creating a description for this image.

Images in these experiments are represented as 50-dimensional vectors. These vectors are obtained using a pre-trained version of the Inception image classifier [Szegedy et al., 2015]. By removing the last classification layer from the Inception model, we are able to obtain a 2048-dimensional vector representing an image. To reduce the complexity of our models, we reduce the dimensionality of the 2048-dimensional vectors into 50-dimensional vectors. This is accomplished using the



Figure 6.13: Example images of three flower species.

principal component analysis (PCA) algorithm described in Section 2.5.1. Words in the image captioning experiments are represented by 50-dimensional Word2Vec-vectors trained on the Oxford-102 flower dataset.

6.5.2 Results

The initial experiment consists of multiple simplifications. The result from this experiment is used to further investigate the image captioning capabilities of GANs.

Controllable text generation

In this experiment, we collect the captions of two different flowers, “Geranium” and “Californian poppy”. The “Geranium” is a red flower with many petals, while the “Californian poppy” is yellow flower with few petals, see Figure 6.13a and Figure 6.13b, respectively. These flowers have clear and distinct features, which make them distinguishable. The model is trained on all captions describing these two images. A major simplification of this experiment is the representation of images. We do not represent each image as an individual control vector obtained using Inception and PCA. Instead we use a single control vector for each of the flower species. The reason behind this simplification is to research whether it is possible to control the generated sentences by changing which of the two vectors is used during inference.

Table 6.19 shows 10 generated sentences after training the image captioning model. The first 5 sentences are generated when the generator of the model is given the red flower vector as control vector. The last 5 are generated when the control vector is equal to the vector representing the yellow flower.

By observation, it is clear that the produced sentences have been influenced by the control vectors. When the generator is given a control vector representing the red flowers, the generator is clearly using the word “red” more frequently than other colors. The same observation can be done when giving the generator the control vector representing the yellow “Californian poppy”. The generator in this case does only produce sentences containing the color yellow. The sentences do, however, lack variation, but previous experiments have shown that this could be related to the training time.

Captions describing the red Geranium

<sos> this flower has petals that are red with short steman <eos>
 <sos> this flower is small red petals with petals red stamens <eos>
 <sos> this flower has petals that are red and bunched together <eos>
 <sos> this flower has petals that are red with small steman <eos>
 <sos> this flower has petals that are red with orange steman <eos>

Captions describing the yellow Californian poppy

<sos> this flower has petals that are yellow with yellow stamen <eos>
 <sos> this flower has petals that are yellow with yellow stamen <eos>
 <sos> this flower has petals that are yellow with yellow stamen <eos>
 <sos> this flower has petals that are yellow with yellow edges <eos>
 <sos> this flower has petals that are yellow with yellow stamen <eos>

Table 6.19: Sentences generated by the controllable text generation model. This model is trained on captions describing the two flowers “Geranium” and “Californian poppy”. Only a single control vector is used for each of the two flower species.



(a) Geranium

(b) Grape hyacinth

(c) Californian poppy

Figure 6.14: Validation images for the three flower species the model is trained on.

Image captioning

The previous experiment shows clear indications that we are able to control the sentences generated by the model. This new experiment tries to demonstrate that this architecture is applicable on the image captioning problem. In this experiment we increase the complexity by introducing two more factors. The first is another flower, the “Grape hyacinth”, see Figure 6.13c. This flower is chosen for its blue and distinct color that is easily distinguished from the two already introduced flowers. The second change from the last experiment, is the introduction of all images of these three flowers, rather than using a single image vector for each flower type like in the previous experiment. The idea behind this choice is to allow the model to learn the role of the control vector. Which in this case, is to control the colors and features used in the generated sentences.

Table 6.20 lists the generated captions when the model is given three different image vectors, one for each flower type. The three images used as control vectors have been used as examples during training. It is apparent by observation that the model is able to recognize and use image vectors to generate sentences relevant to the specific flowers, in terms of both color and features. The table clearly shows

Captions describing the red Geranium

<sos> this flower has petals that are red and bunched together <eos>
<sos> this flower has petals that are red and bunched together <eos>
<sos> this flower has petals that are red and bunched together <eos>
<sos> this flower has petals that are red and bunched together <eos>
<sos> this flower has petals that are red and bunched together <eos>

Captions describing the yellow Californian poppy

<sos> this flower has petals that are yellow with yellow stamen <eos>
<sos> this flower has petals that are yellow with yellow stamen <eos>
<sos> this flower has petals that are yellow with yellow stamen <eos>
<sos> this flower has petals that are yellow with yellow stamen <eos>
<sos> this flower has petals that are yellow with yellow stamen <eos>

Captions describing the blue Grape hyacinth

<sos> this flower has petals that are blue and fused together <eos>
<sos> this flower has petals that are blue and closed together <eos>
<sos> this flower has petals that are blue and closed together <eos>
<sos> this flower has petals that are blue and closed together <eos>
<sos> this flower has petals that are blue and bunched together <eos>

Table 6.20: Sentences generated by the image captioning model. This model is trained on images and captions describing the flowers “Geranium” and “Californian poppy” and “Grape hyacinth”. Sentences are generated using images from the training set as control vectors.

that the image captioning model is able to recognize already seen control vectors, and create meaningful sentences related to an image. However, a more interesting experiment is to see whether the model is able to generalize over the control vector. To test this, we exchange the control vectors from the previous results. Instead of using control vectors from the training data, the model is, in this experiment, given images from a validation set, i.e. previously unseen images.

Figure 6.14 shows the three images used in the validation experiment. As seen in the figure, the three images are quite similar to the three example images illustrated in Figure 6.13. During training, the image captioning model has seen many different images of the three different flower types. Unseen images is chosen to determine if our model sees the correlation between these images, and the images used during training. Table 6.21 lists sentences generated by our model, when the control vectors are represented by previously unseen images. By observation, it is clear that the model has been able to generalize, and is still producing meaningful sentences, describing the three different flowers. The similarity of the generated sentences, demonstrates lacking variety in the generator’s output, but as mentioned earlier, this could be related to training time.

6.6 Pre-Training: Sequence-to-Sequence

In this section, we present a pre-training approach using an autoencoder architecture and the corresponding results from experiments with it. A commonly used approach to give GANs a good initial starting point for adversarial training, is to pre-train it in a supervised manner. A sequence-to-sequence model that encodes

Captions describing the red Geranium

```
<sos> this flower has petals that are red with red stamen <eos>
<sos> this flower has petals that are red with red steman <eos>
<sos> this flower has petals that are red with red stamen <eos>
<sos> this flower has petals that are red with red stamen <eos>
<sos> this flower has petals that are red with red stamen <eos>
```

Captions describing the yellow Californian poppy

```
<sos> this flower has petals that are yellow with yellow stamen <eos>
<sos> this flower has petals that are yellow with yellow stamen <eos>
<sos> this flower has petals that are yellow with yellow stamen <eos>
<sos> this flower has petals that are yellow with yellow stamen <eos>
<sos> this flower has petals that are yellow with yellow stamen <eos>
```

Captions describing the blue Grape hyacinth

```
<sos> this flower has petals that are blue and bunched together <eos>
<sos> this flower has petals that are blue and bunched together <eos>
<sos> this flower has petals that are blue checkered bunched together <eos>
<sos> this flower has petals that are blue and bunched together <eos>
<sos> this flower has petals that are blue and bunched together <eos>
```

Table 6.21: Sentences generated by the image captioning model. This model is trained on images and captions describing the flowers “Geranium” and “Californian poppy” and “Grape hyacinth”. Sentences are generated using images from the validation set as control vectors.

the input, and decodes it to output the same sequence, is able to learn how to produce sequences and can serve as the initial starting point for our generator. The idea is to allow the GAN to have a model that is able to produce intelligible language from the beginning of training. This should also initialize the generator close to a set of weights that makes it able to produce reasonable sentences, which, in turn, could give it a higher chance of converging.

The model is shown in Figure 6.15 and is split into two parts, the encoder and decoder. Before the input sequence is inputted into the encoder part of the network, it is embedded using a pre-trained word embedding with a specified embedding dimension. The encoder part of the network consists of a function, $h = f(x)$, that represents the entire caption as one latent vector, z . This vector is then fed into the decoder, which can be seen as a function that produces a reconstruction, $r = g(z)$. The decoder estimates a vector that needs to be compared to the representations of words provided by the pre-trained word embedding to retrieve the most similar word using cosine similarity. Both the encoder and decoder consists of one or more LSTMs.

Since this is a regression-based model and works by estimating a word vector that needs to be retrieved, the difference between the correct and the estimated word vector needs to be minimized. This is done using the mean squared error (MSE) loss function together with the adam optimizer [Kingma and Ba, 2014].

When experimenting with different hyperparameters it is clear that giving the model sufficient dimensionality allowed it to reach a higher accuracy, see Table 6.22. A large hidden layer size, using 300-dimensional word embeddings, shows that the sequence-to-sequence model is able to reproduce longer sentences with over 90 % accuracy. While deep LSTMs have shown better results [Sutskever et al., 2014], we

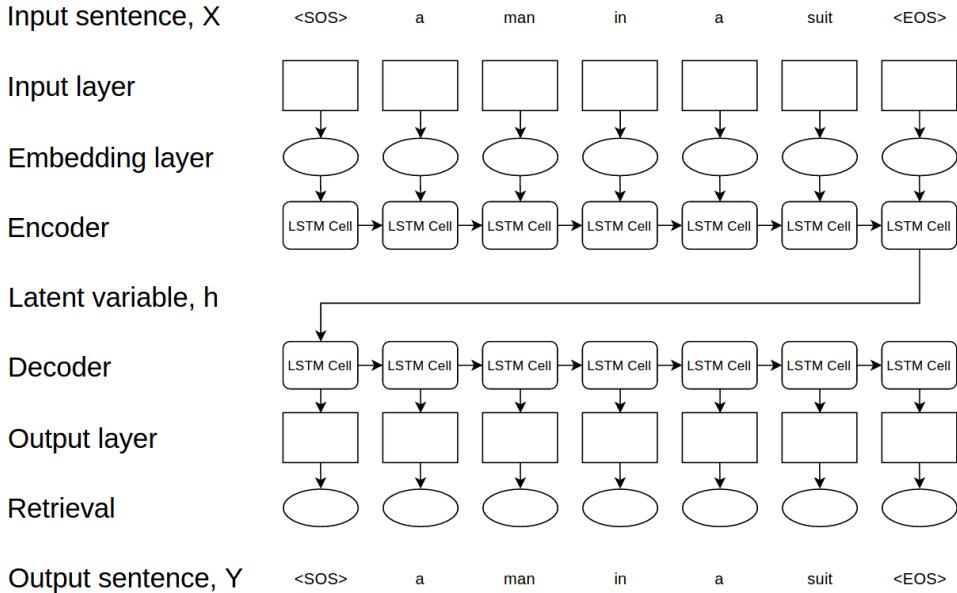


Figure 6.15: The unfolded sequence-to-sequence model.

Hidden layer size	MRR	R@1	R@5	R@10	R@20
256	0.921	0.907	0.913	0.914	0.914
512	0.926	0.914	0.915	0.915	0.915

Table 6.22: Recall at K metrics and mean reciprocal rank of two sequence-to-sequence runs with 256 and 512 hidden units.

mostly experimented with single layer LSTMs. This is because we want to limit the complexity of the GAN, which the weights are transferred to. 50-dimensional word embeddings is a suitable dimensionality to be used for the generator without sacrificing too much accuracy, as seen in Table 6.23. The 20-dimensional embedding showed poor top-1 recall, but showed significantly better top-5 recall results, while both the 50-dimensional and 300-dimensional show good and not very dissimilar results. As seen in the table, the 50-dimensional embedding model has higher 5, 10 and 20 recall values than the 300-dimensional embedding model. A possible reason for this could be that high-dimensionality might make it more difficult to find rare words. Words could be positioned further apart in the 300-dimensional space than the 50-dimensional space, forcing the model to make larger adjustments to its weights.

While both the 50- and 300-dimensional embeddings provide sufficient results, as seen in Figure 6.24, accuracy suffered significantly when going down to the 20-dimensional embedding, see Table 6.23. For example, in five of the six sentences shown in Figure 6.24 wide is retrieved as the first word, while correct word, $<\text{sos}>$, is retrieved as the third closest word.

Embedding dimension	MRR	R@1	R@5	R@10	R@20
20	0.358	0.270	0.459	0.586	0.731
50	0.747	0.683	0.829	0.856	0.881
300	0.783	0.747	0.785	0.806	0.830

Table 6.23: Recall at K metrics and mean reciprocal rank of two sequence-to-sequence runs with 20, 50 and 300 in embedding dimension.

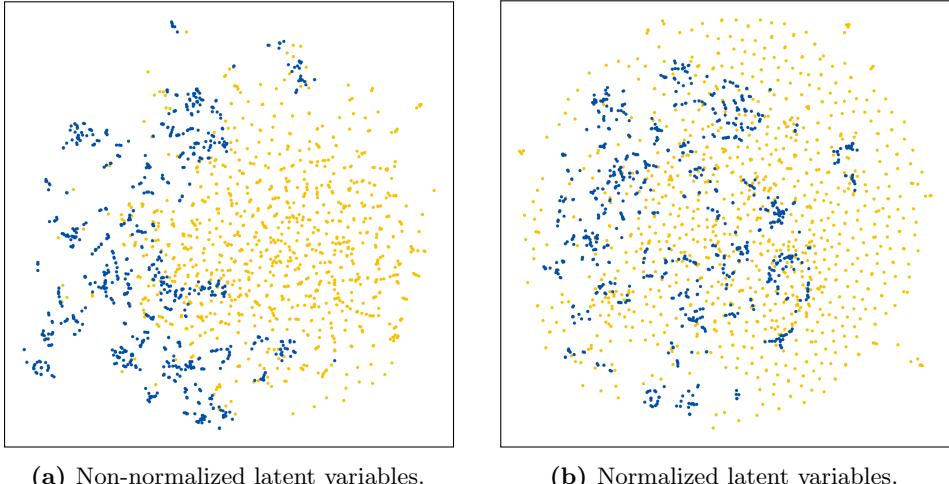


Figure 6.16: Latent variables from the sequence-to-sequence model (blue) compared with a gaussian distribution (yellow) visualized using t-distributed stochastic neighbor embedding (t-SNE) dimensionality reduction

6.6.1 Incompatibility

A challenge of using sequence-to-sequence pre-training to initialize GANs is the incompatibility between the latent representation and the random input given to the generator. While noise used in the GAN is drawn from a gaussian random distribution, the distribution of latent variables produced by the encoder has a different distribution, as seen in Fig 6.16a.

An approach to deal with this is to use a normalization layer in the encoder to normalize the latent variables into a subset of the gaussian distribution, as seen in Figure 6.16b. While this improves the sequence-to-sequence model's ability to generalize, it does lead to a slight decrease in accuracy.

6.6.2 Results

Despite trying different approaches to make the noise and latent distributions compatible and various hyperparameters, the GAN training process shows no significant results. After one epoch, the discriminator is able to correctly classify all real and generated data, see Figure 6.17. The reasoning for this is still unclear.

RS	<sos>	a	surfer	catches	a	huge	wave	:	<eos>	<pad>
I20	drum	filled	skier	concert	each	clown	pier	:	<eos>	pier
I50	<sos>	a	surfer	pulling	a	huge	wave	:	<eos>	.
I300	<sos>	a	woman	next	a	<pad>	street	.	<eos>	<pad>
RS	<sos>	two	women	are	walking	on	the	beach	with	surfboards
I20	wide	many	others	are	standing	on	covered	beach	by	parade
I50	<sos>	two	ladies	are	walking	before	sand	sand	with	UNK
I300	<sos>	two	women	are	walking	on	the	beach	with	UNK
RS	<sos>	two	women	are	walking	down	the	beach	carrying	boards
I20	wide	many	others	are	standing	down	covered	beach	market	cane
I50	<sos>	two	ladies	are	walking	riding	sand	grass	holding	UNK
I300	<sos>	two	women	are	walking	down	the	beach	wearing	UNK
RS	<sos>	two	surfers	preparing	to	enter	the	water	for	a
I20	wide	many	cane	cafe	to	parade	covered	pier	what	each
I50	<sos>	two	UNK	preparing	to	UNK	sand	water	for	a
I300	<sos>	two	UNK	dressed	to	UNK	the	water	for	a
RS	<sos>	2	woman	surfers	carrying	their	boards	across	the	beach
I20	wide	many	woman	cane	market	their	cane	along	drum	beach
I50	<sos>	three	woman	UNK	carrying	his	UNK	through	sand	grass
I300	<sos>	two	woman	UNK	walking	their	UNK	along	the	beach
RS	<sos>	two	women	carrying	surfboards	.	<eos>	<pad>	<pad>	<pad>
I20	wide	many	pedestrians	market	parade	.	<eos>	pier	pier	.
I50	<sos>	two	ladies	carrying	UNK	.	<eos>	.	.	.
I300	<sos>	two	women	carrying	UNK	.	<eos>	<pad>	<pad>	<pad>

Table 6.24: Inferred sentences from model with a embedding dimension of 20, 50 and 300. Each row contains a real sentence (RS) and the output of the model with an embedding dimension of 20 (I20), 50 (I50) and 300 (I300) using the real sentences as input.

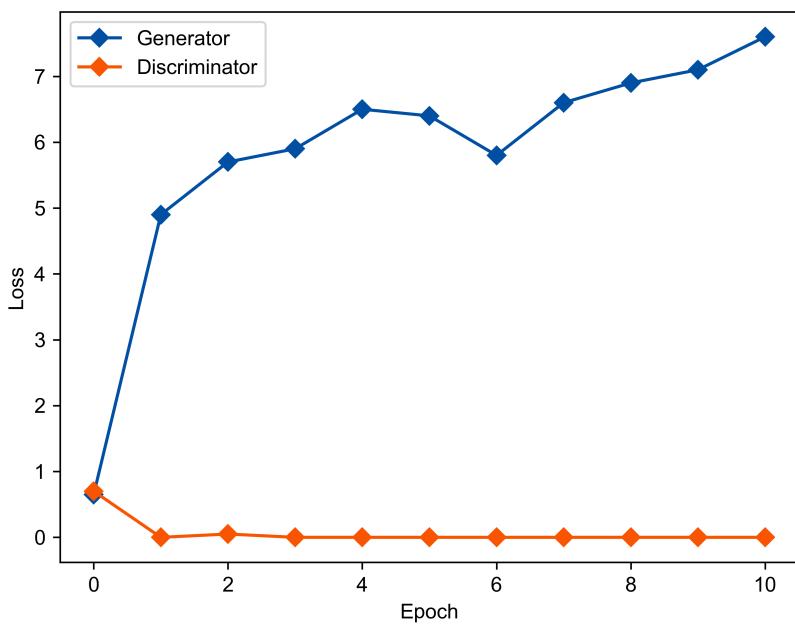


Figure 6.17: The loss functions of pre-trained generator and discriminator. The sequence-to-sequence decoder is used to initialize the weights of the generator in the GAN text generator model, which is described in Section 6.3.

CHAPTER 7

Conclusion

This chapter evaluates and discusses the results and observations made in Chapter 6. It covers both evaluation of architecture-related topics and choices made before, during and after the experiments. The chapter provide insight into the thoughts behind the evaluation method used and contains a discussion related to how our model compares to the sequence generative adversarial nets (SeqGAN) baseline model. The chapter covers topics related to training, performance and architectural advantages and disadvantages. The controllable text generation extension to our model is also discussed in detail. Finally, in the future works section, we will discuss potential improvements and further research that can be applied to our models.

7.1 Evaluation

In this section, we evaluate a variety of concepts related to our models, our evaluation metric and the results obtained through the experimental phase.

7.1.1 Discrete Values in Adversarial Training

In the text generation experiments we show two models that can be trained by representing words as either one-hot vectors or word embeddings. While both word representations gave some results, the word embedding approach generally outperformed the softmax distribution model. One major drawback of the softmax distribution model, using one-hot representation of words, is caused by artificial neural networks being designed to work with continuous values. In a standard supervised learning problem, the trained network would generate vectors with values close to a one-hot vector, but would never lead to an output of an actual one-hot vector. This is not a problem in a normal supervised learning case, where you can translate from a one-hot vector to a word from the vocabulary during inference. However, in our generator-discriminator setup this is a major issue. Intuition would suggest

that when the generator tries to trick the discriminator with vectors of floating point numbers drawn from the softmax distribution, the discriminator knows that real data only contains integers of 1s and 0s. Thus, allowing the discriminator to easily distinguish between real and generated data.

A natural solution to this problem would be to convert the sequence of vectors outputted by the generator into a sequence of one-hot vectors. This would force the discriminator to be trained only on one-hot vectors. However, the gradient used to update the weights in the generator is based on the error of the output from the complete generator-discriminator network. Translating the probability distribution to a one-hot vector after it has passed through the generator would make the loss function non-differential, which in turn stops the gradient from being calculated.

While converting all vectors into one-hot vectors would stop the network's ability to update the weights of the network correctly, there exists another approach. Instead of converting generated sentences, we transform the training examples from one-hot vectors, into vectors of real numbers that sum to 1, $v_i \in \mathbb{R} | 0 < v_i < 1$. This forces the discriminator to look at the actual values and ordering of vectors, not only if there consists a single 1 or not in the vector. Introducing this transformation does not work as expected. Instead of improving performance, it produces worse results, see Figure 6.5. A reason for this could be that instead of deceiving the discriminator, this distorts the training data, preventing the model from learning.

Implementing dropout in the discriminator does, however, improve the training of the models. The model that is only extended with dropout is the one that produces the most human-like sentences. This also requires by far the longest training time. The prevalent issue with the discriminator still exists. It has the potential to learn to separate real data from generated data by looking at differences values, but not necessarily registering the difference in sentences content. We believe this can be the reason why models based on the softmax distribution stops learning after some epochs of training.

7.1.2 Hyperparameters

The experiments show that implementing dropout in the discriminator is essential for training. Table 6.8 indicates that the dropout ratio matters when it comes to performance, and a lower dropout ratio is a better choice when training. Our results show that the lowest dropout ratio tested, achieves the best results. It is therefore interesting to research if an even lower dropout ratio would increase performance further.

A similar observation is done for model complexity, and the number of the long short-term memory (LSTM) hidden units. Table 6.14 displays a correlation between the variety of sentences generated and the number of hidden units. Further experiments could be done by training a larger model to examine whether the diversity of sentences would increase additionally.

Training our models takes a significant amount of time, which is why these subjects are not researched further. Depending on the amount of data and the model complexity, our models require days of training to produce reasonable sentences. However, we cannot exclude that additional training time beyond this point could

still provide better results. It is our impression that the word embedding model trained on Flickr30k could provide meaningful sentences, if it is given sufficient time to train. As our thesis concerns the concept of generating text, rather than optimizing training, we have disregarded rigorous testing of hyperparameters.

7.1.3 Evaluation Method

In the process of creating text generation models using generative adversarial network (GAN), we developed an approach to compare our models with each other, as well as to the baseline model. One of the main problems with evaluating results produced by GANs, is that there is no corresponding real data for each generated sample. Our evaluation method is able to indicate whether text generative models are producing meaningful text or not. The combination of three reference sentence retrieval methods leads to diversity in the evaluation method and, subsequently, improves the results. In some cases the evaluation method fails to provide an appropriate score. We base our evaluation method on bilingual evaluation understudy (BLEU), which is normally used on machine translation tasks. As we have limited experience with BLEU, we configured the algorithm with the same standard values in every evaluation case, causing the method to occasionally grade models as either too good or too poor. One specific example was the calculation of the β score on the model prediction captions from the Oxford-102 flower dataset, using word embeddings from a vocabulary of 400 000 words. The β score received was quite high, see Figure 6.10, but it was clear by observation, see Table 6.15, that the generated sentences had poor syntactic structure and semantic meaning.

One of the major benefits of our evaluation metric is that it works well with differently-sized datasets, even when the dataset contains a large or small number of similar sentences. If the goal for evaluation is to measure how close the generated sentences are to the original data, the evaluation method is suitable for text generation. The method rewards models that creates sentences with similar semantic content as sentences in the dataset. This could, in some cases, result in giving a high score to sentences without any meaning. Another issue with our evaluation method is that our method does not compensate for diversity in the generated sentences. Hence, the models that produce one, and only one, perfect sentence, will be rewarded with a perfect score. If the goal is to evaluate whether or not the model is able to produce syntactically correct sentences, our proposed evaluation method is not necessarily unsuitable. On the other hand, this is not the purpose for proposing this evaluation method. The main goal of the text generation experiments is to extend an existing dataset with plausible new sentences. Our proposed evaluation method manages to automatically evaluate if models are able to do so convincingly.

7.1.4 Pre-Training: Sequence-to-Sequence

Further work on initializing the GAN models using the sequence-to-sequence decoder will not be prioritized, but some assumptions can still be drawn from the results. Our belief is that the space from which the latent variables are drawn

from, lies in a different area than the latent variables produced by the encoder in the sequence-to-sequence model. This mismatch between distributions limits the generators ability to utilize any of the learned language modeling stored in the weights of the decoder. This results in the generator only being able to produce unintelligible sentences.

The dimensionality of the space that words are mapped to has a big impact on the accuracy of the models, as seen in Table 6.23. This is apparent in the retrieval process of the predicted embeddings. While the correct word is in the top-K results, a dimensionality that is too low could lead to consistently retrieving the incorrect words, while still showing high accuracy. If the generator is able to utilize the knowledge learned by the sequence-to-sequence model, the model is able to start training with a basic structure, in which it can produce sentences from. This could decrease the amount of time the model uses to start producing coherent words in the beginning of training.

7.2 Discussion

In this section, we will compare the results from the word embedding model, based on the continuous approach, with results from the baseline model, SeqGAN. The baseline model is regarded as one of the current state-of-the-art models for generating text using GANs. It is therefore interesting to look at differences and similarities of our proposed model and the baseline. In addition, we will discuss the results from our image captioning model, as well as reviewing what limitations and advantages are prevalent for the proposed model.

7.2.1 Text Generation

An advantage of using the continuous approach is that it is not dependent on any pre-training of either the generator, nor the discriminator. Other GAN-based models like the SeqGAN approach, requires pre-training to produce reasonable results. The SeqGAN model pre-trains the generator with supervised learning, and uses the generator to pre-train the discriminator. Our continuous approach enables the training of models using only standard techniques, introduced by Goodfellow et al. [2014]. This does not exclude pre-training as a valid early step in the continuous approach. Although pre-training is not required, it is possible that it could both improve the performance and training time.

7.2.2 Word Representation

Our approach is entirely dependent on representing words as dense vectors. In contrast, SeqGAN is not dependent on the pre-processing of data in this way, and is able to use the data as discrete tokens. Hence, our approach depends on having accurate word embeddings, which represents the correlation between different words in a reasonable fashion. Without this, it would be problematic for the model to create meaningful sentences. This correlation is specifically required during the

generation of sentences. In the generating process, it is desirable that a change in input would cause a slight change in the output, e.g. changing from describing a yellow flower to describing a red flower. This is achieved easier when the words “red” and “yellow” are closer to each other in the high-dimensional word embedding space.

7.2.3 Vocabulary Size

Table 6.10 indicates that our model is sensitive to the density in the high-dimensional word embedding space. The poor results achieved when applying the pre-trained global vectors (GloVe) compared to the custom trained Word2Vec embeddings, suggest our model is unable to learn under such circumstances. We have two theories why the usages of GloVe with a large vocabulary size affects the results. One assumption is that the generator has too many directions to explore in order to produce coherent sentences, and that a small adjustment of the weights would lead to a large change in the words generated. I.e., the large size of the vocabulary creates a “crowded” embedding space, which in turn causes an embedding to change from representing one word, to representing another with a slight change. The second theory to why using the custom word embeddings performs better, is that, in this case, every word in the vector space is relevant. In the GloVe embedding space, it is expected that words describing colors and specific flower-related words such as “stamen” and “petals” are grouped together. The distance between these group can, however, be quite large in the GloVe space. In our custom embeddings there are always relevant words in proximity. Training with the GloVe embedding space would therefore require a larger placement change in the high-dimensional space, forcing the generator to look at irrelevant words while searching for relevant ones.

7.2.4 Intermediary Feedback

Another major difference in our model compared to the SeqGAN baseline, is that our approach requires more time to train. The baseline model’s training time to produce meaningful sentences, including pre-training, is about a one-fourth or less compared to our model. A probable reason for this major difference could be the way feedback is given to the different generators. In our model the generator receives feedback for complete sentences, while the generator in the baseline receives information through the REINFORCE algorithm. SeqGAN is able to provide intermediary feedback to the generator for each timestep of the generated sequence. In contrast, outputting the entire sequence from the generator to the discriminator in one step, without any intermediary feedback, comes with a non-trivial cost. The generator is forced to adjust the network with regards to the cost calculated based on the entire sequence. This makes it difficult for the generator to find sequences that can fool the discriminator, especially at the start of the training process. In this exploring phase, the generator makes large changes to the words produced, and if there is a large space of possibilities to explore, the generator takes a significant amount of time to produce something reasonable. This period of trial-and-error for

the generator could be reduced if sufficient feedback is provided. The SeqGAN pre-training allows the model to skip parts of this time-consuming exploration process. Supervised training allows the model to learn a correct structure for the generated sentences. However, it is difficult to measure the impact of such a pre-training step, if it is to be used in our training process.

However, extending our model to consider each word in the sentence could increase the quality of generated sentences significantly. This could allow our model to consider the short-term versus the long-term goals for each word in the sentence. This could possibly allow the model to choose words based on the accumulated minimization of loss, rather than the loss calculated by looking only at the finished sequence. Such a method is explored further in future works, see Section 7.5.5. Another issue with only giving feedback on a complete sentence, is that the difficulty of training increases dramatically if the length of the sentences generated is increased. This is demonstrated in the experiments, since the generated text progressively improve towards the end of the sentences. An example of this issue is observed in Table 6.18. The sentences generated by our model lacks correct structure toward the end of the sentence. The beginning of the sentences, however, contain correctly structured words.

The baseline model shows a high β score when trained on both the Flickr30k and the Oxford-102 flower dataset. As seen in Table 6.17, the baseline model outperforms our model on both datasets. However, our model shows a higher variety in the sentences that are generated. This means that among N generated sentences, from both the baseline and our model, SeqGAN produces less distinct sentences. Our model produces a high variety of sentences, but with low β score. In general, models that produce a large amount of distinct sentences produce mostly incorrect sentences. When the models approach the point of being able to produce meaningful sentences, the amount of unique sentences is drastically reduced. As a result, we conclude that by having a higher number of distinct sentences, this reflects that our model is still trying new sentences. In other words, models with a high β score generates good, but few unique sentences.

7.2.5 Image captioning

The image captioning model is an extension of the word embedding model that uses images to control text generation. A proof of concept of this is shown in Section 6.5. Our image captioning model provides a framework for further experimentation with using GAN for image captioning. Compared to more traditional image captioning models, our model is able to utilize the benefits of GAN, as described in Section 2.6.2. This includes the ability to generate realistic and sharp samples, rather than samples that blur out in order to generalize [Goodfellow et al., 2014]. The controllable text generation experiment in Section 6.5.2 demonstrate that our architecture allows control over the generated sentences. The experiment shows that changing the control vector clearly affects the sentences generated.

This allowed us to further investigate the controllable properties of our architecture, and start experimenting with image captioning. The last experiment using this architecture shows that it could be developed further to work as an image cap-

tioning model. We show that when the model is trained on multiple images, it is capable of using the image representations to generate meaningful image descriptions. We also show that our model is able to generalize. Given unseen images, it is able to recognize the similarity of other seen examples and create sentences suitable to these new images. We believe that these results show that GANs can be used to do both controllable text generation and image captioning.

The model suffers from the same drawbacks as the word embedding model, as discussed above. It requires a significant amount of training to produce both meaningful and varied sentences. As a result of time constraints, rigorous testing of the image captioning model was not conducted. However, experiments on the text generation models have shown that sufficient amount of training can lead to an increase in sentence quality. One of the main advantages of the image captioning model is that it only requires a straightforward extension of the word embedding model. Additionally, replacing the dense representation of the images allows the control of text with any feature. This makes our architecture not limited to solving tasks related to images. It enables the possibility to control the generation of any sequence of discrete tokens..

7.3 Contributions

Our experiments show that the word embedding approach is able to produce grammatically correct and meaningful text. In addition, we are able to do so by using the standard method of training [Goodfellow et al., 2014], only using sequences of word embeddings, rather than images. To our knowledge, this is the first successful attempt at using word embeddings with GAN to produce meaningful text. Despite the drawbacks previously discussed, we believe that our proposed model can be used as a foundation for further research in the field.

We provide a proof of concept image captioning model capable of correlating generated text with given images. While thorough experimenting is not performed in this work, the model shows promising results. To our knowledge, this is the first attempt at image captioning using GAN. Our provided framework provides a basis for further research and experiments in the domain of image captioning, as well as other domains.

We propose a novel evaluation method for text generative systems, which combines the machine learning evaluation metric, BLEU, with a set of interchangeable information retrieval techniques. This allows automatic evaluation of our models, in addition to the observable human evaluation. The capability of this method is demonstrated with a comparison between our word embedding model and the SeqGAN baseline.

7.4 Answers to Research Questions

The following elaborates on how the research questions presented in Section 1.2 have been answered:

Research question 1 Is there an effective method for dealing with the discrete and sequential structure of natural language when working with GANs?

Both the softmax distribution model and word embedding model are able to produce a high ratio of meaningful sentences when trained on the Oxford-102 flower¹ dataset. While this dataset is limited in its variety, it serves to show that our proposed continuous approach has the potential to produce natural language. Using the Flickr30k dataset [Young et al., 2014], we show that our text generation models are able to do so with varying results. However, the percentage of meaningful sentences generated by the softmax distribution model is quite low. The word embedding model is able to produce sentences with promising structure, and usually starts sentences with words that “make sense” together. However, it rarely creates complete meaningful sentences.

Research question 2 Is the generation of natural language using GANs able to be controlled using images?

Experiments using our image captioning model show that our architecture is clearly able to correlate generated sentences with given images. This approach opens up for a straightforward way of controlling text generation using images, or any other vector representing relevant information. However, the provided results are produced in a setting where several parameters are limited. Hence, the results only serve as a proof of concept and further testing is required. Experiments presented are not sufficient evidence to determine whether the proposed image generation model is able to generate captions that are meaningful and suitable to a provided image. To conclusively determine whether this is possible using our proposed model, further experimentation is required. However, the results provided in this thesis is very promising.

7.5 Future Work

In this section we will provide insight into future work that could be interesting when using our proposed models as a framework for further research. The section covers improvements to the current models regarding the architecture, and possible solutions to different limitations of our models. The section also elaborates on how our proposed models can be used as a basis for further development and research in the field of using GANs in the sequential or discrete domain.

7.5.1 Pre-training

A possible extension to our model is to introduce pre-training to the discriminator. This could be accomplished in a supervised manner, using real data as positive samples and an augmented version of the data as negative samples. This is similar to what is done by Zhang et al. [2016]. The creation of negative examples

¹<http://www.robots.ox.ac.uk/~vgg/data/flowers/102/>

could include re-arranging words, adding noise or some other common approach of augmenting text. Alternatively, if we are able to pre-train the generator, we could pre-train the discriminator using the real data and the generated data in a supervised manner, similar to what is done by SeqGAN, described in Section 5.1.

As an alternative to our attempted generator pre-training approach using sequence-to-sequence, we could implement a maximum likelihood model, such as the one in Section 4.3.2. This could maximize the likelihood of each word given the preceding words. This is also implemented by the SeqGAN model. Such a model could then be able to produce reasonable sentences before being improved further using adversarial training.

7.5.2 Evaluation metric

Our evaluation method does not consider grammatical or syntactical features. Rule-based grammar and syntax analyzer functions are not hard to find, and could be combined to serve as an additional evaluation metric if such is a priority.

7.5.3 Recurrent Neural Network or Convolutional Neural Network

In literature, there are varied results on using recurrent neural networks (RNNs) or convolutional neural networks (CNNs) as the discriminator [Yu et al., 2016; Yang et al., 2017]. To conduct experiments comparing the results of using both of these methods, is a possible future extension to our work. SeqGAN implements a CNN in their discriminator and manages to perform training reasonably well. An intriguing point is made by Yang et al. [2017], who argues that the use of RNN-based discriminators performs much worse than CNN-based networks.

7.5.4 Wasserstein generative adversarial network

Research surrounding GANs has led to the development of several techniques, some of which are elaborated upon in Section 3.3. For further development of our proposed architecture, techniques counteracting mode collapse could help the stability of the training process. Specifically, the results by Arjovsky et al. [2017], who propose the Wasserstein generative adversarial network (WGAN) model, show improved stability, and has no signs of mode collapse.

7.5.5 Multiple Discriminators

To allow our models to provide feedback, not only on finished sequences, but also at intermediary steps, it is possible to extend them to incorporate multiple discriminators, looking at different N-grams of word embeddings. This would allow one discriminator to evaluate the entire sentence, while other discriminators looks at subsequences of a sentence, for example 2-gram or 3-gram. The N-gram discriminators would be able to evaluate whether words fit together locally, which in turn could stabilize the start of the training process. If the generator is provided

with intermediary feedback, it might allow it to make smaller gradient updates in the beginning of the training. Giving the generator the opportunity to catch patterns like “`<sos>` a” and “`<eos> <pad>`”, could provide important feedback to our original model. The discriminators would be trained together with one generator and provide updates to the network interchangeably. The improved stability might decrease training time for our current models, as the generator would search for possible sentences in a more intelligent matter. In combination with an earlier proposed future work, see Section 7.5.3, an approach to applying multiple discriminators is to use both an LSTM and CNN. There is a possibility that these two networks would be good at recognizing different aspects of a sentence, thus providing valuable feedback to the generator.

7.5.6 Control vector

In our experiments, the control vector in the image captioning architecture has only been used to represent a specific image. This vector could be exchanged with a variety of information, making it possible to control the generation of text using dense embedding representations or classification attributes. Examples include gender, semantics, age, sentence length and formality of language. Another use case for our proposed architecture is the ability to generate data from any discrete or sequential data distribution, such as speech or music. We believe that our model could be used as a basis for further research of GANs in several domains.

Bibliography

- Arjovsky, M., Chintala, S., and Bottou, L. (2017). Wasserstein GAN. *arXiv*.
- Baccouche, M., Mamalet, F., and Wolf, C. (2011). Sequential Deep Learning for Human Action Recognition. In *Proceedings of the 2nd International Conference on Human Behavior Understanding*, pages 29–39.
- Bahdanau, D., Brakel, P., Xu, K., Goyal, A., Lowe, R., Pineau, J., Memisevic, A., and Bengio, Y. (2016). An Actor-Critic Algorithm for Structured Prediction. *Arxiv*.
- Baum, L. E. and Petrie, T. (1966). Statistical inference for probabilistic functions of finite state markov chains. *The annals of mathematical statistics*, 37(6):1554–1563.
- Bengio, S., Vinyals, O., Jaitly, N., and Shazeer, N. (2015). Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks. *arXiv*, pages 1–9.
- Bengio, Y. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166.
- Che, T., Li, Y., Jacob, A. P., Bengio, Y., and Li, W. (2017). Mode Regularized Generative Advarsarial Networks. *International Conference on Learning Representations*, (2014):1–16.
- Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 1724–1734.
- Collobert, R. and Weston, J. (2008). A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25 th International Conference on Machine Learning*.

- Conneau, A., Schwenk, H., Barrault, L., and Lecun, Y. (2017). Very Deep Convolutional Networks for Text Classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics*, volume 1, pages 1107–1116.
- Coulom, R. (2006). Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. *International Conference on Computers and Games*, pages 72–83.
- Dong, J., Li, X., and Snoek, C. G. M. (2016). Word2VisualVec: Cross-Media Retrieval by Visual Feature Prediction. *arXiv*.
- Dosovitskiy, A., Springenberg, J. T., and Brox, T. (2015). Learning to generate chairs with convolutional neural networks. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 07-12-June, pages 1538–1546.
- Farhadi, A., Hejrati, M., Sadeghi, M. A., Young, P., Rashtchian, C., Hockenmaier, J., and Forsyth, D. (2010). Every picture tells a story: Generating sentences from images. *European Conference on Computer Vision*, 6314 LNCS(PART 4):15–29.
- Frome, A., Corrado, G., and Shlens, J. (2013). Devise: A deep visual-semantic embedding model. *Neural Information Processing Systems*, pages 1–11.
- G.E. Hinton and Salakhutdinov, R. (2006). Reducing the Dimensionality of Data with Neural Networks. *Science*, 313(June):1504–1508.
- Glorot, X., Bordes, A., and Bengio, Y. (2011). Domain Adaptation for Large-Scale Sentiment Classification: A Deep Learning Approach. In *Proceedings of the 28th International Conference on Machine Learning*, number 1, pages 513–520.
- Goodfellow, I. (2016). NIPS 2016 Tutorial: Generative Adversarial Networks. *Neural Information Processing Systems Tutorial*.
- Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.
- Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative Adversarial Networks. *Neural Information Processing Systems*.
- Graves, A. and Jaitly, N. (2014). Towards End-to-End Speech Recognition with Recurrent Neural Networks. In *Proceedings of the 31st International Conference on Machine Learning*, volume 31.
- Hand, D. J. and Yu, K. (2001). Idiot’s bayes—not so stupid after all? *International statistical review*, 69(3):385–398.
- Hjelm, R. D., Jacob, A. P., Che, T., Cho, K., and Bengio, Y. (2017). Boundary-Seeking Generative Adversarial Networks. *arXiv*.
- Hochreiter, S. (1997). Long Short-Term Memory. *Neural Computation*.

- Hotelling, H. (1933). Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, 24(6):417.
- Hu, Z., Yang, Z., Liang, X., Salakhutdinov, R., and Xing, E. P. (2017). Controllable Text Generation. *arXiv*.
- Ioffe, S. and Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of The 32nd International Conference on Machine Learning*.
- Isola, P., Zhu, J.-Y., Zhou, T., and Efros, A. A. (2016). Image-to-Image Translation with Conditional Adversarial Networks. *arXiv*, page 16.
- Jang, E., Gu, S., and Poole, B. (2016). Categorical Reparameterization with Gumbel-Softmax. *International Conference on Learning Representations*, (2014):1–13.
- Johnson, R. and Zhang, T. (2015). Effective Use of Word Order for Text Categorization with Convolutional Neural Networks. *The North American Chapter of the Association for Computational Linguistics*, (2011):103–112.
- Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. *International Conference on Learning Representations*, pages 1–15.
- Kingma, D. P. and Welling, M. (2013). Auto-Encoding Variational Bayes. *International Conference on Learning Representations*.
- Kulkarni, G., Premraj, V., Dhar, S., Li, S., Choi, Y., Berg, A. C., and Berg, T. L. (2013). Baby Talk : Understanding and Generating Simple Image Descriptions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(12):2891—2903.
- Kusner, M. J., Sun, Y., Kolkin, N. I., and Weinberger, K. Q. (2015). From Word Embeddings To Document Distances. In *Proceedings of The 32nd International Conference on Machine Learning*, volume 37, pages 957–966.
- Kuznetsova, P., Ordonez, V., Berg, A. C., Berg, T. L., and Choi, Y. (2012). Collective Generation of Natural Image Descriptions. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics*, volume 1, pages 359—368.
- Le Cun, Y., Jackel, L., Boser, B., Denker, J., Graf, H., Guyon, I., Henderson, D., Howard, R., and Hubbard, W. (1989). Handwritten digit recognition: applications of neural network chips and automatic learning. *IEEE Communications Magazine*, 27(11):41–46.
- LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, volume 86, pages 2278–2323.

- Li, B. and Han, L. (2013). Distance Weighted Cosine Similarity Measure for Text Classification. *Lecture Notes in Computer Science*, 8206:611–618.
- Li, J., Monroe, W., Shi, T., Ritter, A., and Jurafsky, D. (2017). Adversarial Learning for Neural Dialogue Generation. *arXiv*.
- Li, S., Kulkarni, G., Berg, T. L., Berg, A. C., and Choi, Y. (2011). Composing Simple Image Descriptions using Web-scale N-grams. In *Proceedings of the Fifteenth Conference on Computational Natural Language Learning*, pages 220–228.
- Liang, X., Hu, Z., Zhang, H., Gan, C., and Xing, E. P. (2017). Recurrent Topic-Transition GAN for Visual Paragraph Generation. *arXiv*.
- Lotter, W., Kreiman, G., and Cox, D. (2016). Unsupervised Learning of Visual Structure using Predictive Generative Networks. *Under review of International Conference on Learning Representations*.
- Lynch, C., Aryafar, K., and Attenberg, J. (2015). Images Don’t Lie: Transferring Deep Visual Semantic Features to Large-Scale Multimodal Learning to Rank. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1–9.
- McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133.
- Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2000). Distributed Representations of Words and Phrases and their Compositionality. *Neural Information Processing Systems*, 1:3111–3119.
- Mikolov, T., Karafiat, M., Burget, L., and Khudanpur, S. (2010). Recurrent neural network based language model. *Interspeech*.
- Mitchell, M., Dodge, J., Goyal, A., Yamaguchi, K., Stratos, K., Mensch, A., Berg, A., Han, X., Berg, T., and Health, O. (2012). Midge: Generating Image Descriptions From Computer Vision Detections. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 747–756.
- Nash, J. F. (1950). Equilibrium points in n-person games. *Proceedings of the national academy of sciences*, 36(1):48–49.
- Norouzi, M., Mikolov, T., Bengio, S., Singer, Y., and Mar, L. G. (2015). Zero-Shot Learning by Convex Combination of Semantic Embeddings. *British Machine Vision Conference*, pages 1–9.
- Odena, A. (2016). Semi-Supervised Learning with Generative Adversarial Networks. *arXiv*.
- Olah, C. (2015). Lstm cell diagram. <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>. [Online; accessed 19-April-2017].

- Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002). BLEU: a Method for Automatic Evaluation of Machine Translation. *Computational Linguistics*, (July):311–318.
- Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.
- Pfau, D. and Vinyals, O. (2016). Connecting Generative Adversarial Networks and Actor-Critic Methods. *arXiv*.
- Radford, A., Metz, L., and Chintala, S. (2015). Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. *International Conference on Learning Representations*.
- Reed, S., Akata, Z., Yan, X., Logeswaran, L., Schiele, B., and Lee, H. (2016). Generative Adversarial Text to Image Synthesis. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 1060–1069.
- Rubner, Y., Tomasi, C., and Guibas, L. J. (2000). The Earth Mover’s Distance as a Metric for Image Retrieval. *International Journal of Computer Vision*, 40(2):99–121.
- Rumelhart, D. E., Hinton, G., and Salakhutdinov, R. (1985). Learning Internal Representations by Error Propagation. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*.
- Rumelhart, D. E., Hinton, G., and Williams, R. J. (1986). Learning Representations by Back-propagating Errors. *Nature*, 323(9).
- Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., and Chen, X. (2016). Improved Techniques for Training GANs. *Neural Information Processing Systems*, pages 1–10.
- Shore, J. E. and Johnson, R. W. (1980). Axiomatic Derivation of the Principle of Maximum Entropy and the Principle of Minimum Cross-Entropy. *IEEE Transactions on Information Theory*, 26(1).
- Simonyan, K. and Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. *International Conference on Learning Representations*, pages 1–14.
- Smolensky, P. (1986). Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. chapter Information Processing in Dynamical Systems: Foundations of Harmony Theory, pages 194–281. MIT Press, Cambridge, MA, USA.
- Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15:1929–1958.

- Sutskever, I., Martens, J., and Hinton, G. (2011). Generating Text with Recurrent Neural Networks. In *Proceedings of the 28th International Conference on Machine Learning*.
- Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to Sequence Learning with Neural Networks. *Neural Information Processing Systems*, pages 3104–3112.
- Sutton, R. S., McAllester, D., Singh, S., and Mansour, Y. (1999). Policy Gradient Methods for Reinforcement Learning with Function Approximation. *Neural Information Processing Systems*, 12:1057–1063.
- Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 07-12-June, pages 1–9.
- Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2016). Rethinking the Inception Architecture for Computer Vision. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2818–2826.
- Van Der Maaten, L., Hinton, G., and van der Maaten, G. H. (2008). Visualizing Data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605.
- Vinyals, O., Toshev, A., Bengio, S., and Erhan, D. (2015). Show and tell: A neural image caption generator. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 07-12-June, pages 3156–3164.
- Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Machine Learning*, pages 229–256.
- Xiang, S. and Li, H. (2017). On the effect of Batch Normalization and Weight Normalization in Generative Adversarial Networks. *arXiv*.
- Yang, Z., Chen, W., Wang, F., and Xu, B. (2017). Improving Neural Machine Translation with Conditional Sequence Generative Adversarial Nets. *arXiv*.
- Young, P., Lai, A., Hodosh, M., and Hockenmaier, J. (2014). From image descriptions to visual denotations: New similarity metrics for semantic inference over event descriptions. *Transactions of the Association for Computational Linguistics* 2.
- Yu, L., Zhang, W., Wang, J., and Yu, Y. (2016). SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient. In *Proceedings of The 31st AAAI Conference on Artificial Intelligence*.
- Zhang, Y., Gan, Z., and Carin, L. (2016). Generating Text via Adversarial Training. *Neural Information Processing Systems Workshop*, (Nips).

Appendices

APPENDIX A

Additional Results

0.25 dropout:

```
<sos> this flower has petals that are pink with flowery stigma <eos>
<sos> this flower has large yellow petals and bright yellow stamen <eos>
<sos> this flower has petals that are pink with pink lines <eos>
<sos> this flower has petals that are pink and ruffled together <eos>
<sos> this flower has petals that are white with yellow patches <eos>
<sos> this flower has pistil purple petals and a bell markings <eos>
<sos> this flower has long pink petals as its <pad> <pad><eos>
<sos> this flower has petals that are white with green stamen <eos>
<sos> this flower has petals that are pink with pink stamen <eos>
<sos> the veined pointed in alternately are looking small <eos><pad><pad>
```

0.50 dropout:

```
<sos> this flower has petals that are white and folded together <eos>
<sos> this flower has petals that are pink with white stamen <eos>
<sos> this flower has petals that are white and folded together <eos>
<sos> this flower has petals that are white and folded together <eos>
<sos> the petals on this flower are purple with red stamen <eos>
<sos> this flower has petals that are purple with red stamen <eos>
<sos> this flower has petals that are pink with white stamen <eos>
<sos> this flower has petals that are purple with red stamen <eos>
<sos> a velvet white flower with a purple together stamen <eos><pad>
<sos> this flower has petals that are pink with white stamen <eos>
```

0.75 dropout:

```
<sos> this flower has petals that are pink with red stamen <eos>
<sos> this flower has petals that are pink with red stamen <eos>
<sos> a purple petals with a <eos> number with are pistol <eos>
<sos> a purple petals with yellow stamen a a folded pedicel <eos>
<sos> a petals on purple rounded petals and a yellow steman <eos>
<sos> a red petals with yellow white pink yellow stamen <eos><pad>
<sos> this flower has petals that are pink with red stamen <eos>
<sos> this red petals with yellow white gray violet are pedicel <eos>
<sos> this flower on purple rounded petals and yellow anthers <eos><pad>
<sos> a flower on flower rounded petals and a yellow pedicel <eos>
```

Table A.1: Randomly chosen generated sentences for three different dropout ratios implemented by discriminators. Every sentences is generated after 16000 epochs of training. All models are trained using word embeddings.

Query sentence chosen from dataset					
<sos> this flower has petals that	are	yellow with	white	edges	<eos>
<sos> this flower has petals that	are	yellow with	white	edges	<eos>
<sos> this flower has petals that	are	pink with	yellow	edges	<eos>
<sos> this flower has petals that	are	yellow with	pink	edges	<eos>
<sos> this flower has petals that	are	white with	pink	edges	<eos>
<sos> this flower has petals that	are	pink with	white	edges	<eos>
Top-2 sentences retrieved using Cosine similarity					
<sos> this flower has petals that	are	yellow with	white	edges	<eos>
<sos> this flower had petals that	are	yellow with	white	edges	<eos>
<sos> the flower has petals that	are	yellow with	white	edges	<eos>
<sos> this flower has petals that	are	pink with	white	edges	<eos>
<sos> this flower has petals that	are	purple with	white	edges	<eos>
Top-2 sentences retrieved using term frequency-inverse document frequency (TF-IDF)					
<sos> the flower has petals that	are	yellow with	white	edges	<eos>
<sos> this flower had petals that	are	yellow with	white	edges	<eos>
<sos> the flower has petals that	are	yellow with	white	edges	<eos>
<sos> this flower has petals that	are	pink with	white	edges	<eos>
<sos> this flower has petals that	are	red with	yellow	edges	<eos>
Top-2 sentences retrieved using word mover's distance (WMD)					
<sos> the flower has petals that	are	yellow with	white	edges	<eos>
<sos> this flower had petals that	are	yellow with	white	edges	<eos>
<sos> this flower has petals that	are	yellow with	white	edges	<eos>
<sos> this flower has yellow petals that have	white edges	<eos>	<pad>		
<sos> this flower has petals that	are red	with yellow edges	<eos>		

Table A.2: Table showing the top-5 retrieved reference sentences from Oxford-102 Flower dataset for each of the three retrieval methods; cosine similarity, WMD and TF-IDF for a given query sentence chosen from the dataset.

Query sentence containing errors	<sos> this flower has petals that are yellow with white edges <eos>	<sos> this flower has purple and pink petals with stamen <eos>	<sos> this flower has white petals with green lines and stamen <eos>	<sos> this flower has white and purple petals with green stamen <eos>	<sos> this flower has blue petals with long and green stamen <eos>	<sos> this flower has white and red petals with green stamen <eos>
Top-2 sentences retrieved using Cosine similarity						
<sos> a flower with bright blue leaves and a blue stamen <eos>	<sos> this flower has light blue petals with darker blue veins <eos>	<sos> this flower has blue petals with long and green stamen <eos>	<sos> this flower has petals that are blue with green stamen <eos>	<sos> the flower shown has several blue petals with blue veins <eos>		
Top-2 sentences retrieved using TF-IDF						
<sos> petals that are blue with green stamen <eos>	<sos> the petals of this flower are pink with green stamen <eos>	<sos> this flower has petals that are green with pink stamen <eos>	<sos> this flower has petals that are pink with green stamen <eos>	<sos> the petals on this flower are purple with green stamen <eos>		
Top-2 sentences retrieved using WMD						
<sos> petals that are blue with green stamen <eos>	<sos> the petals of this flower are pink with green stamen <eos>	<sos> this flower has petals that are green with pink stamen <eos>	<sos> this flower has petals that are pink with green stamen <eos>	<sos> the petals on this flower are purple with green stamen <eos>		

Table A.3: Table showing the top-2 retrieved sentences from Oxford-102 Flower dataset for each of the three retrieval methods; cosine similarity, WMD and TF-IDF for a given query sentence containing errors.

Query sentence containing errors												
Top-2 sentences retrieved using Cosine similarity												
<sos> this flower has petals that are yellow with white edges <eos>												
<sos> two soccer players interact on the field <eos> <pad> <eos>												
<sos> soccer players stand on the field <eos> <pad> <eos>												
<sos> soccer players on the field <eos> <pad> <eos>												
<sos> two football players kick soccer balls on the athletic field <eos>												
Top-2 sentences retrieved using TF-IDF												
<sos> a man in swim trunks is juggling on the beach <eos> <pad> <eos>												
<sos> swimmers in the water with swim trunks jumping on beach <eos> <pad> <eos>												
<sos> boy in swim trunks jumping on beach <eos> <pad> <eos>												
<sos> two boys swim on a sunny day <eos> <pad> <eos>												
<sos> two man with swim shorts on beach <eos> <pad> <eos>												
Top-2 sentences retrieved using WMD												
<sos> two soccer players walk on the soccer field <eos> <pad> <eos>												
<sos> two soccer players playing soccer on a soccer field <eos> <pad> <eos>												
<sos> two soccer players on a soccer field <eos> <pad> <eos>												
<sos> two soccer teams walking on a soccer field <eos> <pad> <eos>												
<sos> the soccer players are walking on the field <eos> <pad> <eos>												

Table A.4: Table showing the top-2 retrieved sentences from Flickr30k dataset for each of the three retrieval methods; cosine similarity, WMD and TF-IDF for a given query sentence containing errors.

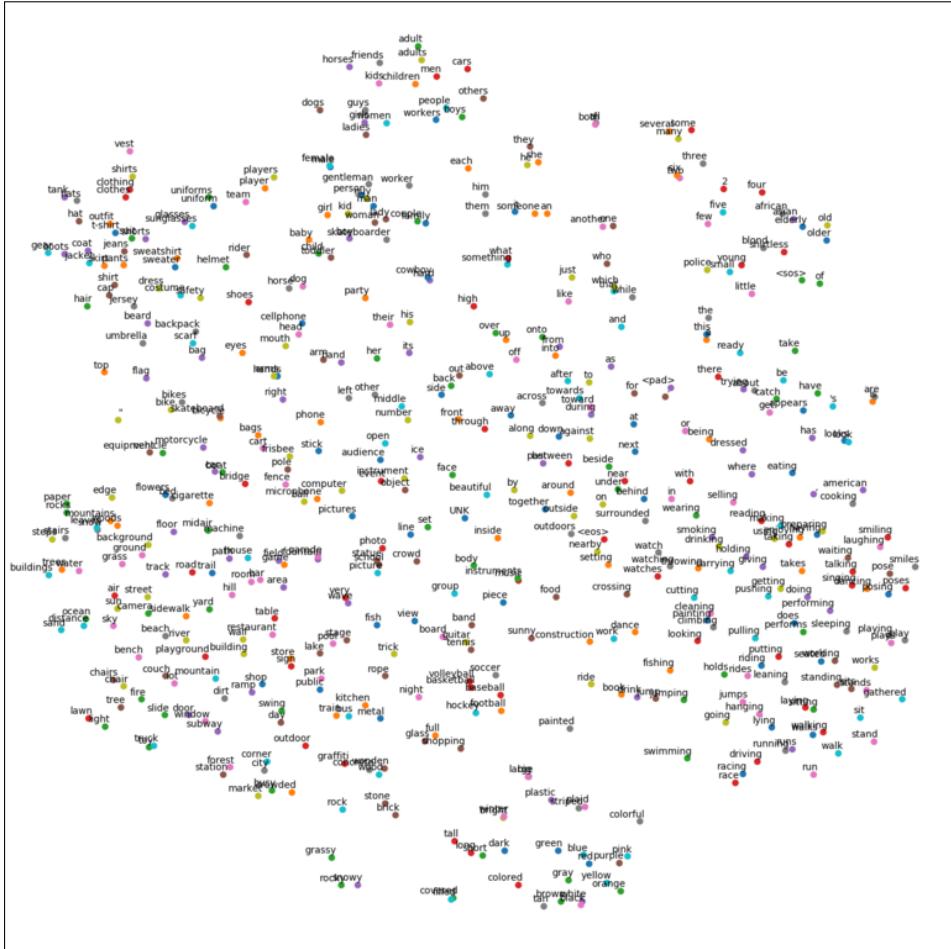


Figure A.1: t-distributed stochastic neighbor embedding (t-SNE) dimensionality reduction on Word2Vec embedding vectors trained on the Flickr30k dataset.