On Generating Mathematical Formulae

Luca Sacchetto

Master's thesis

On Generating Mathematical Formulae

Luca Sacchetto

23. July 2020

Institute for Data Processing
Technische Universität München

Luca Sacchetto. *On Generating Mathematical Formulae.* Master's thesis, Technische Universität München, Munich, Germany, 2020.

Supervised by Prof. Dr.-Ing. K. Diepold and Matthias Kissel, M.Sc.; submitted on 23. July 2020 to the Department of Electrical Engineering and Information Technology of the Technische Universität München.

Institute for Data Processing, Technische Universität München, 80290 München, Germany, `http://www.ldv.ei.tum.de/`.

Acknowledgements

Foremost, I would like to express my sincere gratitude to my advisor Matthias Kissel, M.Sc.: his guidance and his advice were fundamental to the completion of this thesis and his positive outlook was a much needed morale boost throughout these past six months.

I would also like to extend my earnest thanks to Prof.Dr.-Ing Klaus Diepold, who offered me the opportunity to embark on this fantastic journey and without whom this thesis would not have been possible.

Abstract

Generative Adversarial Networks (GANs) have demonstrated great success in image generation in recent years. This thesis proposes a novel application of GANs: the generation of mathematical formulae. In this context, a formula is described as a sequence of discrete symbols that represent a mathematical relation between quantities. These can include integers, operands and constants. Due to the well known "non-differentiability" issue of GANs in discrete data generation [9], the standard GAN architecture cannot be used. To that end, the architectures from two language generation models are modified and evaluated: the Gumbel-Softmax GAN [23] and the Sequence GAN [41]. Experimental and mathematical results demonstrate the unviability of using the Gumbel-Softmax GAN for the generation of mathematical formulae using the proposed approaches, by cause of the difference between its intended application and the one studied in this thesis. Finally, the experiments done on the Sequence GAN show highly promising results, which make this particular architecture the more suitable amongst the analysed ones.

Contents

Abstract **5**

1. Introduction **9**

2. Framework **13**
- 2.1. Generative Adversarial Networks 13
- 2.2. Convergence 15
  - 2.2.1. Mode Collapse 17
  - 2.2.2. Vanishing Gradients 18

3. Related Work **19**
- 3.1. Continuous Generative Adversarial Networks 19
- 3.2. Discrete Generative Adversarial Networks 20
  - 3.2.1. Reinforcement Learning based methods 20
  - 3.2.2. Direct methods 21
  - 3.2.3. Difference between Mathematical Formulae and Natural Language 22

4. Vanilla GAN **25**
- 4.1. Model 25
- 4.2. Results 26

5. Non-differentiability Issue **29**
- 5.1. Outline 29
- 5.2. Straight-Forward Softmax GAN 30

6. Gumbel-Softmax GAN **33**
- 6.1. Model 33
  - 6.1.1. Gumbel-Max-Trick 33
  - 6.1.2. Continuous Relaxation 35
  - 6.1.3. Structure 35
  - 6.1.4. Training Data 36
- 6.2. Results 37
  - 6.2.1. Constrained logits 37
  - 6.2.2. Unconstrained logits 42

7. Sequence GAN **47**
7.1. Reinforcement Learning 47
7.2. Model 47
7.3. Policy Gradient 48
7.4. Results 50

8. Discussion **53**

9. Conclusion **57**

List of Acronyms **59**

List of Symbols **61**

List of Tables **63**

List of Figures **65**

List of Algorithms **67**

Bibliography **72**

Appendices **73**

Appendix A. Straight-Forward Softmax GAN **75**

Appendix B. Gumbel-Softmax GAN with unconstraint logits **77**

1. Introduction

"The most interesting idea in the last ten years in machine learning"

- Yann LeCun, prominent deep learning researcher, director of AI research at Facebook, professor at NYU and co-recipient of the 2018 Turing award, on GANs [24].

Since its introduction by Goodfellow et al. in 2014 [10], GANs have gained widespread research attention and introduced the field of Machine Learning (ML) to new and exciting possibilities, such as image generation, image reconstruction and Natural Language Processing (NLP). This thesis introduces and analyses a novel application for GANs: the generation of mathematical formulae. From mathematics through quantum physics to computer science, formulae are at the core of our understanding of the world: the calculation of the area of a sphere, the knowledge that the ultimate answer is indeed 42, the description of the state of a particle in a quantum-mechanical system and even the simplest, but fundamental notion that 1 plus 1 equals 2 are all concepts precisely and unambiguously described by mathematical formulae. The existence of an algorithmic model capable of generating novel and complex mathematical formulae, given a collection of known ones as training data, could have unforeseeable advantageous effects on the most disparate research fields and its applications could be varied and manifold.

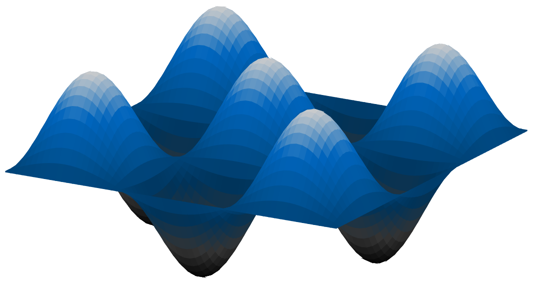


Figure 1.1.: Schrödinger equation: wave-function in an infinite two-dimensional quantum well of unit length for energy levels $n_x = 3$ and $n_y = 3$.

Consider the abundance of undiscovered formulae in all fields of research: among those is the computation of prime numbers, which always garners much attention among mathematics researchers. Although there exists a plethora of formulae for calculating upper and lower bounds of prime numbers and for producing candidates, an efficiently computable and precise mathematical formula for calculating all the prime numbers remains undiscovered. A model, like the one described above, could theoretically help with such search. Further examples include the unification of the concepts of time in quantum physics and its relativistic counterpart. In quantum physics, time is regarded as a fixed and absolute quantity, whereas in relativistic terms, time is dependent on quantities such as gravity and velocity. A formula that can unify these pradigms is yet to be discovered. Although such a model is undoubtedly far fetched and may be implausible or even impossible, such considerations provided nevertheless the motivations that lie behind this thesis. Indeed, this latter sets to lay the first stone of the foundation upon which such model could be built.

Because this thesis sets to analyse the viability of using GANs to generate mathematical formulae and aims to function as a proof-of-concept, the objective of the models is reduced to its simplest possible terms, while still being true to the aim of the thesis. Conceptually, the objective of the systems is to generate novel equations of the following form:

$$a + b = c\,; \quad a, b, c \in \mathbb{Z}$$

Within the scope of this thesis, the mathematical operators $+$ and $=$ will be kept out of the objective of the generative algorithm, so that the formulae can be represented by vectors of length three:

$$x = \left[\ x_1, \quad x_2, \quad x_3\ \right], \quad where \quad x_3 = x_2 + x_1$$

In practice, training data of such form would still require the definition of a large set of possible symbols, i.e. a large vocabulary, as every integer would have to be assigned to an unique characterization. Consequently, to minimize the set of available symbols, the data is restructured as a vector of length 8:

$$n = \left[\; n_1 \quad n_2 \quad n_3 \quad n_3 \quad n_4 \quad n_5 \quad n_6 \quad n_7 \quad n_8 \; \right],$$

$$where \quad n_i \in \{0,1\}\,; \quad \sum_{i=1}^{4} n_i = \sum_{i=5}^{8} n_i$$

The core concept of the objective remains unvaried, yet the vocabulary is thusly restricted to two symbols: one and zero. The reasoning behind the choice of this specific objective is twofold. Firstly, it represents the simplest form of mathematical formula; this reduces the number of variables to a minimum and allows a more clear analysis of the working principle of the models and of the problems that may arise. Secondly, a successful completion of this objective would nevertheless demonstrate the capability of the model to work with more complicated formulae. To that end, the choice to limit the objective to integer values only is fundamental. Namely, such limitation requires the development of a model that operates over a discrete domain, rather than over a continuous one. The discrete domain is what allows this thesis' work to be used as a first step towards the generation of mathematical formulae. Naturally, the generation of the above described vector x would not be outside the means of a continuous model. However, if one extends the objective to include mathematical operands:

$$x' = \left[\; x_1, \quad +, \quad x_2, \quad =, \quad x_3 \; \right]$$

One would have no way to define the tokens "$+$" and "$=$" over a continuous domain and would fail in the construction of the algorithm. However, defining such tokens over a discrete domain is straight forward: operators would be defined as e.g. a position in an one-hot vector [14]; not unlike integers. The notion that operands can be defined in the same way as integers is also why we can omit them from the scope of this thesis: just the integers suffice as proof-of-concept. In conclusion, this objective fulfills two crucial conditions: it is as simple as possible, while insuring that the results of this work are as significant as possible.

2. Framework

2.1. Generative Adversarial Networks

Developed by Ian Goodfellow et al. in 2014 [10], GANs are a game-theorical approach to the conception of deep generative models. The basic structure of a GAN consists of two deep neural networks: a generator and a discriminator. The generator maps a latent noise input to to the distribution of the training data. The discriminator is a binary classifier, whose inputs is both the output of the generator and samples from the training data. Its output is a scalar that represents the network's decision on weather the samples are real or come from the generator.

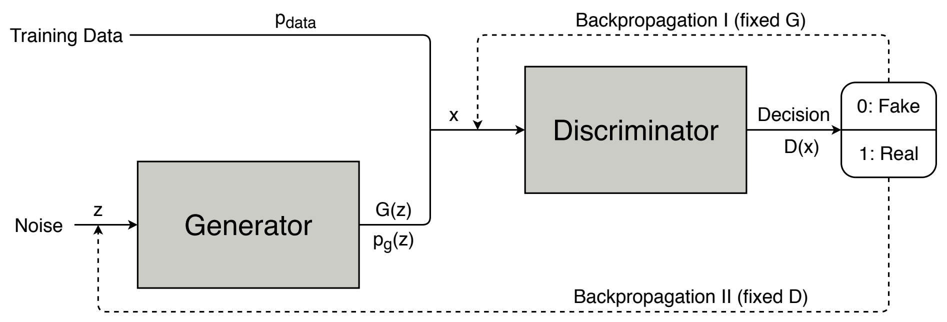


Figure 2.1.: Basic structure of a GAN.

Intuitively, this system can be understood as an adversarial game played by the two models, hence the game-theorical approach: the discriminator tries to catch the generator while the generator tries to fool the discriminator. The system converges when the players reach Nash-equilibrium [28]: the point where both players' strategies cannot be changed to increase their respective payoffs in relation to the game. Effectively, the generator is producing real-looking samples, the discriminator cannot recognize them anymore and both models have no incentive to change their strategies. Mathematically, the objective of the entire system can be expressed as the min-max optimization problem:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))] \quad (2.1)$$

where x is the real data, $p_{data}(x)$ is its distribution, z is the noise input of the generator and $p_z(z)$ is the distribution of the latent noise. Algorithm 1 illustrates the functionality of the GAN: during training, one must alternate between updating the generator and updating the discriminator. To that end, the error is firstly back-propagated through the discriminator, updating only this latter. Subsequently, the error is back-propagated through both the discriminator and the generator: in this case, only the generator is updated.

Algorithm 1 : GAN [10]

```
Generator $G$ , Discriminator $D$;
while not converged do
    for $n = 1$ to $N_D$ do
        Sample batch of $m$ noise samples: $\{z_1, ..., z_m\}$;
        Generate samples with $G$ : $\{G(z_1), ..., G(z_m)\}$;
        Sample batch of $m$ real samples from $p_{data}(x)$: $\{x_1, ..., x_m\}$;
        Update $D$ via gradient descent:
            $$-\nabla_{\theta_D} \frac{1}{m} \sum_{i=1}^{m} [\log D(x_i) + \log(1 - D(G(z_i)))] \quad (2.2)$$
    end
    for $n = 1$ to $N_G$ do
        Sample batch of $m$ noise samples: $\{z_1, ..., z_m\}$;
        Update $G$ via gradient descent:
            $$-\nabla_{\theta_G} \frac{1}{m} \sum_{i=1}^{m} [\log(D(G(z_i)))] \quad (2.3)$$
    end
end
```

2.2. Convergence

Goodfellow et al., in their original paper [10], show that there exists a mathematical description of the Nash Equilibrium in the proposed GAN; i.e. a convergence point. Namely, when the distribution generated by the generator p_g is equal to the distribution of the training data p_{data}:

Lemma 1. *The GAN is optimal when $p_g = p_{data}$*

Proof. [10]
Consider first the objective of the discriminator as per equation 2.1, i.e. to maximize the value function $V(D, G)$. Further, consider freezing the generator, so that equation 2.1 becomes:

$$\begin{aligned}\max_D V(D,G) &= \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1-D(g(z)))] \\ &= \int_x p_{data}(x)\log(D(x))dx + \mathbb{E}_{z\sim p_z(z)}[\log(1-D(g(z)))]\end{aligned} \tag{2.4}$$

Because with a frozen generator the quantity $g(z)$ is a fix input to the discriminator, it can be substituted for x:

$$\max_D V(D,G) = \int_x p_{data}(x)\log(D(x))dx + \mathbb{E}_{z\sim p_z(z)}[\log(1-D(x))] \tag{2.5}$$

Recall the Law of the unconscious statistician:

Theorem 1 (Law of the Unconscious Statistician [6])**.** *Let $g(X)$ be a function on the continuous random variable X. If the distribution of $g(X)$ is not known, then the expectation $\mathbb{E}[g(X)]$ can be computed as:*

$$\mathbb{E}[g(X)] = \int_x g(x)f_X(x)dx \tag{2.6}$$

where $f_X(x)$ is the probability density function of X.

Applying the "Law of the Unconscious Statistician" to equation 2.5 yields:

$$\max_D V(D,G) = \int_x p_{data}(x)\log(D(x))dx + p_g(x)\log(1-D(x))dx \tag{2.7}$$

Note that the function $f(x) = a\log(x) + b\log(1-x)$ *reaches its maximum at* $x^* = \frac{a}{a+b}$ *; so that the optimal discriminator is defined as:*

$$D^* = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)} \tag{2.8}$$

Secondly, consider the objective of the generator as per equation 2.1 with a frozen, optimal Discriminator:

$$\begin{aligned}\min_G V(D^*,G) &= \mathbb{E}_{x\sim p_{data}(x)}[\log D^*(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1-D^*(G(z)]\\ &= \int_x p_{data}(x)\log\frac{p_{data}(x)}{p_{data}(x)+p_g(x)} + p_g(x)\log\frac{pg_x}{p_{data}(x)+p_g(x)}dx\end{aligned} \tag{2.9}$$

Further, consider the Jensen-Shannon divergence between p_{data} *and* p_g*:*

$$\begin{aligned}D_{JS}(p_{data}\|p_g) =& \frac{1}{2}D_{KL}\left(p_{data}\middle\|\frac{p_{data}+p_g}{2}\right) + \frac{1}{2}D_{KL}\left(p_g\middle\|\frac{p_{data}+p_g}{2}\right)\\ =& \frac{1}{2}\left(\log 4 + \int_x p_{data}(x)\log\frac{p_{data}(x)}{p_{data}(x)+p_g(x)}\right.\\ &\left.+ p_g(x)\log\frac{p_g(x)}{p_{data}(x)+p_g(x)}dx\right)\\ =& \frac{1}{2}\left(\log 4 + \min_G(D^*,G)\right)\end{aligned} \tag{2.10}$$

which leads to:

$$\min_G V(D^*, G) = 2D_{JS}(p_{data} \| p_g) - \log 4 \tag{2.11}$$

Since, by definition, the Jensen-Shannon divergence can only be equal or greater than 0; and since it is only equal to zero if the distributions are equal to each other, the generator has its optimum at $p_g = p_{data}$. Plugging this equality into equation 2.7 yields: $D^* = \frac{1}{2}$. Intuitively, the discriminator cannot distinguish between real and generated samples anymore and outputs only $\frac{1}{2}$. The players have reached Nash equilibrium and the GAN has reached its optimum. $\square$

Although the Adversarial Network has a mathematical optimum, in practice, GAN training is not as straight-forward as the mathematical model may suggest: there exist several ways, in which the network fails to converge, most prominently mode collapse and vanishing gradients.

2.2.1. Mode Collapse

Mode collapse is widely regarded as one of the more prominent and challenging mode failures of GAN training. It results in a model which generates samples from an extremely limited subset of the original distribution p_{data}. If, for example, the network's objective is to generate faces, mode collapse would result in the generator only generating one single face or a collection of faces with e.g. the same skin tone. Specifically, the generator has learned to map the entire latent space to a limited number of points in the input space. Various causes have been linked to Mode Collapse in GANs; such as sharp discriminator's gradients [37] and, more commonly, excessive training of the generator. If the generator is updated too frequently, it may tend towards generating a single sample that fools the discriminator: its loss function assumes near-zero values and even a subsequent training of the discriminator does not solve the problem [8]. This tendency is also caused by the architecture of the model: because the discriminator processes samples independently from each other, it has no way of assessing the entropy of the generated samples [32]. Several solutions have been proposed to alleviate the problem of mode collapse; notable amongst them are: the Wasserstein GAN architecture [3] and minibatch discrimination [32]. Nevertheless, these solutions are not always applicable and the problem of mode collapse remains an open one.

2.2.2. Vanishing Gradients

Vanishing Gradients are generally caused by the discriminator converging too quickly. An optimal discriminator D^* outputs zeros and ones for the generated and the real samples, respectively:

$$D^*(G(x)) = 0; \quad D^*(x) = 1 \tag{2.12}$$

In this state, the gradients of the discriminator are zero almost everywhere. Consequently, as the discriminator converges towards its optimality for a given generator, its gradients tend towards zero and therefore may vanish when passed to the generator. Arjovsky and Bottou lay an analytical proof for the above statement in [2]:

$$\lim_{\|D-D^*\|\to 0} \nabla_\theta \mathbb{E}_{z\sim p(z)}[\log(1 - D(G_\theta(z)))] = 0 \tag{2.13}$$

where

$$\|D\| = \sup_{x\in X} |D(x)| + \|\nabla_x D(x)\|_2 \tag{2.14}$$

and $\log(1 - D(G(x)))$ is the objective function as per equation 2.1. They further show, that this problem can be alleviated by setting the generator's objective to $-\log(D(G(z)))$, which is reflected in algorithm 1.

3. Related Work

3.1. Continuous Generative Adversarial Networks

The original and most prominent use of GANs is image processing and image generation; the results of such implementations have advanced remarkably in recent years and can be strikingly indistinguishable from real photographs.

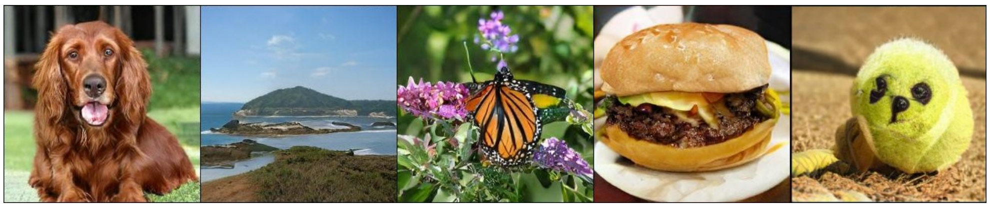


Figure 3.1.: Selection of pictures generated by a Large Scale GAN [4].

Figure 3.1 shows results from the implementation by Andrew Brock et. al. in [4], where the authors scaled up state-of-the-art GANs by drastically increasing batch size and number of parameters of the model.

Figure 3.2 shows a selection of pictures generated by the second iteration of Style-GAN [18] [19]. Herein, the authors propose a generator with an alternative structure: instead of feeding a latent noise vector to the input layer of the generator, the latent vector is a learned constant which is mapped by a Multilayer Perceptron (MLP) to a latent space. This is in turn passed to each layer of the generator through adaptive instance normalization (AdaIN). In addition, gaussian noise is added to each layer. The alternative generator's advantages comprise greater variation within the generated pictures and separation of high-level attributes such as hairs and freckles.

Other important developments in GAN research include the work published by Zakharov et al. in 2019 [42], wherein the authors' model is capable of generating video of a person giving a speech, by feeding the network with only a small number of head shots of the protagonist.

Furthermore, at NVIDIA, Kim et al. introduced Game-GAN in early 2020 [20]. The model is able to recreate the game of Pac-Man starting from just the recording of a match.

Figure 3.2.: Selection of pictures generated by Style-GAN II [18] [19].

3.2. Discrete Generative Adversarial Networks

Notwithstanding the progress that GAN based models have made on continuous domains such as images and videos, the state of the research in GANs on discrete domains is more scattered and has fewer impressive results. This stems partly from the fact that to generate discrete symbols, the generator must contain a hard-decision layer, which does not allow for standard back-propagation through the model. This issue is at the core of discrete GAN research: it is discussed in more detail in chapter 5 and will be referred as "non-differentiability issue" hereafter. Due to the structure of the objective of this thesis, as described in chapter 1, a model which generates sequences of discrete, highly correlated symbols is required. As far as the author of this thesis knows, there is no research on the generation of mathematical formulae. However, the research field which is most related to it is NLP, specifically using GANs to generate strings of original text. To this end, there exist a plethora of different variations on the basic GAN architecture, which try to overcome the non-differentiability issue discussed in chapter 5. However, as of the writing of this thesis, there does not seem to be a clearly dominant method. Nevertheless, these methods can be categorized into two types: the methods that use Reinforcement Learning and the ones that use a direct approach.

3.2.1. Reinforcement Learning based methods

This family of models incorporates concepts of Reinforcement Learning (RL) into a GAN. Specifically, the generator becomes a RL-Agent, which moves through the state-action space defined by the previously generated words and the selection of the next word, respectively. The reward is provided by the discriminator and the model is updated through a policy gradient algorithm such as REINFORCE [40].

The Sequence Generative Adversarial Network (Seq-GAN), developed by Yang et al. in 2017 [41] is one of the more prominent members of this family. Herein, the model is updated by performing a Monte-Carlo search on each word, with the reward given by the discriminator at each step. This method will be discussed in more detail in chapter 7.

In 2017 Guo et al. introduced Leak-GAN [13]: the proposed model addresses the scarcity of information provided by the discriminator, especially with longer sentences. Because the discriminator is a binary classifier, the score it provides is both sparse and lacks intermediate information: a sub-optimal condition for a RL model. To that end, the discriminator is structured to leak its internal state to the generator model, which follows a hierarchical RL [39] architecture to interpret such leaked state. This provides the generator with more meaningful and frequent information.

Model	Generated Sentences
Leak-GAN	A couple of people are riding bikes down an asphalt road. An old photo of a man riding on a motorcycle with some people. A person in a helmet standing next to a red street.
Seq-GAN	A red and white photo of a train station. The bathroom is clean and ready for us to use . A man is walking with his dog on the boardwalk by the beach.

Table 3.1.: Examples of sentences generated by Leak-GAN and Seq-GAN, trained from the COCO dataset, as shown in [13].

A solution to a similar problem was proposed by Lin et al. in 2017 with Rank-GAN [25]: The discriminator is substituted by a Ranker, which is provided a single generated sentence and a collection of human-written ones. The Ranker ranks them according to believability and consequently provides a more rich reward compared to a simple binary classifier.

Hjelm et al. use a slightly different approach and propose a Boundary Seeking-GAN [15], wherein the model performs policy gradient based on the KL-divergence and derived from the objective function of f-GANs [30].

3.2.2. Direct methods

Contrary to section 3.2.1, these models do not use RL based methods to overcome the non-differentiability issue of discrete GANs, instead relying on modifications to the original architecture to allow for the back-propagation through the discriminator-generator ensemble to happen.

Among the several improvements to the Wasserstein Generative Adversarial Network (WGAN) proposed by Gulrajani et al.[11], a discrete WGAN is outlined. The original WGAN proposed by Arjovski et al. [3] sets to alleviate some of the problems connected to GAN training. It uses a critic instead of a discriminator, which provides a score on the generated samples rather than a binary classification. Furthermore, it uses the novel Wasserstein distance as objective to the model, instead on relying on the more common Jensen-Shannon divergence, which is what the original GAN [10] is based on. WGAN garnered much attention and is widely regarded as a successful improvement upon the original GAN. Gulrajani et al.'s discrete WGAN makes use of the same structure and applies it to discrete data. The generator's softmax output is fed directly to the generator, overcoming the non-differentiability issue.

In 2016 Kusner et al. proposed the Gumbel-Softmax-GAN [23], based on the Gumbel-Max trick [12] and its softmax relaxation [26][17], which allows for differential sampling of a categorical distribution. This approach will be discussed in more detail in chapter 6.

The Relational GAN, proposed by Nie et al. [29], uses the same Gumbel-Softmax trick, expands however on the GAN architecture by substituting the more common LSTM [16] modules for relational-memory modules [33].

3.2.3. Difference between Mathematical Formulae and Natural Language

Although many similarities can be drawn between the research field of NLP and the objective of this thesis, inasmuch that both problems can be condensed to the generation of a sequence of discrete symbols, there exist one major difference, which can be summarized as follows:

Lemma 2. *Given a previously generated set of symbols in a sequence $S_{1:t-1} = \{s_1, \dots, s_{t-1}\}$, a symbol s_t to be generated and a set of available symbols $\phi = \{\phi_1, \dots, \phi_N\}$, the true distribution $p_{data}(\phi|S_{1:t-1})$ of a formulae generation problem is either very spiky or one-hot [14], whereas the true distribution of a NLP problem is considerably smoother.*

This concept is best illustrated by means of practical examples: firstly, consider a generative model G for language processing. Assume that , based on the previously generated symbols in that sentence $S_{1:t-1}$, such model generates a probability distribution $p_G(\phi|S_{1:t-1})$. The generated word can be subsequently computed as $s_t = \text{argmax}(p_G(\phi|S_{1:t-1}))$.

Secondly, consider the sentence:

Sentence 1. *The sea is blue*

Further consider the generation process of the fourth word "blue". In this instance we can define $S_{1:t-1} = \{\text{"the"}, \text{"sea"}, \text{"is"}\}$, the learned distribution of G $p_g(\phi|S_{1:t-1})$ and the true distribution $p_{data}(\phi|S_{1:t-1})$. For visualization purposes, the left diagram of figure 3.3 illustrates what the distribution p_{data} may look like. Note that the example distribution is relatively smooth: intuitively, although the word "calm" is indeed the most probable, it is not the only correct one, as sentences 2 and 3 are still plausible and correct

Sentence 2. *The sea is green.*

Sentence 3. *The sea is calm.*

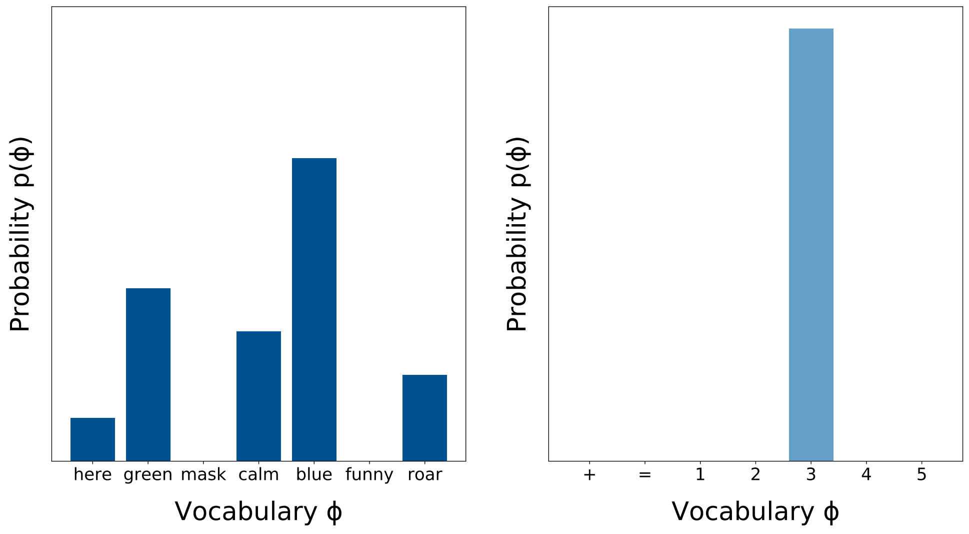


Figure 3.3.: Example true probability distributions of a symbol to be generated s_t given the previously generated symbols $S_{1:t-1}$ and the set of available symbols (vocabulary) ϕ. NLP (left, $S_{1:t-1} = \{$the, sea, is$\}$) and formulae generation (right, $S_{1:t-1} = \{$1, +, 2, =$\}$).

Consider now the same scenario applied to the problem of mathematical formulae generation and substitute sentence 1 for formula 1 and consider the generation of the last symbol "3"

Formula 1. $1 + 2 = 3$

In this instance, we can also define $S_{1:t-1} = \{$"1", "+", "2", "="$\}$, $p_G(\phi|S_{1:t-1})$ and $p_{data}(\phi|S_{1:t-1})$. Herein, the right diagram of figure 3.3 illustrates what the distribution p_{data} may look like. Note that this distribution is very spiky. Intuitively, the symbol "3" is the only one that the distribution allows: the only correct one, as the example formulas 2 and 3 demonstrate.

Formula 2. $1 + 2 = +$

Formula 3. $1 + 2 = 6$

This means that the distribution of the training data of a formulae generation problem is majorly different than the distribution of the training data of a language generation problem. Although it is possible to apply models designed for and tested on NLP problems to the generation of mathematical formulae, this subtle but fundamental difference could have major consequences and should be carefully considered.

4. Vanilla GAN

4.1. Model

The term "Vanilla GAN" indicates the simplest form of GAN; without major modifications to the original architecture proposed by Goodfellow et al. [10] and described in chapter 2. As outlined in chapter 5, Vanilla GANs operate over a continuous domain and are not applicable to discrete sequences as is. The aim of the experiments performed in this section is to analyse the extent of the applicability of continuous GANs to the objective of this thesis. Consequently, the training data for this particular model must deviate from the representation as vectors of length 8 $\in \{0,1\}$ discussed in chapter 1. Accordingly, the data is structured as a collection of integer vectors of length 3, to represent the formula $a+b=c$:

$$X_i = \begin{bmatrix} x_1^1 & x_2^1 & x_3^1 \\ x_1^2 & x_2^2 & x_3^2 \\ \vdots & \vdots & \vdots \\ x_1^N & x_2^N & x_3^N \end{bmatrix} = \begin{bmatrix} 2 & 3 & 5 \\ 6 & 6 & 12 \\ \vdots & \vdots & \vdots \\ 1 & 1 & 2 \end{bmatrix}$$

$$where \quad x_1^i, x_2^i \in [0,10], \quad x_3^i \in [0,20] \quad and \quad x_3^i = x_2^i + x_1^i.$$

The model developed for this chapter is similar to the architecture described in chapter 2: both the discriminator and the generator are feed-forward MLPs, with two layers of 200 neurons each. the generator has an output layer of 3 neurons with a linear activation function. The learning rate is set at $\eta = 5e-4$ and both models are optimized through the Adam optimizer. Note that, although the training data is comprised of integers only, the model still operates over a continuous domain.

4.2. Results

Figure 4.1 shows the evolution of the distribution of generated samples against the distribution of the training data, which indicates that the model correctly learned the probability distribution of the training data.

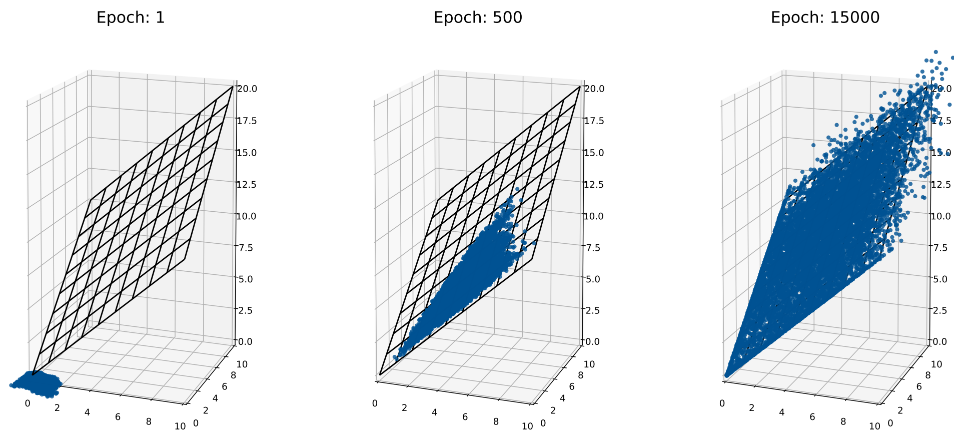


Figure 4.1.: Evolution of generator's output distribution (scatter plot), against the true distribution (wire plot) for epochs 1, 500 and 15000.

Table 4.2 shows a selection of generated vectors after completed training,where the values are truncated after three decimal places to improve readability. The mean square error of the generated samples is $3.94e - 4$, calculated by subtracting the first two entries of the vectors from the last one.

The implication of these results is twofold. Firstly, GANs are fundamentally capable of reproducing the strong correlation between outputs needed for the subsequent work in this thesis. Secondly, although the training data is effectively discrete, the network's outputs don't show any sign of tending towards integer values. Indeed, experiments conducted with real valued data suggest that restricting the training data to integers does not affect the generated samples after training. This further indicates that a model architecture capable of operating over discrete domains is needed.

7.947	-0.006	7.961
3.083	9.556	12.628
3.192	1.251	4.433
2.675	7.723	10.379
5.746	6.493	12.213
5.747	2.320	8.053
6.167	7.776	13.945
6.210	1.827	8.012
6.941	3.026	9.986
0.561	4.820	5.389
3.824	0.504	4.367
2.046	3.820	5.879
5.828	2.555	8.393
4.294	-0.001	4.316
1.833	9.135	10.964
5.405	4.648	10.064
5.531	5.617	11.132
0.020	6.165	6.182
6.570	1.485	8.071
5.684	4.474	10.160
0.566	7.501	8.067

Table 4.1.: Selection of generated vectors after training.

5. Non-differentiability Issue

5.1. Outline

Unlike deep learning models for classification tasks, GANs do not translate easily to a discrete output space. To outline such difference, consider firstly a multinomial classification problem, where the aim of the classifier is to map each input point x_i to one of N labels y_n. The classifier is easily constructed as a deep neural network with input x_i and a dense layer with N neurons as output. The output vector is subsequently passed through the softmax function, which maps it to a probability distribution.

$$\mathrm{softmax}(x_i) = \frac{\exp x_i}{\sum_{j=1}^{N} \exp x_j}, \quad i = 1, \dots, N; \quad x = (x_1, \dots, x_N) \in \mathbb{R} \tag{5.1}$$

The error between the softmax and the one-hot vector of the ground truth is consequently back-propagated during training. In testing, labels can be extracted by sampling the softmax output, hence calculating their argmax. Although such model effectively produces discrete data, in training, gradients exists everywhere and the error can be propagated.

On the other hand, consider the objective of the GAN: equation 2.1. Specifically, if one analyses the objective of the generator and its implementation (see algorithm 1):

$$-\nabla_{\theta_G} \frac{1}{m} \sum_{i=1}^{m} [\log(D(G(z_i)))]$$

one clearly sees, that the error is back-propagated through both the discriminator and the generator. This means, that the generator-discriminator ensemble must be differentiable everywhere. Consequently, contrariwise to the classifier described above, the generator cannot generate discrete data by means of sampling an outputted softmax probability distribution, because that would introduce a non-differentiable argmax layer between the generator and the discriminator. Gradients cannot propagate through a non-differentiable function as is, and the model would fail.

5.2. Straight-Forward Softmax GAN

One straightforward solution to the problem described hitherto, would be to skip the sampling step completely and pass the softmax output directly to the discriminator. Indeed, the generator-discriminator ensemble would be differentiable everywhere, allowing the error to fully back-propagate. Henceforth, the reasons behind the unviability of such solution will be discussed.

As outlined by equation 2.7 of chapter 2, the objective of the generator, assuming a fixed discriminator, is to minimize the Jensen-Shannon Divergence between the real data and the generated data $D_{JS}(p_{data}\|p_g)$:

$$\min_G V(D^*, G) = 2D_{JS}(p_{data}\|p_g) - \log 4$$

The objective is minimized when $p_{data} = p_g$ and $D_{JS} = 0$. If one were to skip the sampling step, then p_g would be the softmax output of the generator: a continuous distribution over the simplex. On the other hand, p_{data} would be the discrete distribution described by the one-hot vectors of the real data. Consequently, the Jensen-Shannon divergence between a discrete and a continuous distribution would have to be calculated. With such distributions, the Kullback-Leibler divergence would be infinite and the Jensen-Shannon divergence would saturate at 1, making the minimization problem impossible [11].

Intuitively, the discriminator learns to promptly reject all generated samples, without leaving time for the generator to properly learn. Nevertheless, in practice the GAN does not explicitly compute the Jensen-Shannon divergence between the true and the generated distributions. Therefore a set of experiments were conducted to assert the behaviour of such model. To that end, a GAN was constructed, where the generator outputs is firstly passed through a softmax layer and subsequently directly fed as input to the discriminator, named Straight-Forward Softmax GAN.

0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0

Table 5.1.: Selection of subsequently generated sequences of a Straight-Forward Softmax GAN.

Table 5.2 shows generated samples of a Straight-Forward Softmax GAN; it is clear that the generator collapsed into one mode. Figure 5.1 shows the error from the discriminator on generated samples and the percentage of unique sequences throughout training. As expected, the discriminator learns quickly to recognize the generator's samples, whereas the generator immediately falls into mode collapse (refer to the Appendix for more examples). In this form, simply skipping the sampling step between the generator and the discriminator is not a viable solution and more complex approaches that address the non-differentiability issue are needed.

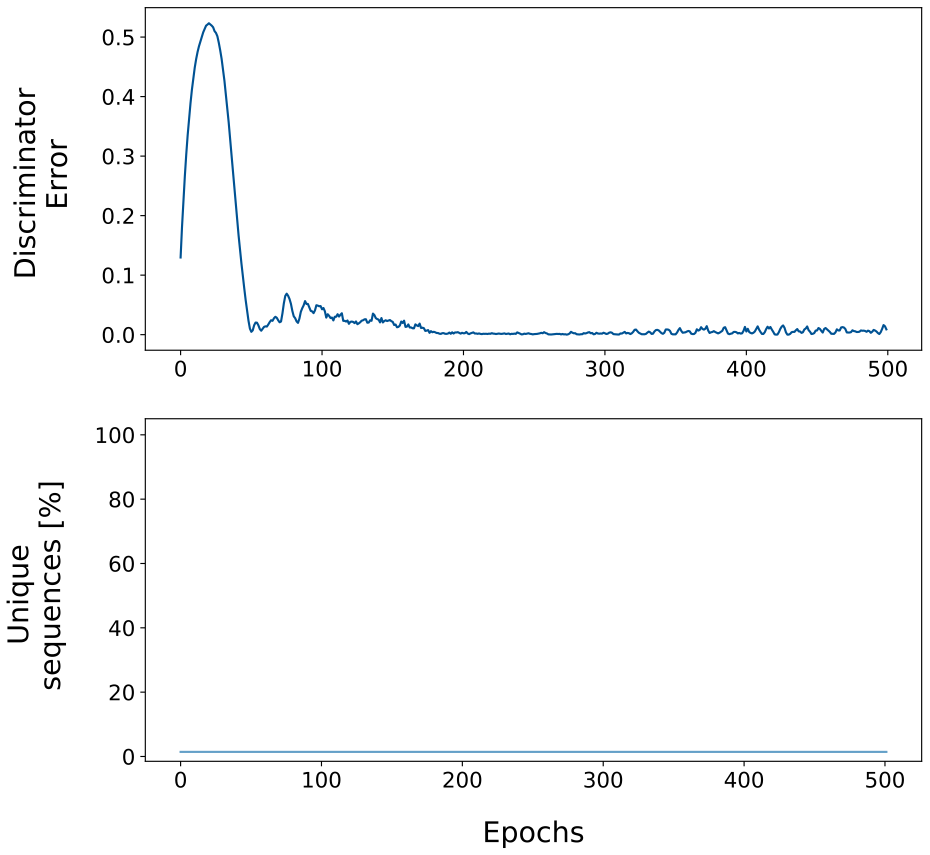


Figure 5.1.: Discriminator Error (top) and percentage of unique sequences (bottom) throughout training.

6. Gumbel-Softmax GAN

6.1. Model

Proposed by Kusner et al. in 2016 [23], the Gumbel-Softmax GAN overcomes the problem outlined in chapter 5 by use of the Gumbel-Softmax distribution. Discovered concurrently by Jang et al. [17] as the Gumbel-Softmax distribution and by Maddison et al. [26] as the Concrete distribution; it is a continuous distribution that lives on the simplex and approximates samples from a categorical distribution. By substituting the hard-decision between the generator and the discriminator with the Gumbel-Softmax distribution, Kusner et al.'s model has gradients everywhere, so that the discriminator error can back-propagate correctly. The Gumbel-Softmax distribution is a continuous relaxation of the Gumbel-Max Trick [27] [12].

6.1.1. Gumbel-Max-Trick

The Gumbel-Max-trick is a type of reparametrization trick, which allows sampling from a categorical distribution. Specifically, sampling is performed by adding i.i.d. samples from Gumbel(0,1) to unnormalized log-probabilities and selecting the highest value among these noisy logits (Lemma 3).

The Gumbel distribution $Gumbel(\mu, \beta)$, or Generalized Extreme Value distribution Type-I, is defined by:

$$\begin{aligned} \text{PDF:} \quad f(z;\mu;\beta) &= \frac{1}{\beta}\exp\{-(z-\mu)-\exp\{-(z-\mu)\}\} \\ \text{CDF:} \quad F(z;\mu;\beta) &= \exp\{-\exp\{-\frac{z-\mu}{\beta}\}\} \end{aligned} \tag{6.1}$$

with mode μ and scale β. The standard Gumbel distribution $Gumbel(0,1)$ can be sampled by means of $-\log(-\log(Uniform(0,1)))$.

Lemma 3. $\operatorname{argmax}(x_k + g_k) \sim \frac{\exp(x_k)}{\sum_{k'}^{K}\exp(x_{k'})}$, *where* $x_k \in \mathbb{R}$ *and* $g_k \sim Gumbel(0,1)$

Proof. [1] *Consider the probability that* $(x_k + g_k) = z_k \sim Gumbel(x_k, 1)$ *is larger than any other* $z_{k' \neq k}$*:*

$$\begin{aligned} P_{z_k > z_{k'}} &= \prod_{k' \neq k} F(z_k, x_{k'}) \\ &= \prod_{k' \neq k} \exp\{-\exp\{-(z_k - x_{k'})\}\} \end{aligned} \tag{6.2}$$

The marginal distribution over z_k *is known, therefore the overall probability can be computed by means of* $p(x) = \int_y f(x|y)f(y)$*:*

$$\begin{aligned} P_{k>k'} &= \int_{\infty}^{\infty} f(z_k, x_k) \times P_{z_k > z_{k'}} dz_k \\ &= \int_{\infty}^{\infty} \exp\{-(z_k - x_k) - \exp\{-(z_k - x_k)\}\} \\ &\quad \times \prod_{k' \neq k} \exp\{-\exp\{-(z_k - x_{k'})\}\} dz_k \\ &= \int_{\infty}^{\infty} \exp\{-z_k + x_k - \exp\{-z_k\} \sum_{k'=1}^{K} \exp\{x_k\}\} dz_k \end{aligned} \tag{6.3}$$

This integral has a closed form:

$$P_{k>k'} = \frac{\exp\{x_k\}}{\sum_{k'=1}^{K} \exp\{x_{k'}\}} \tag{6.4}$$

which corresponds to the softmax probability. □

Indeed, the gumbel distribution is the only one which satisfies lemma 3 [27].

6.1.2. Continuous Relaxation

Although the Gumbel-Max Trick allows for sampling from a categorical distribution, it still requires a hard decision on the noisy probabilities. The Gumbel-Softmax distribution addresses such issue by relaxing the hard decision and substituting the argmax function with a softmax function with temperature:

$$y_k = \mathrm{softmax}\left(\frac{1}{\tau}(x_k + g_k)\right) \tag{6.5}$$

The temperature parameter τ dictates the accuracy of the approximation. Figure 6.1 shows the impact of the temperature parameter on the closeness of the approximation. For high values of τ, e.g. $\tau = 10$, the expectation of the Gumbel-Softmax distribution resembles a Uniform distribution and the equation 6.5 badly approximates a sample, as the bottom-right diagram of figure 6.1 shows. For low values of τ, e.g. $\tau = 0.1$, the expectation of the Gumbel-Softmax distribution resembles the Categorical distribution parametrized by x_k and equation 6.5's approximation of a sample is nearly identical to the categorical one-hot sample, as shown in the two bottom-left diagrams of figure 6.1. Formally,

$$\lim_{\tau \to 0} \mathbb{E}\left[\mathrm{softmax}\left(\frac{1}{\tau}(x_k + g_k)\right)\right] = \mathbb{E}\left[Uniform(0,1)\right] \tag{6.6}$$

$$\lim_{\tau \to \infty} \mathbb{E}\left[\mathrm{softmax}\left(\frac{1}{\tau}(x_k + g_k)\right)\right] = \mathbb{E}\left[Categorical(x_k)\right] \tag{6.7}$$

By gradually annealing the temperature parameter during training, the Gumbel-Softmax GAN is able to learn the Categorical distribution with minimal error.

6.1.3. Structure

Figure 6.3 illustrates the structure of the unrolled Recurrent Neural Network (RNN) that functions as generator. The input at time-step $t = 0$ is the latent noise vector z; at each time-step the RNN generates a probability distribution over the set of

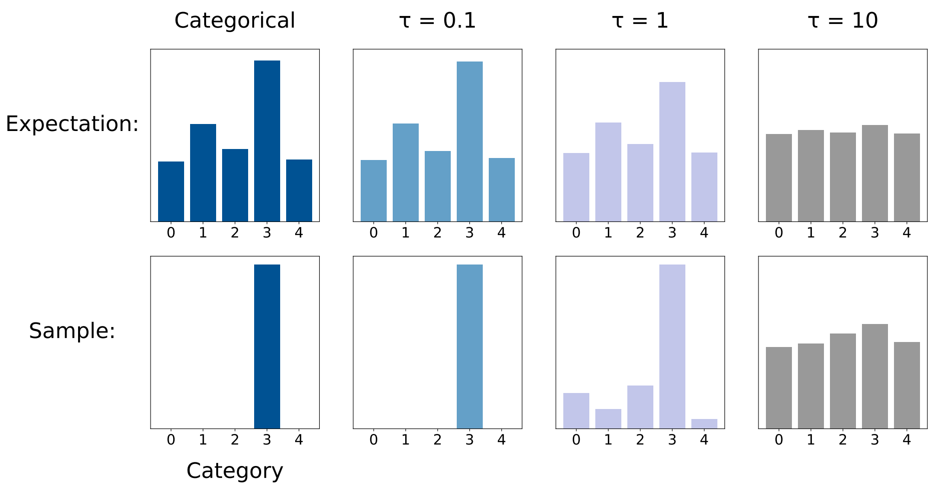


Figure 6.1.: Expectation (top) and samples (bottom) from the Gumbel-Softmax distribution with different temperatures.

possible symbols (in this thesis' case $\phi = \{0, 1\}$). This probability, parametrized by the log-unnormalized probabilities x_k (called logits hereafter) is sampled by means of equation 6.5, i.e. x_k is added to independent Gumbel noise and passed through a softmax layer. The sample is subsequently fed back into the RNN as input at time-step $t + 1$. The module that performs the sampling operation, i.e. that applies equation 6.5 to the output of the generator x_k, will be referred as "Gumbel-Softmax sampler" hereafter. The generated sequence is composed by the sample produced by the Gumbel-Softmax sampler at each timestep. In the adversarial training, the generator firstly produces a full sequence, which is subsequently fed to a dense MLP, that acts as discriminator. The discriminator is a feed-forward MLP with two layers of 400 neurons each. The model is trained for 1000 epochs; both the generator and the discriminator are optimized with the Adam optimizer [21] and a learning rate of $5e - 5$. The temperature parameter τ in equation 6.5 is annealed from ~ 3 to ~ 0.1. Two slight variations of the above described model are analysed: one with constrained and one with unconstrained logits.

6.1.4. Training Data

As discussed in chapter 1, the training data is structured as a collection vectors of length 8:

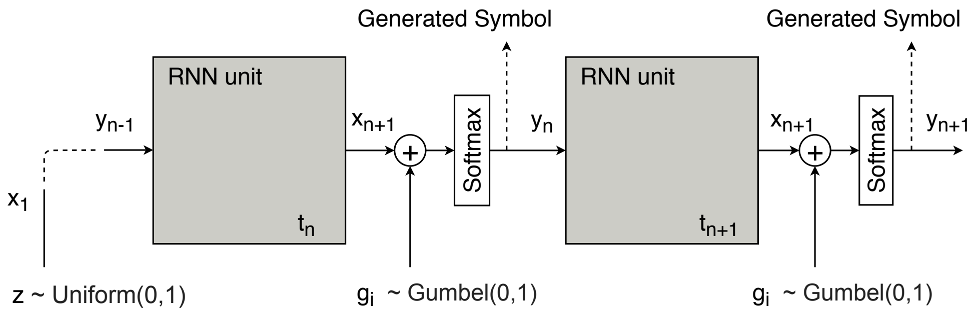


Figure 6.2.: Gumbel-Softmax GAN structure : t_n represents the current timestep, x_n is the output of the RNN at t_n and y_n is the sample of the Gumbel-Softmax distribution. The term "Gumbel-Softmax sampler" indicates the collection of operations performed between the RNN units at different time-steps.

$$X_i = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Each vector is further embedded in a one-hot vector of size 8x2 and it is fed into the model in this form.

6.2. Results

6.2.1. Constrained logits

The logits are constrained between -1 and 1 with the hyperbolic tangent activation function. Table 6.2.1 shows a selection of generated sequences after training: the correct ones, where the sum of the first four digits is equal to the sum of the last four digits, are highlighted.

Figure 6.4 shows the evolution of the amount of correct sequences generated by the generator throughout the training epochs. Note that the x-axis is placed slightly below 27%, which is the expected percentage of correct sequences when sampling

1 0 0 1 0 1 0 1	1 0 1 0 1 0 0 1
1 1 0 0 1 0 1 0	1 1 0 1 0 0 1 1
0 0 1 0 1 0 1 0	1 0 1 1 0 1 1 1
1 0 1 0 1 0 1 1	0 1 0 0 1 0 1 0
0 1 0 1 1 0 1 0	0 0 1 0 1 0 1 0
1 0 0 1 0 1 0 1	1 0 1 0 1 0 1 0
1 1 0 1 0 0 1 0	1 0 1 0 1 0 1 0
0 1 0 1 0 1 1 0	0 1 0 1 0 1 0 1
0 1 0 1 0 0 1 1	1 0 1 1 0 1 1 1
1 0 1 1 0 1 0 1	0 1 0 0 0 1 0 1
0 1 0 1 0 1 0 0	1 0 1 0 1 0 1 0
0 0 1 1 0 1 0 1	1 0 0 1 0 1 0 1
1 0 1 0 1 0 1 0	0 1 0 0 1 0 1 0
0 1 1 0 1 0 1 0	0 1 0 1 0 1 1 1
1 0 1 0 1 0 1 0	0 1 0 1 0 1 0 0

Table 6.1.: Random selection of subsequently generated sequences: correct sequences are highlighted.

from the uniform distribution. Although the model does indeed generate considerably more correct sequences than a random generator, it converges to generating approximately 60% of correct sequences. The reason behind the sub-optimal convergence can be identified in the fact, that the Gumbel-Softmax sampler has a non-zero probability P_s of changing the argmax of its input. Formally:

Lemma 4. *Equation 6.5 maps zero-probabilities in the discrete distribution to non-zero probabilities in the Gumbel-Softmax distribution.*

Proof. *Consider:*

$$y_k = \mathrm{softmax}\left(\frac{1}{\tau}(x_k + g_k)\right)$$

Let entry n of vector x_k be a zero-probability: $x_{k=n} = 0$.

$$y_{k=n} = \mathrm{softmax}\left(\frac{g_k}{\tau}\right) \tag{6.8}$$

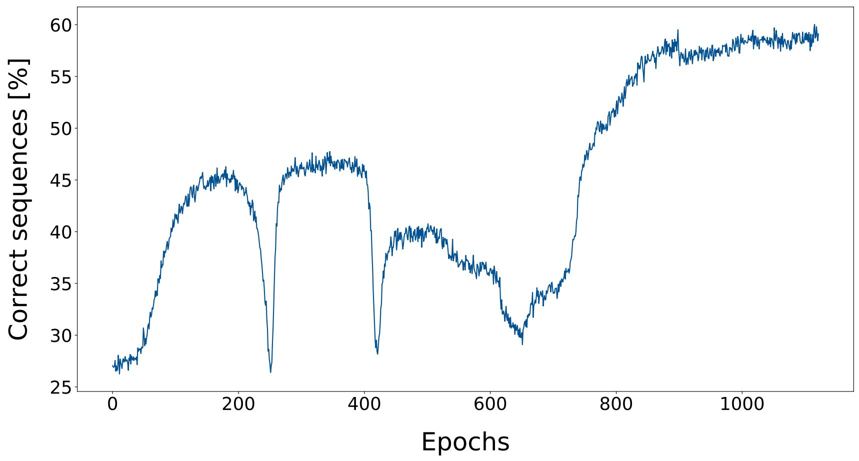


Figure 6.3.: Percentage of correct sequences throughout training (10000 samples per epoch).

Since the probability that a sample from a continuous distribution is exactly equal to a specific value is zero [7], it can be assumed that $g_k \neq 0$ and it follows that:

$$y_{k=n} \neq 0 \tag{6.9}$$

$\square$

Consequently, there is a non-zero probability P_s that a sample from the Gumbel-Softmax distribution corresponds to a category c_i with probability $p_{CAT}(c_i) = 0$ in the categorical distribution. The following illustration shows a practical example of the output of the sampler having a different argmax than the logits:

$$\begin{array}{|c|}\hline 1.0 \\ \hline 0.0 \\ \hline \end{array} \xrightarrow{GS} \begin{array}{|c|}\hline 1.179\text{e-}08 \\ \hline 9.999\text{e-}01 \\ \hline \end{array}$$

The probability P_s can indeed be quantified; recall equation 6.4 of proof , which can be reformulated as:

$$P\left(\operatorname{argmax}(x_k + g_k) \neq \operatorname{argmax}(x_k)\right) = \operatorname{softmax}(x_{k>k' \neq k}) \tag{6.10}$$

This remains valid after the softmax relaxation:

$$\begin{gathered}P\left(\operatorname{argmax}\left(\operatorname{softmax}\left(\tau^{-1}(x_k+g_k)\right)\right)\neq\operatorname{argmax}(x_k)\right)\\=\operatorname{softmax}(x_{k>k'\neq k})\end{gathered} \tag{6.11}$$

Note that, although the temperature parameter controls the degree of approximation of the softmax relaxation, it has no effect on the validity of equation 6.10. Therefore, the probability P_s that the Gumbel-Softmax sampler changes the position of the highest value of the logits x_k can be computed as:

$$P_s = 1-\operatorname{softmax}(x_{k>k'\neq k}) \tag{6.12}$$

If a constraining interval $[a,b]$ on the logits x_k is introduced, the lower bound of $P_s(x_k|[a,b])$ can be calculated: because the maximum value of the softmax function of a vector of length 2 is proportional to the difference between the entries of the vector, in the specific instance of $[a,b]=[-1,1]$, the lower bound of the probability P_s can be computed by evaluating equation 6.12 at the edges of the interval:

$$\begin{gathered}\max(\operatorname{softmax}([a,b]))\propto a-b\\\implies \inf(P_s)=P_s|_{x=[-1,1],x_{k>k'\neq k}=1}=0.1192\end{gathered} \tag{6.13}$$

To analyse the consequences of this behaviour of the Gumbel-Softmax sampler on the percentages of correct sequences a generator might generate, a perfect, deterministic generator was constructed:

Though only applicable to this specific instance, the perfect generator provides a tool to understand the effect that the probability P_s has on the percentage of correct samples among the generated ones. Indeed, by plugging the value of equation 6.13 in algorithm 2, it generates approximately 75% correct sequences. Which can be interpreted as the upper bound a Gumbel-Softmax GAN can achieve, when the logits are constrained between -1 and 1. This empirical upper bound also explains the

Algorithm 2 : Perfect Generator

```
Switch probability $P_s$;
for n= 1 to batch size do
    Initiate sequence with zeros;
    Fill first half of sequence with random numbers $\in \{0, 1\}$
    for symbols in second half of the sequence do
        if Sum of second half < sum of first half then
            symbol = 1 with probability $1 - P_s$,
            symbol = 0 with probability $P_s$
        end
        if Sum of second half > sum of first half then
            symbol = 0 with probability $1 - P_s$;
            symbol = 1 with probability $P_s$;
        end
    end
end
```

sub-optimal convergence shown in figure 6.4 and table 6.2.1. A visual interpretation of why the probability P_s effects the percentage of correct sample is offered in the following illustration:

1	0	1	0	1	1		

$$\downarrow$$

1	0	1	0	1	1	1	

Namely, if the argmax change (highlighted in the picture) happens under certain conditions, the generator cannot correct itself anymore and produces an incorrect sequence.

This property of the Gumbel-Softmax sampler has a further consequence on the generated sequences. Figure 6.4 shows the probability distribution of unique sequences of the training data p_{data} and of the generated samples p_g. The generator doesn't generate every unique sequence with the same probability; in fact, the two spikes in the distribution represent the sequences:

1	0	1	0	1	0	1	0
0	1	0	1	0	1	0	1

The alternating pattern of these two specific sequences is advantageous to the generator, because it allows this latter to mitigate the effects of the change and correct itself up to the last symbol in the sequence. Therefore, the model has learned to generate these two sequences with a starkly higher probability than the rest.

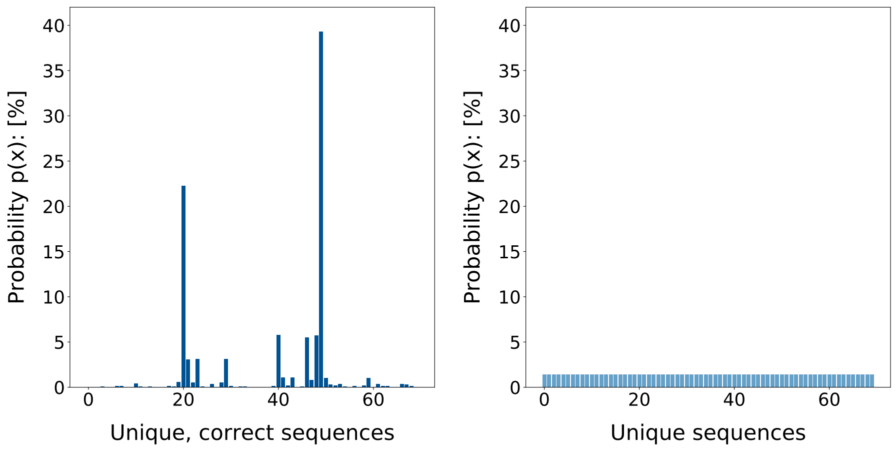


Figure 6.4.: Probability distribution of the training data (right) and of the generated data (left): each bar in the diagram represents one unique sequence of length 8 (computed on 10000 sample, incorrect sequences are omitted).

6.2.2. Unconstrained logits

A logical solution to the problems outlined in section 6.2.1 is to not constrain the logits. To that end, an identical model is constructed, whose only difference is the activation function of the last layer of the generator: herein, a linear one is adopted. In theory, if the difference between the single logits is high enough, the probability of the Gumbel-Softmax sampler changing the argmax of the logits, as per equation 6.12, becomes negligible, e.g.:

$$P_s|_{x=[-10,10],x_{k>k'\neq k}=10} \approx 2.6e-9 \tag{6.14}$$

The generator should therefore learn to space the logits sufficiently apart and mitigate the issue described hitherto. Nevertheless, the experimental results show, that the model with unconstrained logits always falls into mode collapse. Table 6.2.2 shows an excerpt from the output of two different training runs of the model, one with total mode collapse and one with partial mode collapse. Furthermore, to exclude the possibility that this failure simply stems from sub-optimal hyperparameters, several experiments with different parameters have been run: the failure could be observed across all the experiments (refer to the appendix for the other results).

To further analyse the possible reason behind the catastrophic mode collapse

0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

1 0 1 0 1 0 1 0
0 1 0 1 0 1 0 1
1 0 1 0 1 0 1 0
0 1 0 1 0 1 0 1
1 0 1 0 1 0 1 0
0 1 0 1 0 1 0 1
1 0 1 0 1 0 1 0
0 1 0 1 0 1 0 1
1 0 1 0 1 0 1 0
0 1 0 1 0 1 0 1
1 0 1 0 1 0 1 0
0 1 0 1 0 1 0 1
1 0 1 0 1 0 1 0
0 1 0 1 0 1 0 1
1 0 1 0 1 0 1 0

Table 6.2.: Mode Collapse: two different training runs, total mode collapse (left) and partial mode collapse (right).

of the model, a study on the relationship between the value of the logits and the percentage of unique sequences generated has been conducted. As figure 6.2.2 clearly shows, there exists a strong correlation between the difference of the two logits and the percentage of unique sequences generated by the model. Based on this experimental correlation, it can be inferred that:

- The distance between the logits is proportional to the percentage of correct sequences. (refer to equation 6.12)
- The distance between the logits is inversely proportional to the percentage of unique sequences.

Henceforth, three possible explanations of the model's behaviour are discussed.

Hypothesis 1. *The high difference between the logits interferes with the differentiability of the softmax relaxation of the Gumbel-Max trick.*

Recall equation 6.5:

$$y_k = \mathrm{softmax}\left(\frac{1}{\tau}(x_k + g_k)\right)$$

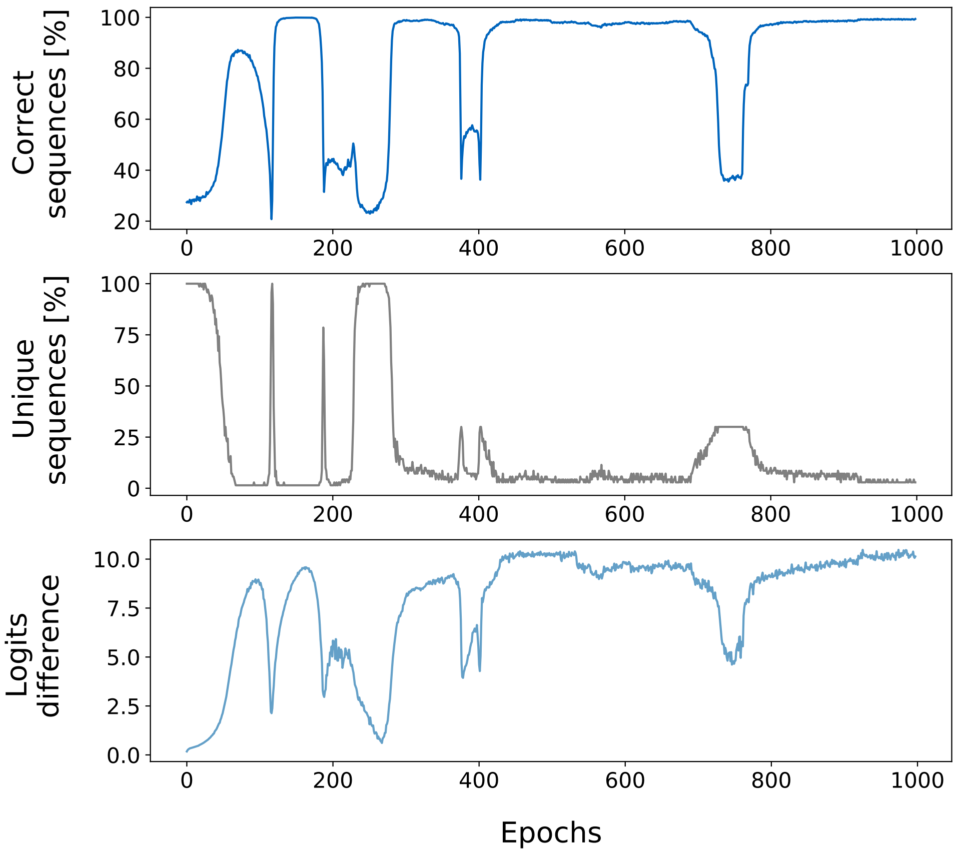


Figure 6.5.: Percentage of correct sequences (top), percentage of unique sequences (middle) and difference between the logits (bottom) against the epochs of one training session.(10000 samples per epoch, refer to appendix for more examples.

The softmax relaxation works by starting with a high temperature and gradually annealing it. At first, the approximation is bad but the gradients are high, then the approximation gets gradually better and the gradients get gradually smaller. The model learns alongside the annealing until it learns to reproduce the true categorical distribution with minimal error. However, if the generator immediately generates logits with a high difference, the approximation becomes immediately nearly categorical and the gradients vanish before the generator has finished learning.

Hypothesis 2. *The high difference between the logits causes sampling the Gumbel-Softmax distribution indistinguishable from taking the softmax of the logits.*

If the difference between the logits is high enough, the contribution of the gumbel noise becomes negligible, as outlined in equation 6.15.

$$\lim_{x_k \to \infty} \operatorname{softmax}\left(\frac{1}{\tau}(x_k + g_k)\right) = \operatorname{softmax}(x_k) \tag{6.15}$$

Hypothesis 2 is corroborated by the fact that the experimental results of this model show strong similarities to the experimental results of the model discussed in chapter 5, wherein the softmax of the logits is directly passed to the discriminator.

Hypothesis 3. *The high difference between the logits causes the softmax function to saturate and its gradients to vanish.*

$$\lim_{x \to \infty} \nabla \operatorname{softmax}(x) = 0 \tag{6.16}$$

It is a known issue, that the gradients of the softmax tend to vanish if the values of the input are too high. Generally, a common solution is to normalize the inputs, though that is difficult to apply in this instance due to the temperature parameter control needed by the Gumbel-Softmax distribution and the erratic behaviour of the difference between the logits.

7. Sequence GAN

7.1. Reinforcement Learning

The Sequence Generative Adversarial Network (Seq-GAN) [41] combines a GAN with a RL methodology, to overcome the issue of back-propagating the error through discrete samples (refer to chapter 5).

Reinforcement Learning indicates a family of ML algorithms, that learn by interacting with an environment. Generally, a RL agent is enclosed in a state-action space; the agent selects actions based on its current policy and the environment communicates the new state and a reward based on the state-action tuple. The agent then updates its policy based on the received reward by performing Policy Gradient , i.e. by maximizing the expected future reward.

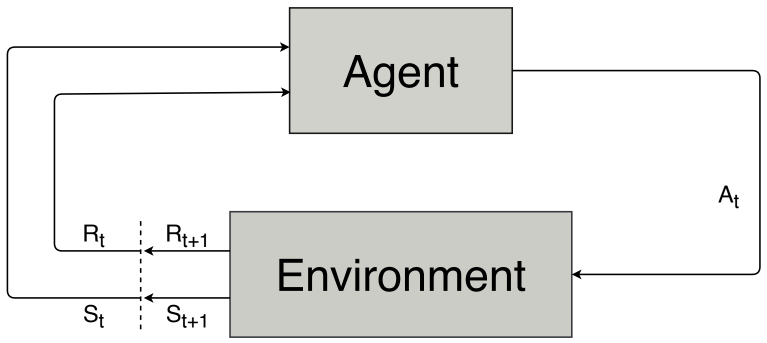


Figure 7.1.: Reinforcement Learning: Agent selects action A_t, Environment returns the Reward R_{t+1} and the new State S_{t+1}.

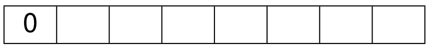
7.2. Model

In Seq-GAN, the generator is treated like an RL agent, whose state-action space is defined by the previously generated symbols in the sequence and the next generated word, respectively. Formally, we can define a sequence $Y_{1:T} = [y_1, \dots, y_T]$, and specify:

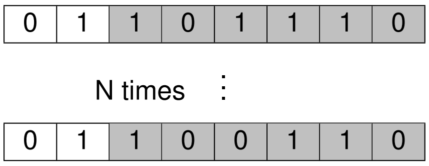
- The state $S_t = Y_{1:t-1} = [y_1, \dots, y_{t-1}]$

- The action $a_t = y_t$
- The final reward $R(S_{T-1}, a_t) = D(Y_{1:T})$, where D denotes the discriminator model.

Note that the reward is only defined for the entire sequence, which means that it is sparse and provides little information about the intermediate actions. To address this issue the Seq-GAN incorporates a roll-out policy. Assume that the current state S_t is represented by the following incomplete sequence $Y_{1:t-1}$:



Further assume, that the action selected by the agent is $a_t = 1$. To compute the intermediate reward $R(S_t, a_t)$, a N-time Monte-Carlo (MC) search is performed to fill in the remaining unknown symbols n different times:



The highlighted squares represent the symbols filled in by the MC search. The reward is computed as the average of the score of the discriminator on the N MC sequences. With the intermediate reward, the generator is updated through policy gradient. Finally, the state is updated with the generated symbol and the process continues until the sequence is complete. Naturally, for the last symbol no MC search is needed, as the discriminator is able to provide the reward for the completed sequence. A visualization of this process is provided in figure 7.2 and in algorithm 3. Since the generator acts as an RL agent and its updated through policy gradient, there is no back-propagation through the generator-discriminator ensemble: the discriminator is merely the model that provides the reward. Therefore, the Seq-GAN does not have the same issues as other GANs with discrete target distributions.

7.3. Policy Gradient

The generator's parameters are updated with the Adam optimizer using the REINFORCE algorithm [40] and following [36].

$$\theta \leftarrow \theta + \eta \nabla_\theta J(\theta) \tag{7.1}$$

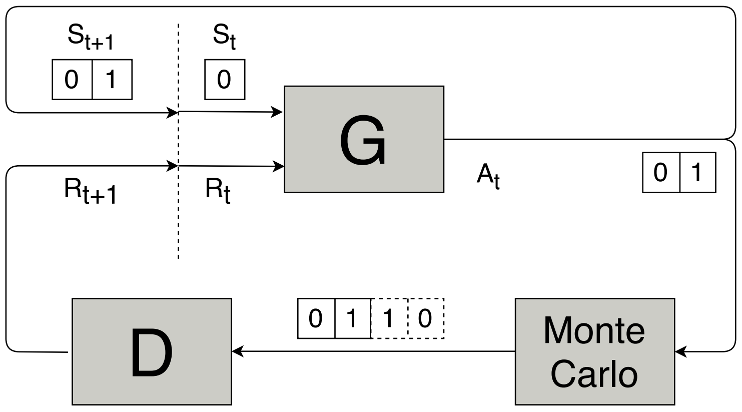


Figure 7.2.: Structure of Seq-GAN, example of generation of binary sequences: herein the sequence is of length 4 to improve the comprehension of the diagram.

where η denotes the learning rate and:

$$\nabla_\theta J(\theta) = \sum_{t=1}^{T} \mathbb{E}_{y_t \sim G_\theta(y_t|T_{1:t-1})}[\nabla_\theta \log G_\theta(y_t|Y_{1:t-1}) Q(Y_{1:t-1}, y_t)] \tag{7.2}$$

where $Q(Y_{1:t-1}, y_t)$ indicates the reward at time-step t based on state $S_t = Y_{1:t-1}$ and action $a_t = y_t$:

$$Q(Y_{1:t-1}, y_t) = \begin{cases} \frac{1}{N}\sum_{n=1}^{N} D(Y_{1:T}^n) \quad Y_{1:T}^n \in MC(Y_{1:t}, N) & \text{for} \quad t < T, \\ D(Y_{1:t}) & \text{for} \quad t = T \end{cases} \tag{7.3}$$

where $Y_{1:T}^n \in MC(Y_{1:t}, N)$ is the nth search of a N-time Monte-Carlo search to complete one sequence $Y_{1:t-1}$. For a detailed derivation of equations 7.1, 7.2 and 7.3 refer to Yu et al.'s paper [41].

Algorithm 3 : Sequence Generative Adversarial Network, adapted from [41]

```
generator policy $G_\theta$; MC-policy $G_\beta$; discriminator D; real data $S = \{X_{1:T}\}$;
Pre-train $G_\theta$ with MLE on $S$;
Pre-train $D$;
for epochs do
    for g-steps do
        Initiate sequence $Y = \{\}$;
        for t in 1:T do
            Generate a symbol $y_t \sim G_\theta$;
            Generate batch of sequences $Y_{t:T}^{MC^i} \sim G_\beta$;
            Compute reward;
            Update generator parameters via policy gradient;
            Append $y_t$ to $Y$;
        end
    end
    for d-steps do
        Use current $G_\theta$ to generate negative examples and draw positive
          examples from $S$;
        Train discriminator $D$ for k epochs;
    end
    $\beta \leftarrow \theta$;
end
```

7.4. Results

Table 7.4 shows an excerpt of sequences generated by the model after training: it is able to generate sequences with $\sim 97\%$ accuracy (See figure 7.3). Furthermore, the probability distribution of the generated sequences shown in figure 7.4 indicates that the model doesn't contain any particular bias and reproduces the true probability distribution to an acceptable degree with the Jensen-Shannon divergence between it and the uniform distribution of:

$$JS\left(p(x) \sim G(z) \| q(x) \sim U(0,1)\right) \approx 0.03 \tag{7.4}$$

0 1 0 1 1 0 0 1	0 1 1 1 0 0 1 1
0 0 1 0 0 0 1 0	0 0 0 1 0 1 0 0
0 0 0 1 1 0 0 0	1 1 1 0 0 1 1 1
1 1 1 0 1 0 1 1	1 1 0 0 1 1 0 0
0 1 1 0 1 1 0 0	0 0 1 1 0 0 1 1
0 0 1 1 1 1 0 0	1 0 1 1 1 1 0 1
0 1 1 1 0 1 1 1	1 0 1 1 0 1 1 1
1 0 1 1 1 1 1 0	0 0 1 1 0 1 0 1
1 0 1 0 1 0 0 1	0 1 1 0 1 0 0 1
0 0 1 0 0 1 0 0	1 0 0 0 1 0 0 0
0 1 0 1 1 0 0 1	0 0 0 0 0 0 0 0
1 0 1 1 1 1 0 1	1 1 0 0 0 0 1 1
1 0 1 0 1 0 0 1	1 1 0 0 1 1 0 0
1 0 1 0 1 0 0 1	1 0 1 0 1 0 0 1
0 1 0 0 1 0 0 0	0 0 1 1 0 1 0 1

Table 7.1.: Random selection of subsequently generated sequences: incorrect sequences are highlighted.

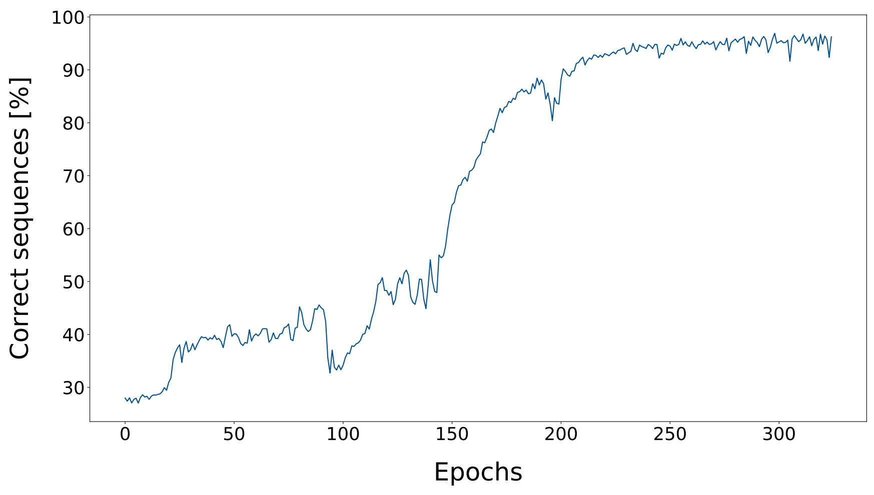


Figure 7.3.: Percentage of correct sequences throughout training (10000 samples per epoch).

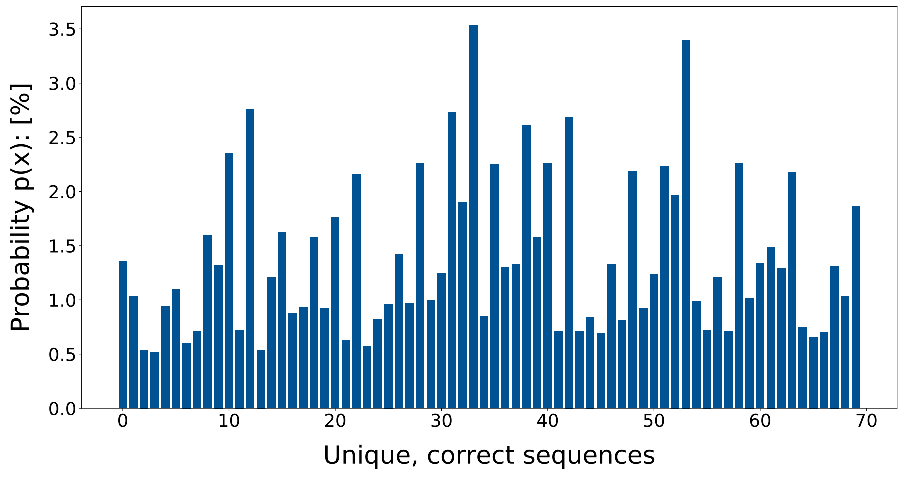


Figure 7.4.: Probability distribution of the generated data, each bar in the diagram represents one unique sequence of length 8 (computed on 10000 samples, incorrect sequences were omitted).

8. Discussion

The results presented in chapter 6 strongly indicate that the Gumbel-Softmax GAN is not a viable option for the objective of this thesis. It is furthermore important to note, that the issues found within the scope of this work, escalate with more complicated formulae. To that end consider firstly the consequences of increasing the length of the training sequences, from 8 to e.g. 16 and maintaining all other variables constant, i.e. constraint $[a, b] = [1, -1]$ and set of possible symbols $\{0, 1\}$: by using the model described in algorithm 2, we can determine that this would result in a decrease of accuracy from $\sim 75\%$ to $\sim 64\%$. Secondly, consider the effect of expanding the set of possible symbols from $\{0, 1\}$ to $\{0, 1, +, =\}$. In chapter 6 it is assumed, that the softmax function on an interval $[a, b]$ has its maximum at the edges of the interval. This allowed the calculation of a lower bound to the probability P_s, that the Gumbel-Softmax sampler's output has a different argmax than its input. While this property holds undoubtfully true for logit vectors of length $n = 2$, it is not as straight-forward for the case $n > 2$ and requires a formal proof:

Lemma 5. $\max(\text{softmax}(x)), \quad x \in [a, b]$ *is maximized by logits of the form:*

$$x = [b, a, \dots, a]$$

Proof.

$$\begin{aligned}
&\min(P_s = 1 - \text{softmax}(x_{k>k'\neq k})) \\
&= \max(\text{softmax}(x_{k>k'\neq k})) \\
&= \max\left(\frac{\exp(x_{k>k'\neq k})}{\exp(x_{k>k'\neq k}) + \sum_{j\neq k}^{K}\exp(x_j)}\right)
\end{aligned} \tag{8.1}$$

Given a fixed $\exp(x_{k>k'\neq k})$ *equation 8.1 is maximized by minimizing the quantity* $\sum_{j\neq k}^{k}\exp(x_j)$*. Given a fixed* $\sum_{j\neq k}^{k}\exp(x_j)$ *, it is maximized by maximizing* $\exp(x_{k>k'\neq k})$*. This is achieved by setting* $x_{k>k'\neq k} = b$ *and* $x_{i\neq k} = a$*.* □

Lemma 5 and its proof provide the knowledge, that a softmax function constrained in the interval $[a, b]$ has indeed its maximum at the edges of the interval for all vectors of length $n \in \mathbb{N}$. This allows the employment of equation 6.10 and algorithm 2 to compute $\inf(P_s)$ for larger vocabularies as well. Therefore, maintaining all other variables constant, the effect of the expansion of the vocabulary on the accuracy of the Gumbel-Softmax GAN can be investigated. Expanding the vocabulary from $\{0, 1\}$ to $\{0, 1, +, =\}$, increases $\inf(P_s)$ from $\sim 11\%$ to $\sim 29\%$ and decreases the maximum possible accuracy from $\sim 75\%$ to $\sim 42\%$. To summarize:

- Increasing the length of the sequence decreases the maximum possible accuracy of the model.
- Increasing the size of the vocabulary decreases the maximum possible accuracy of the model.

This implies that the Gumbel-Softmax GAN, along with not being applicable to the specific objective of this thesis, it is additionally not applicable to possible expansions thereof. The reason behind the inapplicability of the Gumbel-Softmax GAN on the problem of generating mathematical formulae, despite its results on language generation problems, lies entirely in the difference between the two problems outlined in chapter 3. The smoother structure of the underlying distribution of the training data in a language generation problem, considerably mitigates the impact of the argmax change performed by the Gumbel-Softmax sampler. In this context, a smoother probability distribution is interpretable as a higher percentage of correct words in the vocabulary.

Figure 8.1 illustrates the difference between the Gumbel-Softmax sampling operation for a language generation problem and a formulae generation problem. In both cases the Gumbel-Softmax sampler generates a sample different from the most probable in the categorical distribution. On the smoother language generation distribution, this merely causes the selection of a less probable, but likely still appropriate word. However, on the spiky formulae generation distribution, this causes the selection of a completely incorrect symbol.

Moreover, due to the fact that the logits are not normalized, the sum of the probabilities associated with the correct words in the language generation distribution can be higher than the upper value of the constraining interval. In this case, the probability that the Gumbel-Softmax sample selects an incorrect word is lower than with the formulae generation distribution. Note that the constraining interval need not be selected by the developer of the model, but can also refer to an interval learned by the model, if the logits are unconstrained (refer to the Appendix). Furthermore, it is noteworthy that the metrics used to evaluate NLP models are not exact: for example, a slightly out of place word, in an otherwise plausible sentence does not necessarily register as a completely incorrect sentence. Ergo, the effects of the issues described in this thesis may lie within the error margins of the evaluation

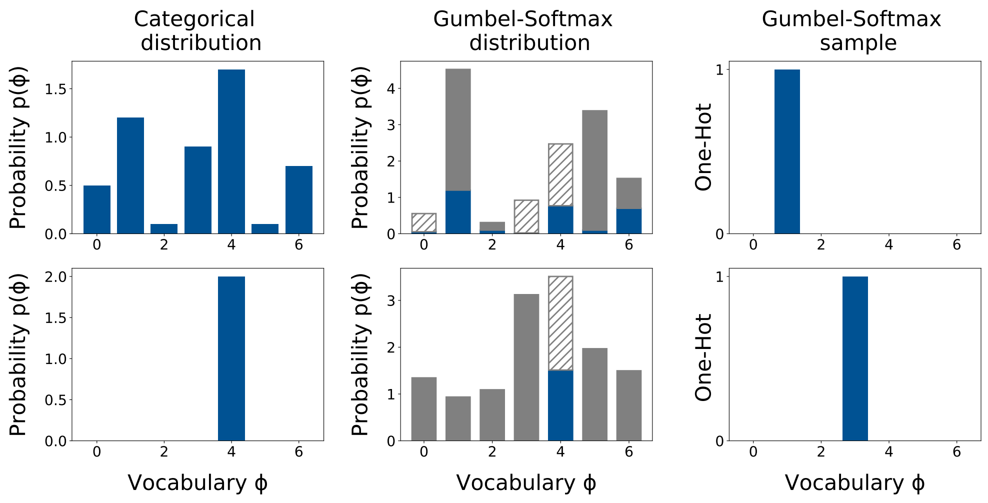


Figure 8.1.: Comparison of the processes of sampling the Gumbel-Softmax distribution for language generation (top) and formulae generation (bottom). The positive gumbel noise is represented in grey, the negative gumbel noise is represented in hatched grey. The right pair of diagrams represents the output of the generator, i.e. the logits. it can be also interpreted as the probability distribution over the vocabulary, given a previously generated incomplete sequence (see chapter 3).

methods used in NLP, whereas they assuredly do not lie within the error margins of any appropriate evaluation method that may be used for mathematical formulae generation.

On the other hand, the results presented in chapter 7 show that the Sequence GAN manages to fulfill the objective of this thesis: the trained model is capable of generating accurate and correctly distributed sequences. The RL approach provides an elegant solution to the non-differentiability issue and seems to be particularly apt for the objective of this thesis. While it was determined, that the issues revealed by the experimental and mathematical results of the Gumbel-Softmax GAN would escalate with the expansion of the objective of this thesis, no such assessment could be made from the results of the Sequence GAN. Both GANs and Deep RL models have demonstrated particular efficacy on highly complex tasks, such as GANs that can generate real looking pictures and Deep RL models, like AlphaGo [35], that can beat the most skillful humans in sophisticated games. This indicates that a future expansion on the objective of this work would not be outside the realm of abilities of a model based on the Sequence GAN architecture.

9. Conclusion

The thesis' aim was to anlyse the viability of using Generative Adversarial Networks to produce mathematical formulae, i.e. discrete sequences of highly corrrelated symbols. Firstly, an attentive investigation into the posssible structures of the training data was conducted, with the intent to fulfill two conditions:

- The training data should be reduced to its simplest form.
- The results derived from it should be as significant and as generalizable as possible.

To that end, a sequence of length eight was chosen, such that the sum of the first half is equal to the sum of the second half. Because the sequence contains only ones and zeros, it fulfills the first condition and, as discussed in chapter 1, its discrete form provides the knowledge, that the results of this thesis are applicable on more complicated formulae, fulfilling the second condition.

Notwithstanding, a first experiment with a continuous GAN was successfully conducted and indicated the capability of the model to generate highly correlated sets of outputs.

Subsequently, an analysis of the issues that arise from trying to operationalize GANs over discrete domains was conducted and it was determined that a modified GAN was needed; it was further asserted, that GANs originally intended for language processing were the best candidates to tackle the task at hand, due to the strong similarities between it and the objective of this thesis. Accordingly, two such models were modified and tested. Firstly, the Gumbel-Softmax GAN, which uses a reparametrization trick derived from the Gumbel-Max trick to differentiably sample a categorical distribution and back-propagate through the model. It was discovered that, because of the differences between language generation and formulae generation, the Gumbel-Softmax GAN is not viable for this thesis' objective, as it could generate correct sequences with only $\sim 60\%$ accuracy. A Mathematical explanation of its issues was provided and it was shown, that the described problems escalate with more complicated formulae.

Finally, the Sequence GAN was tested. It borrows from the field of RL to overcome the issues that GANs have with discrete data. The generator is an RL Agent with the state-action space described by the previously generated symbols and the symbol to generate, respectively. Moreover, the discriminator provides a reward for each state-action tuple through a Monte-Carlo roll-out policy. It was shown that the

Sequence-GAN , contrary to the Gumbel-Softmax GAN, was capable of fulfilling the objective of this thesis with $\sim 97\%$ correct sequences. Future work would therefore be based on the Sequence GAN and entail a gradual expansion of the training data to include further symbols such as mathematical operands, constants, integrals and derivatives, like shown by the following illustration: current structure (top) and possible expansion (bottom).

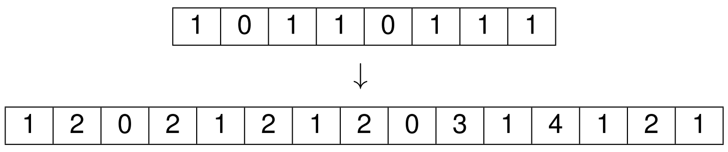


Figure 9.1.: Visualization of one possible expansion of the training data: expanding the sets of possible symbols to include the mathematical operands $+$ (encoded as the value 2), $-$ (encoded as the value 3) and $=$ (encoded as the value 4). The top sequence describes the formula "$1+0+1+1 = 0+1+1+1$", whereas the bottom sequence describes the formula "$1+0+1+1+0-1 = 1+1$".

List of Acronyms

GAN	Generative Adversarial Network.
JS	Jensen-Shannon divergence.
KL	Kullback–Leibler divergence.
MC	Monte-Carlo.
ML	Machine Learning.
MLP	Multilayer Perceptron.
NLP	Natural Language Processing.
RL	Reinforcement Learning.
RNN	Recurrent Neural Network.
Seq-GAN	Sequence Generative Adversarial Network.
WGAN	Wasserstein Generative Adversarial Network.

List of Symbols

η	Learning rate.
D	Discriminator.
G	Generator.
G_β	Monte-Carlo Generator. (In this thesis $G_\beta = G_\theta$).
G_θ	Main Generator of the Sequence GAN. (In this thesis $G_\beta = G_\theta$).
$R(S, a)$	Reward given state S and action a.
S_t	State at time-step t.
$Y_{1:t}$	Set of symbols in a sequence up to time-step t.
a_t	Action at time-step t.
g_i	Sample from Gumbel(0,1).
p_g	Probability distribution of the generator's output.
p_z	Probability distribution of the generator's input, $p_z \sim U(0, 1)$.
p_{data}	Probability distribution of the training data.
z	Generator's input, $z \sim U(0, 1)$.
P_s	Probability, that the Gumbel-Softmax sampler shifts the argmax of its input.

List of Figures

1.1. Schrödinger equation in an infinite postential well 9

2.1. Basic structure of a GAN. 13

3.1. Selection of pictures generated by a Large Scale GAN [4]. 19
3.2. Selection of pictures generated by Style-GAN II [18] [19]. 20
3.3. Comparison between true probability distributions of a NLP and a Formulae Generation problem 23

4.1. Vanilla GAN: evolution of the generated distribution compared the true distribution 26

5.1. Straight-Forward Softmax GAN: discriminator error and percentage of unique sequences throughout training 32

6.1. Expectation and samples from the Gumbel-Softmax distribution with different temperatures 36
6.2. Structure of the Gumbel-Softmax GAN 37
6.3. Gumbel-Softmax GAN (constrained): Percentage of correct sequences throughout training 39
6.4. Gumbel-Softmax GAN (constrained): comparison between generated distribution and true distribution 42
6.5. Gumbel-Softmax GAN (unconstrained): Percentage of correct sequences, percentage of unique sequences and difference between the logits throughout training 44

7.1. Working principle of Reinforcement Learning 47
7.2. Structure of the Sequence GAN 49
7.3. Sequence GAN: Percentage of correct sequences throughout training 51
7.4. Sequence GAN: probability distribution of the generated data 52

8.1. Comparison of the processes of sampling the Gumbel-Softmax distribution for language generation and formulae generation 55

9.1. Visualization of one possible expansion of the training data 58

A.1. Straight-Forward Softmax GAN: baseline 75

A.2. Straight-Forward Softmax GAN: experiment I 76
A.3. Straight-Forward Softmax GAN: experiment II 76
A.4. Straight-Forward Softmax GAN: experiment III 76
A.5. Straight-Forward Softmax GAN: experiment IV 76

B.1. Gumbel-Softmax GAN (unconstrained): baseline 77
B.2. Gumbel-Softmax GAN (unconstrained): experiment I 78
B.3. Gumbel-Softmax GAN (unconstrained): experiment II 78
B.4. Gumbel-Softmax GAN (unconstrained): experiment III 78
B.5. Gumbel-Softmax GAN (unconstrained): experiment IV 78

List of Tables

3.1. Examples of sentences generated by Leak-GAN and Seq-GAN . . . 21

4.1. Vanilla-GAN: Selection of generated vectors 27

5.1. Straight-Forward Softmax GAN: selection of generated sequences . 31

6.1. Gumbel-Softmax GAN (constrained): selection of generated sequences 38
6.2. Gumbel-Softmax GAN (unconstrained): selection of sequences generated in mode collapse 43

7.1. Sequence GAN: random selection of subsequently generated sequences: incorrect sequences are highlighted 51

List of Algorithms

1. Generative Adversarial Network 14

2. Perfect Generator 41

3. Sequence Generative Adversarial Network 50

Bibliography

1. R. Adams. The gumbel-max trick for discrete distributions. 2013. URL `https://lips.cs.princeton.edu/the-gumbel-max-trick-for-discrete-distributions`. Accessed on 23.07.2020.

2. M. Arjovsky and L. Bottou. Towards principled methods for training generative adversarial networks. In *arXiv preprint arXiv:1701.04862*, 2017.

3. M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein gan. In *arXiv preprint arXiv:1701.07875*, 2017.

4. A. Brock, J. Donahue, and K. Simonyan. Large scale gan training for high fidelity natural image synthesis. In *arXiv preprint arXiv:1809.11096*, 2018.

5. K. Chen, Q. Huang, H. Palangi, P. Smolensky, K.D. Forbus, and J. Gao. Natural-to formal-language generation using tensor product representations. In *Thirty-third Conference on Neural Information Processing Systems (NeurIPS) 2019*. 2019.

6. H. DeGroot Morris and M.J. Schervish. *Probability and statistics*. Addison Wesley, 2002.

7. Y. Dodge. *The concise encyclopedia of statistics*. Springer Science & Business Media, 2008.

8. D. Foster. *Generative deep learning: teaching machines to paint, write, compose, and play*. O'Reilly Media, 2019.

9. I. Goodfellow. Nips 2016 tutorial: Generative adversarial networks. In *arXiv preprint arXiv:1701.00160*, 2016.

10. I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pp. 2672–2680. 2014.

11. I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A.C. Courville. Improved training of wasserstein gans. In *Advances in neural information processing systems*, pp. 5767–5777. 2017.

12. E.J. Gumbel. *Statistical theory of extreme values and some practical applications: a series of lectures*. US Govt. Print. Office, 1954.

13. J. Guo, S. Lu, H. Cai, W. Zhang, Y. Yu, and J. Wang. Long text generation via adversarial training with leaked information. In *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

14. D. Harris and S. Harris. *Digital design and computer architecture*. Morgan Kaufmann, 2010.

15. R.D. Hjelm, A.P. Jacob, T. Che, A. Trischler, K. Cho, and Y. Bengio. Boundary-seeking generative adversarial networks. In *arXiv preprint arXiv:1702.08431*, 2017.

16. S. Hochreiter and J. Schmidhuber. Long short-term memory. In *Neural computation*, 9(8), pp. 1735–1780, 1997.

17. E. Jang, S. Gu, and B. Poole. Categorical reparameterization with gumbel-softmax. In *arXiv preprint arXiv:1611.01144*, 2016.

18. T. Karras, S. Laine, and T. Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4401–4410. 2019.

19. T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila. Analyzing and improving the image quality of stylegan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8110–8119. 2020.

20. S.W. Kim, Y. Zhou, J. Philion, A. Torralba, and S. Fidler. Learning to simulate dynamic environments with gamegan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1231–1240. 2020.

21. D.P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *arXiv preprint arXiv:1412.6980*, 2014.

22. N. Kodali, J. Abernethy, J. Hays, and Z. Kira. On convergence and stability of gans, rejected. In *arXiv preprint arXiv:1705.07215*, 2017.

23. M.J. Kusner and J.M. Hernández-Lobato. Gans for sequences of discrete elements with the gumbel-softmax distribution. In *arXiv preprint arXiv:1611.04051*, 2016.

24. Y. LeCun. What are some recent and potentially upcoming breakthroughs in deep learning? 2016. URL `https://www.quora.com/What-are-some-recent-and-potentially-upcoming-breakthroughs-in-deep-learning`. Accessed on 23.07.2020.

25. K. Lin, D. Li, X. He, Z. Zhang, and M.T. Sun. Adversarial ranking for language generation. In *Advances in Neural Information Processing Systems*, pp. 3155–3165. 2017.

26. C.J. Maddison, A. Mnih, and Y.W. Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *arXiv preprint arXiv:1611.00712*, 2016.

27. C.J. Maddison, D. Tarlow, and T. Minka. A* sampling. In *Advances in Neural Information Processing Systems*, pp. 3086–3094. 2014.

28. J. Nash. Non-cooperative games. In *Annals of mathematics*, pp. 286–295, 1951.

29. W. Nie, N. Narodytska, and A. Patel. Relgan: Relational generative adversarial networks for text generation. In *International conference on learning representations*. 2018.

30. S. Nowozin, B. Cseke, and R. Tomioka. f-gan: Training generative neural samplers using variational divergence minimization. In *Advances in neural information processing systems*, pp. 271–279. 2016.

31. S. Rajeswar, S. Subramanian, F. Dutil, C. Pal, and A. Courville. Adversarial generation of natural language, criticized. In *arXiv preprint arXiv:1705.10929*, 2017.

32. T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training gans. In *Advances in neural information processing systems*, pp. 2234–2242. 2016.

33. A. Santoro, R. Faulkner, D. Raposo, J. Rae, M. Chrzanowski, T. Weber, D. Wierstra, O. Vinyals, R. Pascanu, and T. Lillicrap. Relational recurrent neural networks. In *Advances in neural information processing systems*, pp. 7299–7310. 2018.

34. I. Schlag, P. Smolensky, R. Fernandez, N. Jojic, J. Schmidhuber, and J. Gao. Enhancing the transformer with explicit relational encoding for math problem solving. In *Thirty-third Conference on Neural Information Processing Systems (NeurIPS) 2019*. 2019.

35. D. Silver, A. Huang, C.J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot et al.. Mastering the game of go with deep neural networks and tree search. In *nature*, 529(7587), pp. 484–489, 2016.

36. R.S. Sutton, D.A. McAllester, S.P. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pp. 1057–1063. 2000.

37. H. Thanh-Tung and T. Tran. On catastrophic forgetting and mode collapse in generative adversarial networks. In *arXiv*, pp. arXiv–1807, 2018.

38. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008. 2017.

39. A.S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In *arXiv preprint arXiv:1703.01161*, 2017.

40. R.J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Machine learning*, 8(3-4), pp. 229–256, 1992.

41. L. Yu, W. Zhang, J. Wang, and Y. Yu. Seqgan: Sequence generative adversarial nets with policy gradient. In *Thirty-first AAAI conference on artificial intelligence*. 2017.

42. E. Zakharov, A. Shysheya, E. Burkov, and V. Lempitsky. Few-shot adversarial learning of realistic neural talking head models. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 9459–9468. 2019.

Appendices

A. Straight-Forward Softmax GAN

Figure A.1 shows the error of the discriminator and the percentage of unique sequences throughout training. Note that the error of the discriminator converges between 0.2 and 0.5, which is expected from a GAN. Figures A.2 through A.5 show a selection of training runs for a GAN where the softmax layer is passed directly to the discriminator with different hyper-parameters. η_G and η_D denote the learning rates of the Generator and of the Discriminator, respectively: the error of the discriminator becomes near-zero, the generator cannot optimize its loss function and the model's mode collapses immediately.

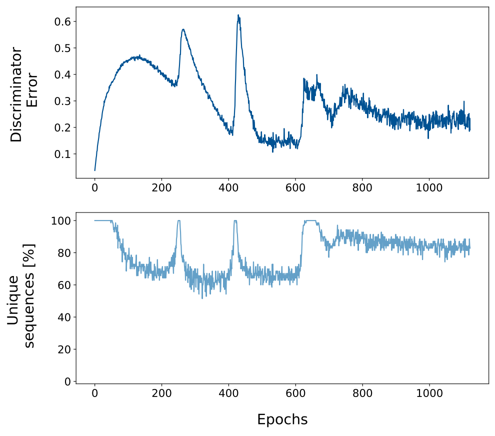


Figure A.1.: GS-GAN $\eta_{D,G} = 5e - 4$

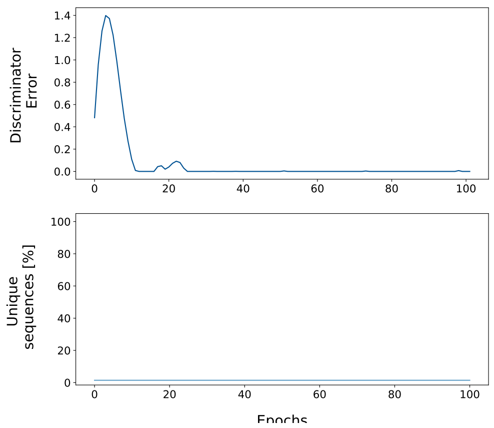


Figure A.2.: $\eta_{D,G} = 1e - 2$

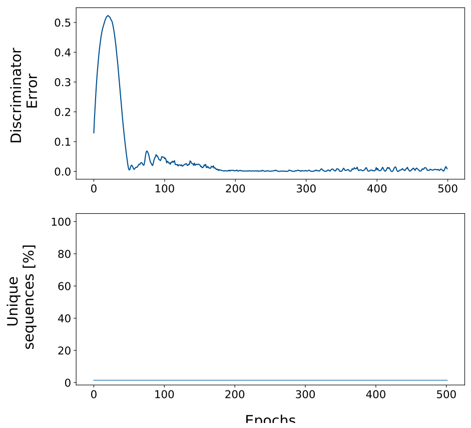


Figure A.3.: $\eta_{D,G} = 1e - 3$

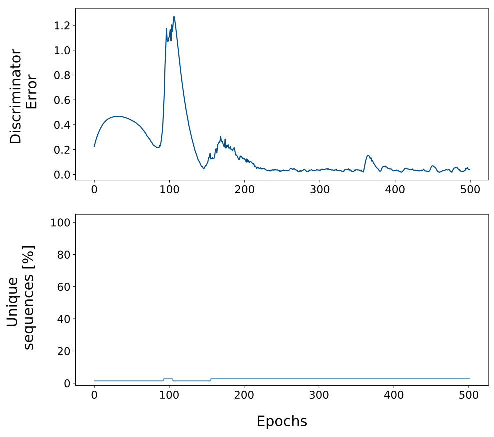


Figure A.4.: $\eta_{D,G} = 5e - 4$

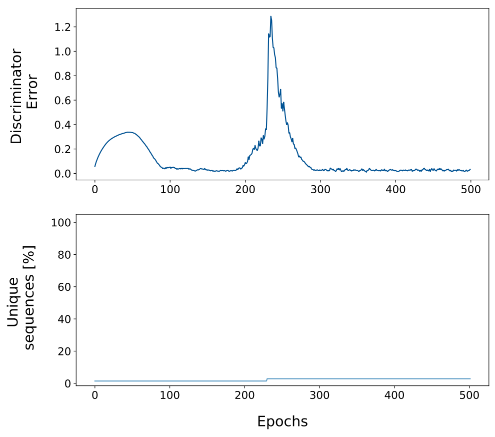


Figure A.5.: $\eta_G = 1e - 2$, $\eta_D = 5e - 4$

B. Gumbel-Softmax GAN with unconstraint logits

Figure B.1 shows a training phase with constrained logits as baseline; Figures B.2 through B.5 show several training phases with unconstrained logits. Each time, the correlation between logit difference, accuracy and mode collapse is clearly recognizable. Note figure B.4 ; although the logits are not constrained, the model does not increase the difference between them: therefore it produces varied sequences, but does not learn the training distribution. Further note, that although the logits are not constrained, the model settles on a defined interval, which varies from run to run.

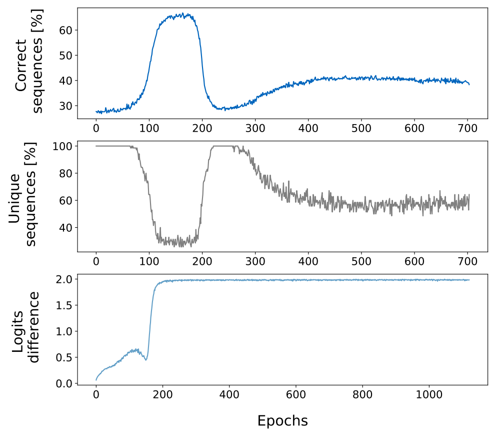
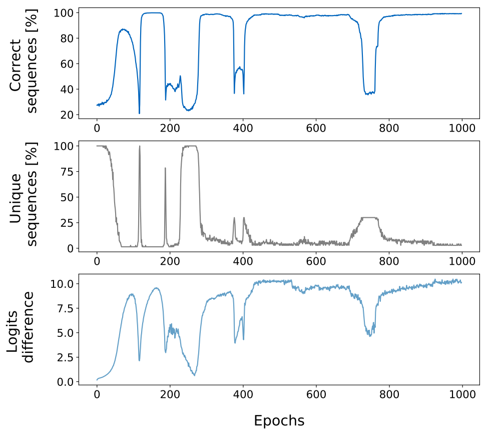


Figure B.1.: Baseline :$\eta_{D,G} = 5e-4$



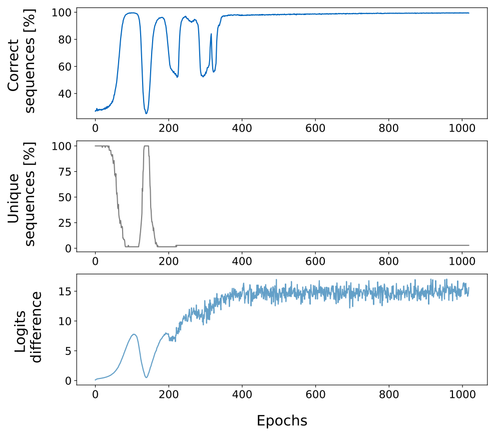
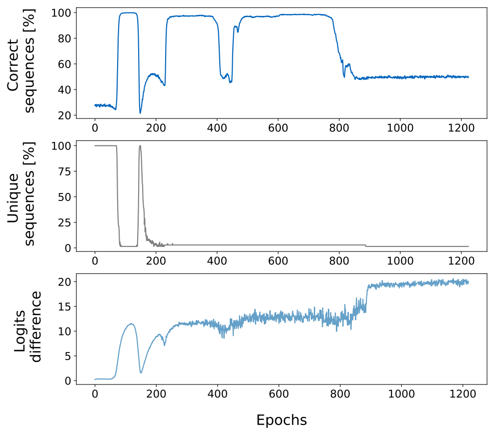


Figure B.2.: $\eta_{D,G} = 1e-4$

Figure B.3.: $\eta_G = 1e-4$, $\eta_D = 5e-3$



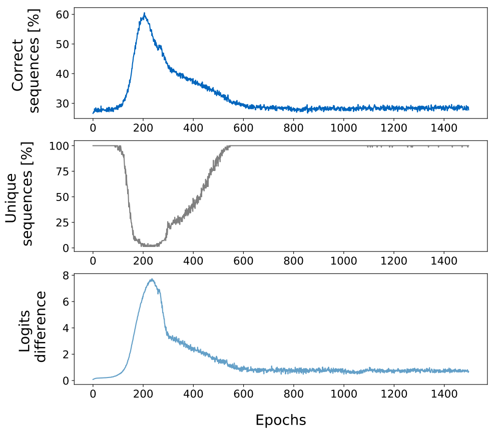


Figure B.4.: $\eta_{D,G} = 5e-4$

Figure B.5.: $\eta_G = 1e-3$, $\eta_D = 5e-4$