POLITECNICO DI MILANO
Corso di Laurea Magistrale in Ingegneria Informatica
Dipartimento di Elettronica Informazione e Bioingegneria

DEEP-FAKE REVIEW DETECTION

Relatore: Prof. Paolo Cremonesi

Tesi di Laurea di:
Giovanni Luca Favuzzi, matricola 898457

Anno Accademico 2019-2020

A tutti coloro che credono e crederanno in me.

Abstract

In the Deep Learning Era it is possible to misuse machine learning techniques to produce audios and videos that can easily deceive humans and machines. This work deals with identifying deep-fake reviews by means of a Variational Autoencoder based on the Attention mechanism. The Variational Autoencoder is trained, validated and tested on three partitions of the Amazon Product Data dataset, then it is compared against the state-of-the-art architecture, the Transformer, on natural language modelling tasks. The Encoder of the Variational Autoencoder produces review embeddings that help disentangle deep-fake reviews from legit ones.

Sommario

Nell'Era del Deep Learning è possibile fare un uso malevolo delle tecniche di Machine Learning per produrre audio e video tali da ingannare sia macchine che umani. Questo lavoro si focalizza sull'identificazione di recensioni deep-fake tramite un Autoencoder Variazionale basato sul meccanismo dell'Attenzione. L'Autoencoder Variazionale è allenato, validato e testato su tree partizioni dell'Amazon Product Dataset ed è paragonato all'architettura dello stato dell'arte Transformer, su moderazione di linguaggio naturale. L'Encoder dell'Autoencoder Variazionale rende possibile separare recensioni deep-fake da recensioni reali.

Acknowledgments

I'm thanking Prof. Paolo Cremonesi for his continuous assistance during the whole thesis and I'm thanking Ing. Massimo Quadrana for his advises in times of uncertainty. I'm thanking Politecnico di Milano for teaching me and for the opportunity of presenting this work.

I'm thanking my parents and my brother. I'm thanking my relatives and my closest friends.

Contents

Chapter 1

Introduction

1.1 General overview

Generative Deep Neural Networks (DNNs) can produce humanlike artifacts starting from big chunks of data. These artifacts take the name of **deep-fakes**. Deep-fakes can follow many different forms, depending on the data the DNN is trained on: they can produce not only images but also video and audio of high-quality. Once the DNN is trained and validated, it is possible to mass-produce deep-fakes which can deceive both humans and machines. This is problematic for many reasons: for instance, recommender systems have a major role in personalizing the user experience and, to achieve the best results, sentiment analyses can really boost the performances; in this case, deep fakes can possibly sabotage any unprotected recommender system by overflowing it with biased reviews, hijacking its recommendations towards some arbitrarily chosen product. Deep fakes have been gathering also the media attention due to their nasty applications over porn, politics and generally misinformation: there are already many deep-learning-based editing softwares that can generate arbitrarily-directed video/audio of any famous public figures, given that enough images/videos of that person are retrievable online. Since deep-fake mass production can't be countered using only human inspection, it is necessary to automate deep-fake detection.

This work deals with identifying deep-fake reviews by means of a **Variational Autoencoder (VAE)** based on the **Attention** mechanism. The VAE is trained on a corpus of reviews and it is compared to the state-of-the-art architecture for Language Modelling: the **Transformer** Decoder as a way to test its competitiveness. The VAE is less perplexed than the Trans-

former in modelling language, therefore it can generate more humanlike reviews using its Decoder. Instead, the VAE Encoder is employed to detect deep-fake reviews: the Encoder produces review embeddings that help disentangle deep-fake reviews from legit ones in the embedding (or latent) space. Since the embeddings of the two types of reviews lay on different parts of the embedding space, a non-parametric clustering algorithm like DBSCAN is enough to label the embeddings as deep-fake or legit. The stronger the VAE, the better the disentanglement.

1.2 Short work description

Considering that the Variational Autoencoder is used as a language model, it is mandatory to compare it to the state-of-the-art in Language Modelling, which is the Transformer. In its original form the Transformer is a sequence to sequence (seq2seq) model composed of an Encoder and a Decoder primarily based on the Attention mechanism. The architecture is employed mostly for natural language translation, however, the Transformer Decoder by itself can deal with language modelling tasks. When talking about language modelling, the Transformer is actually a Transformer Decoder. The Transformer-based state-of-the-art architectures for language modelling, that arose lately, are multi-billion parameters models that are impossible to train without several powerful GPUs or TPUs. Of these powerful language models there are only a few instances, such as GPT-2 (Radford, 2018), which can be considered as the first of them. For this reason there hasn't been as much work on unsupervised deep-fake detection on text as in more traditional machine learning tasks. The only attempts involve Autoencoders, such as in (Nguyen, Nguyen, Tien Nguyen, Thanh Nguyen, & Nahavandi, 2019).

The Transformer-based architectures suffer from the exposure bias issue, which degrades the performances during inference. On the other hand, generative models such as Variational Autoencoders don't suffer from this issue, thus they can perform better. Even so, VAEs haven't got much attention lately due to the Transformer success and up to now there is no study on attention-based VAEs. This work dives deep into this matter, providing a competitive architecture for Variational Attention-based Autoencoders, showing the language modelling performances that can be gained using limited resources.

The VAE is trained, validated and tested on three different partitions of the Amazon Product Data (ADP) dataset (He & McAuley, 2016). The three partitions are sampled randomly from the full ADP using for train, devel-

opment and test sets respectively the percentages $90 - 5 - 5$. The VAE is hyperparameter tuned on the dev-set. The tuning stage consists of trying out different hyperparameter values using a variant of the grid search: first, a grid of values is chosen considering the limited computational resources; second, one hyperparameter at the time, the model is validated on the grid values associated to that hyperparameter (while the others are kept fixed) and the value that gives the best score (Perplexity) on the development set is considered as best; once this process is performed on all the hyperparameters, the process is performed once again on the grid values that are just before or after the chosen ones. If there is 3% or more improvement in the development test scores, the process is repeated again. Finally, the VAE is trained on the train-set using the Adam optimizer. Next, the Transformer is hyperparameter tuned on the dev-set and it is trained using the Adam optimizer. The two architectures are compared on language modelling tasks using Perplexity on the test-set as metric. In order to perform deep-fake review detection, the VAE Encoder produces review embeddings, which help disentangle deep-fake reviews from legit reviews in the embedding (or latent) space. The embeddings of the legit reviews help train a Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm. The training consists of computing the core-points. To detect deep-fakes, the Encoder embeddings of the deep-fake reviews are fed to the DBSCAN algorithm that marks them as outliers (fake samples) if they have a distance which is bigger than ϵ from every core-point, otherwise they are considered as inliers (legit samples). The performances of this deep-fake detection algorithm are tested using Accuracy, Precision, Recall and F1 on three different datasets made of 1024 legit reviews drawn from the test-set and 1024 adversarial reviews generated from: the VAE, the Transformer using temperature annealing, the Transformer using top-k ranking, all of them trained on 10% of the train-set. To provide a better understanding of the several parts that compose the VAE, the latter is subject of an Ablation Study, which helps provide reliable knowledge. The study gives an insight on the benefits that each piece of the VAE has on the training convergence and on the dev-set performances.

For this work there are several interesting future perspectives which consist of: weakening either Encoder or Decoder to boost the performances, incorporating the Transformer-XL mechanism to remove the fixed-context issue, incorporating the BERT objective function to boost the performances.

1.3 Thesis structure

This work is divided into 7 Chapters:

- **Chapter 2** presents the state-of-the-art for Language Modelling and Anomaly Detection. This part explains the successful mechanisms and point out the underlying issues.
- **Chapter 3** summarizes the main contributions of this work and delivers the basic concepts needed to proceed.
- **Chapter 4** explains the motivations under the main idea and the logical process behind the implementation of the idea itself.
- **Chapter 5** dives deep into the implementation details for both the VAE and the deep-fake detection algorithm.
- **Chapter 6** clarifies the experimental set-ups and shows the results obtained, providing some interpretation on why they occurred.
- **Chapter 7** wraps up everything exposing the conclusions and offering some future perspectives.

Chapter 2

State of the art

This section illustrates the state-of-the art techniques up-to-date for both Language Modelling and Anomaly Detection (AD) algorithms.

2.1 Language Modelling

Statistical language modelling refers to drawing a probability distribution over sequences of words, given a certain vocabulary. Considering a sequence $s = [w_1 \dots w_n]$, a language model can output the probability of that sequence belonging to the language L taken into consideration: $P_L(s) = P_L(w_1 \dots w_n)$.

2.1.1 Recurrent Neural Networks

Prior to 2017, Language Modelling hasn't been that successful compared to traditional machine learning tasks (e.g. image classification). Recurrent Neural Networks (RNN) based on Long Short Term Memories (LSTM) or Gated Recurrent Units (GRU) have been the state of the art for most of the time, however, it has been empirically demonstrated that LSTM aren't capable of modelling very long sequences and are rather hard to train due to vanishing/exploding gradient issues (Pascanu, Mikolov, & Bengio, 2013).

2.1.2 Transformer

In the past few years, text generation has been partially achieved thanks to the introduction of a new mechanism named **Attention**. The revolutionary paper in (Vaswani et al., 2017) shows how self-attention can model long-range dependencies on text. The new architecture revolving around the

attention mechanism is named **Transformer** and it is the up-to-date state-of-the-art in Language Modelling. The original architecture of the Transformer is a sequence to sequence (Seq2Seq) model (Sutskever, Vinyals, & Le, 2014) based on attention. More precisely, the Transformer is an Encoder-Decoder model that transforms a sequence into another. The encoder maps an input sequence to a hidden vector which absorbs the relevant information. The decoder maps the hidden vector to an output sequence. This architecture is extremely flexible since input and output sequences can be different: considering machine translation, the sequences can have different lengths; considering image-text matching, the input sequence is a sequence of pixels and the output sequence is a sequence of words. The Transformer employs self-attention in the encoder and both self and normal attention in the decoder. Figure 2.1 shows the in-depth architecture.

Transformer Variants

Considering its success, even though the original Transformer is designed to perform machine language translation, it doesn't take much to redirect it for Language Modelling. In facts, its Decoder is enough to perform this task, given the right objective function. The most successful architectures are GPT-2 (Radford et al., 2018), Megatron-LM (Shoeybi et al., 2019) and Transformer-XL (Dai et al., 2019); all of them are based on the Transformer Decoder. GPT-2 is a (slightly altered) large Transformer Decoder that models language non-recurrently, meaning it can model sentences of fixed-length only. It has 1.5 billion parameters and is trained over 40 GB of text. Megatron-LM is just a stronger GPT with 4.3 times more parameters; Transformer-XL (a.k.a. XLNet) is an autoregressive version of a Transformer Decoder on segment level (a segment is a slice of a sentence) therefore it eliminates the fixed sentence length issue. Since Transformer-XL is slightly worst than Megatron-LM, besides having 32.3 times less parameters, and being better than the GPT-2 full implementation, besides having 5.8 times less parameters, Transformer-XL can be considered a breakthrough in the field. BERT (Devlin, Chang, Lee, & Toutanova, 2018) must be mentioned as well since it is has a peculiar objective function and excellent performances: the LM is trained by guessing randomly hidden words in a sentence (thus not just sequential guessing). This peculiar objective function, however, makes generating text harder than usual since sentences are produced sequentially from left to right.

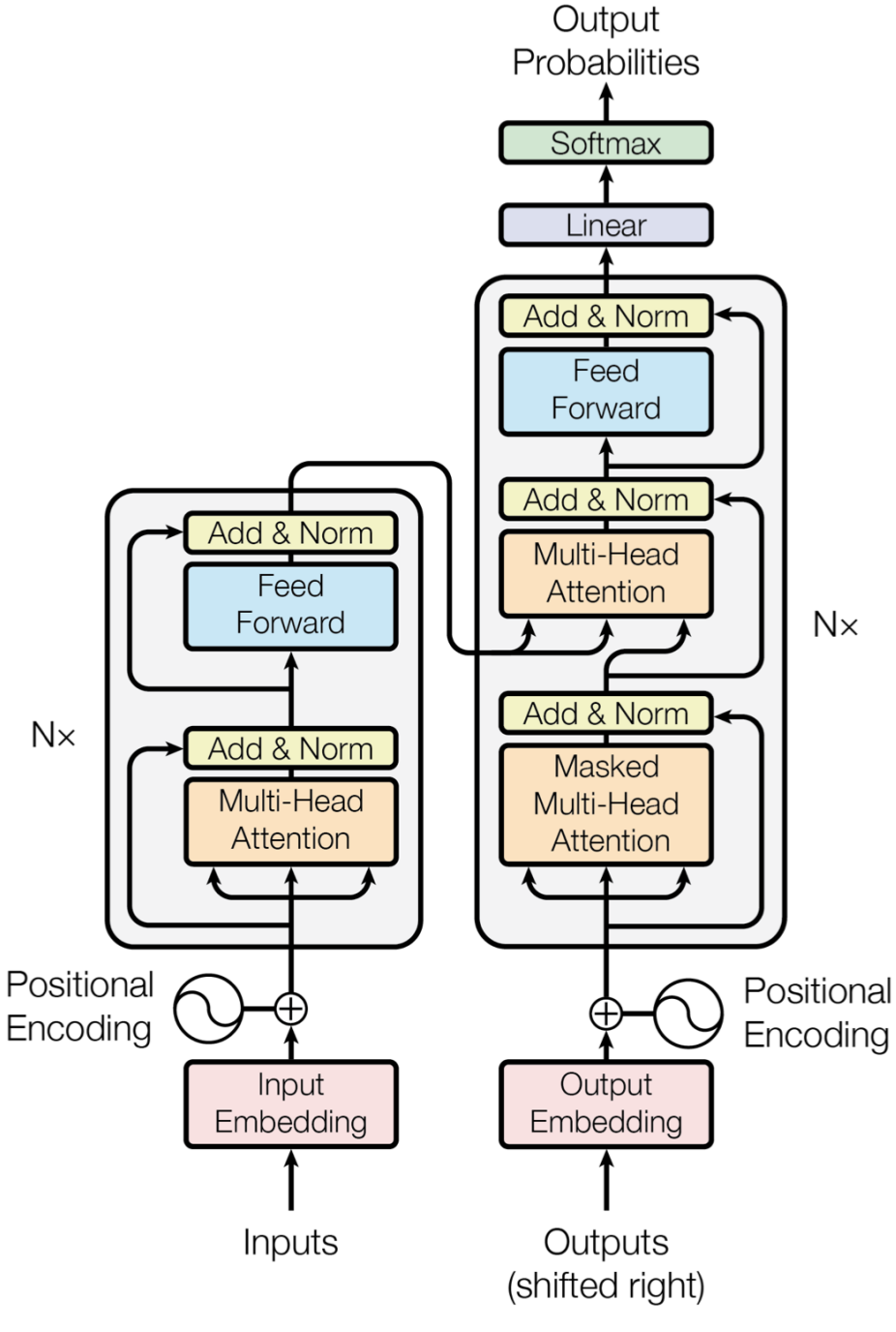

Figure 2.1: Transformer Architecture

2.1.3 Generative Models

Deep Generative Models adapted for modelling language can produce text as well as alter existing content as believably as possible.

Generative Adversarial Networks

Generative Adversarial Networks (GANs) are neural-network-based techniques that have been successful in the recent years (Goodfellow et al., 2014). GANs contain a generative neural net (Generator) and a discriminative neural net (Discriminator): the two nets take part in the adversarial framework, which consists of the Generator generating fake samples and the Discriminator discriminating the fake samples from the real ones. The more fake samples the discriminator believes as legit, the better it is for the generator. Ideally, at convergence time, the generator should be able to output realistic samples and the discriminator should be confident enough to discriminate the suspicious fake samples from the real ones. Even though GANs have had most of their success for computer vision tasks, the recent years have seen some work in the Natural Language Processing area too: (Fedus, Goodfellow, & Dai, 2018), (Press, Bar, Bogin, Berant, & Wolf, 2017).

Variational Autoencoders

Variational Autoencoders (VAEs) embody another powerful class of generative models. They consist of an encoding neural net (Encoder), which embeds the input samples into latent embeddings, and a decoding neural net (Decoder), which reconstructs the latent embeddings and brings them back into the data space. The state of the art for language modelling VAEs employs RNNs for either Decoder or Encoder or both. VAEs are trained using a combined objective function that involves a reconstruction loss and a regularization loss. There are several successful VAEs architectures that can model language, from more traditional RNN-based VAEs as in (Bowman et al., 2015), to hybrid VAEs consisting of RNN encoder and CNN decoder as in (Semeniuta, Severyn, & Barth, 2017).

2.1.4 Issues

Still, these language models do not solve some fundamental issues.

Transformer

Non-generative Language Modelling techniques, such as the Transformer, suffer from **exposure bias**. It refers to the train-test discrepancy that seemingly arises when an autoregressive generative model uses only ground-truth contexts at training time and uses generated contexts at test time (Schmidt, 2019). Generally, exposure bias appears when autoregressive models are trained with the Teacher forcing technique, which consists of passing the target word as the next input to the model. This way, at inference time, when the model needs to rely on its own generations, if an error occurs, it is likely to cause an error cascade making errors stack on top of each other. Non-generative Language Modelling have a second problem, it is more of a pre-condition for high-quality text generation and is the need for **context**. All these language models produce more robust generations when fed with a high amount of high-quality context: the better the context, the better the generation, since the less exposure bias is to occur. This becomes an issue when providing enough context is problematic and polarizing the text on a certain argument and/or sentiment is necessary.

GAN

GANs have theoretically no issues, but in practise they are extremely hard to train. The main issues can be summarized as:

- **Non convergence** arises when the parameters of generator and discriminator oscillate and/or destabilize ending up never converging.
- **Mode collapse** happens in the generator when its limited latent space "pushes" out the modes that are harder to reproduce, ending up reproducing only few of them.
- **Diminished gradient** happens in the discriminator when it gets too successful, making the generator gradient vanish.

Even though there is plenty of techniques that help contrast these issues, when dealing with text, GANs are still hard to train, hence they haven't yet had much success compared to the Transformer achievements.

VAE

Despite VAE performant results, the state of the art architectures suffer from a major problem that makes training harder: the **regularization overpowering effect**. Essentially, the regularization loss is the easiest

term to minimize in the objective function, but, such minimization comes at the price of an increment in the reconstruction loss. As training goes on, the model parameters get stuck in a region of the parameter space that causes the regularization loss to anneal to zero and the reconstruction loss to get bigger. The only way to lessen this hurdle is to either start with small weight of the regularization term (nearly zero) and anneal it to 1; otherwise it is possible to impose a minimum value of the regularization term, so that having a smaller value of the term doesn't affect the overall loss (because the minimum is reached). Both these techniques come at a cost since both the regularization term weight annealing curve and the minimum regularization term value become hyperparameters that need to be tuned. The second technique, which involves imposing a minimum value, is definitely the worst of the two since it still allows the model to go down the parameter region where the reconstruction loss is high and it just doesn't allow the model to go further. However, the first technique, involving annealing, requires much more tuning since it is harder to pick the right annealing curve with the right start and ending values.

2.2 Anomaly Detection

By definition, Anomaly Detection (AD) is the mechanism by which an intelligent entity is able to identify an incoming sensory pattern as being hitherto unknown (E. N. Sokholov, 2020). Basically, AD copes with identifying items that differ significantly from the population and that are problematic (e.g. Fraud Detection). These items are distributed in patterns that don't adhere to the statistical properties of the normal population. There are three main AD categories:

- Supervised: the items are labelled (Kawachi, Koizumi, & Harada, 2018).
- Unsupervised: the items are not labelled and the outliers are usually absent from the training set (Chen, Sathe, Aggarwal, & Turaga, 2017).
- Semi-supervised: the dataset is not fully labelled, thus the problem is mainly unsupervised but the performances benefit from labelled items (Ergen, Hassan Mirza, & Kozat, 2017).

2.2.1 Supervised

Supervised AD is a simpler but rarer task since usually most of the outliers are different from one-another and don't belong to a single class: thus having

a dataset that contains all the outlying classes is unusual. This problem is tackled with a standard multi-class classifier.

2.2.2 Unsupervised

Unsupervised AD is much more common: without knowing the outlying classes, it is better to white-list than it is to black-list items. For this reason it is prime to learn the behavior of the normal items to discriminate the abnormal ones as outliers. One-class classifiers are the popular way-to-go. One-Class Support Vector Machines (OC-SVMs) are popular for this task ("Robust outlier detection using SVM regression", 2004), as well as One-Class Neural Networks (OC-NNs). Both of them use similar objective functions (Chalapathy, Menon, & Chawla, 2018). These techniques can be applied directly on the data space but, this way, the task gets harder. Typically, it is easier to train an Autoencoder and then perform the one-class techniques on the Encoder embedding (or latent) space. The task gets easier if the AE Encoder produces meaningful embeddings that encapsulate the useful information. This way, the embedding space can also provide data separation, therefore, easy clustering.

2.2.3 Issues

Even though OC-SVMs are the state-of-the-art for many problems, they are not faultless: in this work SVMs have random-like performances. OC-NNs represent a powerful technique but they require an increasing amount of parameters and training as the distribution of the data becomes more complex. This becomes problematic when only limited resources are available.

Chapter 3

Research problem setting

The following chapter defines the contributions of this work and defines the basic concepts needed to understand the techniques employed lately.

3.1 Contributions

This work is meant to provide a novel VAE architecture that solves the preexisting issues and provides powerful review embeddings that help disentangle deep-fake reviews from legit ones. The actual contributions are:

1. Introduction of a novel competitive architecture for Language Modelling: Variational Autoencoder based on self-attention. It is the first language modelling AE architecture that doesn't employ RNNs in both encoder and decoder. Furthermore, the architecture is much less affected by the regularization term overpowering effect that affects the previous architectures.

2. Unsupervised Anomaly Detection on deep-fake reviews, generated by adversarial models by means of a non-parameter-intensive technique: DBSCAN applied on the VAE Encoder embeddings.

3.2 Basic concepts

This section provides the basic concepts needed to understand the core concepts of this work that are presented later.

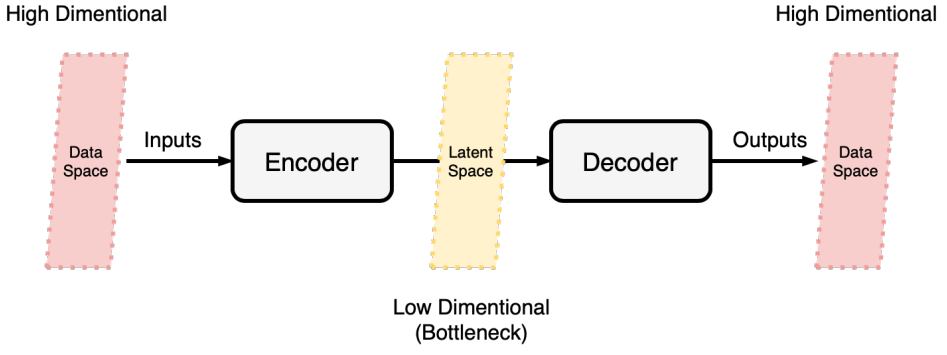

Figure 3.1: Autoencoder Architecture

3.2.1 Autoencoder

The **Autoencoder** (**AE**) is a fundamental deep learning architecture: it is made of an Encoder attached to a Decoder and, in between the two, there is a bottleneck. The bottleneck layer generates a multidimensional space (lower dimensional w.r.t. the data space) where the embeddings (or latent codes) lay on. Figure 3.1 shows the general scheme.

The flow is straightforward: the Encoder maps the input data into a (low-dimensional) latent space, then the Decoder maps the latent items back to the (high-dimensional) data space. Basically, the input samples are encoded and then decoded/reconstructed. The better the reconstruction, the lower the loss. The objective function can be any appropriate reconstruction loss (e.g. Mean Squared Error for images and Negative Log Likelihood for text).

The AE Decoder can be conceived as a sort of Generator that generates new samples starting from a random input vector. Still, AEs by themselves cannot generate new meaningful samples: usually, the hidden prior has an unusual distribution, due to efficiency reasons, that makes it hard to sample from. To generate properly, AEs need to be regularized. There are two main approaches to AE regularization that fork into two different AE sub-categories:

- Variational Autoencoder (VAE): Imposes a well-known distribution as prior (e.g. Normal Standard, Mixture of Gaussians) (Kingma, Mohamed, Jimenez Rezende, & Welling, 2014).

- Adversarial Autoencoder (AAE): Uses a Discriminator to regularize the prior (Makhzani, Shlens, Jaitly, Goodfellow, & Frey, 2015).

Variational Autoencoder

As already mentioned, the VAE forces the latent distribution to be a well known distribution, usually a Gaussian with zero mean and unit variance. This constrain can be imposed by making the Encoder output a distribution instead of a single point. Yet, sampling can't just disrupt the differentiability of the system (since back-propagation is needed) and simply sampling from a distribution is a non-differentiable operation. Fortunately, by using the re-parametrization trick in Equation 3.1, it is possible to sample from a Gaussian distribution keeping the differentiability intact:

$$X = \mu + \epsilon \times \sigma \tag{3.1}$$

$$\sigma = e^{logvar/2} \tag{3.2}$$

$$\epsilon \in N(0, 1) \tag{3.3}$$

The Encoder is trained to generate both bias μ and logarithmic variance $logvar = \log(\sigma^2)$. Then the VAE computes the sample X using these values as in Equation3.1. Yielding a distribution instead of a single point is not enough: the VAE can just anneal the variance of the distribution to zero, reducing the distribution to a mere point. For this reason, the regularization term in Equation 3.4 is mandatory.

$$L_{VAE} = L_{reconstruct}(X|Z) + L_{regularize}(Z|X) \tag{3.4}$$

Here, Z represents samples coming from the latent space distributions. Having a dense convex latent space helps sampling latent points that are associated to well-formed sentences. In few words, given two points in the latent space, which correspond to two different real sentences, each point of the linear interpolation between those two should be decoded into a meaningful sentence.

Usually, the imposed prior is a Normal Standard distribution, however it is not the only well-known distribution that can be imposed as prior. In facts, the Mixture of Gaussians (MoG) is employed in many works as well (Semeniuta et al., 2017). It requires a few tweaks to the architecture of the VAE, but it helps clustering the Encoder embeddings into different parts of the latent space, hence its popularity. For this work the MoG is not employed since it doesn't perform well: the MoG collapses to a Gaussian

with zero mean and unit variance, meaning that the Normal Standard is a better fit to the underlying latent distribution.

Adversarial Autoencoder

The Adversarial Autoencoder (AAE) is a probabilistic AE that resorts to the same adversarial framework of GANs to perform variational inference by matching the aggregated posterior of the latent code vector of the AE with an arbitrary prior distribution. This framework ensures that generating from any part of the prior results in meaningful samples. Additionally, the AAE Discriminator can output a confidence level $Dis(z) \in [0, 1]$ that indicates the fakeness of input x. It can be easily employed for Anomaly Detection. The AAE is made of an Encoder, a Decoder and a Discriminator, distributed as in Figure 3.2. The AAE training workflow requires the forward propagation first, then the following steps are performed sequentially:

1. Encoder and Decoder are updated using the backward propagation computed using the reconstruction loss in Equation 3.5.
2. Encoder and Discriminator are updated using the backward propagation computes using the adversarial loss in Equation 3.6.

$$L_{rec} = -E_{x \sim P_x}\Big[\log\big(Dec(Enc(x))\big)\Big] \tag{3.5}$$

$$L_{adv} = \min_{Enc}\max_{Dis} E_{z \sim P_z}\Big[\log Dis(z)\Big] + E_{x \sim P_x}\Big[\log\big(1 - Dis(Enc(x))\big)\Big] \tag{3.6}$$

The AAE is a powerful technique, but it requires more parameters and more training than the VAE: 1 forward propagation and 2 backward propagations are needed. This gets even more problematic since empirically Autoencoder and Discriminator need m and n straining steps respectively. Generally $m \neq n$ and $n > 1$ *or* $m > 1$ and they are considered as hyperparameters. For this work, the adversarial framework introduces high-significance hyperparameters, extends the training time needed and doesn't provide a robust performance enhancement. For these reasons, it is not the right architecture choice.

3.2.2 Attention

As already mentioned, the VAE is based on the Attention mechanism. This mechanism is deemed to be one of the most powerful in Deep Learning and

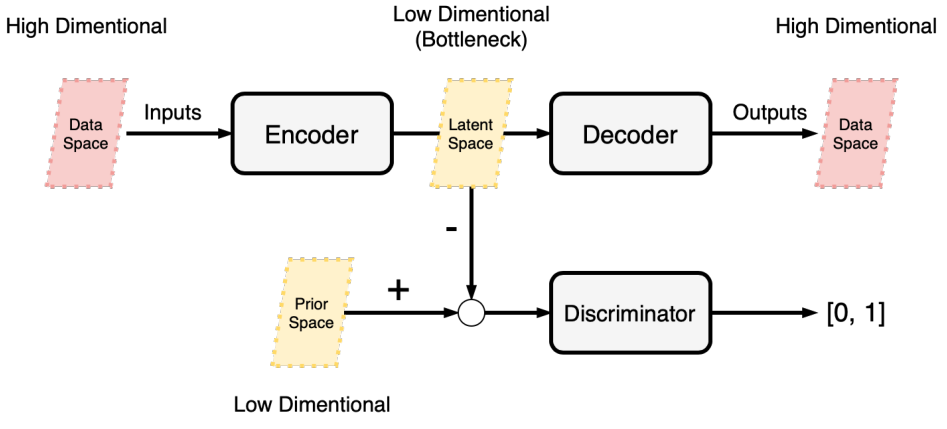

Figure 3.2: Adversarial Autoencoder Architecture

it is currently holder of the state of the art for many Natural Language Processing (NLP) and Computer Vision (CV) tasks. Generalizing, the attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values and output are all vectors. The output is then computed as a weighted sum of the values, where the weight assigns to each value is computed by a compatibility function of the query with the corresponding key (Vaswani et al., 2017). Equation 3.7 summarizes the above.

$$attention(x, y_t) = \sum_{i=1}^{n} \alpha_{t,i} v_i(x) \tag{3.7}$$

$$\alpha_{t,i} = align(x_i, y_t) = softmax\Big(score\big(k(x_i), q(y_t)\big)\Big) \tag{3.8}$$

Queries and keys have dimension d_k, while values have dimension d_v. Considering a source sequence $x = [x_1 ... x_n]$ and a target sequence $y = [y_1 ... y_m]$, the source sequence forms both keys and values, thus x is projected onto those dimensions, as in Equations 3.9, 3.10, instead, the target sequence y is projected onto the query space as in Equation 3.11.

$$v(x) = v = W_v^T \cdot x \tag{3.9}$$

$$k(x) = k = W_k^T \cdot x \tag{3.10}$$

$$q(y) = q = W_q^T \cdot y \tag{3.11}$$

Name	Score Function
Content Based	$score(k, q) = \cos(k, q)$
Additive	$score(k, q) = M_A^* \tanh(W_A^{T*} \cdot [k; q])$
Location-Based	$align(x_i, y_t) = softmax(W_A^* \cdot k(y_t))$
General	$score(k, q) = k^T \cdot W_A^* \cdot q$
Dot-Product	$score(k, q) = k^T \cdot q$
Scaled Dot-Product	$score(k, q) = (k^T \cdot q)/\sqrt{d_k}$

Table 3.1: Attention types

Attention Variants

Since its conception, have been developed several sub-types of attention that differ mainly on how to compute the *score* function. Table 3.1 presents all the attention types, where $(*)$ means: matrices to be learned. The attention types come from:

- Content Based: (Graves, Wayne, & Danihelka, 2014)
- Additive: (Bahdanau, Cho, & Bengio, 2014)
- Location-Based: (Chorowski, Bahdanau, Serdyuk, Cho, & Bengio, 2015)
- General: (Luong, Pham, & Manning, 2015)
- Dot-Product: (Luong et al., 2015)
- **Scaled Dot-Product**: (Vaswani et al., 2017)

The Scaled Dot-Product attention is what is used in this work, due to its positive results in the state of the art, and from now on it is considered as the default one.

Packing together queries, keys and values into matrices Q, K and V respectively, attention can be expressed in matricial form as in Equation 3.12.

$$attention(Q, K, V) = softmax\Big(\frac{Q \cdot K^T}{\sqrt{d_k}}\Big) \cdot V \tag{3.12}$$

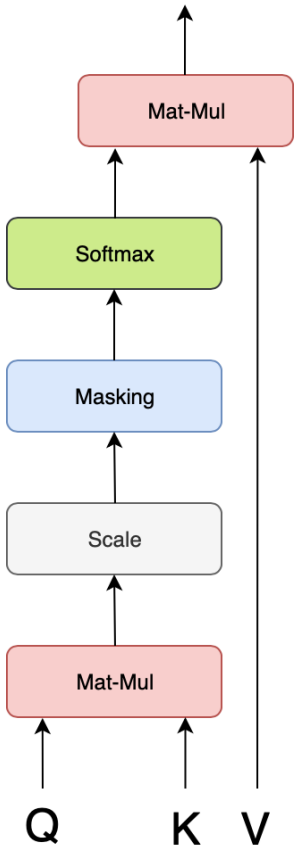

Figure 3.3: Scaled Dot Product Attention

The overall workflow is visually represented in Figure 3.3.

Self-Attention

Self-attention can be defined as in Equation 3.13. It is the fundamental building block of the VAE architecture and models the dependencies between the sequence of words.

$$self\_attention(y_t|y) = attention(y, y_t); \quad y_t \in y \tag{3.13}$$

Self-attention can be either global or local: considering sentences, it can either be applied to the whole sentence (global) or at segment level (local), meaning every $segment\_length$ words. For the sake of clearness, considering an input sentence $s = [w_1 \dots w_n]$, self-attention models how each word w_k is related to the other words $w_{\tilde{k}}$, using real numbers. The higher the value, the higher the correlation. Figure 3.4 offers the visualization of an example from (Vaswani et al., 2017).

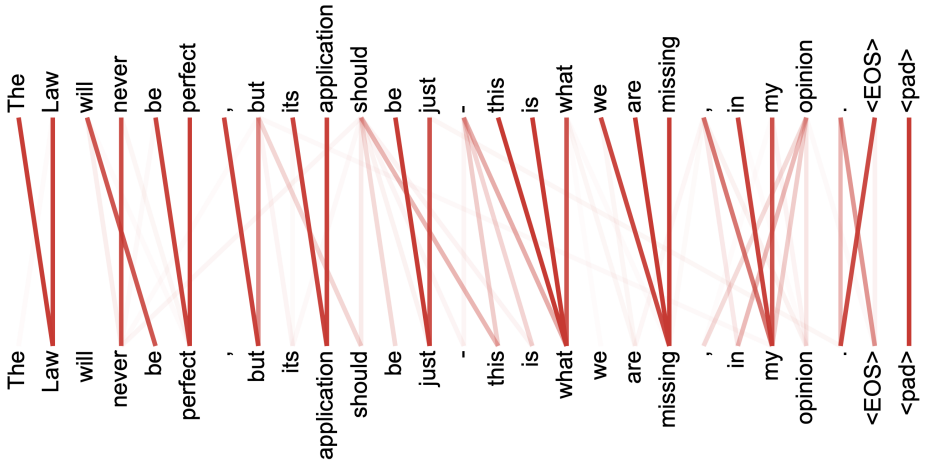

Figure 3.4: Self-Attention Visualization

Mechanism	Layer Complexity	Operations	Max Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolution	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(\log_k(n))$

Table 3.2: Complexity Comparison

From a complexity point of view, self-attention is more attractive than the older mechanisms: RNNs and convolutions. Table 3.2 offers the complexity comparison.

Multi-Head-Attention

Instead of using a single attention function with d_k, d_v, d_k dimensional keys, values and queries respectively, it is beneficial to linearly project the queries, keys and values h times with different, learned linear projections of dimensions d_k, d_v, d_k respectively. On each of these projected versions of queries, keys and values we then perform the attention function in parallel, yielding d_v-dimensional output values. These are concatenated and once again projected, resulting in the final values as in Equation 3.14 (Vaswani et al., 2017). This is interpretable as some sort of ensembling between multiple attentions models. Basically, Multi-Head-Attention (MHA) allows the model to jointly attend to information from different representation subspaces at different positions. With a single attention head, averaging inhibits

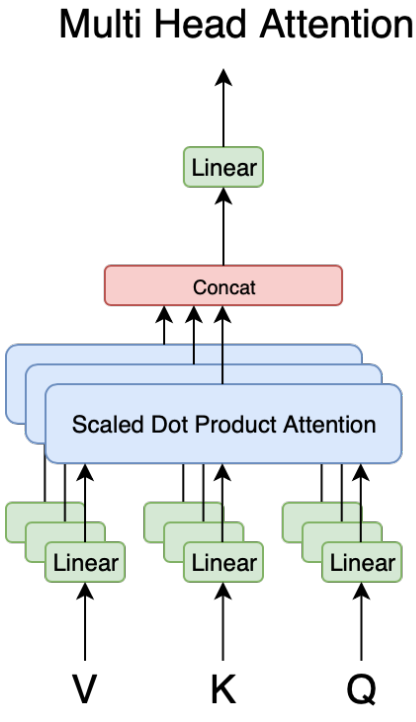

Figure 3.5: Multi Head Attention

this.

$$MHA(Q, K, V) = Concat(head_1 \dots head_h) \cdot W^O \tag{3.14}$$

$$head_i = attention(Q \cdot W_i^Q, K \cdot W_i^K, V \cdot W_i^V) \tag{3.15}$$

The matrices in Equation 3.15 have dimensions: $W_i^Q \in \mathbb{R}^{emb \times d_k}$, $W_i^K \in \mathbb{R}^{emb \times d_k}$, $W_i^V \in \mathbb{R}^{emb \times d_v}$, since they need to map the words from the (emb-dimensional) embedding space to either query, key or value space. Moreover, $W^O \in \mathbb{R}^{d_v h \times emb}$. The above is visually represented in Figure 3.5.

3.2.3 Optimizations

The VAE architecture employs two important architectural enhancements: residual connections and normalization on the layers. These techniques boost the convergence of the model by helping with the vanishing/exploding gradient issue and stabilizing the gradients, respectively.

Residual Connections

Deep neural networks are powerful mathematical models but known to be harder to optimize as the parameter count and/or depth of the model grows. Lately, neural nets got deeper and deeper, therefore, harder to optimize. To cope with these issues, Residual Connection are introduced in (He, Zhang, Ren, & Sun, 2015) as way of making training easier. Neural nets based on Residual Connections are named ResNets. Residual connections are identity connections that are added to the output of a certain layer. If the connections are weighted (instead of simple identities) they are named Highway connections. The connections must be weighted if the dimension of the output of the previously-skipped layer are different from the dimension of the input of the downstream layer. During training, the weights adapt to mute the upstream layer, and amplify the previously-skipped layer. Learning this might take some iterations.

Residual connections speed up the convergence by reducing the impact of vanishing/exploding gradients, as there are fewer layers to propagate through. The network then gradually restores the skipped layers as it learns the feature space. A neural network without residual connections explores more of the feature space, which makes it more vulnerable to perturbations and necessitates extra training data to recover (ResNet, 2020).

Layer Normalization

As already mentioned, neural nets are hard to optimize, however there are several techniques that make things easier. Normalization is one of them: it makes the model invariant to dataset re-scaling. There are several types of Normalization, as shown in Figure 3.6 for images.

- **Batch Normalization**: normalization along the batch dimension.
- **Group Normalization**: normalization separately for each group of channels.
  - **Layer Normalization**: normalization over all the channels (maximum group possible).
  - **Instance Normalization**: normalization over each single channel (minimum group possible).

Batch Normalization is the first normalization technique to be developed and it is introduced in (Ioffe & Szegedy, 2015). Given a mini-batch of M samples, it normalizes the mini-batch by computing its mean and variance.

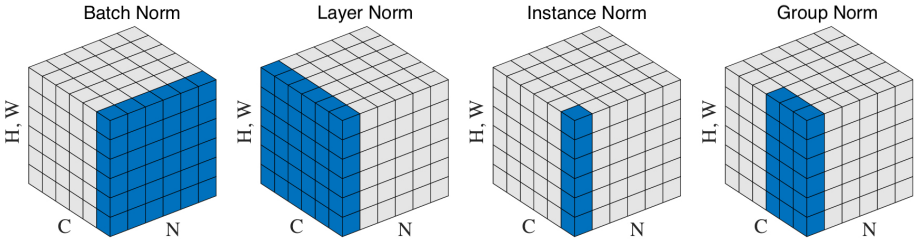

Figure 3.6: Normalization Variants

It is supposed to reduce the covariate-shift problem. Batch normalization has an issue though: when the batch size is not a good approximation of the full dataset, mean and variance get too inaccurate. This usually happens when the mini-batch is too small and it causes the performances to deteriorate.

Layer Normalization (Ba, Kiros, & Hinton, 2016) is an sub-type of Group Normalization (Wu & He, 2018) where the group contains all the channels. It is meant to substitute Batch Normalization. It computes the mean and variance (used for normalization) from all the summed inputs to the neurons in a layer on a single training case. This way the normalization is independent on the batch size and makes the model invariant to weight rescaling and shifting. The idea comes from the fact that changes in the output of a certain layer can cause correlated changes in the summed inputs of the next layer. This type of covariate shift can be dealt with by fixing mean and variance of the summed inputs within each layer: that is why the statistics are computed on all the (H) hidden units in that layer (l) as in Equations 3.16, 3.17.

$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l \tag{3.16}$$

$$\sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(a_i^l - \mu^l)^2} \tag{3.17}$$

Applying layer normalization is easy and it has been demonstrated to work well with generative models. For these reasons it is the normalization technique employed for this work.

3.2.4 Nonlinearity

Activation functions ϕ determine what and when neurons burst out $out = \phi(in)$, given an input in. Mathematically, they are (mostly) needed to intro-

duce non-linearity in the network, however, their influence on the training is much more than just that: they can better/worsen gradient vanishing/-exploding, they can help regularize the model and, in general, unexpectedly change the convergence of the model. Early neural networks are designed to work with binary threshold units smoothed as sigmoids due to their probabilistic interpretation, however the past decade has seen the rise of Rectified Linear Units (ReLU) that depend on the sign of the input (Agarap, 2018). Its formulation is in Equation 3.18. Loosing the interpretation is rather unimportant compared to the performance boost that they offer, however, ReLUs can fit the data so well that usually stochastic regularization techniques are needed (e.g. dropout or adding noise in the hidden layers). Recently in (Hendrycks & Gimpel, 2016) is introduced a new type of nonlinearity: GELUs. GELUs are designed to combine the stochasticity of dropout (Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov, 2014) and zoneout (Krueger et al., 2016) whilst keeping the performances of ReLUs. By multiplying the neuron input x by $m \sim \text{Bernoulli}(\Phi(x))$, having $X \sim \mathcal{N}(0,1)$, it is possible to impose stochasticity: Equation 3.19. It is possible to approximate GELU with Equation 3.20.

$$\text{ReLU}(x) = \max(0, x) \tag{3.18}$$

$$\text{GELU}(x) = x\ P(X \leq x) = x\ \Phi(x) \tag{3.19}$$

$$\text{GELU}(x) \simeq 0.5\ x\left(1 + \tanh\left(\sqrt{2/\pi}(x + 0.044715\ x^3)\right)\right) \tag{3.20}$$

Chapter 4

Logical project of the solution of the problem

4.1 Overview

4.1.1 Motivation

Deep-fake detection consists of identifying reviews generated by deep neural networks in a population of reviews. The reviews are not labelled and there is no way to produce a complete labelled dataset since there are infinite ways of modelling language with neural networks. Even a slight hyperparameter variation could lead to significantly different neural language models. Since supervised methods cannot be applied successfully, deep-fake review detection must be dealt unsupervisedly.

Unsupervised deep-fake detection can be tackled with anomaly detection algorithms. These algorithms usually employ Autoencoder-generated features, which are fed to a second algorithm designed for detecting outliers in a population of items. These techniques are usually one-class SVMs or one-class NNs. Both of them are parametric-intensive algorithms, hence they require a non-trivial amount of training, therefore, higher computing power. Moreover, the better the AE features, the better the performances. The AE features are "better" if they encapsulate useful information. Of course, the features possess these properties if the AE is "strong" enough to generalize on them by means of its Encoder. For this reason, having a well-performing AE is prime for solving this task successfully.

4.1.2 Idea

Following the success that the attention mechanism had in the state of the art, this work studies attention-based variational autoencoders: these models can produce powerful enough embeddings to disentangle legit reviews from deep-fake ones in the embedding space. This way, it is enough to employ a non-parametric outlier-detection algorithm, such as a slight variation of DBSCAN, to separate legit reviews from deep-fake ones with good performances. As a disclaimer, these techniques are supposed to work under the hypotheses that deep-fake reviews are non-perfect, meaning that they are either syntactically or semantically incorrect, hence they have slight (or severe) grammatical errors (e.g. this coffee has good) or they are incoherent (e.g. this coffee is bad but it is good).

4.2 Design

4.2.1 Dataset

Language models learn from a large corpus of text. Depending on the dataset, the language model can learn sub-types of language. More specifically, if the dataset contains only reviews, the language model will be able to generate reviews but it won't be able to generate poems and vice-versa. This means that, in general, a dataset containing reviews of a certain language will have a different word-distribution w.r.t. the general word-distribution of that spoken language. It is possible to demonstrate this property using the Zipf's law (Zipf, 2020), by estimating the Zipf exponent on the corpus and comparing it to the empirical Zipf exponent known for that language. Knowing this is mandatory if two language models have to be compared: it would not be fair to compare a language model (e.g. the VAE) tested on a certain corpus and another language model tested (e.g. the Transformer) on another corpus. For this reason, we cannot compare the performances of the Transformer claimed on some papers with the performances of the VAE on the Amazon Product Dataset.

The corpus must be preprocessed before being fed to the neural network: a maximum length max_len of sentences is selected; the length of a sentence is given by the sum of the number of words and punctuation inside it). Next, to each sentence is attached a Start of String (SOS) token to its start and an End of String (EOS) token to its end. Next, for each sentence that is smaller than $max\_len + 2$, a list of padding (PAD) tokens is added to match that length. The data is almost ready to be encoded, however,

when dealing with Natural Language Processing tasks, the practice is to trim the vocabulary to a reasonable number of words, so as to grant a high word-coverage for the corpus and then encode the sentences using 1-hot encoding (One-hot, 2020). This is mandatory since a vocabulary that is too big would lead to high-dimensional 1-hot encodings that cannot be processed by standard GPUs. The dataset is partitioned into three different parts using random sampling: train-set with 90%, dev-set with 5% and test set with 5%. Random sampling is considered a good-enough way of partitioning considering the number of reviews of the full dataset (roughly 2 millions). The dataset is split only into three parts because even a 5-folds cross validation would be too computationally expensive.

4.2.2 Architecture

Given the idea of a neural network, the in-depth design must go through some standard passages. First, as mentioned above, the input has to be encoded in such a way that the neural network can process it and perform computation efficiently. The number of input neurons must match the number of dimensions of the 1-hot encodings, that is the number of words in the vocabulary. Next, is needed an appropriate number of output neurons for the task considered. In this case, since the task is to reconstruct sentences, the number of output neurons is the same as the number of input neurons. Moreover, in order to carry out the training successfully, an appropriate loss function is required. For this work the loss function is the weighted sum of two terms, a reconstruction term and a regularization term. Given a more defined structure of the net, it is time to select the optimizer and the learning rate associated with it. Picking the right optimizer with the right learning rate is vital for the success of the work since the convergence of the network depends mainly on those two. Since deep neural networks are hard to train, the next step is to deal with vanishing/exploding gradient. This issue can be tackled in multiple ways, some better than others: it is a matter of trying out what we think will work and see the effects the adjustments have on the training and overall performances (on the development set). Gradient clipping can cope with this issue, however it introduces high-relevance hyperparameters. Another way of dealing with this is to add residual connections: they can shorten the path between input and output, lessening the gradient vanishing. Activation functions have a major effect on vanishing gradient as well, so they have to be picked carefully. Furthermore, to speed-up the convergence of the model, a suitable normalization technique has to be chosen. As final step, the hyperparameters of the net-

work have to be tuned on the development set: the number of hidden layers and hidden neurons, the batch size, the number of epochs (using early stopping). Scheduling the learning rate can also help reach better performances: lowering the learning rate it is possible to get nearer to a local minimum, however there is the possibility to overdo it, causing overfitting.

4.3 Experimentation

In order to understand whether the new architecture is competitive or not with the state-of-the-art, both quantitative and qualitative experiments are needed. The quantitative experiments help compare the VAE with the Transformer and they provide a number that measures how well one is doing w.r.t. the other. The qualitative experiments don't provide a number but they provide higher-level evidence, such as visualizations, that can be enjoyed by humans. Qualitative experiments are necessary to validate or negate hypotheses that can't be assessed with a mere number.

4.3.1 Tuning

Tuning the hyper-parameters is a long but necessary step for developing a successful model. In few words, this phase consists of trying out several values for the hyperparameters, following a certain strategy, then selecting the best value accordingly to the dev-set performances. The strategy that is employed to pick the hyperparameters value to be tried is a modification of grid search. First, a grid of values is chosen considering the limited computational resources. Second, one hyperparameter at the time, the model is validated on the grid values associated to that hyperparameter (while the others are kept fixed) and the value that gives the best scores on the development set is considered as best; once this first step is completed on all the hyperparameters, the process is performed once again: this time the values to be tried out are those ones that are just before or just after the ones currently considered as best. If there is 3% or more improvement in the development test scores, the process is repeated again until convergence. This technique is sub-optimal, but it helps lowering the computational efforts that would go into trying out all the possible grid-search combinations. The right hyperparameter tunes are chosen by looking at the development-set performances for each differently tuned model.

4.3.2 Language Modelling

Metrics

Natural Language Processing models are known to be hard to evaluate, since it is hard to tell when one model has learnt the language properly. In particular, when dealing with language modelling, it is not enough to consider just the number of words that are mispredicted by the model since there are multiple ways to tell the same concept. Perplexity fulfills exactly such task: it provides a continuous value that measures how "wrong" the model is being at predicting words in a sentence. Even though alone it is not enough, it can be used as loss function, therefore it enables back-propagation in the network. Perplexity is usually paired with human evaluation, therefore this section must provide language modelling example as well.

Generation

Even though quantitative results are important, they are still not enough. There have to be qualitative experiments on generation, that can be inspected by humans directly. The two qualitative experiments on generations that are done in this work are:

- Random generation
- Interpolation generation

Both experiments deal with generating sentences given some points in the latent space, but there is one key difference: how to get the latent points. As the name suggests, random generation generates sentences given latent points randomly sampled from a Gaussian distribution (the imposed prior) having same bias and variance as the latent code distribution (such distribution is computed by encoding true sentences from the dataset). Instead, the interpolation generation consists of sampling randomly two (real) sentences from the dataset, encoding them to get their latent representation, interpolating the latent representations and finally decoding points sampled from that interpolation. More precisely, given two sentences s_1 and s_2, the interpolation of their latent representation $l_1 = Encoder(s_1)$, $l_2 = Encoder(s_2)$ can be computed as in Equation 4.1. By choosing the α value, it is possible to navigate the interpolation.

$$l_{intp}(\alpha) = l_1 * \alpha + l_2 * (1 - \alpha) \tag{4.1}$$

Once the interpolation is computed, the associated sentence is derived by decoding the interpolation through the Decoder: $s_{intp} = Decoder(l_{intp})$.

By interpolating two reviews in the latent space, it is possible to observe how the review shifts. This is important evidence for understanding whether the model is overfitting or underfitting. Having correct representation of the extremes of the interpolation (the original sentences) along with weird jumps in the sentence meaning and/or grammatical correctness on the interpolation would mean overfitting the dataset (memorizing the seen reviews without generalizing on the dataset). Having incorrect representation of the extremes and few changes of the sentence on the interpolation segment can mean underfitting.

Embedding Visualization

The VAE performances can be evaluated by looking at the Encoder embedding space as well: studying the latent code properties is important to understand the prowess of the VAE in representing sentences. Since the embedding space can't be witnessed on 2D graphs, it has to be projected onto a visualizable lower-dimensional sub-space. Principal Component Analysis (PCA) can be used for this task: it extracts the Principal Components (PC_i), that are the "directions" of the embedding space that explain most variance. PCA is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables (entities each of which takes on various numerical values) into a set of values of linearly uncorrelated variables called principal components (Karl Pearson, 2020). There are several properties that can be witnessed in the latent space, such as: review length, review sentiment and review topic. In order to select which topics to consider, Latent Dirichlet Analysis (LDA) is performed on the full dataset for topic extraction.

Attention Visualization

As already mentioned, the VAE is based on attention. The attention mechanism is well known for being interpretable. The way the model is built, it is possible to visualize the relationship between high-level features as well as low-level ones before and after each down-sampling blocks (in the Encoder) or up-sampling blocks (in the Decoder). This visualization can tell us whether the model is giving high relevance to non-grammatical relationships, thus possibly overfitting, or it is generalizing, thus learning the grammatical patterns of the language. Moreover, if there is a substantial number of heads that are too similar, the number of heads can be decreased. Alternatively, if the heads capture different interesting relationships, the number can be increased. Given the attention matrix, which models the correlation

between concepts, it is possible to project it onto a discrete two-dimensional color-map.

4.3.3 Anomaly Detection

In order to evaluate properly the performances of the AD algorithm, it would be better to have deep-fake reviews generated by multiple deep-neural-networks since, as already said, we want a robust detection algorithm that can generalize on all the deep-fake reviews: it would be useless to have a detector that detects deep-fake reviews generated by only one type of deep-neural-net. Consequently, the AD is tested on three different datasets, half of which is made of legit reviews sampled from the test-set and half of which is made of deep-fake reviews generated by: one attentive VAE, one Transformer using top-k sampling, one Transformer using temperature sampling, all of them trained on a sub-portion of the train-set. Once the three datasets are completed, assessing the performances of the AD can be done computing Accuracy, Precision, Recall and F1, on unseen data. In addition, the reader is going to be provided with a visualization of the latent space projected onto the 2-dimensional spaces described by its principal components. This visualization helps understanding whether every principal components is useful or not. If there are principal components that aren't useful, then it could be a good idea to project the latent space onto the lower-dimensional space described by the useful principal components, which would lessen the computational resources needed by the AD.

4.3.4 Ablation Study

The ablation study is a crucial step for any deep learning research. It helps to figure out causality into the model, which is the most straightforward way to generate reliable knowledge. In facts, considering the stochasticity of the parameter initialization and of the optimization routines in artificial neural networks, chances are that removing a few modules (or replacing some trained features with random ones) there is no loss in performance. By dissecting the net (removing each component one by one) and by observing the results, it is possible to understand the role that each component takes and their relationship between each other. For these reasons, the VAE is dissected of its main parts and its broken down variants are trained on a sub-part of the train-set for few epochs (due to computational power issues) and checked on the dev-set. The ablation study provides also visualizations of the comparison between training curves of the full VAE and of its broken down versions to show how the convergence of the model improves.

Chapter 5

System architecture

This section provides an in-depth description of the attentive Variational Autoencoder (attnVAE) and Anomaly Detection (AD) algorithm foe detecting deep-fakes. For the VAE, the architecture comes first, then the training and optimization technique are explained. For the AD, multiple techniques are explained in general and the best one is explained in depth.

5.1 Attentive Variational Autoencoder

The overall VAE architecture is depicted in Figure 5.1. The VAE is made of two main modules: the Encoder and the Decoder. The Encoder encodes the reviews into the latent space, turning them into embeddings, which are lower-dimensional w.r.t. the original data space. The Decoder decodes the embeddings back into the data space. Encoder and Decoder are almost specular and both of them contain two types of secondary modules: Self-Attentive Blocks (SABs) and Point-Wise Feed Forward nets (PWFFs).

5.1.1 Encoder

The Encoder is represented in Figure 5.1 in the VAE context. It is composed of three main parts:

1. **Embedding**: embeds words using an unbiased linear projection.
2. **Core**: outputs a continuous dense vector after down-sampling the input word-embeddings.
3. **Gaussian**: generates the bias vector and logarithmic variance vector needed to produce the latent codes, starting from a continuous dense

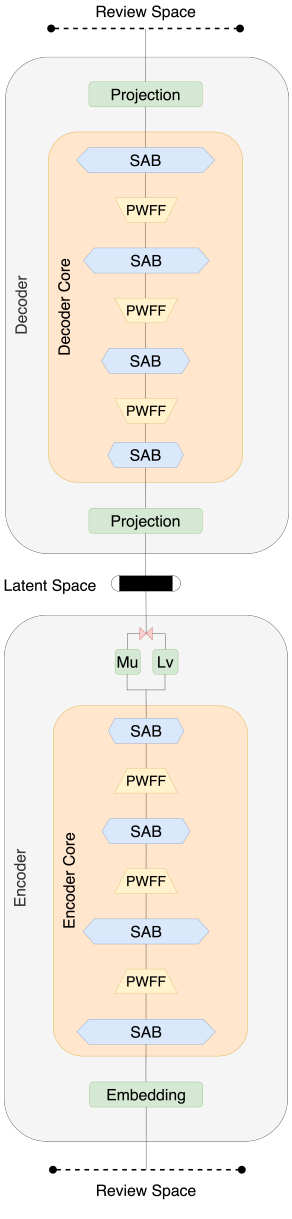

Figure 5.1: Variational Autoencoder Architecture

vector.

Core

The Encoder Core module involves two main mechanisms: self-attention (in the SABs) and down-sampling (in the PWFFs). The two mechanisms are alternated *n_blocks* times, starting with self-attention: the (word-embedded) input sequence goes through the self-attention first and then is down-sampled. The process is repeated *n_blocks* times and each time the input sequence is reduced in length, until it reaches the right length (which is an hyperparameter). The sequence length reduction can be linear or exponential, but in general it can be any non-increasing function. For the sake of simplicity, this work is realized with linear reduction, therefore, for each repetition, the sequence length is reduced of k units. For example, given $k = 4$ the input sequence is reduced as: $16 \to 12 \to 8 \to 4$. Applying self-attention at different sequence lengths helps the network correlate not only words, but higher-level features too, which take the name: concepts. In addition, down-sampling alleviates the Gaussian module from having to produce the *bias* and *logvar* vectors starting from a full-length sequence, which would be harder and would require many more parameters.

Workflow

The Encoder Embedding module takes as input a mini-batch of 1-hot encoded sentences (dimension := $batch\_size \times seq\_len \times vocab\_size$), then it linearly projects the sequence onto the *emb*-dimensional space where the words embeddings lay on (dimension := $batch\_size \times seq\_len \times emb\_dim$). Next, the Encoder core is fed with the word-embedded full sequence. The Self-Attentive Block (SAB) computes the attention of the sequence and passes its output to the Down-Sampling Block (DSB), which consists of a Point-Wise Feed Forward (PWFF) network followed by a one-dimensional convolution with kernel size of 1. The SAB-DSB process is repeated *n_blocks* times until the sequence has seq_final_len length (dimension := $batch\_size \times seq\_final\_len \times vocab\_size$). Afterwards, the tensor is flattened (dimension := $batch\_size \cdot seq\_final\_len \times vocab\_size$) and passed to the Gaussian module. The Gaussian module is made of two separate Linear Feed Forward networks that compute both bias and logarithmic variance. Those are then mixed as in Equation 3.1 to produce the latent embeddings (dimension := $batch\_size \times code\_dim$) which are ready to be fed to the Decoder.

5.1.2 Decoder

The Decoder is represented in Figure 5.1 in the VAE context. Its architecture is mostly specular to the Encoder one, having three modules:

1. **Linear**: projects the latent embeddings back to the word-embedded sequences.
2. **Core**: up-samples starting from a small-length sequence, arriving to a full-length sequence.
3. **Embedding**: maps the input sequence from an embedding-dimensional space to a 1-hot word-dimensional space using an unbiased linear projection.

Core

Similarly to the Encoder, the Decoder Core module involves self-attention (in the SABs) and up-sampling (in the PWFFs). The two mechanisms are alternated n_blocks times, starting with self-attention. The sequence length increment can be linear or exponential, but in general it can be any non-decreasing function. For the sake of simplicity, this work is realized with linear increment, therefore, for each repetition, the sequence length is incremented of k units.

Workflow

The latent embeddings are fed to a the linear layer that turns them back into a word-embedded sequence (dimension := $batch\_size \times seq_len \times emb\_dim$). This is then fed to the Core, which starts with a new SAB and subsequently with an Up-Sampling Block (USB) which up-samples to increase the length of the sequence. The USB consists of a PWFF followed by a one-dimensional convolution with kernel size of 1,needed to up-sample. The SAB-USB process is repeated n_block times until the sequence is full-length. Finally, the full-length sequence is projected back to the 1-hot word space by means of a linear projection (tensor dimension := $batch\_size \times seq\_len \times vocab\_size$).

5.1.3 Self-Attentive Block

The Self-Attentive Block (SAB) performs MHA as explained in Equation 3.14, which is an ensemble of self-attentions that model the relationships between words in a sentence. However, considering that sentences are created sequentially from left to right, modelling the relationship w_k to w_{k+C}, $C > 0$

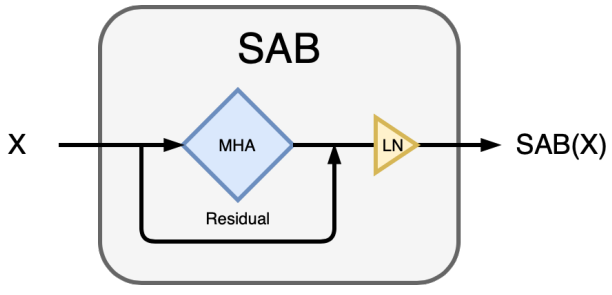

Figure 5.2: SAB architecture.

would mean "looking at the future". The only relationships allowed are those between w_{k+C} and w_k, "looking at the past". Breaking these rules, just like in (Devlin, Chang, Lee, & Toutanova, 2018), is problematic during the generation phase, again, because sentences are formed left to right. To avoid computing self-attention on every pair of words, the input sentence is sub-sequentially masked: given a sequence $s = [s_1 \dots s_n]$, sub-sequential masking generates n sequences masked sequentially, as in Listing 5.1.

Listing 5.1: Masking example

```
SOS     MASK    MASK    MASK    MASK    MASK    MASK
SOS     this    MASK    MASK    MASK    MASK    MASK
SOS     this    is      MASK    MASK    MASK    MASK
SOS     this    is      an      MASK    MASK    MASK
SOS     this    is      an      example MASK    MASK
SOS     this    is      an      example   .     MASK
SOS     this    is      an      example   .     EOS
```

The masking value (MASK) is as close as possible to $-\infty$ so that, by computing the attention between an actual word and a masked word, the result is as low as it can get, hence, any possible correlation is neglected by the model. The SAB presents residual connections (He, Zhang, Ren, & Sun, 2016) that connect the input to the output. Residual connections help the optimization process by reducing the impact of vanishing/exploding gradient. For the sake of optimization Layer Normalization (Lei Ba, Kiros, & Hinton, 2016) is applied to the output too. Since layer normalization stabilizes the output of the layer it is applied to, convergence accelerates. The SAB is depicted in Figure 5.2.

5.1.4 Point-Wise Feed Forward Net

The PWFF is a type of Network in Network (NIN) (Lin, Chen, & Yan, 2013) which generalizes Convolutional Neural Networks (CNN). Convolutional layers apply a linear filter on the receptive field, as in Figure 5.3. A linear filter is a Generalized Linear Model (GLM) for the underlying data patch. NINs replace the GLM with a micro Neural Network, as in Figure 5.4.

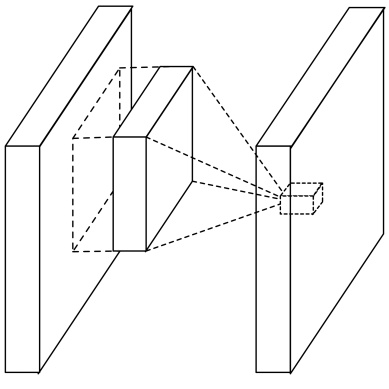

Figure 5.3: Convolutional net

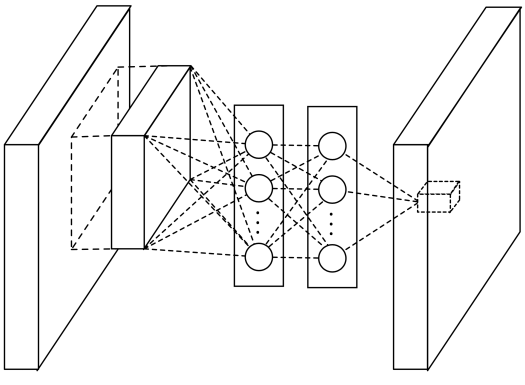

Figure 5.4: Network in network

The PWFF is a NIN that implements the micro net as a Linear Feed Forward net and has filter-size of 1. This means that it is applied to each position of the input separately and identically, just like a one-dimensional convolutional layer with filter size of 1, therefore, it saves parameters w.r.t. a standard linear projection on all the parameters at once. Mathematically speaking, the micro-net takes as input x, which is the receptive field, and computes the function in Equation 5.1 which describes two linear transformations with a ReLU activation in between. That being said, the PWFF is easier to implement with two convolutions with kernel size 1, as depicted in Figure 5.5.

$$micro(x) = \max(0, x \cdot W_1 + b_1) \cdot W_2 + b_2 \tag{5.1}$$

5.1.5 Positional Embeddings

Since the VAE itself contains no recurrence and the Attention mechanism is shift-invariant hence, in order for the model to make use of the order of the sequence, the information about the relative or absolute position of the tokens in the sequence must be added explicitly. There are several types of positional encodings which depend on the type of attention considered.

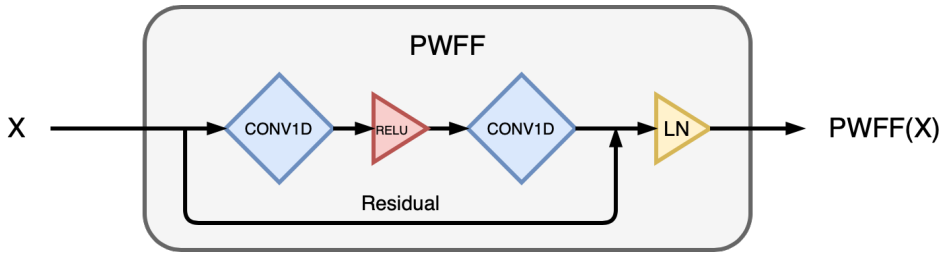

Figure 5.5: PWFF architecture

- **Fixed**: designed to work with fixed-context attention. The attention is operated over the full sentence.
- Relative: designed to work with segmented-context attention. The attention is operated over segments of the full sentence.
- **Frozen**: the encodings stay the way they are initialized, they can't change.
- Learnable: the encodings are initialized in a certain way, but the back-propagation of the neural network can still affect them, thus, they act as parameters of the model.

Independently from the type of embedding that is used, positional encodings (P) are summed to the word embeddings (E) before each SAB: $input = P + E$. Of course, positional encodings have the same emb dimension as the word embeddings, so the two can be summed.

Following the success of the positional encodings in (Vaswani et al., 2017), **sine** and **cosine** embeddings are employed. These are fixed, frozen embeddings. Their formulation is expressed in Equations 5.2, 5.3.

$$PE(pos, 2i) = \sin\left(pos/10000^{2i/emb}\right) \tag{5.2}$$

$$PE(pos, 2i+1) = \cos\left(pos/10000^{2i/emb}\right) \tag{5.3}$$

Here, pos is the position of the token in the sentence and i is the dimension. That is, each dimension of the positional encoding corresponds to a sinusoid. The wavelengths form a geometric progression from 2π to $10000 \cdot 2\pi$. This function should help the model learn how to attend by relative positions, since for any fixed offset k, PE_{pos+k} can be represented as a linear function of PE_{pos}. The positional embeddings are fixed from the start since learned positional embeddings produce nearly identical results but they introduce more learnable parameters in the model.

5.2 Training

Training the VAE is not straightforward: both the **objective function** and the **optimization algorithm** have a big impact on the convergence of the model, therefore they must be picked consciously. As already mentioned, the VAE objective function L_{VAE} in Equation 3.4 involves a reconstruction term and a regularization term. The actual loss equation, however, is a weighted loss if the two terms: Equation 5.4.

$$L_{VAE} = L_{reconstruct} + \alpha \; L_{regularize} \tag{5.4}$$

The α value is initialized to 0.1 and is set to 1 after the first 8 epochs. This is enough to avoid the reconstruction term overpowering the other term, explained in Paragraph 2.1.4:

1. no continuous annealing is needed.
2. no minimum value of the reconstruction term has to be found.

This is an important improvement from the previous architectures since the former techniques employed need high-influence hyperparameters to be tuned.

The reconstruction term is supposed to indicate the model how similar the reconstruction is to the input, while, the regularization term tells how similar is the latent space distribution to the chosen prior. For Language Modelling, two appropriate losses are:

- Reconstruction Loss: **Negative Log Likelihood** (**NLL**)
- Regularization Loss: **Kullback Liebler Divergence** (**KLD**)

5.2.1 Reconstruction Loss

The log-likelihood function is a logarithmic transformation of the likelihood function. Since concavity plays a key role in the maximization and, as the most common probability distributions in particular the exponential family—are only logarithmically concave, it is usually more convenient to work with the log-likelihood function. Moreover, the log-likelihood is particularly convenient for maximum likelihood estimation since logarithms are strictly increasing functions, hence, maximizing the likelihood is equivalent to maximizing the log-likelihood (Log-likelihood, 2020). Still, since it is better to work with minimization and maximizing the log-likelihood is equivalent to minimizing its negative, negative log-likelihood (NLL) is better suited.

When dealing with text, it is possible to compute the likelihood of a sequence of words $s = [w_1 \dots w_n]$, as in Equation 5.5.

$$likelihood(s) = \prod_{i=1}^{n} P(w_i | w_1 \dots w_{i-1}) \tag{5.5}$$

Finally, NLL can be computed as in Equation 5.6.

$$NLL(s) = -\frac{1}{n} \sum_{i=1}^{n} \log \big(P(w_i | w_1 \dots w_{i-1}) \big) \tag{5.6}$$

Furthermore, NLL is always used in most NLP problems because of its correlation with **Perplexity** (**PP**). In information theory, Perplexity is a measure of how well a probability distribution or probability model predicts a sample, thus it may be used to compare probability models. A low PP indicates the probability distribution is good at predicting the sample (Perplexity, 2020). In NLP, PP is a way of evaluating language models since language models are probability distributions of words over sentences (or texts). In few words, the (likely) best model is the less perplexed one, thus, the one that minimizes Equation 5.7.

$$PP(s) = \sqrt[n]{\prod_{i=1}^{n} \frac{1}{P(w_i | w_1 \dots w_{i-1})}} \tag{5.7}$$

It is rather simple to demonstrate that: $PP(s) = e^{NLL(s)}$.

5.2.2 Regularization Loss

The Kullback–Leibler divergence, also known as relative entropy, is a measure of how one probability distribution D_1 is different from a second, reference probability distribution D_2 (Kullback–Leibler divergence, 2020). The KLD equals zero when the two distributions in question are identical. The formulation is in Equation 5.8.

$$KLD\ (D_1 || D_2) = \sum_{x \in \mathcal{X}} D_1(x) \log \left(\frac{D_1(x)}{D_2(x)} \right) \tag{5.8}$$

Considering prior $\mathcal{P}$ and latent code distribution $\mathcal{L}$, the regularization loss $KLD(\mathcal{P}||\mathcal{L})$ forces $\mathcal{L}$ to be shaped similarly to $\mathcal{P}$. There are several alternatives for $\mathcal{P}$ and, ideally, any distribution should work given a "strong enough" encoder, but usually it is either a Normal Gaussian (as in this work) or a Mixture of Gaussians for unsupervised or semi-supervised clustering. Normal Gaussians have many useful properties:

- they make the optimization tractable.
- they have an analytical evaluation of the KLD.
- they enable systematic gradient computation (backward pass) thanks to the reparametrization trick in Equation 3.1.

Additionally, since Normal Gaussians are easy to reproduce, it is more likely that, by sampling random points from the latent space, their decoded images are meaningful. For these reasons, this work employs a Gaussian prior $\mathcal{P}$ with zero mean and unit variance.

5.2.3 Optimizer

Optimizers are optimization algorithms that update the learnable parameters of the model (e.g. all the parameters that are affected by the backward pass) to get to the lowest possible loss value. Adam is an optimizer that employs first-order gradient-based optimization of stochastic objective functions and is based on adaptive estimates of two lower-order moments (e.g. first and second moment). The method is computationally efficient, has little memory requirements, is invariant to diagonal re-scaling of the gradients, and is well suited for problems that are large in terms of data and/or parameters. The method is also appropriate for non-stationary objectives and problems with very noisy and/or sparse gradients. The hyper-parameters have intuitive interpretations and typically require little tuning (Kingma & Ba, 2015). Adam keeps the advantages of two well-known optimization algorithms: Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp).

5.2.4 Scheduling

Deep learning optimizer such as Adam have few hyperparameters: the learning rate is the value that mostly controls how much to jump towards the direction of minimum gradient in the parameter space during gradient descent. By decaying the learning rate it is possible to reach far better performances rather than keeping it fixed. This way, it is possible to simulate annealing in the model. However, (Smith, Kindermans, Ying, & Le, 2017) shows how it is possible to simulate annealing by increasing the batch size instead. This speeds up training not only due to the bigger batch size, but because is possible to increase the learning rate ϵ and scaling the batch size B accordingly ($B \propto \epsilon$) and reach pretty much the same results (under the assumption $B \ll N$, having N as the number of samples in the dataset:

e.g. $B < N/10$). This comes from the equation $g = \epsilon(N/B - 1)$ that is empirically demonstrated in several studies. In fact, recently it is increasingly accepted that small batches help the model generalize better due to the difference in stats between the whole dataset and a mini-batch. More interestingly, in (Keskar, Mudigere, Nocedal, Smelyanskiy, & Tang, 2016) and (Smith & Le, 2017) is argued how the optimal batch size is $B_{opt} = \epsilon N$.

Following the studies the batch size is initialized to 256 and is doubled every 8 epochs, following an exponential trend, up to 2048. By increasing the batch size (along with the learning rate) the training really does speed up however, it makes it more unstable. Most of the times, the training reaches similar results but it suffers from bigger fluctuations (for bigger batch sizes).

5.3 Anomaly Detection

Detecting neural-generated reviews is not a trivial task: this task clearly can't be supervised at all since there is no clear definition of what a complete neural-generated dataset could be. This being due to the infinite ways to neurally produce text. The only way to overcome this is unsupervisedly: modelling the natural language (in consideration) as better as possible should help the model tell the difference between natural and neural language.

This idea holds up if two main hypotheses are satisfied:

1. The dataset used to train the model is free enough from neurally-generated reviews (e.g. infinitesimal values like less than 0.1%).
2. The natural language is different enough from the neural language (e.g. in grammar and/or coherence).

The first hypothesis is likely to hold since the review dataset employed has been extrapolated from Amazon in 2018 and the first strong LMs started being released in 2019 (e.g. GPT-Full). The second hypothesis is more problematic since it is exactly the ascent of these powerful LMs that makes it harder to hold. Yet, this works doesn't aspire to solve the deep fakes problem completely, it is meant to provide a cheap and efficient first barrier. If both both hypotheses hold and the model is powerful enough, it should be able to either have an high reconstruction error for the outliers (the deepfakes) or should provide some separation hyperplane in the latent space. For this reason, there are two main approaches that are based on:

1. Reconstruction Error

5.3.1 Reconstruction Error

Approach number one consists of computing the reconstruction error ERR on the train set samples $s_i, i \in [1 \dots N]$, cutting the tail, keeping a high percentage of the samples (e.g.$95 - 99\%$) and imposing the threshold: $thr = \max_{i \in [1 \dots N]} \big(ERR(s_i)\big)$. If a test sample s_T has reconstruction error $ERR(s_T) > thr$ then s_T is considered as an outlier (deep-fake), alternatively it is an inlier (legit).

Theoretically, this approach is the most robust one since it uses the whole model (both Encoder and Decoder) to compute the rejection score and it doesn't depend on any external hyperparameter.

5.3.2 Encoder Embeddings

Approach number two uses the Encoder: it generates the latent representations (the embeddings) of the inputs and hopefully it can provide a clear separation between legit and adversarial generations, so that they can be predicted as inliers (legit) or outliers (deep-fake) respectively. Of course, the Encoder alone can't do this: an additional algorithm is needed to perform outlier detection on the Encoder embeddings. Usually one-class Support Vector Machines (OC-SVM) are employed to fulfill this task: SVMs have bees the state of the art before Artificial Neural Nets started being trainable, moreover, since their objective function enforces finding a hyperplane that separates points belonging to different classes, it should make finding outliers (points not belonging to the one-class) rather easy. Even though SVM are popular, they are not faultless: SVMs are based on the kernel used to project the input space to the higher-dimensional kernel space where the hyperplane can be effective, but such kernel is not easy to find. Another helpful technique is clustering, which consists of partitioning the dataset into sets containing similar points. Clustering the train-set, which contains the real reviews, helps computing a per-cluster confidence level CL_k that indicates to which cluster k a certain point p_i belongs. In order to perform outlier detection using clustering, it is enough to set a threshold thr: given p_i, thr is compared to the max confidence level between all clusters (C) for p_i so that:

- $thr \geq \max_{k \in C} CL_k(p_i) \Rightarrow p_i$ is inlier
- $thr < \max_{k \in C} CL_k(p_i) \Rightarrow p_i$ is outlier

There are several kinds of clustering techniques as Gaussian Mixture Models (GMM), K-Nearest Neighbors (KNN) and many more, but these approaches don't model noise and outliers explicitly as Density-Based Spatial Clustering of Applications with Noise (DBSCAN) does. In facts, both GMM and KNN have random-like performances for this work.

DBSCAN models the clusters depending only on the hyperparameter ϵ: the density-based algorithm clusters the points that aren't more distant than ϵ and marks as outliers points that lie alone in low-density regions. More specifically, a set of points is considered as cluster if it contains at least min_pts points. Hence, DBSCAN is a non-parametric algorithm that requires tuning just two hyperparameters. For these reasons, DBSCAN is an appropriate algorithm for outlier detection. Still, this technique doesn't explicitly take into consideration how to predict new samples: clustering every time that a new point is added can be too expensive. DBSCAN can be slightly modified to make online predictions: given a new point p_i it is possible to run through all the core-points cp_k belonging to the real reviews cluster, determining each time whether p_i is within distance ϵ from cp_k. If this hypothesis is satisfied, then the point is considered as an inlier (real), alternatively it is considered as an outlier (deep-fake). In order to work properly, the DBSCAN algorithm must be tuned on the train-set in such a way to have one big cluster of points and few outliers. This big cluster of points represents the family of real reviews; each review that is not associated to that cluster is considered as deep-fake.

Chapter 6

Experimental results and evaluation

This chapter provides and explains all the results about the novel attention-based VAE and some characteristic of the APD dataset.

6.1 Dataset

The Amazon Product Data (APD) comes from (He & McAuley, 2016). It contains million of reviews from the popular e-commerce website Amazon. The reviews are aggregated in multiple categories, ranging from electronics to books to food supplies. Picking only reviews belonging to a single category simplifies the language modelling task since it makes the word vocabulary significantly smaller, therefore the model is trained and tested on reviews regarding "Grocery and Gourmet Food" only. The APD Grocery and Gourmet Food dataset contains $5,074,160$ reviews for $287,209$ products and $2,695,974$ reviewers, amounting to roughly $2GB$ of data. Each review has on average 17.89 reviews, with a minimum of 1 review and a maximum of 11526 reviews. By aggressively de-duplicating the data and keeping only the reviews with max length of 14 (sum of number of words and punctuation), the dataset shrinks to $1,908,771$ reviews with $1,178,441$ reviewers. Each review now has on average 10.33 reviews, with a minimum of 1 review and a maximum of 5415 reviews. Since the dataset contains reviews in English, the Zipfian approximation must have empirical exponent $exp_{zipf} = 1.07$, however, the best fit has an Zipf exponent of $exp_{zipf} = 1.50$, indicating the natural language expressed by these reviews is a sub-type of English.

Variant	# Parameters	# n_blocks
VAE_4	1159064	4
VAE_8	1110076	8
VAE_{16}	1086142	16
VAE_{32}	1075295	32

Table 6.1: VAE variants per depth

6.2 Tuning

6.2.1 Experimental Setup

The VAE versions are trained on a sub-part of the the train-set, which consists of 10% of the train-set, and it is developed on the dev-set. The quantitative performances are evaluated using the Negative Logarithmic Likelihood to assess the reconstruction loss and Kullback Liebler Divergence to assess the regularization loss. Each experiment provides a visualization of the training, since it helps choosing the parameters that make the convergence faster, and provides a table of the performances computed on the dev-set, which helps check that overfitting is not occurring. The first experiment deals with picking the right depth of the model; here, the model has already been hyper-parameter tuned and uses only GELU as activation function. Next, the second experiment copes with picking the right activation function: both GELU and ReLU are tried out.

6.2.2 Model Depth

The VAE implemented for this work has a peculiar property: independently of how many layers are employed to encode and decode the sentence, the parameter count remains roughly the same, actually, decreasing slightly as the depth increases, as depicted in Table 6.1. The variants are named VAE_k after k which is the total number of attention-blocks in the architecture, which is the total number of layers minus 4.

As empirically demonstrated in several works, the deeper the nets, the more complex the functions that they can properly approximate. Having an increased depth doesn't come for free though: the training time increases substantially since the signal must traverse more layers, moreover, having more layers increases vanishing gradient issues, which is still not-completely solvable even using multiple residual connections. Regarding optimization, Figure 6.1 shows the convergence of the different models on the train-set. It

shows how the NLL term seems to be lower for the less deep models, while the KLD behaves oppositely. This means that the deeper models find easier to minimize the KLD instead of the NLL, which is not a good sign, since it represents an increment of the regularization overpowering effect. The only model that seems to optimize the NLL as well as the KLD at the same rate is the VAE_8: it reaches the same performances as the VAE_4 on reconstructing sentences and reaches the same performances as VAE_{32} on regularizing. These realizations are confirmed on the dev-set in Table 6.2. Considering the success of this version w.r.t. its shallower and deeper peers on both train-set and dev-set, from now on, the VAE taken into consideration is the VAE_8 version.

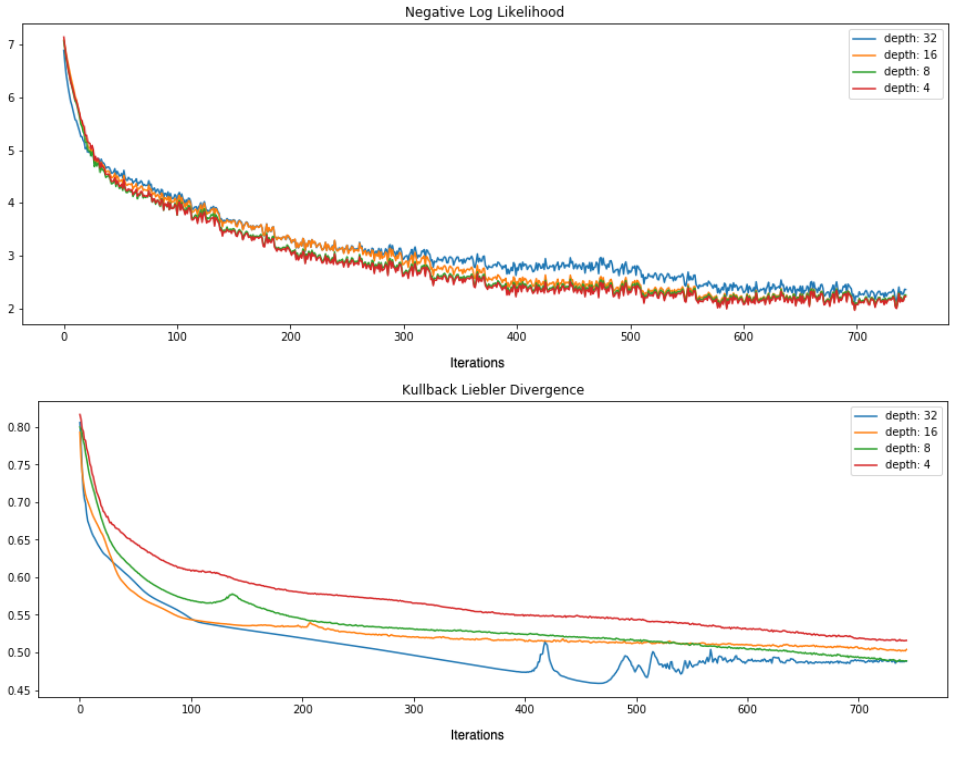

Figure 6.1: Depth Variants Convergence

Version	PP	NLL	KLD
VAE_4	9.034	2.201	0.542
VAE_8	9.234	2.223	0.5148
VAE_{16}	9.7376	2.276	0.5251
VAE_{32}	12.1581	2.498	0.5192

Table 6.2: Depth Dev-Set Performances

6.2.3 Nonlinearity

Figure 6.2 shows how ReLU is clearly a game-changer, fitting the train-set much better than GELU. Nonetheless, choosing this nonlinearity comes with the burden of paying more attention to avoid overfitting, which is more likely to happen. Still, Table 6.3 shows in the dev-set how overfitting hasn't happened. For this reason, from now on, the ReLU activation is substituted to the GELU.

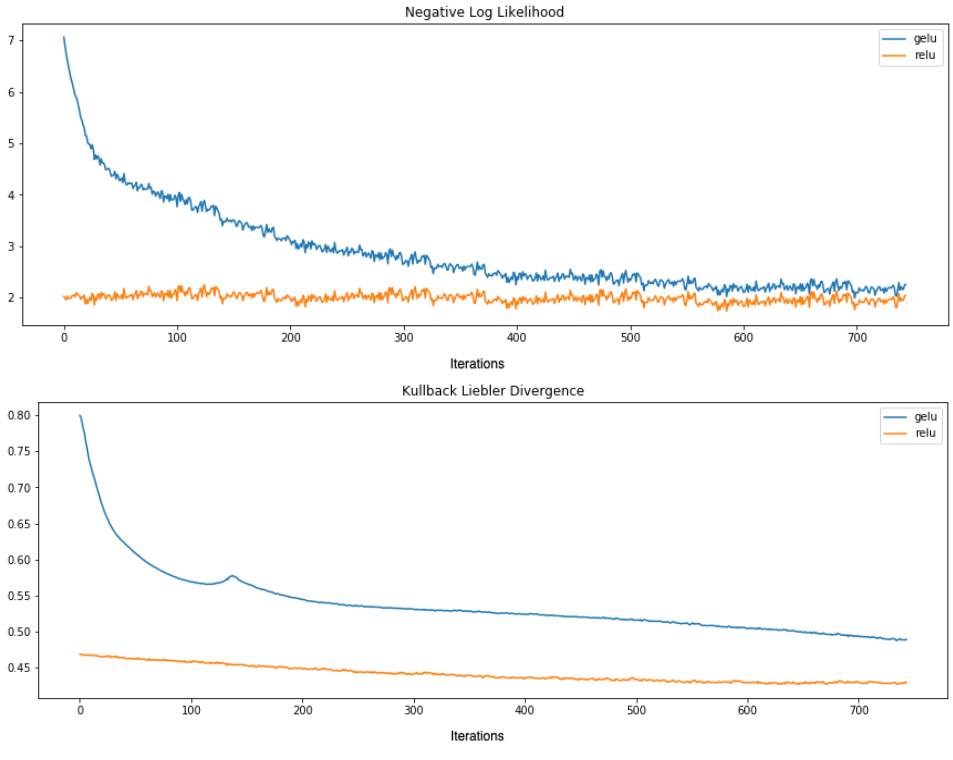

Figure 6.2: GELU vs ReLU Variant Convergence

Version	PP	NLL	KLD
VAE_{8-GELU}	9.2349	2.223	0.5148
VAE_{8-RELU}	7.5157	2.017	0.4337

Table 6.3: Activation Dev-Set Performances

6.3 Language Modelling

This section explores the capabilities of the VAE in modelling reviews.

6.3.1 Experimental Setup

In these experiment, the VAE performances are compared quantitatively to the Transformer performances as a way of understanding whether it is a competitive language model or not. Perplexity (PP) is used as metric for the comparison but Negative Logarithmic Likelihood (NLL) is shown as well. Each model now is trained on train-set: 90% of the full data-set; it is tested on the test-set: 5% of the full data-set. Both models have been trained for 32 epochs with the Adam optimizer, doubling the batch size every 8 epochs starting from a batch size of 256. Both models have been hyperparameter-tuned considering limited computational resources. The hyperparameter-related information is in Appendix A.

Next, the VAE is qualitatively tested as two types of generations are assessed. The first, random generation, consists of sampling random latent points using a Normal Standard and decoding them in the review space. The second, interpolation generation, consists of interpolating the latent points associated to two real reviews and decoding the selected interpolations in the review space. Additionally, the latent space properties are inspected looking at review length, sentiment and topic. Since the dataset doesn't include sentiments, the reviews are fed to a robust sentiment analysis neural classifier that outputs a continuous value between 0 (negative sentiment) and 1 (positive sentiment) which is clustered in 10 uniformly distributed values. More information regarding the classifier are provided in Appendix B. The dataset doesn't include topics either, therefore, Latent Dirichlet Analysis (LDA) is used to extract the top-5 food-related topics. Additionally, all the attention heads are visualized for each layer of both Encoder and Decoder using color-mapped matrices.

6.3.2 Quantitative Performances

Table 6.4 offers the comparison of the main hyperparameters for each model.

Version	hid_dim	emb_dim	key_dim	heads	depth	# parameters
Transformer	1024	256	32	3	8	1149728
VAE_8	512	64	32	3	8	1110076

Table 6.4: Transformer vs VAE Hyperparameters

The result of training both models with the Adam routine with $lr = 1e-4$ and betas $\beta_1 = 0.9, \beta_2 = 0.999$ for the first 16 epochs is shown in Figure 6.3. The training is cut to the first 16 epochs to visualize better the start of the

curve and the mid, which is where the weight of the reconstruction loss is increased from 0.1 to 1. Clearly, the VAE fits the data much better from the start, reaching a significant gain by the end of the experiment. The VAE confirms its dominance on the test set as well, as shown in Table 6.5. The table shows the test-performances of both models at the end of the training.

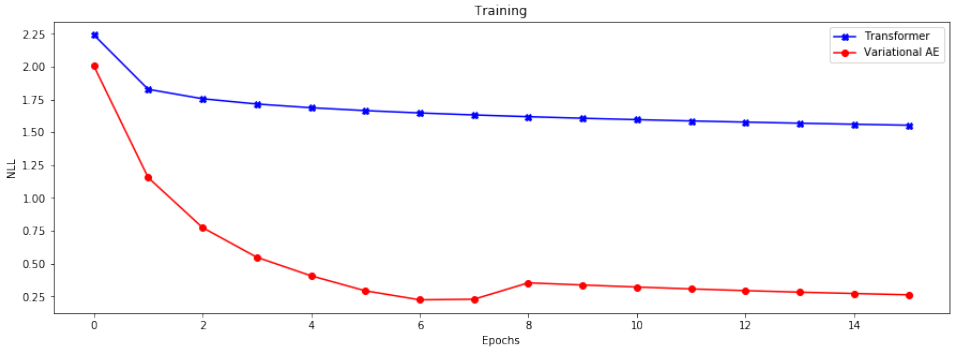

Figure 6.3: Transformer vs VAE training

Version	PP	NLL
Transformer	1.8505	0.6155
VAE_8	1.099	0.0944

Table 6.5: Test-Set Performances

6.3.3 Qualitative Performances

Generation

For the sake of readability, the auxiliary tokens are substituted as:

- SOS $\rightarrow$ '>>'
- EOS $\rightarrow$ '<<'
- PAD $\rightarrow$ ' '

Listing 6.1 shows 10 random generations. The attentive VAE can generate reviews of different length and different topics. It is clear how the generations are not perfect and exhibit errors: reviews (4) and (5) are missing the EOS token, while review (7) is presenting more than one EOS, however the model can capture diversity and review coherence.

Listing 6.1: Random Generation

```
0) >> best of energy ! <<
```

```
1) >> this is my chocolate cup ! <<
2) >> great unk <<
3) >> good and fast shipping <<
4) >> i loved these product so much , i just unk .
      pleased ! !
5) >> great snack , as expected
6) >> best coffee coffee gift !  <<
7) >> nice sweet . amazon << unk << <<
8) >> my unk coffee bag ! <<
9) >> refreshing sweet . clean almonds . great on pieces
      . good ... flavor . <<
```

Listing 6.2 shows just 10 interpolations for 10 values of α. The starting and ending reviews are presented at the top of the example. This visualization provides us with a deeper understanding of the model behavior: it is capable of keeping the positive tone of both reviews over the whole interpolation while handling the length change pretty well too. The reviews are not flawless here too, as interpolation (3) is missing an EOS token. It is interesting to notice how the first two interpolations have the same structure but different subject, hinting that the word-embeddings for 'oatmeal' and 'butter' must be very similar; this behavior is common in other analogous cases (e.g. great in review (4) and good in review (5)). We can also notice how there are replicated reviews, which suggests that those reviews might have more relevance than others (it might be due to their high frequency in the dataset) thus taking up more latent space than others.

Listing 6.2: Interpolation Generation

```
START)  SOS best oatmeal on the UNK . EOS PAD PAD PAD PAD
        PAD PAD PAD PAD
END)    SOS good EOS PAD PAD PAD PAD PAD PAD PAD PAD PAD
        PAD PAD PAD PAD

0) >> best oatmeal on the unk . <<
1) >> best butter on the unk .
2) >> best chai shake unk <<
3) >> great chai ! unk
4) >> great honey ! <<
5) >> good honey ! <<
6) >> excellent <<
7) >> excellent <<
8) >> good <<
9) >> good <<
```

Latent Embeddings

The review properties that are inspected are:

- Sentence Length: Figure 6.4
- Sentence Sentiment: Figure 6.5
- Sentence Topic: Figure 6.6, Figure 6.7.

Figure 6.4 shows how, the VAE can model the review length pretty well, presenting a continuous partitioning in almost all the graphs. This means that it is possible to change the sentence length by just moving along those directions of the latent space. Still, the sub-space described by the first two principal components describes the length better than the other two sub-spaces, as there is more displacement of colors, meaning that the first principal can explain better the lengths of the reviews.

Figure 6.5 displays how the latent space is much shyer in distinguishing positive (max positive: 9) from negative (max negative: 0) reviews. Still, it is possible to notice a slight polarization in the graphs delineated by principal components PC_0 which can be definitely considered an important asset for modelling the review sentiment. Oppositely, the sub-space described by the second PC_1 and third PC_2 principal components presents sentiment randomness.

The top 5 food-related key-words returned by the LDA are: coffee, tea, chocolate, candy, sugar (in order of occurrence from highest to lowest). The latent representation of the reviews containing these topics are depicted in Figure 6.6. We can notice that there is no clear separation between topics, especially considering them all together, however, Figure 6.7 illustrates how by plotting only reviews containing the 'coffee' topic, it is possible to see how it follows some cluster-pattern. The latter is a positive behavior since it means that the attentive VAE is powerful enough to group the reviews that contain the same topic. Once again, the sub-spaces described by the PC_0 show this behavior better than the others, meaning that PC_0 is the most important principal component for this aspect.

Attention

Figure 6.8 illustrates the Encoder attentive modelling, while Figure 6.13 shows the Decoder attentive modelling. The visualizations include actual words only for the first layer of the Encoder and the last layer of the Decoder since those deal with word-embeddings, while the other visualizations

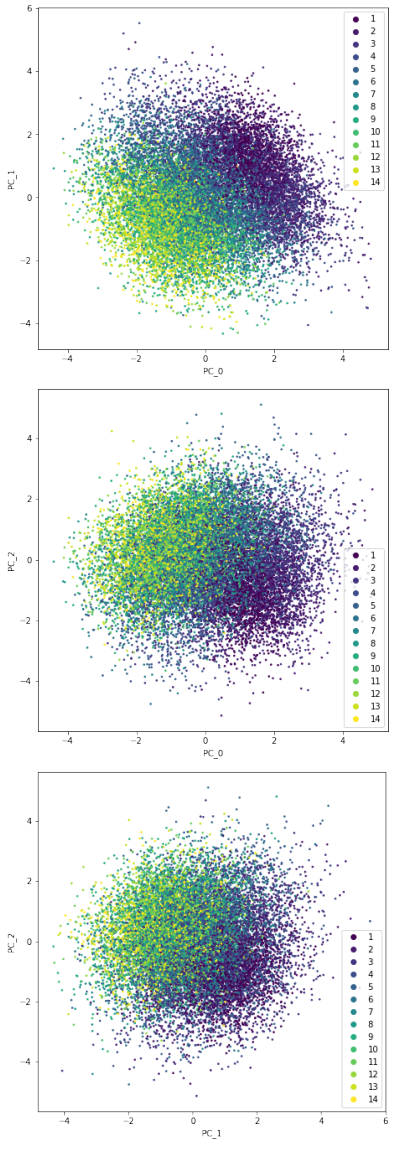

Figure 6.4: Latent Space: Review Length

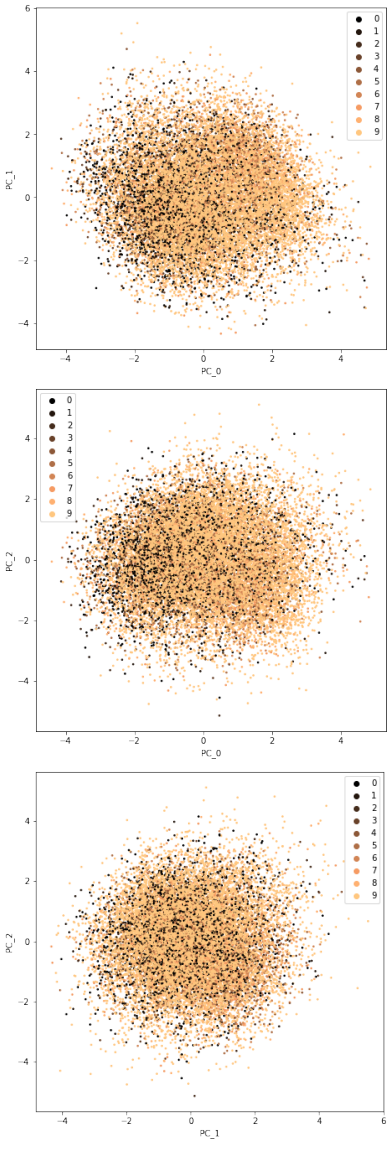

Figure 6.5: Latent Space: Review Sentiment

don't since they deal with concepts (high-level features). For each layer, the attention heads are plotted from left to right: the first attention head is on the left, the second attention head is in the middle and the third is on the right.

For the word-level attention we can notice how the model can capture non-trivial relationships as sentence constructs: for instance in Figure 6.16 in all the attention heads, the model understands that 'what' and 'I' have

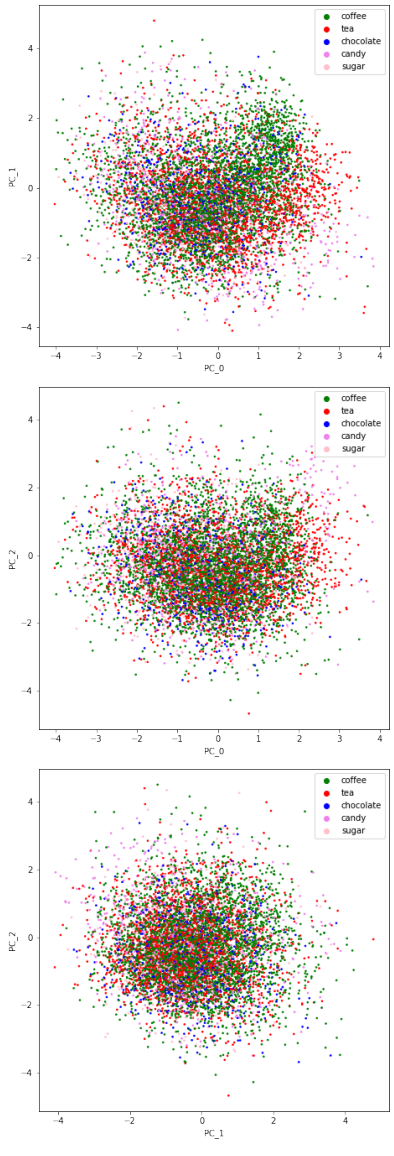

Figure 6.6: Latent Space: Review top-5 Topics

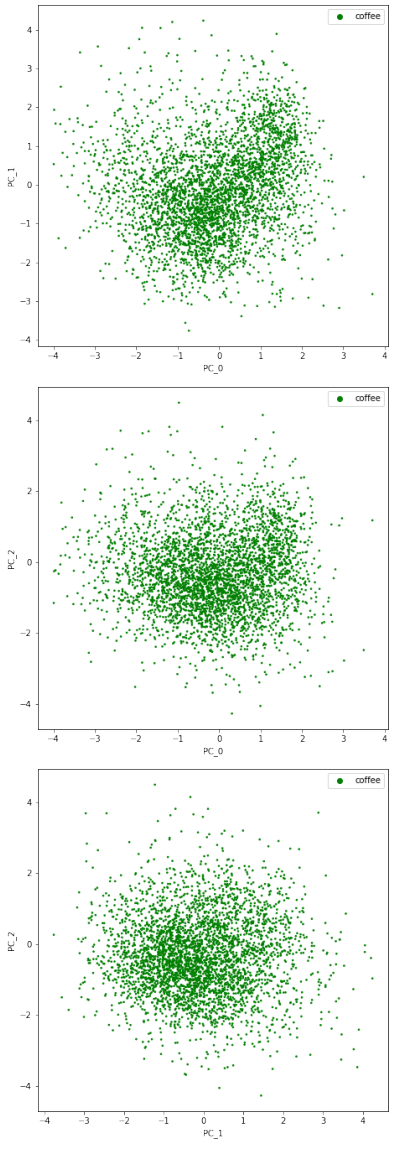

Figure 6.7: Latent Space: Review Topic

high correlation in the 'what I wanted' sequence of words. Moreover, in Figure 6.8, all the attention heads learn that after the first padding (PAD) it is useless to model the attention for the other ones, since they are inevitably going to repeat, therefore the attention correlation shrinks down as the paddings follow. The model, however, captures trivial properties too: in Figure 6.8 the third head gives high correlation to the matches between the SOS token and every other word. This is a trivial relationship since if there

is an SOS token, there will inevitably be other words, except in the case of empty review (empty review: SOS EOS PAD PAD ...).

Looking at the higher level-features, we can notice how, for most attention heads, we have non-null correlation between concepts, neither the correlation is random, meaning that the attention correlation is useful to the model. In the case there would be an attention head showing just a diagonal attention matrix, we could cut off that head, since it would add nothing to the sentence understanding: a diagonal attention matrix would mean that each element of the sequence is important to itself only.

6.4 Anomaly Detection

6.4.1 Experimental Setup

The Anomaly Detection algorithms are trained on the train-set and tested on three different datasets made of 1024 legit reviews from the test-set and 1024 adversarial-generated reviews. The adversarial models used to generate the deep-fake reviews are:

- Attentive VAE, random sampling: adversarial_vae.jsonl
- Transformer, top-10 sampling: adversarial_k10.jsonl
- Transformer, temperature sampling: adversarial_temp4.jsonl

Each model is trained on 10% of the train-set.

The performances of the AD are assessed computing Accuracy, Precision, Recall and F1. In addition, the reader is provided with a visualization of the latent space projected onto the 2-dimensional spaces described by its first three principal components.

6.4.2 Reconstruction Error

The performances of the Reconstruction Error based AD are in Table 6.6. The performances are definitely bad, which suggests that this is not the right algorithm to use. This might be because for VAE it is easier to reconstruct smaller sentences than longer ones, thus the error grows as the sentence length does too, ending up with having a reconstruction error that gets as noisier as the review length grows. The performances are rather good only for the adversarial_temp4.jsonl dataset, suggesting it might be the most grammatically incorrect one.

Figure 6.8: Encoder self-attention

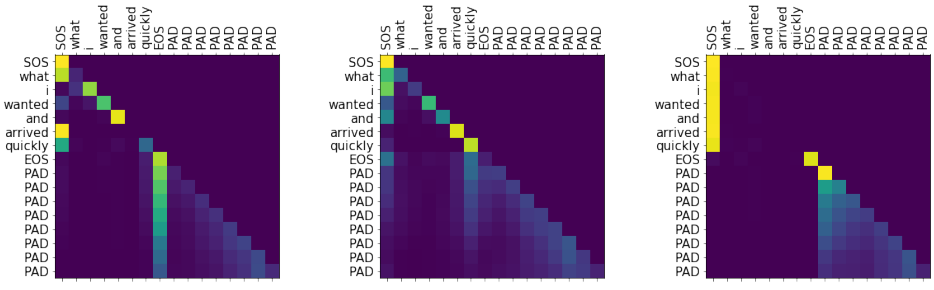

Figure 6.9: First attention block

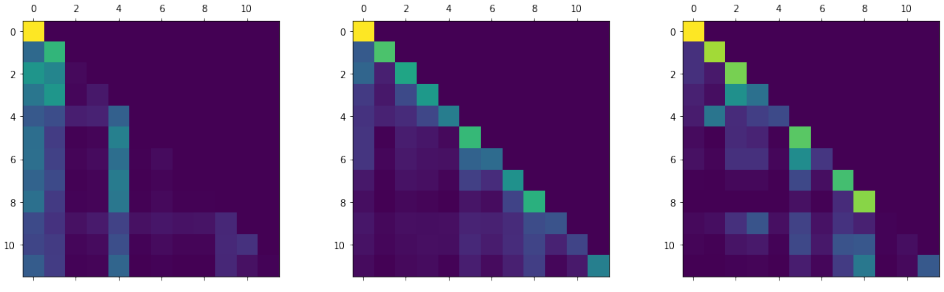

Figure 6.10: Second attention block

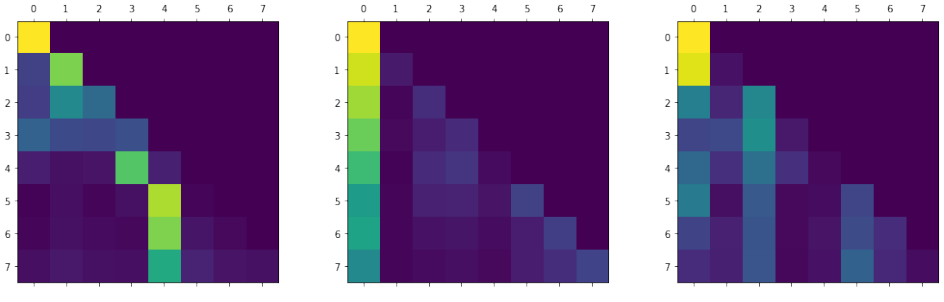

Figure 6.11: Third attention block

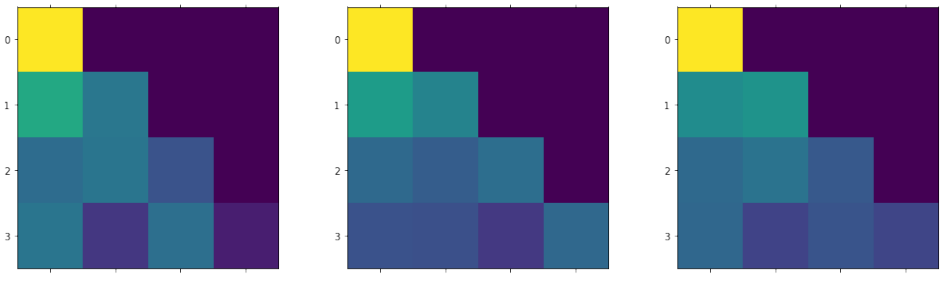

Figure 6.12: Fourth attention block

Figure 6.13: Decoder self-attention

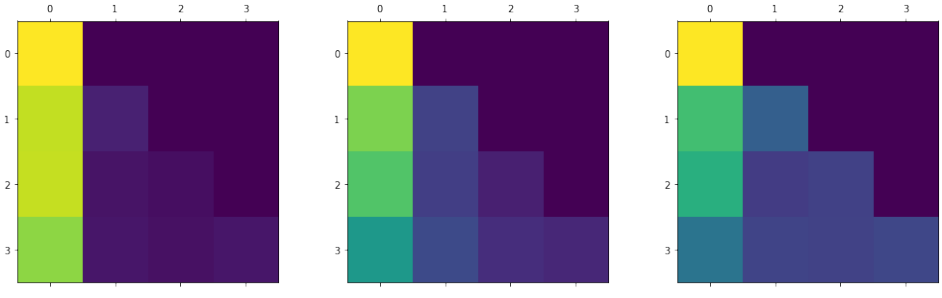

Figure 6.14: First attention block

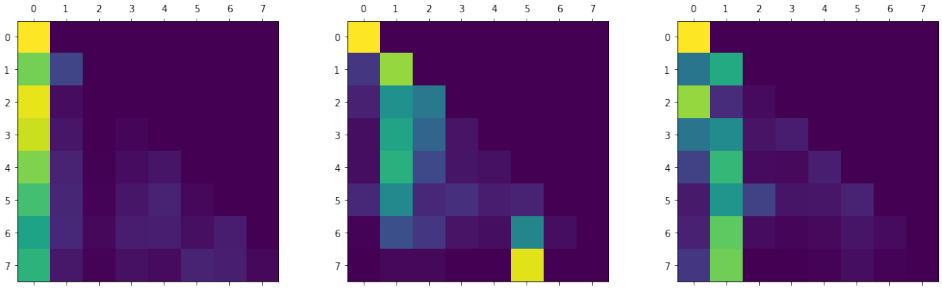

Figure 6.15: Second attention block

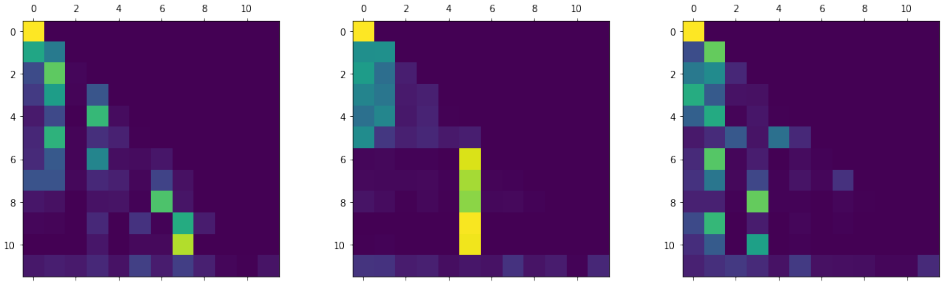

Figure 6.16: Third attention block

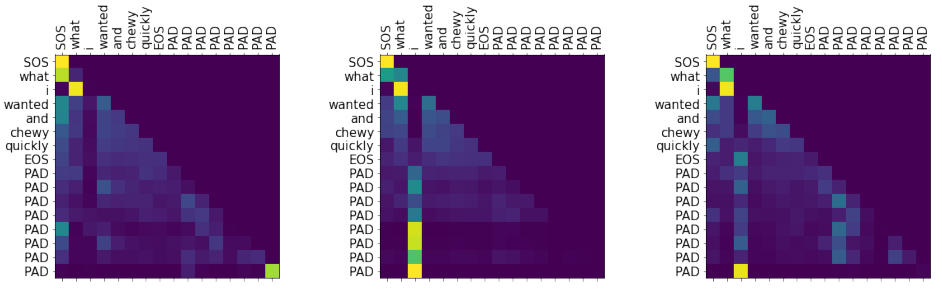

Figure 6.17: Fourth attention block

Dataset	Accuracy	Precision	Recall	F1
adversarial_vae.jsonl	0.5309	0.5163	0.5959	0.5595
adversarial_temp4.jsonl	0.7290	0.6486	0.9960	0.7861
adversarial_k10.jsonl	0.5561	0.5307	0.6425	0.5914

Table 6.6: Reconstruction Error Performances

6.4.3 Encoder Embeddings

Figure 6.18, 6.20 and 6.19 show the predictions on the latent embeddings of the reviews of the three adversarial datasets, projected on the first 3 principal components and Table 6.7 presents the performances on the three datasets. It is clear how the first principal component can disentangle the legit reviews from the deep-fake ones. Of course, this is possible thanks to the VAE Encoder that can capture grammatical correctness and coherence in the reviews. Popular clustering algorithms as GMM or KNN have random-like performances. This means that having explicit handling of outliers in the clustering algorithm (as DBSCAN) is mandatory.

Dataset	Accuracy	Precision	Recall	F1
adversarial_vae.jsonl	0.7607	0.7074	0.7021	0.7459
adversarial_temp4.jsonl	0.7422	0.6973	0.627	0.7086
adversarial_k10.jsonl	0.8565	0.7827	0.9609	0.8696

Table 6.7: DBSCAN Performances

6.5 Ablation Study

6.5.1 Experimental Setup

The VAE broken-down versions are trained on a sub-part of the the train-set, which consists of 10% of the train-set, and they are validated on the dev-set. The quantitative performances are evaluated using the Negative Logarithmic Likelihood to assess the reconstruction loss and Kullback Liebler Divergence to assess the regularization loss. Each experiment provides a visualization of the training curves, since it helps checking if the convergence gets faster, and provides a table of the performances computed on the dev-set, which helps check that the model variant hasn't overfitted.

6.5.2 Residual Connections

Figure 6.21 displays the training of the VAE_8 version with and without residual connections. It is easy to recognize how the non-residual version struggles optimizing the NLL term (the reconstruction term), which means it has an hard time modelling correctly the sentences, instead it just keeps lowering the KLD error, which is easier to do: this behavior brings to the overpowering of the regularization term, which is not desirable. On the other hand, the residual version is able to gain a solid reduction in NLL after a while, which means it is succeeding at giving the right importance to both reconstruction and regularization terms at the same time. Of course, a sudden drop in NLL provokes a sudden rise in the KLD but still, the absolute value of the NLL drop is about 10 times more than the absolute value of the KLD rise. This probably happens because the network has to learn when to use the residual connections and when to ignore them, thus it takes some time to properly use this optimization. Table 6.8 shows how the performances translate into the dev-set as well, confirming the residual connection are helpful.

Version	PP	NLL	KLD
VAE_{8-NO_RES}	16.8272	2.8236	0.1304
VAE_{8-RES}	7.7679	2.0581	0.4288

Table 6.8: Residual Dev-Set Performances

6.5.3 Layer Normalization

Figure 6.22 shows the influence of normalization. This technique makes both training curves more stable during training: less steep in the beginning and makes strong fluctuations less common especially for the KLD term. It takes some iteration to show the importance of layer normalization, which is due to the fact that, when normalized, layers can't change their output too quickly. Table 6.10 shows that the performances on the dev-set are similar to the ones in the test-set, meaning no overfitting has occurred.

Version	PP	NLL	KLD
VAE_{8-NO_NORM}	8.8463	2.18	0.7142
VAE_{8-NORM}	7.2572	1.982	0.4542

Table 6.9: Normalization Dev-Set Performances

6.5.4 Positional Encodings

For this work are employed fixed-frozen parameters since there is no recurrence in the model (the attention is applied over the full sentence) and learning the embeddings doesn't add much performance compared to the number of parameters that needs to be learned. Figure 6.23 shows the training curve of the VAE with and without adding positional embeddings to the word-embeddings. The regularization error is the most affected term here: imposing a Gaussian $\mathcal{N}(0, 1)$ prior over the latent space is much simpler on the positional VAE, rather than the positional embedding-free version. Also, the NLL loss remains pretty much unaltered, validating the adoption of these embeddings. The performances on the dev-set confirm that applying the embeddings doesn't hurt the model.

Version	PP	NLL	KLD
VAE_{8-NO_POS}	13.3965	2.595	0.5517
VAE_{8-POS}	9.2349	2.223	0.5148

Table 6.10: Positional Embeddings Dev-Set Performances

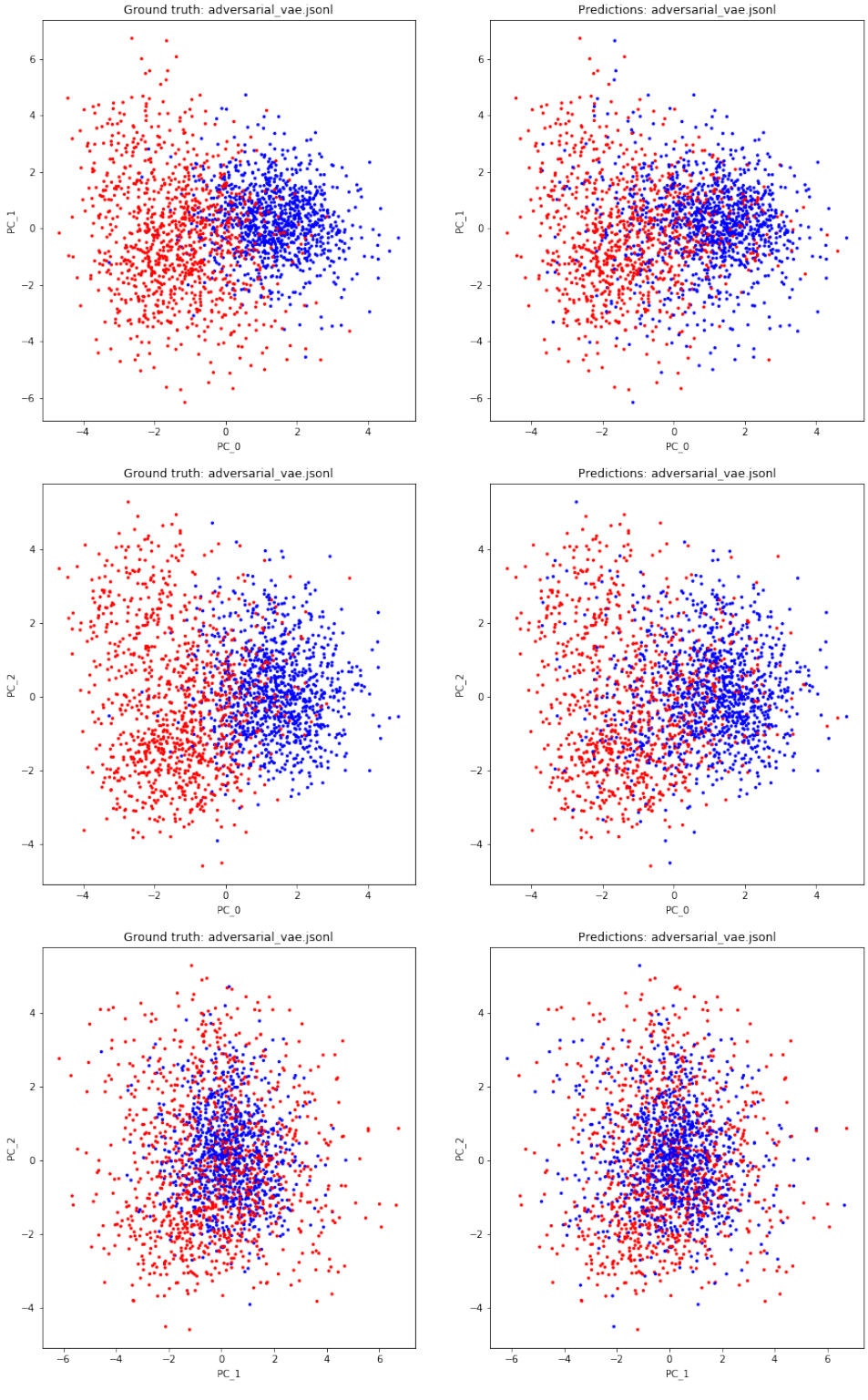

Figure 6.18: adversarial_vae.jsonl

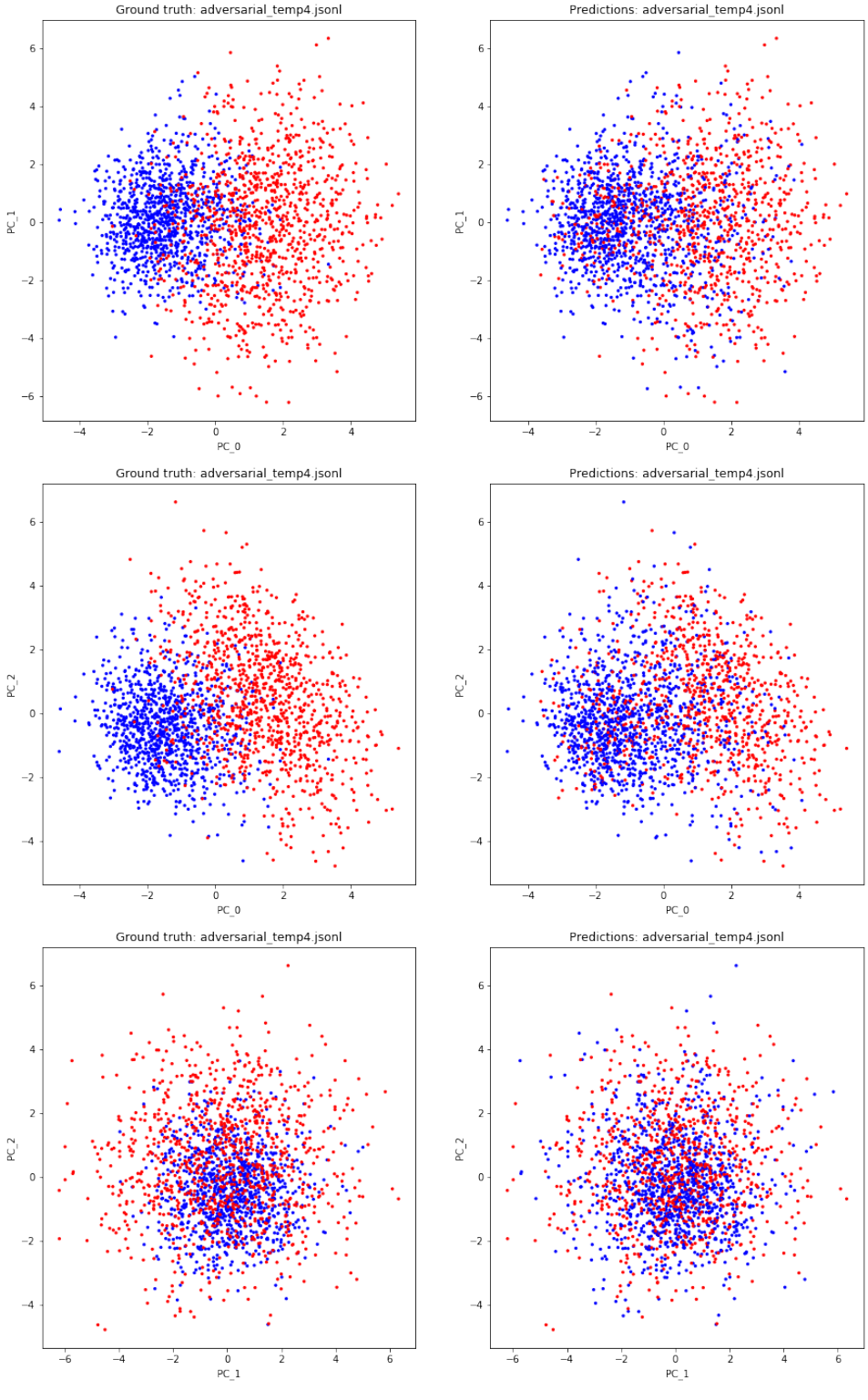

Figure 6.19: adversarial_temp4.jsonl

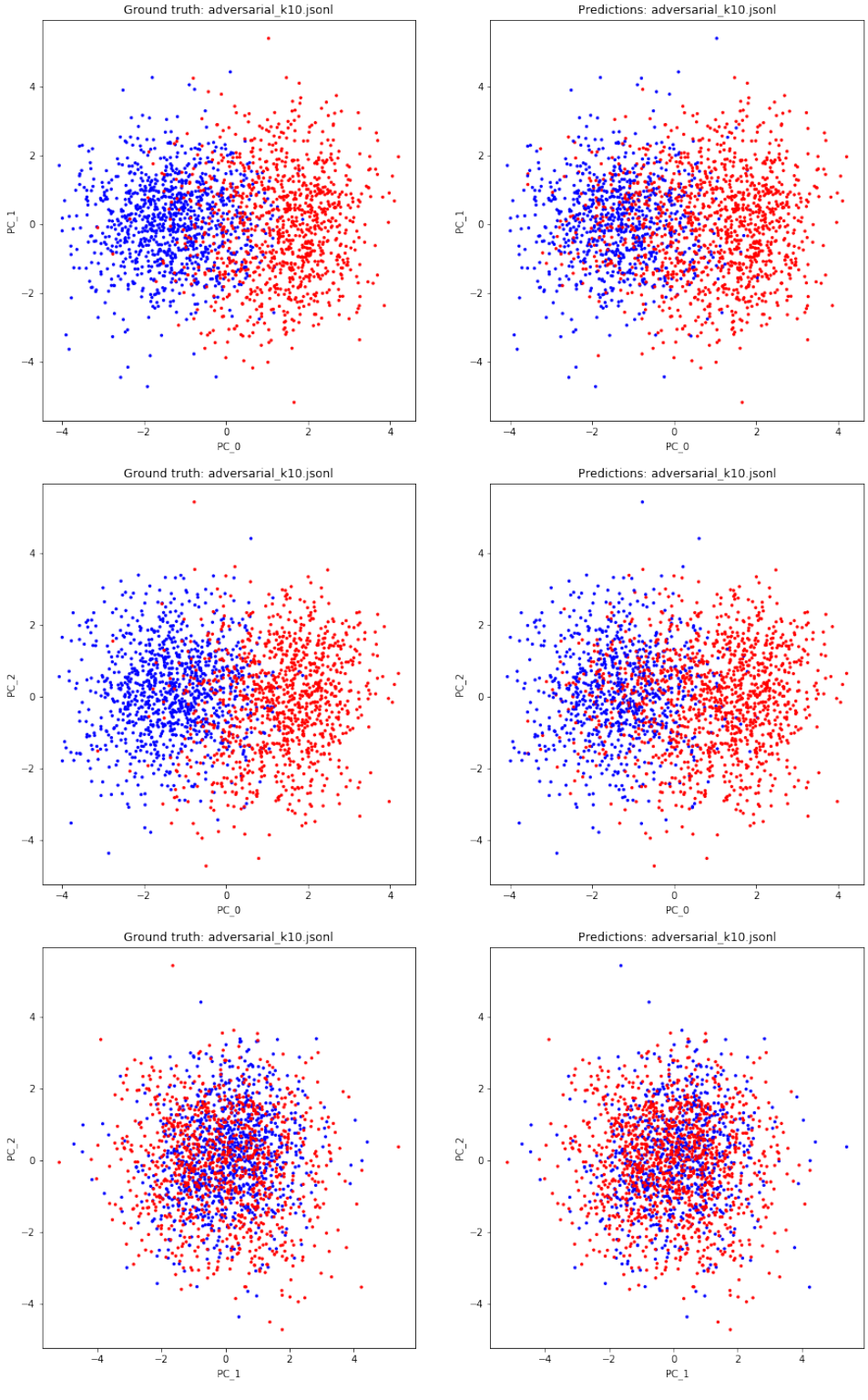

Figure 6.20: adversarial_k10.jsonl

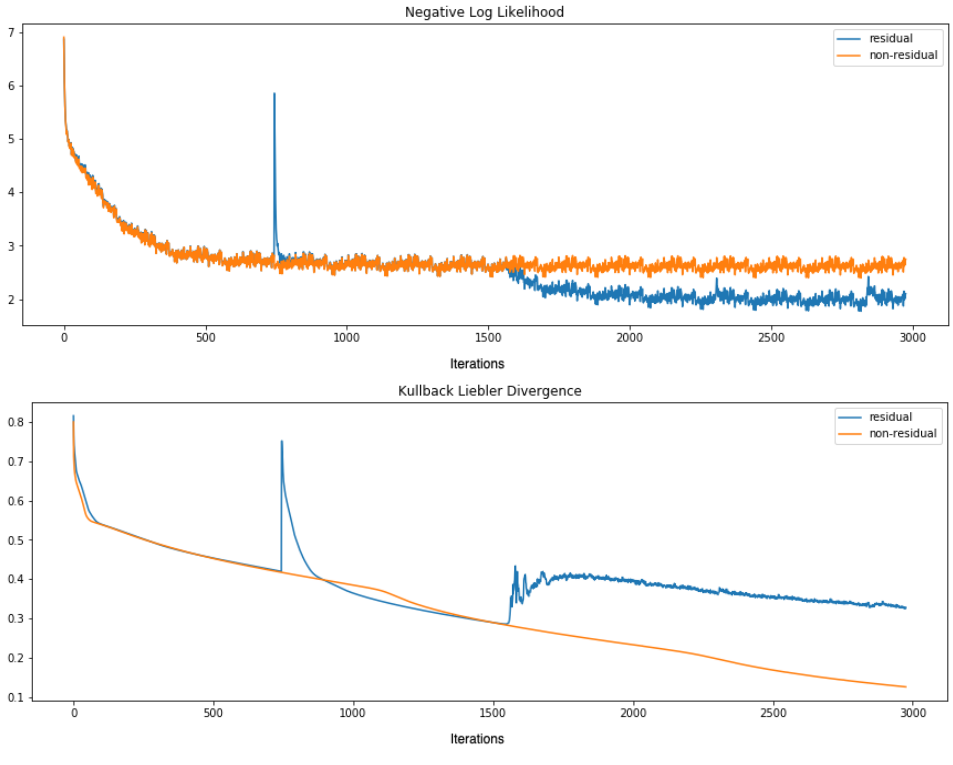

Figure 6.21: Residuality Ablation

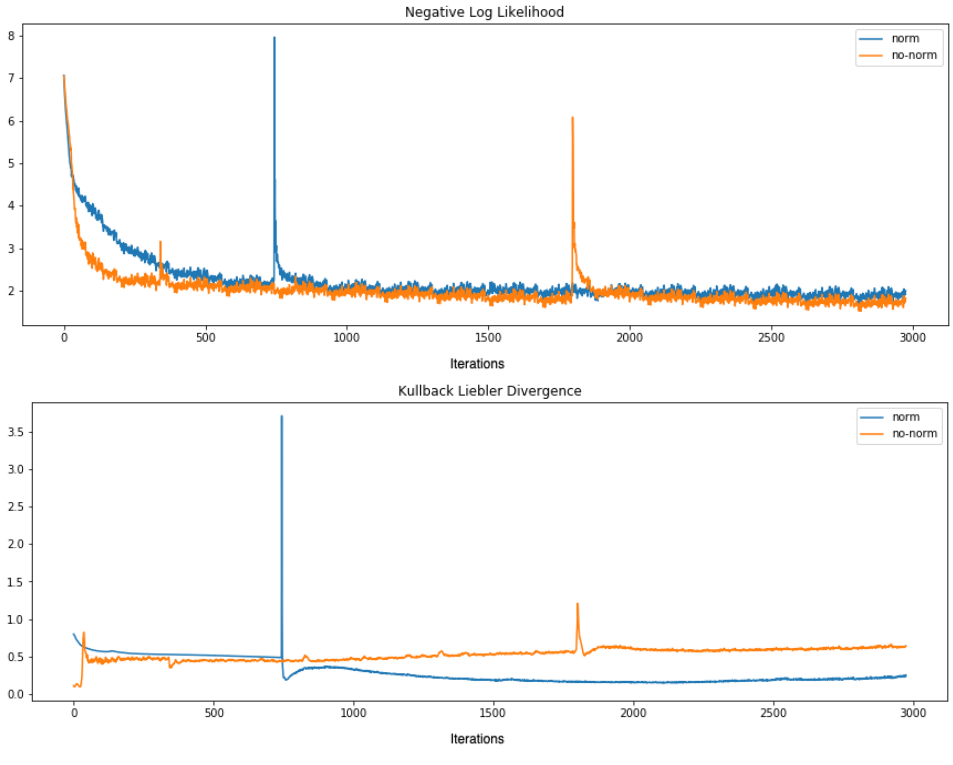

Figure 6.22: Normalization Ablation

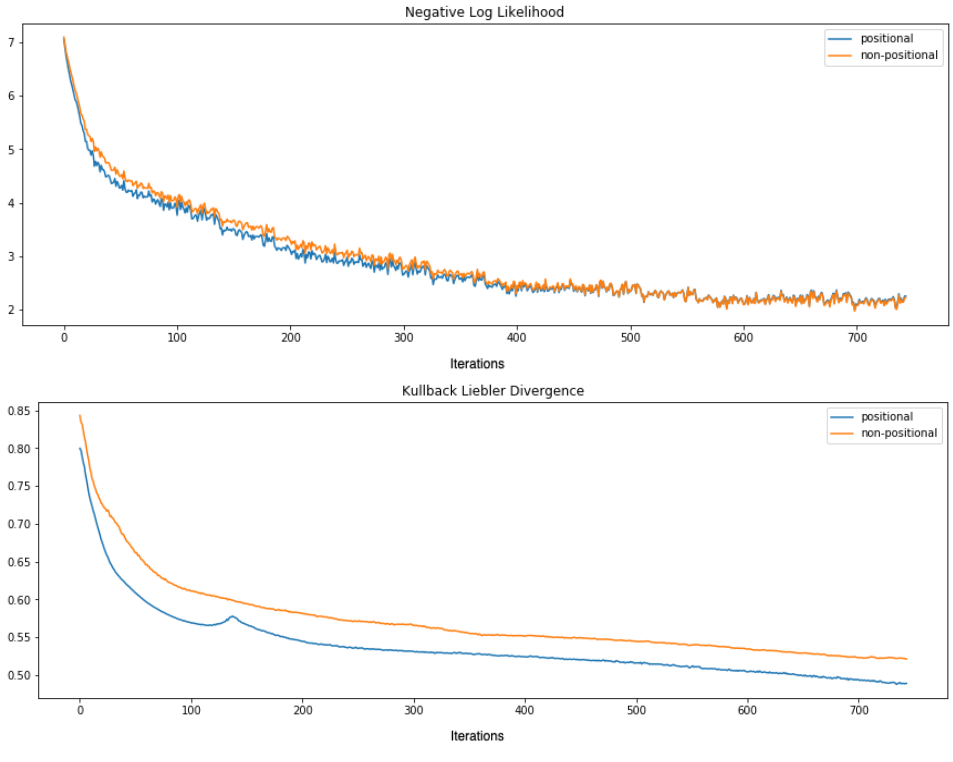

Figure 6.23: Positional Embeddings Ablation

Chapter 7

Conclusions and future perspectives

This work has demonstrated that it is possible to unsupervisedly detect deep-fake reviews with reasonable performances using unsupervised Anomaly Detection algorithms on Variational Autoencoder generated features. The AD solution is parameter-cheap compared to algorithm based on SVMs or NNs, is faster and delivers better performances. The Variational Autoencoder that is employed has a novel architecture and is the first VAE to employ the attention mechanism successfully.

There are several interesting future perspectives:

- Weakening either Encoder or Decoder could lead to better overall performances as in (Yang, Hu, Salakhutdinov, & Berg-Kirkpatrick, 2017).
- Incorporating the Transformer-XL mechanism would remove the fixed-context issue without any significant performance loss, as explained in (Dai et al., 2019).
- Incorporating the BERT objective function could increase the performances, as in (Devlin et al., 2018).

numbers

Bibliography

Agarap, A. F. (2018). Deep learning using rectified linear units (relu). *CoRR*, *abs/1803.08375*. Retrieved from `http://arxiv.org/abs/1803.08375`

Ba, J., Kiros, J. R., & Hinton, G. E. (2016). Layer normalization. *ArXiv*, *abs/1607.06450*.

Bahdanau, D., Cho, K., & Bengio, Y. (2014). *Neural machine translation by jointly learning to align and translate.* Retrieved from `http://arxiv.org/abs/1409.0473`

Bowman, S. R., Vilnis, L., Vinyals, O., Dai, A. M., Jozefowicz, R., & Bengio, S. (2015, November). Generating Sentences from a Continuous Space. *arXiv e-prints*, arXiv:1511.06349.

Chalapathy, R., Menon, A. K., & Chawla, S. (2018). Anomaly detection using one-class neural networks. *CoRR*, *abs/1802.06360*. Retrieved from `http://arxiv.org/abs/1802.06360`

Chen, J., Sathe, S., Aggarwal, C. C., & Turaga, D. S. (2017). Outlier detection with autoencoder ensembles. In *Sdm.*

Chorowski, J., Bahdanau, D., Serdyuk, D., Cho, K., & Bengio, Y. (2015). Attention-based models for speech recognition. In *Proceedings of the 28th international conference on neural information processing systems - volume 1* (p. 577–585). Cambridge, MA, USA: MIT Press.

Dai, Z., Yang, Z., Yang, Y., Carbonell, J. G., Le, Q. V., & Salakhutdinov, R. (2019). Transformer-xl: Attentive language models beyond a fixed-length context. *CoRR*, *abs/1901.02860*. Retrieved from `http://arxiv.org/abs/1901.02860`

Devlin, J., Chang, M., Lee, K., & Toutanova, K. (2018). BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, *abs/1810.04805*. Retrieved from `http://arxiv.org/abs/1810.04805`

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018, October). BERT:

Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv e-prints*, arXiv:1810.04805.

E. N. Sokholov. (2020). *Novelty detection — Wikipedia, the free encyclopedia.* Retrieved from `https://en.wikipedia.org/wiki/Novelty_detection` ([Online; last edited 30 September 2019])

Ergen, T., Hassan Mirza, A., & Kozat, S. S. (2017, October). Unsupervised and Semi-supervised Anomaly Detection with LSTM Neural Networks. *arXiv e-prints*, arXiv:1710.09207.

Fedus, W., Goodfellow, I., & Dai, A. M. (2018, January). MaskGAN: Better Text Generation via Filling in the_____. *arXiv e-prints*, arXiv:1801.07736.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... Bengio, Y. (2014). Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, & K. Q. Weinberger (Eds.), *Advances in neural information processing systems 27* (pp. 2672–2680). Curran Associates, Inc. Retrieved from `http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf`

Graves, A., Wayne, G., & Danihelka, I. (2014). Neural turing machines. *CoRR*, *abs/1410.5401*. Retrieved from `http://arxiv.org/abs/1410.5401`

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep residual learning for image recognition. *CoRR*, *abs/1512.03385*. Retrieved from `http://arxiv.org/abs/1512.03385`

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *2016 ieee conference on computer vision and pattern recognition (cvpr)* (p. 770-778).

He, R., & McAuley, J. J. (2016). Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. *CoRR*, *abs/1602.01585*. Retrieved from `http://arxiv.org/abs/1602.01585`

Hendrycks, D., & Gimpel, K. (2016). Bridging nonlinearities and stochastic regularizers with gaussian error linear units. *CoRR*, *abs/1606.08415*. Retrieved from `http://arxiv.org/abs/1606.08415`

Ioffe, S., & Szegedy, C. (2015, 07–09 Jul). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In F. Bach & D. Blei (Eds.), *Proceedings of the 32nd international conference on machine learning* (Vol. 37, pp. 448–456). Lille, France: PMLR. Retrieved from `http://proceedings.mlr.press/v37/ioffe15.html`

Karl Pearson. (2020). *Principal component analysis — Wikipedia, the free encyclopedia.* Retrieved from `https://en.wikipedia.org/wiki/`

Principal_component_analysis ([Online; last edited 28 March 2020])

Kawachi, Y., Koizumi, Y., & Harada, N. (2018). Complementary set variational autoencoder for supervised anomaly detection. In *2018 ieee international conference on acoustics, speech and signal processing (icassp)* (p. 2366-2370).

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., & Tang, P. T. P. (2016). On large-batch training for deep learning: Generalization gap and sharp minima. *CoRR*, *abs/1609.04836*. Retrieved from http://arxiv.org/abs/1609.04836

Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. In Y. Bengio & Y. LeCun (Eds.), *3rd international conference on learning representations, ICLR 2015, san diego, ca, usa, may 7-9, 2015, conference track proceedings.* Retrieved from http://arxiv.org/abs/1412.6980

Kingma, D. P., Mohamed, S., Jimenez Rezende, D., & Welling, M. (2014). Semi-supervised learning with deep generative models. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, & K. Q. Weinberger (Eds.), *Advances in neural information processing systems 27* (pp. 3581–3589). Curran Associates, Inc. Retrieved from http://papers.nips.cc/paper/5352-semi-supervised-learning-with-deep-generative-models.pdf

Krueger, D., Maharaj, T., Kramár, J., Pezeshki, M., Ballas, N., Ke, N. R., ... Pal, C. (2016). Zoneout: Regularizing rnns by randomly preserving hidden activations. *CoRR*, *abs/1606.01305*. Retrieved from http://arxiv.org/abs/1606.01305

Kullback–Leibler divergence. (2020). *Kullback leibler divergence — Wikipedia, the free encyclopedia.* Retrieved from https://en.wikipedia.org/wiki/Kullback_Leibler_divergence ([Online; last edited 13 March 2020])

Lei Ba, J., Kiros, J. R., & Hinton, G. E. (2016, July). Layer Normalization. *arXiv e-prints*, arXiv:1607.06450.

Lin, M., Chen, Q., & Yan, S. (2013, December). Network In Network. *arXiv e-prints*, arXiv:1312.4400.

Log-likelihood. (2020). *Log-likelihood — Wikipedia, the free encyclopedia.* Retrieved from https://en.wikipedia.org/wiki/Likelihood_function#Log-likelihood ([Online; last edited 26 March 2020])

Luong, M., Pham, H., & Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. *CoRR*, *abs/1508.04025*. Retrieved from http://arxiv.org/abs/1508.04025

Makhzani, A., Shlens, J., Jaitly, N., Goodfellow, I., & Frey, B. (2015, November). Adversarial Autoencoders. *arXiv e-prints*, arXiv:1511.05644.

Nguyen, T. T., Nguyen, C. M., Tien Nguyen, D., Thanh Nguyen, D., & Nahavandi, S. (2019, September). Deep Learning for Deepfakes Creation and Detection. *arXiv e-prints*, arXiv:1909.11573.

One-hot. (2020). *One-hot — Wikipedia, the free encyclopedia.* Retrieved from `https://en.wikipedia.org/wiki/One-hot` ([Online; last edited 20 January 2020])

Pascanu, R., Mikolov, T., & Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *Proceedings of the 30th international conference on international conference on machine learning - volume 28* (p. III–1310–III–1318). JMLR.org.

Perplexity. (2020). *Perplexity — Wikipedia, the free encyclopedia.* Retrieved from `https://en.wikipedia.org/wiki/Perplexity` ([Online; last edited 13 February 2020])

Press, O., Bar, A., Bogin, B., Berant, J., & Wolf, L. (2017, June). Language Generation with Recurrent Generative Adversarial Networks without Pre-training. *arXiv e-prints*, arXiv:1706.01399.

Radford, A. (2018). Improving language understanding by generative pre-training..

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2018). Language models are unsupervised multitask learners. Retrieved from `https://d4mucfpksywv.cloudfront.net/better-language-models/language-models.pdf`

ResNet. (2020). *Residual neural network — Wikipedia, the free encyclopedia.* Retrieved from `https://en.wikipedia.org/wiki/Residual_neural_network` ([Online; last edited 14 February 2020])

Robust outlier detection using svm regression. (2004). In *2004 ieee international joint conference on neural networks (ieee cat. no.04ch37541)* (Vol. 3, p. 2017-2022 vol.3).

Schmidt, F. (2019, October). Generalization in Generation: A closer look at Exposure Bias. *arXiv e-prints*, arXiv:1910.00292.

Semeniuta, S., Severyn, A., & Barth, E. (2017, February). A Hybrid Convolutional Variational Autoencoder for Text Generation. *arXiv e-prints*, arXiv:1702.02390.

Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., & Catanzaro, B. (2019, September). Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. *arXiv e-prints*, arXiv:1909.08053.

Smith, S. L., Kindermans, P.-J., Ying, C., & Le, Q. V. (2017, November). Don't Decay the Learning Rate, Increase the Batch Size. *arXiv e-prints*, arXiv:1711.00489.

Smith, S. L., & Le, Q. V. (2017, October). A Bayesian Perspective on Generalization and Stochastic Gradient Descent. *arXiv e-prints*, arXiv:1710.06451.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014, January). Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, *15*(1), 1929–1958.

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. *CoRR*, *abs/1409.3215*. Retrieved from `http://arxiv.org/abs/1409.3215`

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Polosukhin, I. (2017). Attention is all you need. *CoRR*, *abs/1706.03762*. Retrieved from `http://arxiv.org/abs/1706.03762`

Wu, Y., & He, K. (2018). Group normalization. *CoRR*, *abs/1803.08494*. Retrieved from `http://arxiv.org/abs/1803.08494`

Yang, Z., Hu, Z., Salakhutdinov, R., & Berg-Kirkpatrick, T. (2017). Improved variational autoencoders for text modeling using dilated convolutions. *CoRR*, *abs/1702.08139*. Retrieved from `http://arxiv.org/abs/1702.08139`

Zipf. (2020). *Zipf's law — Wikipedia, the free encyclopedia.* Retrieved from `https://en.wikipedia.org/wiki/Zipf%27s_law` ([Online; last edited 19 March 2020])

Appendix A

Hyperparameter Tuning

Hyperparameter tuning values that have been tried out for the Transformer in Table A.1 and Variational Autoencoder in Table A.2. The table values represent:

- - → not tried
- o → tried
- **x** → best

	2	3	4	8	16	32	64	128	256	512	1024	2048	4096
hid_dim	-	-	-	-	-	-	o	o	o	o	**x**	o	o
emb_dim	-	-	-	-	-	o	o	o	**x**	o	o	-	-
key_dim	-	-	-	-	o	o	**x**	o	o	-	-	-	-
# heads	o	**x**	o	o	o	-	-	-	-	-	-	-	-

Table A.1: Transformer Hyperparameter Grid

	2	3	4	8	16	32	64	128	256	512	1024	2048	4096
hid_dim	-	-	-	-	-	-	o	o	o	**x**	o	o	-
emb_dim	-	-	-	-	o	o	**x**	o	o	o	-	-	-
key_dim	-	-	-	o	o	**x**	o	o	o	-	-	-	-
# heads	o	**x**	o	o	o	-	-	-	-	-	-	-	-

Table A.2: VAE Hyperparameter Grid

Appendix B

Sentiment Analysis

The neural classifier that is used to predict the sentiment of the reviews is trained on the IMDB train-set. Even though the IMDB contains movie reviews, the classifier is tested on the APD dataset (the one employed for training the VAE) using human inspection, holding high-enough performances. Its architecture presents a word-embedding layer, connected to a two-layered bidirectional LSTM with dropout in-between layers. The LSTM layers are connected to a linear feed forward layer which ends up in a dropout layer. The training is carried on using the Adam optimizer for 50 epochs, holding than 94.83% balanced accuracy on the IMDB test-set.

Table B.1 presents the hyperparameters of the Neural Sentiment Analyzer.

Model	hid_dim	emb_dim	LSTM depth	dropout
Analyzer	256	100	2	0.5

Table B.1: Neural Sentiment Analyzer Hyperparameters

Appendix C

Implementation

The code for most of the implementation can be found at: `https://colab.research.google.com/drive/178l7oXOQR-W-yYmuS7M79LTm9aPJ5i6X`.

The main computational resource used for this work is the free GPU provided by Google Colab at: `https://colab.research.google.com`.