DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2019

Attribute-Driven Generation of Drug Reviews using Deep Learning

SYLWESTER LILJEGREN

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

Attribute-Driven Generation of Drug Reviews using Deep Learning

SYLWESTER LILJEGREN (syllil@kth.se)

Master Degree Project (DA231X) in Computer Science
Date: July 1, 2019
Supervisor: Jonas Moll
Examiner: Joakim Gustafson
School of Electrical Engineering and Computer Science
Host company: Omicron Ceti AB (Supervisor: Karin Wallén)
Swedish title: Attributdriven generering av läkemedelsrecensioner med djupinlärning

Abstract

In the last years, the demands on different models using deep learning to generate textual data conditionally have increased, where one would like to control what textual data to generate from a deep learning model. For this purpose, a couple of models have been developed and achieved state-of-art performance in the field of generating textual data conditionally. Therefore, the purpose of this study was to develop a new model that could outperform the relevant baseline models with respect to the BLEU metric. The alternative model combined some of the properties from the state-of-art models and was given the name the *Variational Attribute-to-Sequence decoder model* (shortened to the V-Att2Seq model) that paraphrases the name of one of the state-of-art models and "variational" refers to its application of variational recurrent autoencoders (VRAE). The data set used in this study contained drug reviews that were written by patients to express their opinion about the drug that they have used to treat a certain condition. The drug review texts were accompanied by the following attributes: the (name of the) drug, the condition, and the rating that the patient has given to the drug. The results in this study show that the V-Att2Seq model did not outperform all the baseline models, which concluded that the V-Att2Seq model did not satisfy the requirements imposed on the model itself. However, there are some future work that is suggested by this study to hopefully improve the performance of the V-Att2Seq model in the future such as including other mechanisms that are present in the state-of-art models, testing with e.g. other sizes and settings of the V-Att2Seq model, and testing different strategies for generating sequences since there is still potential that has been observed in the model that should be further investigated to improve its performance.

Sammanfattning

De senaste åren har efterfrågan på djupinlärningsmodeller, som villkorligt genererar textuell data, ökat då det önskas att kunna kontrollera vilka typer av data som genereras från en djupinlärningsmodell. För detta ändamål har ett par olika modeller tagits fram och uppnått bästa prestanda vad gäller att generera textuell data villkorligt. Syftet med denna studie var därmed att ta fram en modell som kunde prestera bättre än relevanta basmodeller med hänsyn till BLEU-måttet. Denna alternativa modell kombinerade några av egenskaperna hos de bästa modellerna och gavs namnet *Variationell Attribut-till-Sekvens-avkodarmodell* (förkortat till V-Att2Seq från den engelska benämningen *Variational Attribute-to-Sequence decoder model*), som parafraserar namnet på en av de bästa modellerna och där "variationell" syftar på modellens tillämpning av variationella autokodare. Datan som använts i denna studie innehöll läkemedelsrecensioner skrivna av patienter för att uttrycka sina åsikter om ett läkemedel som de använt för att behandla en viss sjukdom. Läkemedelsrecensionstexterna åtföljdes av följande attribut: (namnet på) läkemedel, sjukdom och betyget som patienten har gett till läkemedlet. Resultaten i denna studie visar att V-Att2Seq-modellen inte presterade bättre än samtliga basmodeller, vilket gjorde att slutsatsen om att V-Att2Seq-modellen inte tillfredsställde samtliga krav som fanns på denna modell drogs. Denna studie föreslår dock vidare arbete för att förhoppningsvis förbättra prestandan av V-Att2Seq-modellen i framtiden såsom att tillämpa andra mekanismer som återfinns hos de bästa modellerna, testa med bl.a. olika storlekar och inställningar av V-Att2Seq-modellen samt testa andra strategier för att generera sekvenser då det finns mycket potential, som har observerats kring denna modell, som borde vidare undersökas för att förbättra modellens prestanda.

Acknowledgements

I would like to thank all involved parties that have provided support and guidance that made it possible for me to finish my master thesis in spring 2019 at Royal Institute of Technology (KTH). More specifically, I would like to thank these following parties that altogether provided support and guidance:

- The hosting company Omicron Ceti AB, with Karin Wallén as my supervisor over there, for giving me the opportunity to do the master thesis that concerned a very interesting subject, and for giving me support whenever I needed to finish the master thesis.

- The supervisor Jonas Moll for supervising the course of the conduction of my master thesis by giving me very good constructive feedback that contributed to improving my master thesis, and for showing both patience and understanding when I ran into some issues that implied delays for delivery of report draft at certain occasions.

- The examiner Joakim Gustafson for giving me important insights about risks within certain moments of my master thesis that helped me to re-define certain conditions of the master thesis to increase the chances of finishing this master thesis with positive outcomes.

Beside these parties, I would like to thank all other that have helped with e.g. administration for helping me with proceeding my master thesis from the beginning until the end.

Contents

1 Introduction

In the past years, the interest of generating textual data in different contexts has steadily increased due to the enormous potential around this ability. Examples of this are generation of different reviews on various websites like Amazon [19], generation of headlines that summarize contents of various articles written by journalists [21], and generation of textual data to different scientists for their biography section at the Wikipedia web page [22]. Many companies have shown great interest for these generative abilities, since the application of these techniques could lead to both maximization of revenues in diverse ways and, simultaneously, reduce the need (and therefore the costs) of having a large human staff at disposal for performing corresponding tasks. However, the application of such an ability in the business world could simultaneously be ethically dubious as this ability could be exploited for e.g. generating textual data marketed as genuine to increase the degree of deception among textual data that is published on the Internet to the public, which could harm its trust in the information services that are provided.

A lot of research has been conducted to investigate different approaches to enhance the generative performance of various models for similar means. One common approach today is to implement some kind of recurrent neural network such as traditional recurrent neural network models or Long Short-Term Memory (LSTM) network models, which are trained on textual data and thereafter generate corresponding textual data by guessing the next e.g. word that will appear within some sentence provided previous words that have appeared. The LSTM models, especially, have succeeded at generally reaching good performance in modelling sequences, which includes generation of textual data [12]. However, there are some drawbacks with generating textual data with e.g. the LSTM models. One of the most significant drawbacks is that the generation of textual data through LSTM models is broken into next-step predictions and as a consequence, the models do not expose any interpretable representation of global features that are associated with the textual data [1]. This particular drawback could make the textual data generation more difficult if one wishes to generate particular textual data that are influenced by other features that are associated with the same data point as the concerned textual data. One attempt was made to solve this by the model as proposed by Dong et al. [5], where the generation of reviews on Amazon products was conditioned on the user, the product and the rating that the user has given to that product. Intuitively, all the attributes belonging to the user and the product, respectively, along with the given rating constituted the feature space of all user-product-rating tuples represented as vectors. These vectors were then transformed into sequential data that were passed to a LSTM model learning to transform the sequential data to a review that were strings of variable length. Another attempt was made in the study by Hu et al. [10], where an approach combined the use of variational recurrent autoencoders and a discriminator to produce texts that were conditioned on attributes that were associated to the same texts.

This study investigated an alternative model to the models that were proposed by Dong et al. [5] and Hu et al. [10], respectively, for the means of generating drug reviews regarding the drug used to treat a condition (a data set that contains information about the name of the drug, the name of the condition for which the drug was used, the rating of the patient to the drug and a review of the drug written by the patient was used for conduction of this study). The generative performance was measured with respect to a metric evaluating the lexical similarity between the actual textual data and the textual data generated from a model. The alternative model being investigated has some similarities to the model proposed by Dong et al. [5] since they both are making next-step predictions at each consecutive time step based on a pre-determined encoding of the relevant attributes and the previous predictions that were made at previous time steps to generate a suitable drug review. However, the alternative model being investigated applies *variational recurrent autoencoders* (VRAE) to transform any sequence (i.e. a drug review of variable length) into a hidden vector of fixed length, which in turn is combined with the encoded attributes to make a prediction for the current time step. This idea resembles the strategy that was applied for the model proposed by Hu et al. [10], where they have transformed the sequential data into a hidden vector that was combined with the relevant attributes, and which together were passed as input to a learning discriminator to generate sentences. Just like in the case of the model by Dong et al. [5], the predictions are continuously done by the alternative model until a termination token was predicted. Throughout the rest of this report, this alternative model will be referred to as the *variational attribute-to-sequence decoder* (V-Att2Seq) model, a name that paraphrases the name of the similar model that was proposed by Dong et al. [5] and where the word "variational" refers to its application of variational recurrent autoencoders. Since such a model has not yet been proposed in any previous study within this field, this therefore motivates the need of this study to investigate such a model empirically to bring further knowledge and another alternative for conditionally generating textual data in general.

1.1 Purpose and research question

The purpose for conduction of this study was to investigate whether the V-Att2Seq model for generating drug reviews could eventually outperform the baseline models that are applied for generating textual data in general. Therefore, the following research question served as a guideline throughout the entire study:

- *Can the V-Att2Seq model outperform the baseline models at generating drug reviews in terms of ability to generate real-world drug reviews?*

1.2 Scope

For this study, two baseline models were used to establish a comparison to the V-Att2Seq model. Those two baseline models were the LSTM networks and the Nearest Neighbor (NN) model. The first baseline model operates (as described in section 1) by making next-step predictions of tokens given the previous tokens that were predicted at previous time steps. This study considers a valid token to be either a single word and/or an allowed punctuation beside the initial- and termination tokens (i.e. "<s>" and "</s>", respectively, as in the study by Dong et al. [5]). The V-Att2Seq model also follows this interpretation when generating its own sequences of tokens. The second baseline model operates, provided some input drug review, by selecting a partition of its stored training data where the drug reviews have the same surrounding attributes as those of that input drug review and then randomly selects one of the associated drug review texts as its prediction. These two baseline models constitute a good standard to any other model that is proposed when it comes to conditionally generating textual data since similar baseline models have been used in previous studies in the same field.

All the models were evaluated using a metric called BLEU (shorthand for *bilingual evaluation understudy*) to assess the quality of the texts that were produced by each concerned model. The BLEU metric is also the most widely used metric when it comes to assessing the generative performance of models that generate textual data. The BLEU scores that were obtained for the models during testing were analyzed using statistical methods to provide an answer to the research question as formulated in section 1.1. The BLEU metric is explained further in section 2.3.

2 Theoretical background

In this section, an overview of the relevant concepts and theorems is given to provide to the reader a better understanding of the contents that are presented in this report. Firstly, the key concepts and theorems related to the V-Att2Seq model are introduced to make the reader aware of them. Secondly, the metric to be used in this study to compare the models to each other and the components enabling this measurement in this study are also presented. Finally, some insights about previous works that have been done by others in the same field that this study concerns are highlighted to give the reader a state-of-art picture of the relevant field.

2.1 Recurrent Neural Networks

A recurrent neural network is a family of artificial neural networks that consider the dependencies between data points that occur at different time steps, compared to traditional feed-forward neural networks that treat each data point as independent of other data points within the same data set. The fundamental idea behind recurrent neural networks is based on the unfolding of recurrent computations [7]. It generally refers to the operation of transforming a recurrent computation into an equivalent non-recurrent (but repetitive) computation, across which the hyper-parameters (referring to the associated weights in this case) are shared and tuned with respect to. Figure 1 shows how the principle of unfolding recurrent neural networks looks like in practice.

Based on the illustration of the unfolding principle of recurrent neural networks as provided in Figure 1, one gets the following formulas of the recurrent computations after unfolding of the recurrent computations (using matrix notation):

$$\mathbf{a}^{(t)} = \mathbf{b} + \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)} \tag{1}$$

$$\mathbf{h}^{(t)} = tanh(\mathbf{a}^{(t)}) \tag{2}$$

$$\mathbf{o}^{(t)} = \mathbf{c} + \mathbf{V}\mathbf{h}^{(t)} \tag{3}$$

$$\hat{y}^{(t)} = softmax(\mathbf{o}^{(t)}) \tag{4}$$

Furthermore, it could also be observed from Figure 1 that $y^{(t)}$ is the corresponding true value to which $\hat{y}^{(t)}$ is compared and $L = \sum_t L^{(t)}$ is the total loss between the sequences y and $\hat{y}$ across all time steps, where $L^{(t)}$ is the loss between $\hat{y}^{(t)}$ and $y^{(t)}$ at some time step t.

To train a recurrent neural network, one usually applies an adjusted version of the back-propagation, originally developed for the feed-forward neural networks, that is commonly called *back-propagation through time* (or, for short, BPTT). Based on Equations 1-4, the

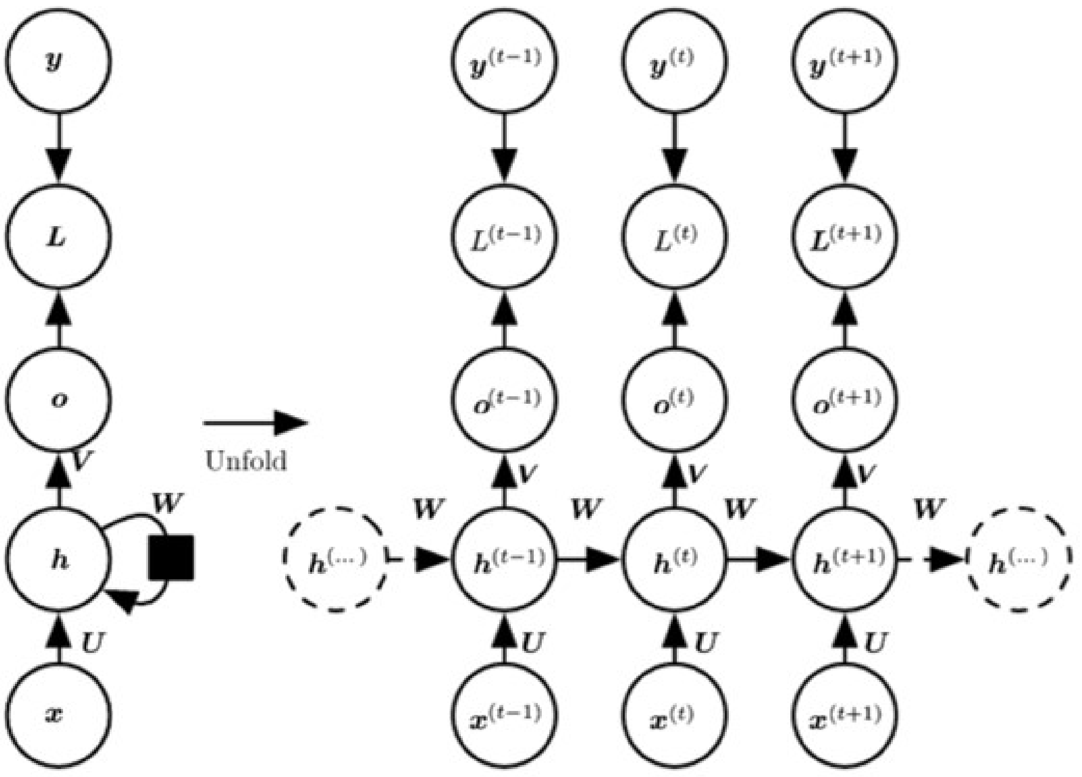


Figure 1: Illustration of how the principle of unfolding of recurrent neural networks [7] looks like in practice.

following equations are used for calculating the partial derivatives for the gradient to update all weights in $\mathbf{W}$, $\mathbf{U}$, and $\mathbf{V}$:

$$\frac{\partial L^{(t)}}{\partial w_i} = \sum_{k=1}^{t} \frac{\partial L^{(t)}}{\partial \hat{y}^{(t)}} \frac{\hat{y}^{(t)}}{\mathbf{o}^{(t)}} \frac{\partial \mathbf{o}^{(t)}}{\partial \mathbf{h}^{(t)}} \left(\prod_{j=k+1}^{t} \frac{\partial \mathbf{h}^{(j)}}{\partial \mathbf{a}^{(j)}} \frac{\partial \mathbf{a}^{(j)}}{\partial \mathbf{h}^{(j-1)}} \right) \frac{\partial \mathbf{h}^{(k)}}{\partial \mathbf{a}^{(k)}} \frac{\partial \mathbf{a}^{(k)}}{\partial w_i} \tag{5}$$

$$\frac{\partial L^{(t)}}{\partial u_i} = \sum_{k=1}^{t} \frac{\partial L^{(t)}}{\partial \hat{y}^{(t)}} \frac{\hat{y}^{(t)}}{\mathbf{o}^{(t)}} \frac{\partial \mathbf{o}^{(t)}}{\partial \mathbf{h}^{(t)}} \left(\prod_{j=k+1}^{t} \frac{\partial \mathbf{h}^{(j)}}{\partial \mathbf{a}^{(j)}} \frac{\partial \mathbf{a}^{(j)}}{\partial \mathbf{h}^{(j-1)}} \right) \frac{\partial \mathbf{h}^{(k)}}{\partial \mathbf{a}^{(k)}} \frac{\partial \mathbf{a}^{(k)}}{\partial u_i} \tag{6}$$

$$\frac{\partial L^{(t)}}{\partial v_i} = \sum_{k=1}^{t} \frac{\partial L^{(t)}}{\partial \hat{y}^{(t)}} \frac{\partial \hat{y}^{(t)}}{\partial \mathbf{o}^{(t)}} \frac{\partial \mathbf{o}^{(t)}}{\partial v_i} \tag{7}$$

Equations 5-7 are obtained using the chain rule for computing the (partial) derivatives of functions that are composed by other functions using all Equations 1-4, as previously stated. One could further observe that all the weights are re-used through the entire time spectrum, which is the most common practice to design recurrent neural networks since it controls the level of complexity within the model to minimize the risk of overfitting the model when training it on a data set.

Recurrent neural networks have become very useful for many application areas such as time series analysis and language modelling. However, there is one significant drawback that usually arises with recurrent neural networks when modelling larger sequences and that is the one that is related to the problem of exploding- and vanishing gradients. In case of exploding gradients, the recurrent neural network will end up with too large gradients (due to large derivatives being multiplied with each other), which could prevent the recurrent neural network to converge towards a minimum point in the error landscape with respect to the tuned weights. In case of vanishing gradients, the gradients will close to zero leading to no update of the weights at all (due to small derivatives being multiplied with each other). What is more remarkable in case of vanishing gradients is that the recurrent neural network becomes less capable of modelling the long-range key dependencies between data points that are farther from each other in time, which also could limit the performance of the recurrent neural network in the long run. To alleviate this drawback of the recurrent neural network, one could use the LSTM network instead. The LSTM network was proposed by Hochreiter and Schmidhuber in 1997 to alleviate the specified drawback with recurrent neural networks [9]. With time, the LSTM network has ultimately outperformed the recurrent neural network and has become a state-of-art-approach within diverse application areas.

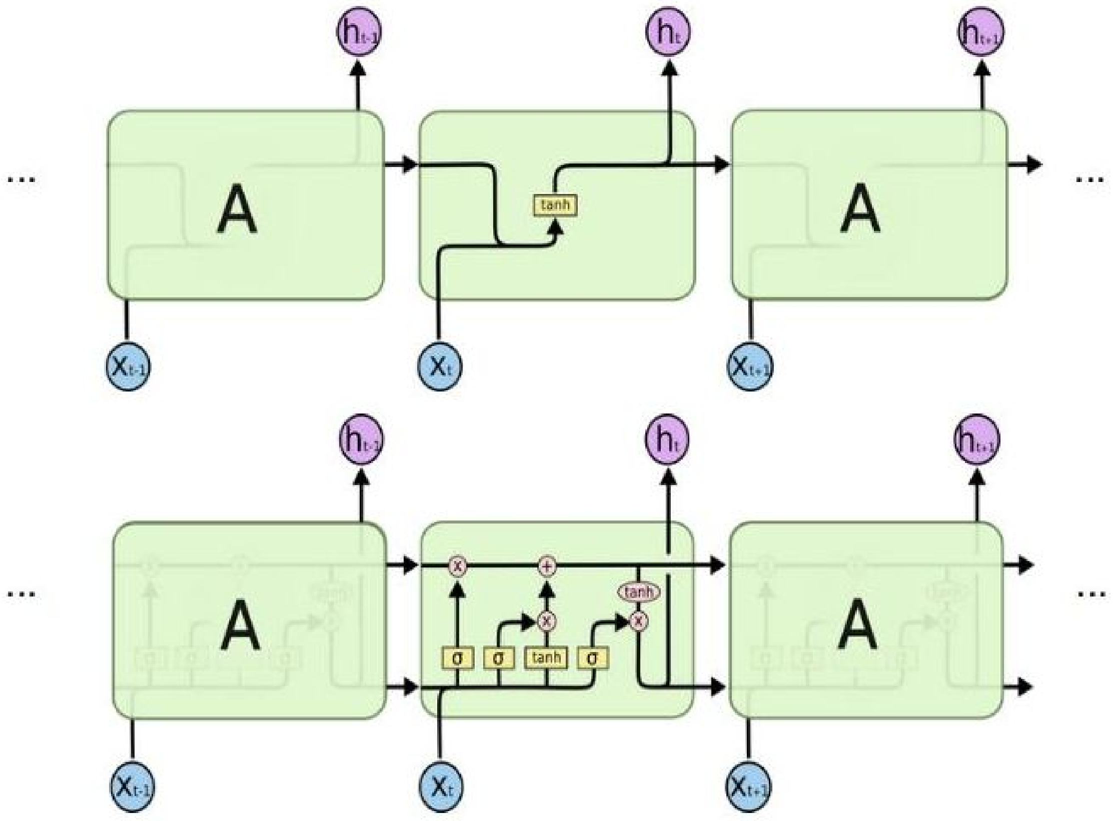


Figure 2: The upper figure depicts an architecture that describes the cells (A), the content within each cell and their inter-dependencies within a traditional recurrent neural network, whereas the lower figure depicts an architecture that describes corresponding cells, content within each cell and their inter-dependencies within a LSTM network [15].

2.1.1 Long Short-Term Memory

The Long Short-Term Memory (LSTM) network is a specific type of recurrent neural network [15] (sometimes known as a "gated" recurrent neural network) which is extensively used in various deep learning fields. The LSTM network distinguishes from the traditional recurrent neural networks by introducing some gates that are associated with each memory cell within a LSTM network. Please consider Figure 2 to obtain an overview over the architecture of the LSTM cell and also how it distinguishes itself from corresponding cells within traditional recurrent neural networks.

While there is only one computational layer within each cell of a recurrent neural network (i.e. the layer in which the tanh activation is computed based on the linear combination of weights and input), then there are in total *four* computational layers within each cell of a LSTM network. In each of those cells at some time step t, the following computations are performed:

1) In the first layer, the output f_t from the *forget gate* is computed, which intuitively is relevant in deciding what past information should be thrown out or not. Therefore, the following formula is used:

$$f_t = \sigma(\mathbf{W}_f[\mathbf{h}_{t-1}, \mathbf{x}_t] + b_f) \tag{8}$$

   where $[A, B]$ denotes concatenation of A and B. Furthermore, $\mathbf{W}_f$ and b_f denote the weights and biases, respectively, that are associated with the forget gate as its parameters. The corresponding notation will be applied to the upcoming procedures in describing the computational flow within a cell in a LSTM network.

2) In the second layer, two different outputs are to be computed: one is i_t from the *input gate* and the other one is $\tilde{C}$ denoting the candidate state of the entire cell. The input gate in a cell of a LSTM network is responsible for deciding which values within the current cell should be updated and after deciding which values to update, the candidate values that $\tilde{C}$ suggests will be stored into those values that were pointed out to be updated (which is done in the third layer). This leads to the following formulas to be used in this layer:

$$i_t = \sigma(\mathbf{W}_i[\mathbf{h}_{t-1}, \mathbf{x}_t] + b_i) \tag{9}$$
$$\tilde{C} = \tanh(\mathbf{W}_C[\mathbf{h}_{t-1}, \mathbf{x}_t] + b_C) \tag{10}$$

3) In the third layer, it is time to update the cell state by computing C_t based on the previous cell state C_{t-1} and all the previously computed outputs in previous layers during the current time step t. This implies Equation 11 which is used for computing the updated cell state C_t:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{11}$$

4) Finally, in the fourth layer, the output from the *output gate* is computed and used to select all the values to output and pass to another cell in next time step. The output gate is responsible to select which values that should be outputted by the current cell given its updated state. Here are the formulas that are used for computing relevant outputs in this layer:

$$\mathbf{o}_t = \sigma(\mathbf{W}_\mathbf{o}[\mathbf{h}_{t-1}, \mathbf{x}_t] + b_\mathbf{o}) \tag{12}$$
$$\mathbf{h}_t = \mathbf{o}_t * \tanh(C_t) \tag{13}$$

The training of LSTM networks is done using the BPTT algorithm (just like in the case of traditional recurrent neural networks) and the update gradients for all the weights and biases as observable in Equations 8-13 are obtained using the chain rule for (partial) derivatives in the same way as correspondingly for the recurrent neural network.

It is worth emphasizing that there are many different variations of the underlying architecture behind the LSTM network. The one described in this study is one of the simpler approaches that is usually introduced when obtaining knowledge about LSTM networks in general. The other variants on the architecture of the LSTM network could refer to e.g. different connection/paths, compositions of calculations and other similar details that are relevant for the overall computation within each cell. One example of different LSTM network architectures as opposed to the one introduced in this section is the one that was proposed by Gers and Schmidhuber by adding so called "peephole connections" [6] that make the computation of outputs from different gates (forget gate, input gate and output gate) also consider the previous cell state C_{t-1}. Another example is the Gated Recurrent Unit network proposed by Cho et al. [2] in 2014, which is a simplification of the LSTM network since the forget gate and the input gate are combined into one single gate which is called the *update gate*.

2.2 Autoencoders

The autoencoder is a type of feed-forward artificial neural network that learns efficient data codings in an unsupervised manner [25]. Usually, the autoencoder aims to learn a lower-dimensional representation of the input data to e.g. remove noise that occurs within the original data, reduce the dimensionality of the data in order to alleviate presumptive problems related to the curse of dimensionality in some contexts, or learn generative models of the data upon which it is trained.

The general architecture of an autoencoder usually consists of an input layer, a hidden layer (sometimes called a *bottleneck*), and an output layer, where the size of the hidden layer is less than the input/output layer (with some exceptions, e.g. sparse autoencoders). Figure 3 illustrates the fundamental components around the architecture of the autoencoders in general. It is observable from Figure 3 that there are two operational computations that are enabled within such an autoencoder, which are the encoder function $\phi : \mathcal{X} \rightarrow \mathcal{H}$ and the decoder function $\psi : \mathcal{H} \rightarrow \mathcal{X}$. The encoder function is responsible for transforming the input $\mathbf{x}$ into another encoding $\mathbf{h}$ described by units in the hidden layer, whereas the decoder function is responsible for (attempting at) transforming the hidden units back to the original input x, i.e. it performs a reconstruction of the original input $\mathbf{x}^{'}$ based on the hidden units to which the original input $\mathbf{x}$ was transformed by the encoder function. The objective function for an autoencoder is to properly select functions ϕ and ψ such that:

$$\phi, \psi = \underset{\phi,\psi}{\operatorname{argmin}} \, ||\mathbf{x} - (\psi \circ \phi)(\mathbf{x})||^2 \tag{14}$$

where $(A \circ B)(x)$ denotes function composition, i.e. $A(B(x))$. Assuming that the weights matrix W (with bias included) is a hyperparameter of the encoder function, and the weights matrix V (with bias included) is a hyperparameter of the decoder function, $\mathbf{x}^{'}$, both the encoder- and decoder function involves two computational steps: 1) compute the linear combination of the input features and the associated weights for each neuron (i.e. $W[\mathbf{x}, 1]$ and $V[\mathbf{h}, 1]$ for the encoder- and decoder function, respectively), and 2) compute the activation (e.g. sigmoid, logistic, ReLU and so forth) of the linear combination to obtain the final values (i.e. to obtain hidden units from the input, or conversely).

Autoencoders are trained using the traditional back-propagation algorithm. However, due to problems related to vanishing gradients when developing deeper versions of autoencoders, it is usually pretrained using e.g. greedy layer-wise pretraining procedure to greedily find initial optimal weights before being ultimately tuned with the back-propagation algorithm to obtain optimal autoencoders.

Although autoencoders are useful within various applications, they are prone to overfitting since they tend to easily learn the identity function and therefore only copies over the input to the output without actually learning a hidden representation of the data, making it less able to generalize its knowledge to un-seen inputs [25]. This has led to proposals of many different versions of autoencoders that resolve this problem, which includes denoising autoencoders, sparse autoencoders, variational autoencoders, contractive autoencoders and so forth. The variational autoencoders, especially, are very relevant for this study.

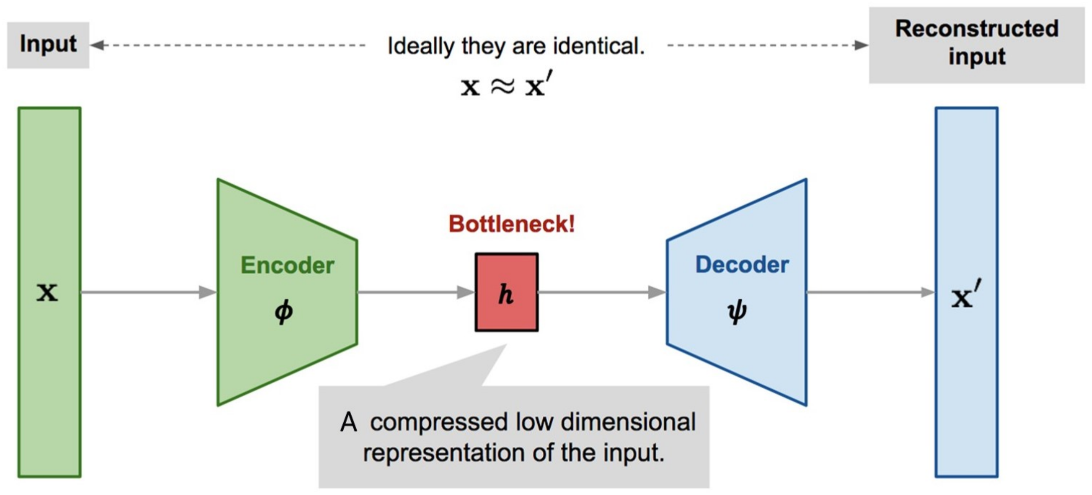

Input
Ideally they are identical.
$\mathbf{x} \approx \mathbf{x}'$
Reconstructed input
$\mathbf{x}$
Encoder
ϕ
Bottleneck!
h
Decoder
ψ
$\mathbf{x}'$
A compressed low dimensional representation of the input.

Figure 3: An illustration of the common architecture of autoencoders in general.

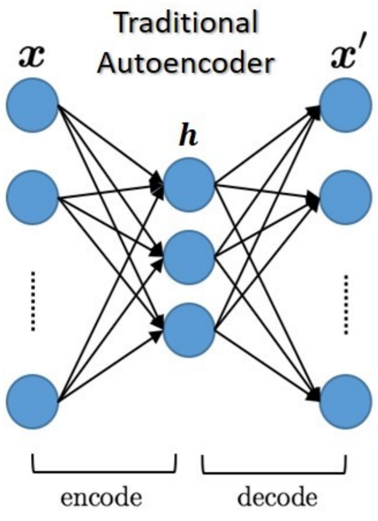
2.2.1 Variational Autoencoders

The variational autoencoder is a special type of autoencoder that aims to learn distributions over hidden vectors based on linear combinations of the input features and the corresponding weights [4]. Therefore, instead of learning to map inputs to hidden vectors, the variational autoencoder learns rather to map the inputs to the hyperparamaters that are associated with the distribution over the hidden vectors. Please consider Figure 4 to obtain an overview over the architectural difference between the traditional autoencoder (as presented previously) and the variational autoencoder. It could be observed from Figure 4 that the encoder within the variational autoencoder aims to learn to map the inputs to the hyperparameters of the distribution over the hidden vectors, whereas the associated decoder learns to map a sampled hidden vector to outputs. One additional detail to emphasize is that in Figure 4, it is assumed that the hidden vectors follow the normal distribution (based on the presence of the mean μ and the variance σ^2 in the architecture of the variational autoencoder) that the encoder function aims to learn based on the input that it receives from the input layer.

Compared to the objective function of the traditional autoencoder as defined by Equation 14, the objective function of the variational autoencoder is defined in the following way:

$$\log P(X) - \mathcal{D}[Q(z)||P(z|X)] = E_{z\sim Q}[\log P(X|z)] - \mathcal{D}[Q(z|X)||P(z)] \tag{15}$$

where $\mathcal{D}[P(x)||Q(x)] = E_{x\sim P}[\log P(x) - \log Q(x)]$ is defined to be the Kullback-Leibler divergence, which measures the difference between two probabilistic distributions P and Q, X the original input and z the associated hidden vector of the input X. Provided the formulation of the objective function as defined by Equation 15 above, a variational autoencoder is optimized by maximizing $\log P(X) = E_{z\sim Q}[\log P(X|z)]$ and minimizing $\mathcal{D}[Q(z)||P(z|X)] = \mathcal{D}[Q(z|X)||P(z)]$ simultaneously. Keep in mind first that $Q(X)$ is a probability distribution that produces some hidden vector z provided some input X and $P(z)$ is another probability distribution that produces a reconstruction of the provided input $X^{'}$ provided some hidden vector z (one could here observe an analogy between these functions to corresponding encoder- and decoder functions that were presented for the autoencoders previously, where in this case $Q(\cdot)$ is the "encoder" function entity and $P(\cdot)$ is the "decoder" function entity). The idea behind this objective function for variational autoencoders is to compare the corresponding distributions of both $Q(\cdot)$ and $P(\cdot)$ when generating the same samples, e.g. to see that both the "encoder"- and "decoder" distributions do not differ too much from each other when generating the same samples (otherwise, if they are not modelling corresponding distributions, then a discrepancy will exist between them making the variational autoencoder less able to reconstruct the original inputs). Usually, variational autoencoders are trained using the right hand-side of Equation 15, to which e.g. stochastic gradient descent along with the backpropagation algorithm is applied. One potential problem is that the combination of stochastic gradient descent and the backpropagation algorithm cannot be successfully



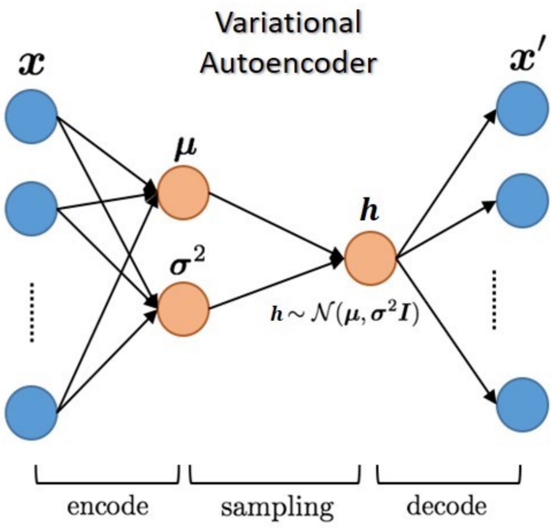


Figure 4: A comparison between a traditional autoencoder (left figure) and a variational autoencoder (right figure) with respect to architecture

applied whenever the units within the hidden layer are *stochastic* (although it can handle stochastic inputs and outputs). To overcome this problem, one usually applies the "re-parametrization trick" which is to move the sampling procedure from the hidden layer to the input layer. This would make it possible to apply stochastic gradient descent and the backpropagation algorithm successfully in the same way as for the traditional autoencoders [4].

2.3 BLEU metric

BLEU (shorthand for *bilingual evaluation understudy*) is a metric that was originally developed by Papineni et al. [16] in 2002 for assessing the quality of a translation, that was made by a machine, between two different languages (e.g. Chinese and English). The underlying idea behind the BLEU metric is that there are many different ways of translating e.g. a sentence from one language into a sentence in another language. The translations of one same sentence from one language to another language usually distinguish from each other through e.g. the use of different vocabulary, the sequence of the words and so forth. However, although there are many different ways of translating e.g. one sentence from one language to another language, one could realize that some translations are worse than other translations of the same sentence. With respect to this background, the BLEU metric aims to measure how well an arbitrary translation of a sentence is with respect to other accepted translations that are used as references.

The formula for computing the BLEU score, $BLEU$, based on an input translation T_i using a set of reference translations $\mathbf{T}_r = \{T_{r1}, T_{r2}, ..., T_{rm}\}$ is as following:

$$p_n = \frac{\sum_{n\text{-gram} \in T_i} Count_{clip}(n\text{-gram})}{\sum_{n\text{-gram} \in T_i} Count(n\text{-gram})} \tag{16}$$

$$BP = \begin{cases} 1 & \text{if } c > r \\ e^{(1-c/r)} & \text{if } c \leq r \end{cases} \tag{17}$$

$$BLEU = BP * \exp\left(\sum_{n=1}^{N} w_n \log(p_n)\right) \tag{18}$$

where the weights w_i are (commonly equal) positive floating numbers such that $\sum_{n=1}^{N} w_n = 1$, and r is the total length of the closest reference translation $T_{rj} \in \mathbf{T}_r$ (i.e. the reference translation T_{rj} with a length that is most equal to the length of the candidate translation T_i) and c the total length of the candidate translation T_i. The value p_n indicates the fraction of n-grams in a candidate translation T_i that are contained within the respective reference translation to which the candidate translation T_i is compared. The function $Count_{clip}(\cdot)$ is a function that operates on a set of counts of each n-gram associated with the candidate translation T_i, where the maximum count for each such n-gram is limited by the corresponding count for the same n-grams within the reference translations in $\mathbf{T}_r$. Practically, it means that whenever the count of some n-gram is larger than the corresponding count within the reference translations in $\mathbf{T}_r$, then that count value will be modified to be the same count value that prevails within the reference translations in $\mathbf{T}_r$. In this way, the calculations when computing p_n will penalize words that occur in the candidate translation T_i significantly more than correspondingly within the reference translations. Furthermore, BP is the translation brevity penalty factor, whose purpose is to make sure that too short candidate translations are being penalized whenever $c \leq r$. The reason why $BP = 1$ whenever $c > r$ is because it is already being handled through the computation of p_n due to the n-gram count clipping procedure as previously explained and to avoid calculative redundancy, $BP = 1$ is set whenever $c > r$.

The range of the BLEU score is $BLEU \in [0, 1]$, where $BLEU = 0$ indicates that the translation T_i has nothing in common with any of the reference translations in $\mathbf{T}_r$ (which would indicate that the translation is not good) and $BLEU = 1$ indicates that the translation T_i matches up to one of the reference translations in $\mathbf{T}_r$ perfectly (which would indicate that the translation is good).

BLEU is currently the most widely used metric for assessing the quality of texts that have been generated by different machines/algorithms due to its empirically stated high correlation with the human assessment of the quality of translations in general and is also relatively inexpensive to compute [16]. However, there are newer variants of BLEU that have been developed such as NIST, where e.g. another distribution of the weights across n-grams is being used (where the less frequent n-grams are given higher weights than the more frequent n-grams) and another formula for calculating the corresponding brevity penalty factor is re-formulated and applied, as an alternative metric to BLEU [3].

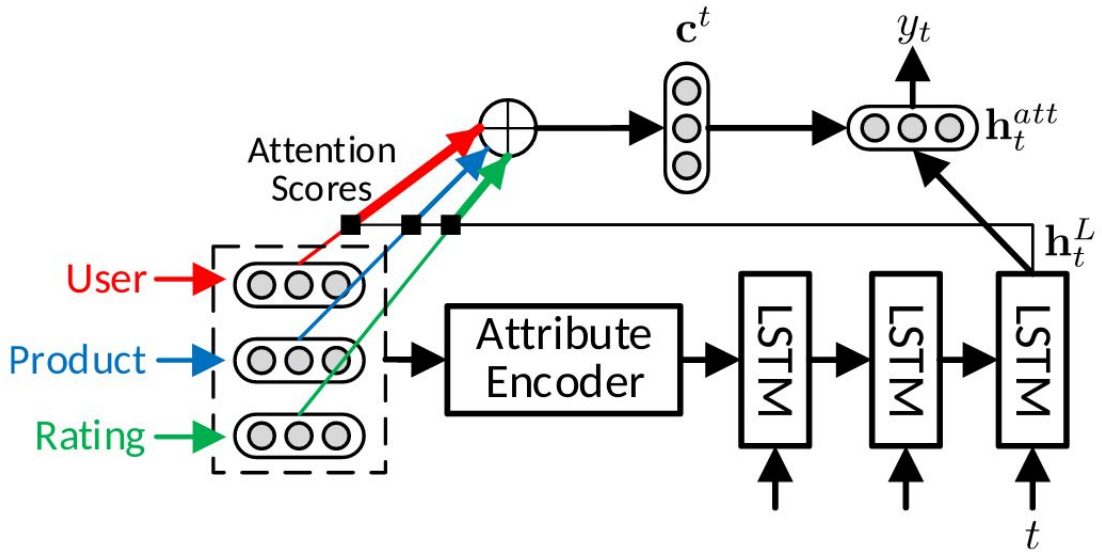


Figure 5: The final architecture of the model proposed by Dong et al. [5]

2.4 Related work

As already hinted in section 1, some efforts have been put by other people in the past to investigate the ability of generating textual data with respect to surrounding attributes (i.e. generating conditionally), using other models than e.g. recurrent neural networks and LSTM networks due to their inability to generate textual data conditionally. For the means of generating certain textual data conditionally, there are two models that were proposed in two different studies (Dong et al. [5] and Hu et al. [10]). The first model proposed in [5] is built on two main phases in mapping attributes to some textual data: 1) encode the attributes into a specific format (and use them to initialize the hidden vectors at the first time step of a LSTM model), and 2) predict the next token at a current time step using the predictions at previous time steps, until a termination token was predicted (using the initialized hidden vectors at the first time step that contains the information about the attributes). Please consider Figure 5 for overview of this architecture.

However, the same study has shown that the performance of the proposed model was further enhanced by introducing the so called *attention mechanisms*, whose purpose was to better utilize the encoder information which would help the model to be better at managing larger sequences in general. This model was applied to the Amazon data set that contained reviews and the associated meta data, where duplicates were removed and the maximum length of the reviews was set to 60. Each review was paired with the user's ID, the product's ID and the rating that the user has given to a product. At the end, the final data set (after imposing the said limitations) contained 937,033 reviews with the said attributes. The other models beside the one as illustrated in Figure 5 included

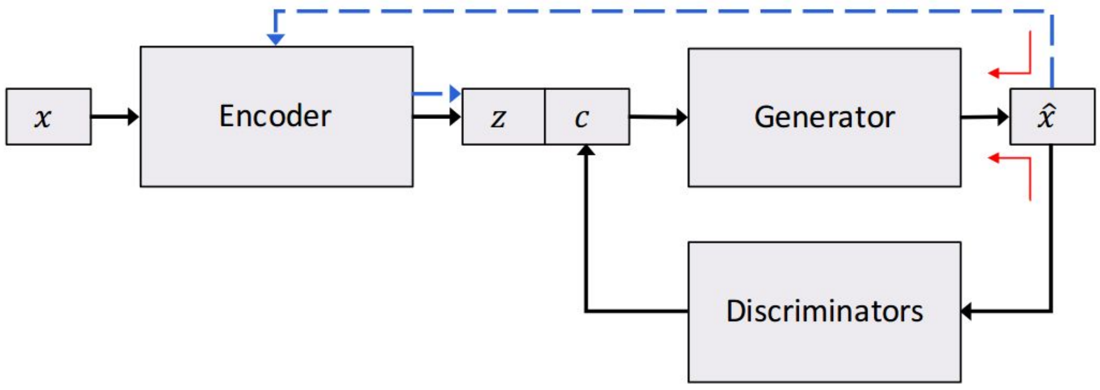


Figure 6: The architecture of the model proposed by Hu et al. in [10]

selecting random reviews, Maximum Entropy Language Model, Nearest Neighbors model (either using product's ID and rating or user's ID and rating as attributes) where a random review among other samples that have the same attribute values is selected, and the same architecture as illustrated in Figure 5 without the attention mechanisms (i.e. attention scores). Based on e.g. BLEU metric, the model as illustrated in Figure 5 performed the best in the experiments.

The other model that was proposed by Hu et al. [10] is based on combining the variational recurrent autoencoders and discriminator learning altogether to conditionally generate textual data. For an overview of the architecture, please consider Figure 6.

The model by Hu et al. [10] initially transforms texts to a vector representation with fixed length, which would make the features in such a vector, hopefully well associated with the input text, be able to become incorporated together with surrounding non-textual attributes, thus creating a fixed feature space of the data set. This is done using a recurrent version of variational autoencoders, which was previously explored in [1] about how to perform transformation between an input text of variable length and a continuous vector space of fixed size. Thus, provided the vector $\mathbf{z}$ (to which an input text x was transformed) and surrounding attributes $\mathbf{c}$, a reconstruction of the original input $\hat{x}$ is performed (based on the attributes in $(\mathbf{z}, \mathbf{c})$) and compared to the original input x. The role of the discriminator in this model is to infer and evaluate the error between the real attributes $\mathbf{c}$ and the corresponding predicted attributes $\hat{\mathbf{c}}$ based on the reconstruction of the original input, $\hat{x}$, since this is believed to penalize generation of sentences that do not correspond to the surrounding attributes. As the study concerning this model [10] reveals, all the generative models were trained using a large IMDB text corpus that contained approximately 350K movie reviews. Two modifications, though, were performed on this data set: 1) all sentences containing more than 15 words were excluded, and 2) all the infrequent words were replaced with "<unk>". This leads to a resulting data set that contained approximately 1.4M sentences. The test data includes the following data sets: 1) Stanford Sentiment Treebank-2 (full size), 2) Stanford

Sentiment Treebank-2 (small size), 3) Lexicon, and 4) IMDB. Each of these data sets contained some kind of reviews that were associated with certain attributes, e.g. movie reviews that were labelled as either positive or negative. The model was compared to semi-supervised variational autoencoder (S-VAE), which is able to conditionally generate sentences given some attributes and unlike the proposed model, it does not include any discriminator. The experiments suggest that the proposed model outperforms S-VAE on all testing data sets. Both the texts generated by these models were evaluated using a trained sentiment classifier and if the sentiment classifier could perform well on a certain set of texts generated by one model compared to another model, this would indicate that the first model produces better texts than the other one.

Further models for generating textual data conditionally include e.g. those proposed in the studies by Wiseman et al. [27] and Subramanian et al. [23], respectively. The model proposed by Wiseman et al. [27] makes use of a Hidden Markov Model decoder that learns discrete templates that describe the underlying structure behind the textual data of concern, whereas the other model proposed by Subramanian et al. [23] makes use of generative adversarial neural networks (GANs) together with conditional recurrent neural networks to generate textual data. However, they concern use case scenarios that are far from the corresponding use case scenarios that this study concerns. Despite this fact, they deserve to be mentioned along with other works that have been conducted in the field of conditionally generating textual data.

3 Method

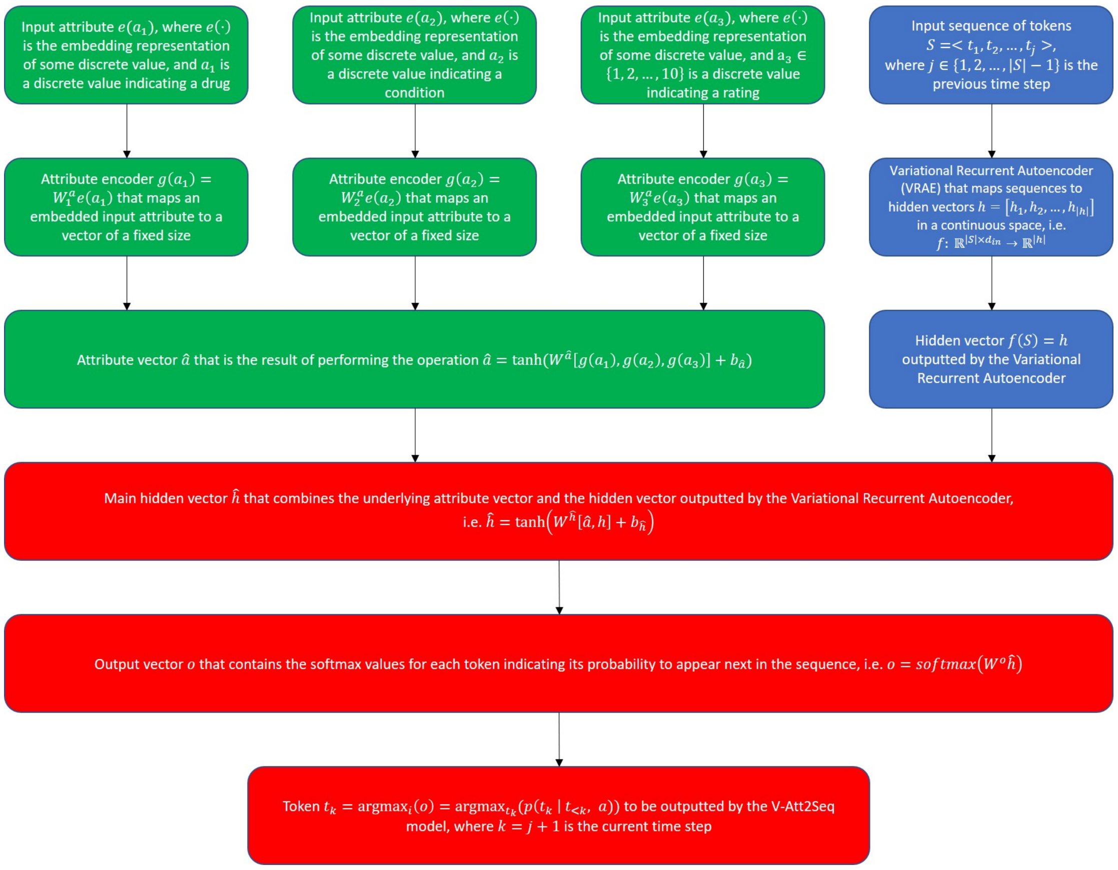
This section aims to provide all the relevant details regarding the conduction of the experiments and how the results from those experiments will be used and interpreted to give an answer to the research question as formulated in section 1.1. This includes a description of the V-Att2Seq model, a description of the data set and the pre-processing strategy used for the experiments, a description of the experiments themselves and a description of a statistical hypothesis based on the results that were obtained from the experiments which will ultimately provide the final answer to the specified research question. Finally, all the necessary information about chosen architecture and associated hyper-parameters for each concerned model to produce the results in section 4 is presented to the reader in an own section.

3.1 Variational Attribute-to-Sequence decoder model

The V-Att2Seq model is heavily inspired by the model which was proposed by Dong et al. [5], which predicts one token at a time in a sequence representing a drug review to output from the model based on the attributes that are provided to the model. The V-Att2Seq model used in this study also works by predicting one token at each consecutive time step of a sequence that represents a drug review. The V-Att2Seq model, however, delivers the predictions of those tokens by using different mechanisms: while the model by Dong et al. [5] initializes the hidden vectors of the LSTM sequence decoder at $t = 0$ with encoded attribute vectors and applies attention mechanisms to make the model to better utilize the encoder information, the V-Att2Seq model instead applies variational recurrent autoencoders that are responsible for mapping sequences to a vector in a continuous space. This vector is subsequently paired with the encoded attribute vector that both constitute a larger combined vector which is, through an intermediate hidden layer (that applies tanh activation), provided to a softmax activation function that ultimately computes the probability for each available token to be the next one in the sequence. Figure 7 illustrates a high-level scheme over the architecture behind the V-Att2Seq model. The idea of combining the hidden vector and the surrounding attributes, upon which the direct predictions are conditioned, comes from the model proposed by Hu et al. [10], where the hidden vector was combined with the attributes which were later passed to a discriminator that contributed to generating sequences.

When applying the V-Att2Seq model for the means of generating drug reviews, there are three main phases in the V-Att2Seq model that are performed routinely: 1) processing the individual attribute inputs altogether (as the green boxes in Figure 7 illustrate) to an attribute vector, 2) transforming sequences of variable length to a hidden vector, which consists of some fixed number of units (as the blue boxes in Figure 7 illustrate) and 3) computing the probabilities of each available token to be the next one in the current sequence (as the red boxes in Figure 7 illustrate) using both the attribute vector from

1) and the hidden vector from 2) that were combined in prior to the calculations of the probabilities for each available token. These three main phases will be explained further in their own respective subsections to give the reader a better understanding of the underlying mechanisms behind the V-Att2Seq model.



Input attribute $e(a_1)$, where $e(\cdot)$ is the embedding representation of some discrete value, and a_1 is a discrete value indicating a drug
Input attribute $e(a_2)$, where $e(\cdot)$ is the embedding representation of some discrete value, and a_2 is a discrete value indicating a condition
Input attribute $e(a_3)$, where $e(\cdot)$ is the embedding representation of some discrete value, and $a_3 \in \{1, 2, \ldots, 10\}$ is a discrete value indicating a rating
Input sequence of tokens $S = < t_1, t_2, \ldots, t_j >$, where $j \in \{1, 2, \ldots, |S| - 1\}$ is the previous time step
Attribute encoder $g(a_1) = W_1^a e(a_1)$ that maps an embedded input attribute to a vector of a fixed size
Attribute encoder $g(a_2) = W_2^a e(a_2)$ that maps an embedded input attribute to a vector of a fixed size
Attribute encoder $g(a_3) = W_3^a e(a_3)$ that maps an embedded input attribute to a vector of a fixed size
Variational Recurrent Autoencoder (VRAE) that maps sequences to hidden vectors $h = [h_1, h_2, \ldots, h_{|h|}]$ in a continuous space, i.e. $f\colon \mathbb{R}^{|S| \times d_{in}} \rightarrow \mathbb{R}^{|h|}$
Attribute vector $\hat{a}$ that is the result of performing the operation $\hat{a} = \tanh(W^{\hat{a}}[g(a_1), g(a_2), g(a_3)] + b_{\hat{a}})$
Hidden vector $f(S) = h$ outputted by the Variational Recurrent Autoencoder
Main hidden vector $\hat{h}$ that combines the underlying attribute vector and the hidden vector outputted by the Variational Recurrent Autoencoder, i.e. $\hat{h} = \tanh\left(W^{\hat{h}}[\hat{a}, h] + b_{\hat{h}}\right)$
Output vector o that contains the softmax values for each token indicating its probability to appear next in the sequence, i.e. $o = softmax\left(W^o \hat{h}\right)$
Token $t_k = \text{argmax}_i(o) = \text{argmax}_{t_k}(p(t_k \mid t_{<k},\ a))$ to be outputted by the V-Att2Seq model, where $k = j + 1$ is the current time step

Figure 7: The scheme over the architecture behind the V-Att2Seq model for used in this study. The green boxes indicate the part of the model that is dedicated to process the input attributes that are passed to the model and the blue boxes indicate the part that is dedicated to process the current sequence. Ultimately, the red boxes indicate the part of the model that is dedicated to merge and combine the vector features using the processed input attributes and hidden vectors before computing the probabilities of each token to be the next one in the current sequence and make a final prediction by selecting that token that maximizes the probability of being the next in the sequence.

3.1.1 Processing of input attributes

The mechanisms for processing all relevant input attributes into a larger hidden attribute vector as depicted among the green boxes in Figure 7 are inspired by the corresponding mechanisms that were involved in the model by Dong et al. [5]. Since no other alternative seems to exist that could be better than the mechanisms as used in the study by Dong et al. [5], the same formulations and computations that were performed within the model by Dong et al. [5] for processing all relevant input attributes were to a large extent used by the V-Att2Seq model in this study.

Therefore, assume that some set of input attributes $\mathbf{a} = [a_1, a_2, ..., a_{|\mathbf{a}|}]$ is provided. For this study, there are three attributes that are considered when generating drug reviews: the drug, the condition and the rating, which implies that $|\mathbf{a}| = 3$. Firstly, the individual hidden vectors of the input attributes a_i are calculated using $g(a_i) = W_i^a e(a_i)$, where $e(a_i)$ is the embedded representation (of fixed size n_e) of the input attribute a_i, $W_i^a \in \mathbb{R}^{m \times n_e}$ is the parameter weights matrix, and m is the dimension of the encoding of the concerned input attribute. The encoding of each input attribute is learned using a traditional multilayer perceptron (MLP) with one hidden layer that transforms an input attribute into some encoding. Next, after successfully encoding each respective input attribute, all these encoded input attributes are combined into one larger hidden vector by computing $\hat{\mathbf{a}} = \tanh(W^{\hat{\mathbf{a}}}[g(a_1), g(a_2), ..., g(a_{|\mathbf{a}|})] + b_{\hat{\mathbf{a}}})$, where $[g(a_1), g(a_2), ..., g(a_{|\mathbf{a}|})]$ are concatenated attribute vectors, $W^{\hat{\mathbf{a}}} \in \mathbb{R}^{|\hat{\mathbf{a}}| \times |\mathbf{a}|m}$ the parameter weights matrix, and $b_{\hat{\mathbf{a}}} \in \mathbb{R}^{|\hat{\mathbf{a}}|}$ the associated biases. After producing the attribute vector $\hat{\mathbf{a}}$, the processing of the input attributes is considered to be done, which allows the model to proceed to transforming relevant sequential data into hidden vectors.

3.1.2 Transformation of sequences to hidden vectors

Assuming that some input string $S =< t_1, t_2, ..., t_{|S|} >$ (where t_i is a token at position i in the string S) is provided, the general idea behind the application of the variational recurrent autoencoder is to reconstruct that input string by outputting $\hat{S} =< \hat{t_1}, \hat{t_2}, ..., \hat{t}_{|\hat{S}|} >$ such that $S \approx \hat{S}$. During the computation, the input string is encoded into a hidden vector representation $\mathbf{h} = [h_1, h_2, ..., h_{|\mathbf{h}|}]$, where $|\mathbf{h}|$ is kept fixed while the lengths of the input- and output strings, $|S|$ and $|\hat{S}|$, are variable (worth noticing is that it must *not* necessarily always be the case that $|S| = |\hat{S}|$).

Both the encoder- and decoder networks are some recurrent neural networks that transform sequential data (strings in this case) to/from the hidden vector representation. In this study, both these networks were selected to be LSTM networks to facilitate the overall generative performance of the V-Att2Seq model due to the good generative performance that has been observed in other previous studies concerning LSTM networks. One could further realized by e.g. inspecting Figure 4 that there are two components, namely the mean and the standard deviation, which have the purpose of modelling the statistical distribution over the hidden vectors provided some input string. Using this architecture, it implies that the units in the hidden vectors are $\mathcal{N}(\mu, \sigma^2 I)$-distributed, which is a standard assumption that is made by the variational autoencoders in general due to various reasons. The reason why the probabilistic components of the variational autoencoders could be useful in this case is because that they will help at preventing possible overfitting of the autoencoder, since there will be some inherent "noise" within the outputs of the autoencoder due to the sampling from the statistical distribution over the units in the hidden vector that will make it more difficult for the concerned autoencoder to learn to merely copy the input sequence over to the output sequence. This will force the autoencoder to actually learn some concrete patterns that could help the autoencoder to perform better at generating data that has not been considered by the autoencoder during the training phase. In the long run, this will improve the generalization capabilities of the concerned autoencoder, which has a positive impact on the overall generative performance of the V-Att2Seq model.

3.1.3 Prediction of the next token provided the attribute vector and the hidden vector

Provided the attribute vector $\hat{\mathbf{a}}$ and the hidden vector $\mathbf{h}$ that were computed accordingly to the instructions in section 3.1.1 and 3.1.2, respectively, these vectors are combined into another vector $\hat{\mathbf{h}} = \tanh(W^{\hat{\mathbf{h}}}[\hat{\mathbf{a}}, \mathbf{h}] + b_{\hat{\mathbf{h}}})$, where $W^{\hat{\mathbf{h}}} \in \mathbb{R}^{|\hat{\mathbf{h}}| \times (|\hat{\mathbf{a}}| + |\mathbf{h}|)}$ is the associated parameter weights matrix, and $b_{\hat{\mathbf{h}}} \in \mathbb{R}^{|\hat{\mathbf{h}}|}$ the associated biases. The primary reason why $\hat{\mathbf{h}}$ is computed and used in subsequent calculations instead of $[\hat{\mathbf{a}}, \mathbf{h}]$ is to introduce non-linearity between the output layer $\mathbf{o}$ and $[\hat{\mathbf{a}}, \mathbf{h}]$ to make the V-Att2Seq model be able to learn more complex patterns and thus improve the overall performance. Another reason for doing so is to provide a more compact representation of the units in the combined vector, in which all the values of those units are bounded by the same interval (in this case to the interval $[-1, 1]$ as a consequence of performing tanh activation over $W^{\hat{\mathbf{h}}}[\hat{\mathbf{a}}, \mathbf{h}] + b_{\hat{\mathbf{h}}}$). Finally, provided the computed vector $\hat{\mathbf{h}}$, one is able therefore to compute the output vector $\mathbf{o} = softmax(W^{\mathbf{o}}\hat{\mathbf{h}})$, where $W^{\mathbf{o}} \in \mathbb{R}^{v \times |\hat{\mathbf{h}}|}$ is the parameter weights matrix, and v the size of the vocabulary (i.e. the number of available tokens). Given all the calculations that have been done, the final prediction of the next token is done by selecting a token t_k at time step k such that $t_k = \operatorname{argmax}_i \mathbf{o} = \operatorname{argmax}_{t_k} p(t_k | t_{<k}, \mathbf{a})$.

For generating an entire sequence of tokens, the V-Att2Seq model simply appends the current sequence with the last predicted token and then performs the same procedure repeatedly by completing all the main phases as described in section 3.1.1, in section 3.1.2, and in this section as well, until a termination token ("</s>") is predicted signalling the end of the sequence, after which the sequence is outputted by the V-Att2Seq model. In the beginning of the generation task, the initial sequence consists of only one token indicating the start of the sequence ("<s>").

3.2 Data set & format

The raw data set contains 215,063 drug reviews that are associated with the drug, the condition to which the drug was applied, the rating of the patient to the drug, the date of submission of the drug review and the number of other patients that found the review useful [20]. This data set originally comes from [8], in which sentiment analysis on the specified data set was performed.

There are six attributes in total that are associated with each submitted drug review and these six attributes are as following:

- **Drug** - An attribute represented by a string, which indicates the name of the used drug. A preliminary inspection of the data set revealed that there are 3,617 different drugs. Consider Table 1 to see what drugs were most occurring in the data set.

- **Condition** - An attribute represented by a string, which indicates the name of the condition that was treated with the specified drug. A preliminary inspection of the data set revealed that there are 917 different conditions. Consider Table 2 to see what conditions were most occurring in the data set.

- **Review** - The target attribute represented by a string, which indicates the review that the patient has given to the specified drug for treating the specified condition to provide an own opinion about using the drug. An inspection of the drug reviews revealed that the minimum number of characters in a drug review is 3, the maximum number of characters in a drug review is 10,787, the mean length (i.e. number of characters) of a drug review is 458.62, and the standard deviation of the length of a drug review is 240.99.

- **Rating** - An attribute represented by an integer, which indicates the rating (on a scale 1-10) that the patient has given to the specified drug for treating the specified condition to indicate how satisfied the patient was with using the drug. Figure 8 illustrates how the ratings among all the drug reviews are distributed.

- **Date** - An attribute represented by a string, which indicates the date when the

Table 1: Top 3 most occurring drug names that were addressed by the drug reviews in the data set.

Drug name	**Number of occurrences**
Levonorgestrel	4,930
Etonogestrel	4,421
Ethinyl estradiol/norethindrone	3,753

Table 2: Top 3 most occurring conditions that were addressed by the drug reviews in the data set

Condition	Number of occurrences
Birth Control	38,436
Depression	12,164
Pain	8,245

review was given by the patient. In this data set, the date interval for the drug reviews spans from the end of February 2008 to the middle of December 2017.

- **usefulCount** - An attribute represented by an integer, which indicates the number of votes from other patients that found the review useful.

Out of the attributes listed above, **Drug**, **Condition**, and **Rating** were used when both training and testing all the relevant models in this study. The strategy was to present these attributes by using *one-hot encoding vectors*. The one-hot encoding vectors are binary vectors, whose indices correspond to all possible discrete states (in this case the (name of) the drug and the condition, respectively) and each of its elements is a binary value that indicates whether the discrete state is active or not. In such a binary vector, only exactly one discrete state must be active (i.e. 1 or true), whereas the other values are non-active (i.e. 0 or false). The reason for doing like this is firstly due to making categorical data process-able for any machine learning model and, secondly, due to the fact that one would make sure that the discrete states are not ordinal toward each other

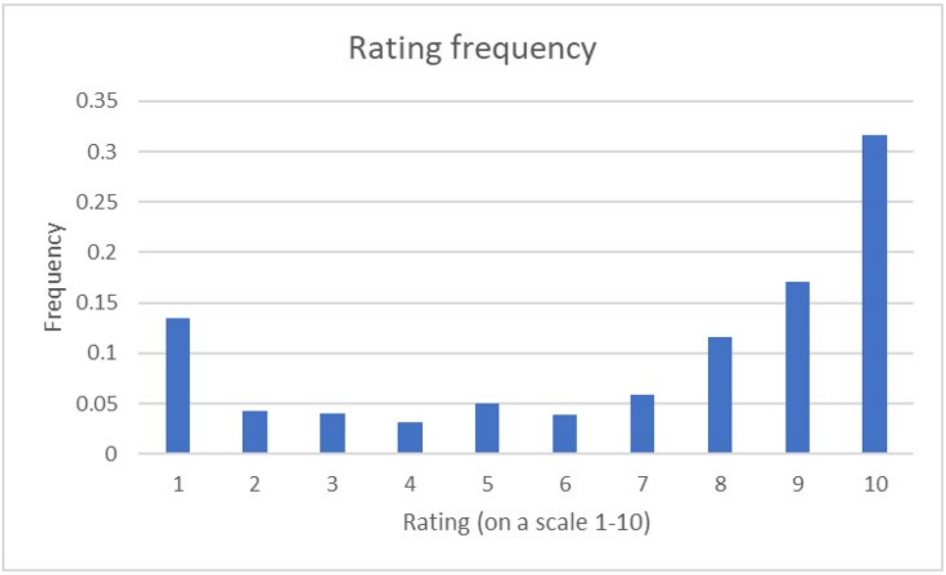


Figure 8: Distribution of the ratings in the entire data set.

(i.e. cannot be numerically ordered), which otherwise would contradict against the actual nominal characteristic of the attribute(s). Although **Rating** is actually ordinal, it was still onehot-encoded due to consistency toward the other two attributes. One possible problem with this strategy is that this could lead to sparse vectors if there are many different discrete states being involved, which would be the case here if one would like to consider all possible drugs and conditions in the data set and this could create problems for models that are worse at handling sparsity in the data set. To resolve this, many low-frequent drugs and conditions were filtered out to reduce the number of units needed to represent the one-hot encoding vectors, but at the same time still have a lot of drug reviews left to be used for training and testing of each respective model. Furthermore, it is also likely to impose limitations on the number of characters that could be present in a drug review, since there are drug reviews that contain over 10,000 characters while the average drug review contains approximately 400-500 characters and to eliminate possible risks for poor generative performance of models in general, one had to filter out drug reviews that deviate too much from the other drug reviews with respect to the length of a drug review. Finally, it is worth noting that there were in total 899 drug reviews where there is no information about the condition that was treated and the solution applied here was to simply remove them for a simpler pre-processing procedure.

What is important to emphasize is that all the candidate pre-processing strategies recently described were not exactly the same for each concerned model since they could react differently to those strategies (beyond the mandatory pre-processing strategy that is described in 3.2.1). Therefore, each model applied an own pre-processing strategy that are used to maximize their generative performance on this data set to avoid any pre-processing bias that could favor some models in front of others. Otherwise, this could potentially decrease the reliability in the results that are obtained from the experiments and the conclusions that were drawn in latter phases using the obtained results as basis.

3.2.1 Pre-processing strategy of the drug review texts

As already hinted in the last paragraph in section 3.2, a pre-processing strategy was applied to the raw data set when it comes to the drug review texts that the models are supposed to generate. The pre-processing strategy for the data set as described in section 3.2 was based on firstly cleaning the data set, where HTML entity references (such as "&#039;" that is a HTML entity reference for an apostrophe ("'")) were transformed into Unicode to present those HTML entity references as strings, the English contractions in the text were extended (e.g. "I'm" to "I am"), repetitions of same punctuation (such as "...", "??", and so forth) were removed (to get ".", "?", and so forth), and all punctuation except for dot ("."), comma (","), colon (":"), semi-colon (";"), exclamation mark ("!"), question mark ("?"), the parentheses ("()"), and hyphen ("-") were removed before correcting misspellings of words. For further insights about the applied pre-processing strategy upon the data set, it is advised to consider the code project, whose link is included in section 7.1 in the Appendices section.

After the cleaning process had been completed, the following criteria were applied to decide whether a drug review should be filtered or not, where it was enough for the drug review to satisfy one of the criteria to be filtered out from the data set:

- The drug review was erroneously parsed, where there were values that do not match up against the expected format of the attribute field.
- The drug review was not longer than 175 characters since it was observed that longer drug reviews tend to become less expressive and more arbitrary, while the shorter ones were more concise and tend to be more expressive. Along with the upper bound of number of characters that was imposed on the drug reviews, the drug review could not be shorter than 15 characters since such drug reviews do not convey much useful information that the models could learn and thus are not really useful at all.
- The occurrence of the set of attributes associated with the drug review itself does occur less than 24 times, at which point the particular set of attributes was considered to be infrequent and thus should be filtered out from the data set.

After following the instructions as given above, the pre-processed data set would be therefore process-able for any of the models that were considered in this study.

3.3 Evaluation

Provided the pre-processed data set as according to the details in section 3.2.1, the entire data set of drug reviews (i.e. the texts and their associated attributes) was randomly split into a training data set and a testing data set, where the training data set was used for training the models whereas the testing data set was used for evaluation of the models with respect to the BLEU metric. For this experiment, 70 % of the entire data set was used for training and the rest of the data set for testing. The split was done in a stratified fashion with respect to the sets of attributes associated with each drug review text to maintain the distribution of the sets of attributes to become as equal as possible in both training- and testing data set. During the training, each of the models (except for the Nearest Neighbor model) minimizes their own pre-defined loss function during training before eventually being evaluated on both the training- and testing data set with respect to the BLEU metric.

To calculate the BLEU score provided a trained model and a data set used as testing data set, each model was given distinct sets of attributes $\mathbf{a} \in \mathbf{A}$ to generate some drug review text from. This drug review text from the model was then evaluated against all other drug reviews that shared the same set of attribute values using the formula for BLEU as presented in section 2.3. The BLEU score was computed for $n = 1, 2, 3, 4$ when calculating the BLEU score based on n-grams (over which an arithmetic average was computed to get the desired BLEU score). Given all the provided BLEU scores that were obtained by the concerned model for each distinct set of attributes, the final BLEU score was ultimately obtained by calculating the weighted average of the obtained BLEU scores where the weighting was based on how frequent the set of attributes was in the testing data set. Mathematically, this was obtained by computing $BLEU_{test} = \Sigma_{\mathbf{a} \in \mathbf{A}} w_{\mathbf{a}} BLEU^{(\mathbf{a})}$, where $w_{\mathbf{a}}$ (such that $\Sigma_{\mathbf{a} \in \mathbf{A}} w_{\mathbf{a}} = 1$ and $0 \leq w_{\mathbf{a}} \leq 1$) denoted the relative frequency of the attributes $\mathbf{a}$ in the testing data set, and $BLEU^{(\mathbf{a})}$ was the obtained BLEU score of the concerned model on the partition of the testing data set where the testing drug reviews shared the same set of attributes as the set of attributes $\mathbf{a}$, from which the model generated a drug review. The reason behind the choice of applying the weighted BLEU score was that the models were mostly optimized with respect to the more frequent sets of attributes in the training data set and to get an accurate measurement of their performance with respect to the distribution of the sets of attributes, this weighting would provide a more nuanced overview of how it actually performed for generating drug review texts for each distinct set of attributes. The weighting of the BLEU scores of the model for each set of attributes in the testing data set was also further enabled due to the stratification property of the training- and testing data set while splitting the data set since this property ensured that each model is evaluated with respect to what kind of data they have considered during training. Computing $BLEU_{test}$ for a model gave the final BLEU score of the same model on the entire testing data set.

This evaluation was, for each concerned model, repeated $N = 10$ times since the BLEU scores between consecutive evaluations for a model may vary more or less significantly and to get a general view of the generative performance of the models with respect to the BLEU metric, a pre-defined number of random evaluations had to be run. The BLEU scores from each such evaluation (both for the training- and testing data set) were collected for each model and they were used for a statistical hypothesis test that would indicate whether there was a statistically significant different between the BLEU scores of the concerned models or not.

3.4 Hypothesis

Due to presence of numerous stochastic components (e.g. randomization within initialization of weights among the models, randomization within updates of the same weights during training and so forth), all the experiments had to be repeated some number of times. For this study, all the experiments (accordingly to a defined setup as specified in section 3.3) were repeated $N = 10$ times for each concerned model. From each of those experiments, the final BLEU scores that were obtained for each model from the testing were collected. Provided those $N = 10$ BLEU scores from the experiments, a statistical hypothesis test was performed using the Kruskal-Wallis H test method. The null hypothesis for the statistical hypothesis test was that provided the $N = 10$ BLEU scores for each of the concerned models, the BLEU scores were generated from the same distribution over the BLEU scores regardless of choice of model. Otherwise, the alternative hypothesis was that some set of $N = 10$ BLEU scores of a model has another distribution than another set of $N = 10$ BLEU scores of at least one other model. The significance level for testing the hypothesis was set to $\alpha = 0.05$.

3.5 Software and libraries

All the relevant models were coded in the Python programming language since the Python programming language offers a lot of available libraries and implementations, respectively, that are most likely not available in other programming languages. Furthermore, to develop all the models that applied deep learning in this study, the Keras deep learning framework (using Tensorflow as backend) [11] was used since it provides a simple and user-friendly API that enables efficient development of deep learning models.

For calculating the BLEU scores across the experiments, the implementation that is distributed by *Natural Language Toolkit* (NLTK) [14] was used. The NLTK library was also used for other means when it comes to working with the data set in the pre-processing phase, e.g. text tokenization at particular occasions during the conduction of the experiments. Furthermore, the Python libraries *PyContractions* [17] and *SymspellPy* [18] provide functions that were important for cleaning drug review texts during the pre-processing phase as described in section 3.2.1. The *PyContractions* library is a Python library that provides functions to expand contractions that occur in the English language (*"don't"* to *"do not"*, *"we're"* to *"we are"*, and so forth) using a pre-compiled list of contractions in the English language. This library is applied to reduce the amount of unnecessary words that were added to the vocabulary that is formed based upon the data set in a latter phase of the experiments, although there are edge cases at which it will fail to expand (e.g. in sentences like *"Jack's a good person"* where *"Jack's"* should be expanded to *"Jack is"* but would not be expanded by the implementation). The *SymspellPy* library is another Python library that provides an implementation that can go through entire texts and correct them using the Damerau-Levenshtein distance metric. The correction includes e.g. editing misspelled words to correct ones, separating two different words that happened to be composite (e.g. *"goodfeelings"* to *"good feelings"*). This implementation helped to avoid trivial misspellings of words that would only lead to unnecessarily expanding the vocabulary to be able to save memory while performing the experiments.

3.6 Pre-processing strategy, settings & hyper-parameters of the models

In this section, the pre-processing strategy of the data set being passed as training to the model along with the settings and hyper-parameters of the model to produce the results in 4 are presented to the reader to give insights about relevant details and setups that influence the behavior of the concerned model during the evaluations. Note that this section only goes through the LSTM model and the V-Att2Seq model since the NN model does not possess any settings or hyper-parameters and its mechanisms are too trivial to be mentioned alongside with the LSTM model and the V-Att2Seq model. Note further that only those settings and hyper-parameters that were manually decided are highlighted, whereas other settings or hyper-parameters that were not highlighted were given their default settings or hyper-parameters (since the models were developed using Keras and Keras has provided a lot of default settings and hyper-parameters whenever not explicitly defined by the user).

3.6.1 Long Short-Term Memory network model

As indicated at previous locations in this report, the LSTM model was one of the two baseline models (while the other was the NN model). The LSTM model ignored the attributes completely and learned to generate tokens in a sequence by merely considering the previous tokens in the same sequence. Therefore, given a set of review texts to optimize the LSTM model with respect to, the drug review texts were first transformed into sequences of fixed length (which was defined by the maximum number of tokens found in a drug review in the training data set). The shorter drug reviews were transformed into sequences that were padded with zero tokens that indicate non-existential tokens at certain positions in the sequences to obtain the desired length. Each token in the sequences was given its own pre-trained word embedding. For this study, the 50-dimensional GloVe word embeddings [24] were applied by the LSTM model to the texts. The reason for applying word embeddings was due to the fact that it helped to reduce the amount of time that was spent on performing the calculations due to lower input dimensionality and, simultaneously, it has been shown in previous studies that word embeddings generally improve the performance of deep learning models [13]. However, word embeddings could not be found for every token that was present in the training data. Therefore, for such tokens, a random vector whose units were uniformly distributed on $[0, 1]$ of the same size as the Glove word embeddings were assigned to such tokens within the frames of this study.

Given a sequence with some fixed size consisting of GloVe-embedded tokens that were passed to the LSTM model, the LSTM model consisted of 256 LSTM cells, where this number of LSTM cells was experimentally derived. Furthermore, given the output vector (containing 256 units) from the LSTM model, a dropout regularization was performed on this vector using a dropout rate set at 0.1 (note that it was only during training that the dropout regularization was actually performed). Thus, the output vector (after undergoing dropout regularization) was ultimately mapped to a softmax probability vector that held the probability for each available token to be the next one in the current sequence.

During the training phase, the LSTM model was optimized by minimizing the categorical cross-entropy loss that was defined as $L(t_i, f_i) = -\Sigma_i t_i \log(f_i)$ where t_i was the actual probability of a data point belonging to class i and f_i the predicted probability of a data point belonging to class i. The LSTM model applies the Adam optimizer with the default parameters as found in the Keras documentation [11]. One could argue that it should have been the RMSProp instead since that is the default choice when it comes to learning with recurrent neural networks in general, but the experimenting with these optimizers has shown that the LSTM model with the Adam optimizer was (marginally) better than the LSTM model with the RMSProp optimizer. The maximum number of epochs that could elapse during the training of the LSTM model was set to 50. Finally, early stopping was also applied to the LSTM network, where the condition for aborting training in advance was if the categorical cross-entropy loss on the training data set has not dropped below the observed minimum categorical cross-entropy loss over two consecutive epochs. The best weights rather than the latest weights were used for the model if the training was aborted due to early stopping.

The LSTM model applies the following strategy for generating sequences: the initial sequence consists only of one token, which is the initial token (i.e. "<s>"). Given the current sequence, the LSTM model predicts the next token by computing softmax probability vector containing the probability of each available token to be the next one in the sequence and selecting the one token that maximizes this probability. This procedure is done repeatedly until a termination token (i.e. "</s>"), which would end the generation and make the LSTM model to return this generated sequence.

3.6.2 Variational Attribute-to-Sequence decoder model

When it comes to the pre-processing strategy of the V-Att2Seq model, it also used GloVe word embeddings with a dimensionality of 50 for the same specified reasons as the LSTM model during the evaluations, i.e. to save some time on calculations and to enhance the performance of the concerned model. Therefore, this implies that the pre-processing scheme for the V-Att2Seq model is equal to the corresponding one of the LSTM model as presented in section 3.6.1.

When it comes to the processing of the input attribute, the dimensionality of the input was $n_e = 50$ which is derived from the size of the Glove word embeddings that were used within the V-Att2Seq model. Next, the size of the output of the attribute encoder for each concerned attribute (i.e. $g(a_i)$ for an attribute a_i) was set to $m = 64$ which in turn was based on the corresponding attribute encoding output size that was used in the model by Dong et al. [5]. Furthermore, the size of the attribute vector $\hat{\mathbf{a}}$ was set to

$|\hat{\mathbf{a}}| = |\mathbf{a}|m = 3 * 64 = 192$. The reason for setting $|\hat{\mathbf{a}}| = |\mathbf{a}|m$ was due to the fact that one wanted to store as much information as possible of the encoded attributes altogether without having too large attribute vector and $|\hat{\mathbf{a}}| = |\mathbf{a}|m$ was found to be a sensible choice to satisfy this as good as possible since a greater size would imply increased risks for redundant complexity that could contribute to possible overfitting of the overall model to some extent.

What regards the variational recurrent autoencoder, the intermediate layers (i.e. both the intermediate layer between the input layer and the hidden layer, and the intermediate layer between the hidden layer and the output layer, respectively) constituted of a LSTM network that itself consisted of 256 LSTM cells. This number was based on the corresponding number of LSTM cells that was used for the LSTM model that served as baseline. Furthermore, the size of the hidden layer was also set to 256 due to the same rationale as the one when setting the size of the attribute vector $\hat{\mathbf{a}}$, i.e. one would like to store as much information as possible without adding too much complexity redundancy in the overall model to increase the risks of occurrence of overfitting within the overall model. Thus, the choice of setting the size of the hidden layer to 256 was considered to be a sensible one.

During the training phase of the V-Att2Seq model, the variational recurrent autoencoder was first trained on transforming any sequence S in the training data set into a hidden vector $\mathbf{h}$ and reversely. The variational recurrent autoencoder minimized the loss function as defined by Eq. 15 in section 2.2.1 to achieve optimal performance at transforming any sequence S to a hidden vector $\mathbf{h}$ and reversely. The variational recurrent autoencoder made use of the RMSProp optimizer that has shown to give better results compared to using the Adam optimizer instead on the contrary to the corresponding case with the LSTM model in section 3.6.1. Finally, to ensure that the variational recurrent autoencoder did not get overfitted, an early stopping was applied where 10 % of the training data set was used as validation data set. The early stopping was activated during the training of the variational recurrent autoencoder if no lesser loss than the observed minimum loss (defined accordingly to Eq. 15 in section 2.2.1) was observed over two consecutive epochs. The maximum number of epochs that may elapse was set to 50. When the variational recurrent autoencoder has finished with its training, the training phase proceeds to the next step with training the entire model that includes the trained variational recurrent autoencoder. The overall model made use of the Adam optimizer. The loss function and training of the V-Att2Seq model at this point is done in the same way as it was done for the LSTM model, i.e. using categorical cross-entropy as loss function and use the same type of early stopping for exiting the training if the categorical cross-entropy loss on the training data set has not decreased below the observed minimum categorical cross-entropy loss over two consecutive epochs. Given the trained V-Att2Seq model, the model generated sequences according to the details provided in section 3.1.3.

4 Results

In this section, all the results for each concerned model are presented to the reader provided all the details and instructions that have been given throughout all the subsections in section 3 that have contributed to producing those results. In conjunction with the results, some sample drug review texts from each concerned model are presented to provide the reader some examples of drug review texts that were outputted by all models. In the following step, a statistical hypothesis test using Kruskal-Wallis H test method will be performed (provided the results that have been presented for each concerned model in each own subsection) to conclude whether there are any statistically significant differences between the BLEU scores from the different models that were considered in this study.

4.1 Long Short-Term Memory network model

The results that were obtained for the LSTM model both on the training- and testing data set across $N = 10$ experiments can be observed in Table 3. Examples of drug review texts that were generated by the LSTM model could be observed in Table 4.

Table 3: BLEU scores from best-performing LSTM network model across $N = 10$ experiments.

#	**Training BLEU score**	**Testing BLEU score**
1	0.19	0.18
2	0.19	0.17
3	0.22	0.19
4	0.2	0.18
5	0.19	0.17
6	0.18	0.18
7	0.2	0.18
8	0.18	0.18
9	0.19	0.18
10	0.21	0.18

Table 4: Examples of drug review texts generated by the LSTM model during the experiments.

Drug	**Condition**	**Rating**	**Text**
ondansetron	nausea/vomiting	10	ambien is the best ever . s took or a few weeks ago . i had constant in the past . i was acting irrational , the only side effect was not well . s my dosage has changed . s one pill
dextromethorphan	cough	1	i have been taking the no co ten my three hundred and twenty-five for a month now and they work great for my back and hip pain . s one pill has helped me a lot . s s one ! s you
levonorgestrel	birth control	10	five days to go on a twelve week treatment plan . undetectable after four the week unbelievable . no side affects whatsoever . s one relief the us a month . two thousand and sixteen lbs . s my or . s five

4.2 Nearest-Neighbor model

The results that were obtained for the NN model both on the training- and testing data set across $N = 10$ experiments can be observed in Table 5. Examples of drug review texts that were generated by the NN model could be observed in Table 6.

Table 5: BLEU scores from best-performing Nearest-Neighbor model across $N = 10$ experiments.

#	Training BLEU score	Testing BLEU score
1	1	0.21
2	1	0.24
3	1	0.22
4	1	0.24
5	1	0.25
6	1	0.23
7	1	0.23
8	1	0.23
9	1	0.22
10	1	0.23

The reason why the NN model obtains 1 in BLEU score on the training data set is because the training data set always contains the text that the model outputs since the NN model selects such a text from the training data set and while calculating the BLEU score, it will be discovered that the outputted text is identical to one of the texts in the training data set, thus leading to the BLEU score 1 (sometimes to BLEU scores such as 0.99 due to errors in the floating precision). With respect to this background, the BLEU scores of the NN model on the training data set will not be considered in the statistical test results that are presented in this section as well.

Table 6: Examples of drug review texts generated by the NN model during the experiments.

Drug	Condition	Rating	Text
ondansetron	nausea/vomiting	10	fran saved my life ; literally . my morning sickness was horrendous , and this is the only thing that made me feel great !
dextromethorphan	cough	1	my four year old had major diarrhoea after taking this medication .
levonorgestrel	birth control	10	it really helped me out .

4.3 Variational Attribute-to-Sequence decoder model

The results that were obtained for the V-Att2Seq model both on the training- and testing data set across $N = 10$ experiments can be observed in Table 7. Examples of drug review texts that were generated by the NN model could be observed in Table 8.

Table 7: BLEU scores from best-performing V-Att2Seq model across $N = 10$ experiments.

#	**Training BLEU score**	**Testing BLEU score**
1	0.41	0.19
2	0.38	0.2
3	0.4	0.19
4	0.38	0.19
5	0.39	0.19
6	0.45	0.19
7	0.34	0.19
8	0.36	0.2
9	0.44	0.2
10	0.4	0.19

Table 8: Examples of drug review texts generated by the V-Att2Seq model during the experiments.

Drug	**Condition**	**Rating**	**Text**
ondansetron	nausea/vomiting	10	fantastic medicine for nausea medicines to keep pregnancy . none a lot of medicines for the nausea , none the nausea that worked . it has me ! i need to go out ! need get when i need get any
dextromethorphan	cough	1	! it helps with the cough but gave me uncontrollable diarrhoea . i would not recommend this product again . this happens . one thing that helps me with the de ! you not stop ! let things of my benadryl
levonorgestrel	birth control	10	! , very easy and i have had a good experience with it will recommend it . my periods had the four months . it is only related my acne , no cramping . i swear by this ! also me

4.4 Statistical hypothesis test

Based on the results that were presented in Table 3 for the LSTM model, Table 5 for the Nearest Neighbor model, and Table 7 for the Variational Attribute-to-Sequence model, the hypothesis tests were conducted with the Kruskal-Wallis H test method on training data and testing data set, respectively. Table 9 presents the results that were obtained across $N = 10$ tests on training data and testing data, respectively, and provides an answer whether the results imply a statistically significant difference or not.

Table 9: Hypothesis test results using Kruskal-Wallis H test on training data and testing data, respectively.

	H test static	***p*-value**
Training data (w/o NN)	14.286	0.0002
Testing data	24.935	0

When it comes to the statistical analysis with respect to the H test static in the case of the testing data set, the rejection region is $R = \{\chi^2 : \chi^2 > 5.991\}$ provided the significance level $\alpha = 0.05$. Thus, the computed H test static (24.935) clearly is greater than 5.991, which indicates that the formulated null hypothesis is rejected and the alternative hypothesis is assumed instead. This conclusion is further justified considering that the computed p-value (0) was less than $\alpha = 0.05$, which implies the same conclusion.

Finally, when it comes to the training data set (where the BLEU scores of the NN model on the training data set were not considered), the corresponding rejection region for the computed H test static is $R = \{\chi^2 : \chi^2 > 3.841\}$. Thus, the computed H test static (14.286) clearly is greater than 3.841, which indicates that the formulated null hypothesis (which was about that the groups of the BLEU scores of different models followed the same distribution) is rejected and the alternative hypothesis is assumed instead. This conclusion is further justified when considering the computed p-value (0.0002) that was less than $\alpha = 0.05$, which implies the same conclusion that was drawn previously.

5 Discussion

Based on the results that have been presented in section 4, an analysis of them is done in the first part of this section where the results are reflected upon and then put into a greater context. Furthermore, the connection between the research question and the obtained results is established before giving the final answer to the research question with respect to the obtained results. Beside the specified analysis, attempts that have been done within the frames of this study are highlighted and the reasons to why those attempts were unsuccessful are explained as well. Based on these mentioned attempts, future work on the concerned topic of this study will be suggested to overcome the shortcomings that the V-Att2Seq model suffers from. Further future work could help the V-Att2Seq model to become more reliable in future applications.

5.1 Analysis of the obtained results

The statistical hypothesis test results in section 4 suggest that there were statistically significant differences between the BLEU scores from the different models. However, provided those statistically significant differences, the V-Att2Seq model does not outperform the NN model, although it outperforms the LSTM model (with a statistically significant margin). The research question as formulated in section 1.1 was whether the V-Att2Seq model was able to outperform all the baseline models in the context of generating drug reviews. Therefore, since the V-Att2Seq was not able to outperform the NN model in this study, the response to the research question is negative.

Despite this response to the research question, there are a couple of insights to cover in this section based on all the effort that was put into producing the results in section 4 through the entire study. One such insight was that the idea of combining variational recurrent autoencoders with a core LSTM network has contributed to make the LSTM network be better at conditionally constructing sequences from input attributes compared to a core LSTM network that ignored input attributes. The fact that the V-Att2Seq model is better than the LSTM model (with a statistically significant margin) proves this reasoning and that could be considered a step forward. One could also compare the examples of drug review texts that were generated by the LSTM model in Table 4 and the V-Att2Seq model in Table 8, respectively. The LSTM model generated texts that may be somewhat linguistically cleaner than the corresponding texts by the V-Att2Seq model, but those generated texts have either little or nothing to do with the specified attributes. Meanwhile, the V-Att2Seq occasionally succeeded with generating texts that directly address the attributes to various extent through e.g. appropriate adjectives provided the rating and addressing the name of the drug and/or condition in the text itself. All these details indicate that the V-Att2Seq model may possess some potential to be developed to become a more sophisticated model for similar means in the future.

Another insight that was obtained from this study is that one possible reason to why the V-Att2Seq model did not outperform the NN model could be that the transformation of the sequences to hidden vectors is done independently from surrounding attributes. Worth noting is that the model as proposed in the study by Dong et al. did initialize the hidden vectors at the first time step $t = 0$ within its sequence LSTM decoder with encoded attributes. This feature may have helped that model to outperform the corresponding NN model when conducting experiments in that study. The absence of this feature within the V-Att2Seq model could probably lead to a worse generative performance of the model as a consequence of not considering the surrounding attributes. Therefore, it could be worth to actually try to incorporate this feature into a future version of the V-Att2Seq model to hopefully increase its generative performance.

One final insight that was obtained from this study is that the applied strategy for generating sequences by greedily predicting the next token at each consecutive time step may not be the most optimal one for the V-Att2Seq model. Perhaps another more comprehensive strategy for generating sequences such as a beam search could help the V-Att2Seq model to generate texts that were more linguistically cleaner as those examples of drug review texts that as seen in Table 8. It has already been hinted in the study by Dong et al. [5] that the beam search strategy for generating sequences could be a possible candidate for the same means, although their model also ended up using an analogous strategy for generating sequences in the experiments. This could also be worth spending some time at when developing the V-Att2Seq model in a future study.

All the results from this study are primarily of interest for the researchers that work with deep learning in general since this entire study aims at providing further empirical knowledge about an alternative model (i.e. the V-Att2Seq model) that could be used for the means of generating textual data conditionally (which in turn has a broad scope of possible applications). The results from this study do not imply any direct commercial- or industrial consequences in the near future. However, despite these premises, there are still some aspects to highlight when it comes to possible ethical dilemma and different kinds of sustainability issues that more large-scaled future studies concerning similar topics may imply. One example regarding ethical dilemma is the application of the data set for developing similar models where similar models as those concerned in this data set may use confidential data that would go against the integrity of the people in different contexts. In the case of e.g. generating drug reviews, the data set used in this study was anonymized in advance to make the data set not to expose confidential user data to third parties. However, there are real-world examples where companies have used confidential user data for developing own data models to achieve a certain objective. Those examples include the data scandal around Facebook and Cambridge Analytica, where Cambridge Analytica managed to acquire personal data about up to 87 million Facebook users without their consent for political advertising purposes in conjunction with the US presidential election 2016 [26]. Therefore, it is entirely possible that something corresponding may occur when gathering data for development of deep learning models that generate different textual data with respect to information that could improve the generative performance of such models, although the acquisition of the data may go against some data protection regulation law. Furthermore, regardless of whether the acquisition of the data was legal or not, the deep learning models themselves could impose a threat against people's trust in the information services if such deep learning models actually would be successful at generating textual data conditionally and it would be difficult to recognize whether the generated textual data is genuine or fake. This could lead to a higher degree of deception among the public if large amount of real data can be reproduced by different deep learning models and market itself as genuine to the public for e.g. gaining commercial advantages over other competitors, advancing political agenda, producing fake scientific work to feed a certain narrative, and so forth. All these things are a threat to the sustainability that is founded upon people's trust in these information services that are important for further development in the world. If the people cannot trust the most important tools for achieving further knowledge to educate themselves about the world and to come in contact with other people around the world, the goal of achieving absolute sustainability would become a mere ideal that is impossible to achieve. All these considerations are something that should be given some attention by any researcher that works with developing deep learning models for generating textual data conditionally, although these considerations mostly apply when this research has progressed even further in the future.

5.2 Attempts and suggested future work

Initially, when this study started, the first model before the V-Att2Seq model was established was to try with applying matrix completion to tune the units of the hidden vectors (that were outputted by the variational recurrent autoencoder), from which one could apply a one-to-many LSTM decoder to produce sequences. However, this method failed since the best noted BLEU scores for such a model were approximately 0.05, which was way below the corresponding BLEU scores of the baseline models. This has forced the study to revise the literature an additional couple of times, from which the idea of the V-Att2Seq model was arisen and was afterwards dedicated much more focus toward the end of this study. However, it was in a late phase that the idea of the underlying concepts behind V-Att2Seq model was established before both developing the model, optimizing it and evaluating it within the frames of this study. This has therefore led to that the amount of time spent on the V-Att2Seq model was not much in overall and its settings as presented in section 3.6.2 are highly preliminary, meaning that there might be settings that could work more optimally for the V-Att2Seq model given the data set of drug reviews.

Provided all the details that have been provided in this section, there are a couple of future work that could be conducted as a next step when it comes to developing the current V-Att2Seq model as applied in the experiments within the frames of this study. The first suggested future work as brought up in the previous subsection was to make the variational autoencoder consider the surrounding attributes associated with any sequence while transforming sequences to/from hidden vectors (in a similar fashion as in the model by Dong et al. [5]) to hopefully improve the overall generative performance of the V-Att2Seq model. The second suggested future work would be to develop other custom strategies for generating sequences such as beam search in which e.g. the V-Att2Seq model outputs three most likely sequences and selects the one sequence that maximizes the overall likelihood of the concerned sequence given the contained tokens that were predicted at previous time steps. The third, and last, suggested future work would be to try out different settings of the V-Att2Seq model by changing the sizes of the components within the V-Att2Seq model to explore further what kind of parameters that could facilitate the performance of the V-Att2Seq model. In conjunction with this, it would further be optimal to select a data set that is commonly used as benchmark for deep learning models whenever applied for the means of generating textual data conditionally (e.g. the same data set as the one in the study by Dong et al. [5]) to easier draw some parallels between own obtained results and results that other people have obtained in other similar studies to establish better premises for comparison.

6 Conclusions

Based on the results that were presented in section 4 and its analysis in section 5, it is hereby concluded that the V-Att2Seq model is *not* able to outperform (with statistically significant margin) all the baseline models that were chosen for this study, where the NN model turned out to be the best performing model when it comes to generating drug reviews in this study. However, although the V-Att2Seq model did not outperform all the baseline models in this study, it could be still worth to investigate further and put more effort into developing this model to become more sophisticated for similar means in the future. Examples of suggested future work with the purpose of improving the performance of this model include to incorporate surrounding attributes within the variational recurrent autoencoder when transforming sequences to/from hidden vectors in a similar fashion as it was done within the LSTM sequence decoder in the model by Dong et al. [5], testing other sizes of the relevant components within the V-Att2Seq model, and developing other custom strategies for generating final sequences (where the beam search is suggested as a possible candidate for the V-Att2Seq model in a future work).

7 References

[1] Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew M. Dai, Rafal Józefowicz, and Samy Bengio. Generating Sentences from a Continuous Space. *CoRR*, abs/1511.06349, 2015.

[2] Kyunghyun Cho, Bart van Merriënboer, Çağlar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, October 2014. Association for Computational Linguistics.

[3] George Doddington. Automatic evaluation of machine translation quality using n-gram co-occurrence statistics. In *Proceedings of the Second International Conference on Human Language Technology Research*, HLT '02, pages 138–145, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

[4] Carl Doersch. Tutorial on variational autoencoders, 2016. cite arxiv:1606.05908.

[5] Li Dong, Shaohan Huang, Furu Wei, Mirella Lapata, Ming Zhou, and Ke Xu. Learning To Generate Product Reviews From Attributes. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, volume 1, pages 623–632, 2017.

[6] Felix A. Gers and Jürgen Schmidhuber. Recurrent nets that time and count. In *IJCNN (3)*, pages 189–194, 2000.

[7] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* MIT Press, Cambridge, MA, USA, 2016. `http://www.deeplearningbook.org`.

[8] Felix Grä, Surya Kallumadi, Hagen Malberg, and Sebastian Zaunseder. Aspect-based sentiment analysis of drug reviews applying cross-domain and cross-data learning. In *Proceedings of the 2018 International Conference on Digital Health*, DH '18, pages 121–125, New York, NY, USA, 2018. ACM.

[9] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[10] Zhiting Hu, Zichao Yang, Xiaodan Liang, Ruslan Salakhutdinov, and Eric P. Xing. Controllable Text Generation. *CoRR*, abs/1703.00955, 2017.

[11] Keras. Keras: The python deep learning library. `https://keras.io/`, May 2019. Online; accessed May 16, 2019.

[12] Shiwei Liu, Decebal Constantin Mocanu, and Mykola Pechenizkiy. Intrinsically sparse long short-term memory networks. *CoRR*, abs/1901.09208, 2019.

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

[14] Natural Language Toolkit (NLTK). Natural language toolkit. `https://www.nltk.org/index.html`, May 2019. Online; accessed May 16, 2019.

[15] Christopher Olah. Understanding LSTM Networks. `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`, August 2015. Online; accessed February 27, 2019.

[16] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, 2002.

[17] PyPI. pycontractions 2.0.0. `https://pypi.org/project/pycontractions/`, May 2019. Online; accessed May 16, 2019.

[18] PyPI. symspellpy 6.3.8. `https://pypi.org/project/symspellpy/`, May 2019. Online; accessed May 16, 2019.

[19] Alec Radford, Rafal Józefowicz, and Ilya Sutskever. Learning to Generate Reviews and Discovering Sentiment. *CoRR*, abs/1704.01444, 2017.

[20] UCI Machine Learning Repository. Drug Review Dataset (Drugs.com) Data Set. `https://archive.ics.uci.edu/ml/datasets/Drug+Review+Dataset+(Drugs.com)`, October 2018. Online; accessed March 4, 2019.

[21] Abigail See, Peter J. Liu, and Christopher D. Manning. Get To The Point: Summarization with Pointer-Generator Networks. *CoRR*, abs/1704.04368, 2017.

[22] Tom Simonite. Using Artificial Intelligence to Fix Wikipedia's Gender Problem. `https://www.wired.com/story/using-artificial-intelligence-to-fix-wikipedias-gender-problem/`, March 2018. Online; accessed February 10, 2019.

[23] Sandeep Subramanian, Sai Rajeswar Mudumba, Alessandro Sordoni, Adam Trischler, Aaron Courville, and Christopher Pal. Towards text generation with adversarially learned neural outlines. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

[24] Stanford University. Glove: Global vectors for word representation. `https://nlp.stanford.edu/projects/glove/`, August 2014. Online; accessed May 16, 2019.

[25] Wikipedia. Autoencoder. `https://en.wikipedia.org/wiki/Autoencoder`, February 2019. Online; accessed February 21, 2019.

[26] Wikipedia. Facebook–cambridge analytica data scandal. `https://en.wikipedia.org/wiki/Facebook\T1\textendashCambridge_Analytica_data_scandal`, May 2019. Online, accessed May 29, 2019.

[27] Sam Wiseman, Stuart Shieber, and Alexander Rush. Learning neural templates for text generation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3174–3187. Association for Computational Linguistics, 2018.

Appendices

In this section, all the technical details that regard the setup of the conducted experiments are presented where links to relevant pages such as the code project associated with this study are provided to the reader that has the possibility to e.g. control the codes and check the experimental frameworks on its own to get better overview over the technical procedure behind producing the results that were presented in section 4.

7.1 Code project

The code project associated with this study could be found in the following link: `https://github.com/smasl7/Master-Thesis-Code-Project-Spring-2019`.

Note that not all necessary files may exist in the code project due to size limits that are imposed on the GitHub repos. This has lead that the larger files have been excluded to be able to upload the latest version of the code project into the relevant GitHub repo. However, despite this fact, links and necessary guidelines are provided in latter subsections of this section to the reader to be able to track the same files that were used within this code project.

7.2 Source of the used GloVe word embeddings

Those GloVe word embeddings that were used by both the LSTM model and the V-Att2Seq model, respectively, originate from a project by Natural Language Processing Group at Stanford University. The link to that project could be found as following: `https://nlp.stanford.edu/projects/glove/`. The exact download link that supplied the exact GloVe word embeddings that were used in this study could be found at the following link: `http://nlp.stanford.edu/data/glove.6B.zip`.

7.3 SymspellPy implementation & frequency dictionary

During the pre-processing phase of the data set as presented in section 3.2, SymspellPy was used to e.g. correct trivial misspellings of different words, add necessary spaces between two words that were composite and remove potentially unnecessary spaces between two words. The implementation of SymspellPy could be found in its Github repo with the following link: `https://github.com/mammothb/symspellpy`, which includes the frequency dictionary that is necessary for running the implementation in the code project. The link to that frequency direction is as following: `https://github.com/mammothb/symspellpy/blob/master/symspellpy/frequency_dictionary_en_82_765.txt`.