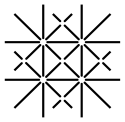


University of Basel

Graph-based Molecule Design with Deep Latent Variable Models

Master Thesis

Natural Science Faculty of the University of Basel
Department of Mathematics and Computer Science
Biomedical Data Analysis

Examiner: Prof. Dr Volker Roth
Supervision: Vitali Nesterov and Mario Wieser

Elias Arnold
elias.arnold@stud.unibas.ch
2014-930-770

17 April, 2020

Acknowledgements

I would like to thank Prof. Dr Volker Roth for the opportunity to write this thesis in his research group. Thanks to my supervisors Vitali Nesterov and Mario Wieser for their guidance and help. I would also like to thank all those who supported this work despite the extraordinary situation during the final stage of this thesis.

Abstract

Material design and drug discovery are research and industry branches which ensure prosperity and significantly contribute to the standards of living. The major potential of the enormous amount of possible molecules has yet to be realised. However, classical *ab initio* methods are computationally very demanding. Recent publications propose a statistical approach, involving generative models based on the variational autoencoder (VAE) architecture. In this thesis, we evaluate the constraint graph variational autoencoder (CGVAE), using a graph-based molecule representation and domain-specific constraints. In a row of experiments, we first discuss the descriptive power of graph-based molecule representations for chemical property prediction. Next, we investigate the reconstruction ability of a molecule using the CGVAE framework. Lastly, we evaluate the graph reconstruction and property prediction as a conjoint optimisation problem.

Our results show that predicting the energy gap with a graph-based descriptor yields a mean absolute error of 5.28 kcal/mol. By adding spatial information to the molecular graph, we achieve an error of 4.76 kcal/mol. Predicting the energy gap from the hidden molecule representation causes an error of 6.49 kcal/mol. As a baseline, the accuracy of an *ab initio* approach is ≈ 1 kcal/mol. Each of the generated molecules is chemically valid, while 87 % are novel. The chemical molecule validity and the decent property prediction results show that a statistical approach promises a considerable alternative to conventional methods.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The need for new materials is steadily increasing. Furthermore, they must be highly functional to meet the requirements of cutting edge technology. The development of new materials is essential for various fields of application, such as semiconductors, batteries, or for drug discovery. As the demand grows, research should be as resource-efficient as possible (Kim et al. 2018). Conventional approaches to molecule generation are no longer sufficient, especially for the more complex and specialised materials.

Moreover, the methods do not cope with the enormous variety of possible molecules. Even for small organic molecules, the number of stable compounds is estimated to be greater than 10^{60} (Kirkpatrick & Ellis 2004). To find new molecules, conventional methods scan a database of already known compounds and try to find molecules which best match the desired properties. Such traditional search methods are computationally very exhausting (Nesterov et al. 2019). In the next step, the selected molecules are further tested and evaluated until the most suitable one is determined. This can be achieved by solving Schrödinger's equation or by conducting costly and time-consuming experiments. Solving the problem algorithmically would take a lot of time and money because such algorithms would scale very poorly.

However, the availability of large databases of known molecules has led to a new approach. Instead of using exhaustive search approaches to find new compounds, machine learning models have taken over the task. In recent years, a great deal of research has been published in this area. A prevalent model architecture for molecule generation is the variational autoencoder (VAE) (Kingma & Welling 2013), which is a concatenation of two artificial neural networks (ANNs). First, a molecule is encoded into a low-dimensional probability distribution – the so-called latent space. Then, a corresponding sample from this latent distribution is decoded back to the original molecule. Once both mappings are trained, one can explore the latent distribution by sampling and decoding molecules from it that were unseen during the training process. Usually, both the encoding and decoding processes are realised with machine learning models; hence, the models are called encoder and decoder models. However,

this approach is also beset by significant challenges. For example, the representation of the molecules and the chemical validity of the generated molecules are two crucial aspects of any work that deals with molecule generation using machine learning.

At first glance, a handy molecule representation would be the widely used, text-based, simplified molecular-input line-entry specification (SMILES) representation of molecules (Gómez-Bombarelli et al. 2018). SMILES strings are less suitable, since one SMILES string can produce various molecule configurations, and producing semantically valid SMILES strings is challenging. Therefore, newer approaches use graph-based representations of the molecule as input. These representations better reflect the networked structure of graphs. In contrast to SMILES, a graph can be easily extended with 3-d information of the molecule, e.g., with edge weights corresponding to the bond lengths. However, there are several drawbacks. Although graph-structured data is prevalent, it is not so easy to learn from such data. The challenge is to handle the arbitrary connectivity of the nodes. Unlike images, where all neighbouring pixels can be arranged as a planar graph, there is no clear way in which to represent a graph (Simonovsky & Komodakis 2018).

Another difficulty when generating new molecules is chemical validity. To ensure the stability of the final compound, some chemical constraints must be applied (Samanta et al. 2018). Usually, a machine learning model is not capable of learning the underlying rules only from examples; those often have to be enforced by hand-crafted masks. The research and testing of automatic methods for molecule generation are still young. However, they hold tremendous potential, and there are many uses for such practices.

The goal of this thesis is to evaluate the constraint graph variational autoencoder (CGVAE) (Liu et al. 2018), which is a VAE approach for molecule generation. Like every ANN, a VAE minimises a loss function during training too. CGVAE optimises three separate loss terms. We divide our experiments into three sections. While the first section uses one loss term, the following sections each use an additional loss term for the objective function. The three loss terms are shortly introduced in the following list:

- The first term represents CGVAE's ability to deduce quantum chemical properties from the molecular graphs. For this task, a combination of the encoder network, which outputs latent vectors, and a densely connected network, which learns a scalar from the latent vectors, is used. In the first section of our experiments, we compare CGVAE's approach to other network architectures. As reference models, we use classical ANN types such as the CNN or the RNN, as well as state-of-the-art methods which were specially developed for learning on graphs.

- Second, CGVAE is trained to reconstruct given molecules. Initially, we observe the learning process and determine the required duration of the training to obtain optimal reconstructions. Then, the generated molecules are related to

the input molecules and compared by different aspects.

- The last loss term shapes the latent space according to a tractable distribution. Random latent vectors from this distribution are drawn and decoded to molecules. Statistics of the found molecules for different parameter settings are presented. Then, we seek for a two-dimensional representation of the latent vectors and illustrate the structure of CGVAE's latent space graphically.

1.1 Related Work

In the following section, we try to place CGVAE within a temporal context. Therefore, we present four approaches to molecule generation using machine learning methods.

1.1.1 Chemical VAE

An early variational autoencoder to generate new molecules was built by Gómez-Bombarelli et al. (2018). The encoder takes SMILES representations of molecules as input and spans a multidimensional and continuous latent space. Ideally, a decoder can reconstruct the string from a sample of this distribution, but the discrete character structure and complex syntax of SMILES strings made this problematic. To tackle this problem, they chose to filter out the mainly invalid SMILES strings from the decoder output. The chemical properties of a latent vector were determined by a multilayer perceptron. They showed that by jointly training this multilayer perceptron together with the VAE, the latent space could be structured into regions where the molecules have similar chemical properties. Despite the flaws in SMILES strings, the continuous latent space allows sampling of molecules unseen during training. Although the results show that this direction taken holds enormous potential, conventional genetic algorithms still have a slight edge when optimising the properties of a molecule.

1.1.2 GraphVAE

The first work which represented molecules as graphs was done by Simonovsky & Komodakis (2018). Previous efforts dealt with a linearisation of molecules,e.g., SMILES strings. Nevertheless, they faced similar difficulties to Gómez-Bombarelli, that is, the optimisation of discrete data structures. Recent advances in text generation have constructed the final output in a sequence of steps. However, owing to the sophisticated connectivity of graphs, a sequential generation would require discrete decisions, which in turn cannot be differentiated.

To overcome these issues, the decoder of Simonovsky & Komodakis (2018) outputs fully connected graphs with a fixed number of nodes. Each edge and node is modelled as a Bernoulli variable, which gives it a probability of existence. Although their

method only works with smaller molecules, it provides a good basis for further research.

1.1.3 Constrained Graph Variational Autoencoder

The CGVAE (Liu et al. 2018) uses a gated graph neural network (GGNN) (Li et al. 2015) for both encoding and decoding purposes. The decoder assembles new molecules sequentially. Based on the number of atoms in the original molecule, they sample a pool of nodes in the latent space. Then, an artificial stop node is added to the group. Finally, nodes from the pool are randomly picked and appended to the final molecule. Every time a new atom is added, CGVAE (Liu et al. 2018) performs a full decoding step on the partial graph, which is the key to the good results. So, to make a graph $\mathcal{G}$ in $t+1$ steps, t partial graphs are generated $(\mathcal{G}^{(0)}, \mathcal{G}^{(1)}, ..., \mathcal{G}^{(t-1)})$. Some rules for the correlation of two atoms are learnt by the VAE, and the remaining chemical rules are enforced by masking.

Unlike GraphVAE, CGVAE chooses to build the molecular graph sequentially. In general, the objective function of every VAE depends on the likelihood of a reconstructed molecule. Since the final graph is constructed node by node, all possible generation traces for a molecule have to be marginalised out, and the sheer number of all possible ways to generate the same molecular graph is not tractable. Therefore, CGVAE supervises its training routine with the ground truth generation traces of the original molecules.

To train the VAE, the loss function is constructed in such a way that it structures the latent space according to the quantum chemical properties of the molecules in the used dataset. This allows CGVAE to find new molecules which are optimised for a particular feature. The smooth latent space allows for gradient ascent to seek for the desired property value.

1.1.4 NeVAE

The NeVAE variational autoencoder (Samanta et al. 2018) addresses the problem of how to handle graphs of different sizes. In contrast to GraphVAE and CGVAE, NeVAE does not have a pool of nodes of a fixed size for the generated molecule. Instead, the number of nodes is given by a sample from a Poisson distribution, which is optimised during training. After the nodes of the final molecule are known, the edges that connect them are sampled. This makes it more efficient than using a distribution for each imaginable edge like GraphVAE. The aggregator function is specifically designed to handle different numbers of nodes. The models used as the encoder and decoder are very similar to GraphSAGE or Graph Convolutional Networks (see Chapter 2). This leads to convincing performance in the diversity and validity of the new molecules.

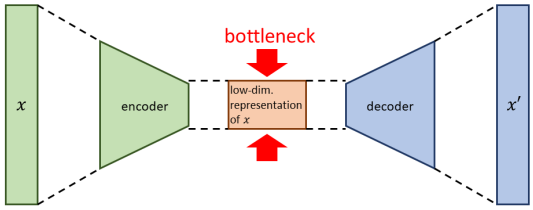
Chapter 2

Foundations

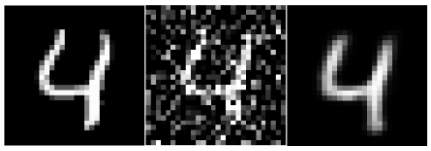
This chapter introduces the main concepts of this work. First, the theory of a variational autoencoder (VAE) is presented. This is a generative machine learning model which was originally proposed by Kingma & Welling (2013). As already mentioned in the introduction, a VAE is composed of two arbitrary ANNs. In the second part of this chapter, we focus on such networks, which are specifically designed for graph-structured input data.

2.1 Variational Autoencoder (VAE)

To introduce VAEs, we first have to review the concept of an autoencoder (AE) in general. It is, like a VAE, a series of two ANNs; the first network, the encoder, learns a low-dimensional representation of the input $\boldsymbol{x}$, while the second neural network, the decoder, raises the dimensionality of the learnt representation to the dimension of the input (see Figure 2.1a). An autoencoder simply optimises its ability to reconstruct the input. The key concept is that the reconstruction is made from a low-dimensional representation. The autoencoder propagates the input through a bottleneck at the intersection of the two networks. This low-dimensional part of the network is called latent space. A vector in this space is often denoted by $\boldsymbol{z}$, which is named a latent variable. Because of the bottleneck, only the most important features of the input persist in the latent vector, and the reconstruction of the input is based on the 'most important information' only. The behaviour of this procedure is very similar to principal components analysis (PCA). Imagine an autoencoder that consists of an input layer, a hidden layer, the bottleneck, of size p, and an output layer. If we would train this model as well as a PCA on the same dataset, the learned weights of the p-dimensional layer span the same vector subspace as the first p principal components of the PCA. Since the weights of the low-dimensional representation do not have to be orthogonal to each other and the autoencoder can use non-linear activation functions, the reconstruction of an autoencoder is often better than one



(a) Autoencoder architecture



(b) original image + noisy image + denoised image (left to right)

Figure 2.1: A denoising example using classical autoencoder architecture. **(a)** Architecture of an autoencoder. The input data is referred to as x. The encoder network learns a sparse representation of x so that it fits through the bottleneck. The remaining parts of x are then upsampled again by the decoder. **(b)** Denoising of an MNIST image. The original (left) is furnished with noise (middle), after which an autoencoder replicates the original (right); however, part of the information is lost. This can be seen by the blurry output.[2]

with PCA. Autoencoders are used for information retrieval, anomaly detection and image compression or denoising (see Figure 2.1b) because of this dimensionality reduction property[1].

Unfortunately, an autoencoder cannot be used as a data generator. Although we may want to sample random latent representations and feed them into the decoder to produce customised, unobserved $\boldsymbol{x'}$, this is not possible, since the distribution from which we would like to sample is unknown in an autoencoder. Therefore, we introduce the concept of variational autoencoders.

To generate new data with graphical models, we have to deal with inference. Let us assume that the visible output $\boldsymbol{x'}$ depends on a latent vector $\boldsymbol{x}$, which is hidden from our view. Remember that $\boldsymbol{z}$ corresponds to the low-dimensional representation of the input $\boldsymbol{x}$. In order to obtain the distribution from which we want to sample, we need to compute $p_\theta(z|x) = \frac{p_\theta(x|z)p_\theta(z)}{p_\theta(x)}$. This is intractable, because the marginal density $p_\theta(x) = \int p_\theta(x|z)p_\theta(z)dz$ would require us to evaluate all possible configurations of the latent vector $\boldsymbol{z}$. There are two main strategies for approximating $p_\theta(z|x)$: monte Carlo methods or Variational inference.

For our setting, the first approach would be too costly and very expensive timewise, especially if the parameters are updated for each single data point on a large dataset (Kingma & Welling 2013). Therefore, the variational inference is the key to solving this problem. Variational inference approximates $p_\theta(z|x)$ by a tractable distribution $q_\phi(z|x)$, where ϕ denotes the set of parameters of q that have to be learnt. The graphical model can be seen in Figure 2.2.

[1] https://en.wikipedia.org/wiki/Autoencoder#Dimensionality_Reduction

[2] Image source: https://towardsdatascience.com

[3] Image source: Kingma, Diederik P., and Max Welling. "Auto-encoding variational bayes." arXiv preprint arXiv:1312.6114 (2013).

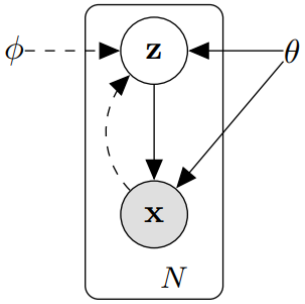


Figure 2.2: A graphical model showing the dependence of two variables, where x represents the observed variable and z is the latent variable that x depends on[3]. The ground truth generative model parameters θ and the variable z are hidden. With variational inference, the variational parameters ϕ are searched to bring a tractable distribution $q_\phi(z|x)$ close to the groud truth distribution $p_\theta(z|x)$.

Our goal is to find an encoder distribution $q_\phi(z|x)$, which is as similar as possible to a decoder distribution $p_\theta(z|x)$. The distance between two probability distributions can be determined using the Kullback-Leibler divergence (KL divergence). So, our goal is to compute the following term:

$$q_\phi^*(z|x) = \arg\min_{q_\phi(z|x)} KL(q_\phi(z|x)||p_\theta(z|x)) \tag{2.1}$$

Now, the KL divergence is substituted by its definition:

$$\begin{aligned} KL(q_\phi(z|x)||p_\theta(z|x)) &= \mathbb{E}_{q_\phi(z|x)}[log(q_\phi(z|x))] - \mathbb{E}_{q_\phi(z|x)}[log(p_\theta(z|x))] \\ &= \mathbb{E}_{q_\phi(z|x)}[log(\frac{q_\phi(z|x)}{\frac{p_\theta(x,z)}{p_\theta(x)}})] \\ &= \mathbb{E}_{q_\phi(z|x)}[log(\frac{q_\phi(z|x)}{p_\theta(x,z)}) + log(p_\theta(x))]. \end{aligned} \tag{2.2}$$

However, we cannot compute this directly because it depends on the intractable marginal density $p_\theta(x)$. As before, computing $p_\theta(x)$ would require us to integrate the whole latent space. To overcome this problem, an alternative objective is searched for:

$$\begin{aligned} &\underbrace{\mathbb{E}_{q_\phi(z|x)}[log(\frac{q_\phi(z|x)}{p_\theta(x,z)}) + log(p_\theta(x))]}_{KL(q_\phi(z|x)||p_\theta(z|x))} = \mathbb{E}_{q_\phi(z|x)}[log(\frac{q_\phi(z|x)}{p_\theta(x,z)})] + \underbrace{\mathbb{E}_{q_\phi(z|x)}[log(p_\theta(x))]}_{log(p_\theta(x))} \\ &KL(q_\phi(z|x)||p_\theta(z|x)) - log(p_\theta(x)) = \mathbb{E}_{q_\phi(z|x)}[log(\frac{q_\phi(z|x)}{p_\theta(x,z)})] \\ &log(p_\theta(x)) - KL(q_\phi(z|x)||p_\theta(z|x)) = \mathbb{E}_{q_\phi(z|x)}[log(\frac{p_\theta(x,z)}{q_\phi(z|x)})]. \end{aligned} \tag{2.3}$$

Where the right side can be simplified to:

$$\begin{aligned}
&\mathbb{E}_{q_\phi(z|x)}[log(\frac{p_\theta(x,z)}{q_\phi(z|x)})] = \mathbb{E}_{q_\phi(z|x)}[log(p_\theta(z,x))] - \mathbb{E}_{q_\phi(z|x)}[log(q_\phi(z|x))] \\
&= \mathbb{E}_{q_\phi(z|x)}[log(p_\theta(x|z))] - (\mathbb{E}_{q_\phi(z|x)}[log(q_\phi(z|x))] - \mathbb{E}_{q_\phi(z|x)}[log(p_\theta(z))]) \\
&= \mathbb{E}_{q_\phi(z|x)}[log(p_\theta(x|z))] - KL(q_\phi(z|x)||p_\theta(z)).
\end{aligned} \tag{2.4}$$

So, our equation becomes:

$$log(p_\theta(x)) - KL(q_\phi(z|x)||p_\theta(z|x)) = \mathbb{E}_{q_\phi(z|x)}[log(p_\theta(x|z))] - KL(q_\phi(z|x)||p_\theta(z)). \tag{2.5}$$

Remember, our goal was to minimise the divergence $KL(q_\phi(z|x)||p_\theta(z|x))$. However, this unknown term also appears in Equation 2.5. The two distributions, $q_\phi(z|x)$ and $p_\theta(z|x)$ do not necessarily have to be of the same type. Therefore, this cannot be calculated, and equality cannot be guaranteed. Therefore, the final inequality becomes

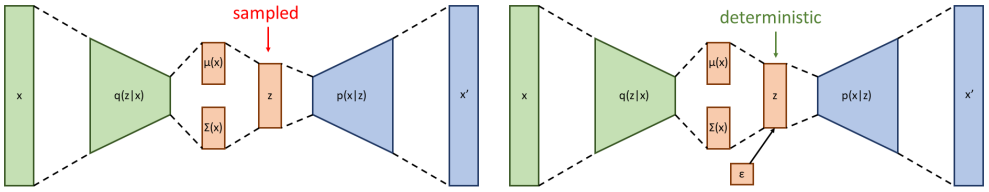
$$log(p_\theta(x)) \geq \underbrace{\mathbb{E}_{q_\phi(z|x)}[log(p_\theta(x|z))] - KL(q_\phi(z|x)||p_\theta(z))}_{\text{lower bound } \mathcal{L}}. \tag{2.6}$$

Since the KL divergence, according to Jensen's inequality, will never be negative, maximising the lower bound corresponds to minimising the KL divergence. We have now found a solvable sub-problem, the maximum is at the same time the minimum of the actual problem. In the literature, the lower bound is often called $\mathcal{L}(\theta, \phi, x)$ or ELBO, short for evidence lower bound. To optimise the lower bound, the most commonly used algorithms are based on a mean-field approximation such as CAVI, short for 'coordinate ascent variational inference' (Blei et al. 2017). Such algorithms assume that $q_\phi(z|x)$ is a product of mutually independent factors, where each can be optimised independently.

The task of maximising the first expectation term could be considered as a maximum likelihood estimation. The second term acts as a regulariser and requires $p_\theta(z|x)$ to be close to $q_\phi(z|x)$ (Doersch 2016). On the one hand, we want to maximise the reconstruction probability of the observed variable x, while on the other hand, we want the distribution $q_\phi(z|x)$ to be tractable.

To train a variational autoencoder, the negative of the lower bound -ELBO acts as the loss function. The encoder $q_\phi(z|x)$ should be designed in such a way that it learns the ground truth parameters θ of the encoder distribution. Under the assumption that $q_\phi(z|x)$ is a Gaussian, the parameters to be learned are μ and σ. To calculate the KL divergence, we substitute $p_\theta(z)$ with a simple Gaussian prior (Kingma & Welling 2013).

In order to obtain a z in the latent space, it has to be sampled (see Figure 2.3). The problem is that we cannot run backpropagation on a stochastic node. The



(a) without reparametrization trick **(b)** with reparametrization trick

Figure 2.3: A visual illustration of the reparametrisation trick. **(a)**: if the encoder of the VAE is forced to learn parameters for $q_\phi(z|x)$, a sample has to be from that distribution to get a latent representation z. The issue is that such a stochastic layer is not-differentiable and the VAE cannot be trained with backpropagation. **(b)**: therefore, we reparametrise the sample z, such that the noise is independent of the parameters. In both subfigures, backpropagation only takes place between the dashed lines.

solution is called a reparameterisation trick, where we decouple the sample from the parameterisation of the normal distribution whose parameters we are learning. Instead of sampling the latent vector $\boldsymbol{z}$ directly, we first sample an $\epsilon \sim \mathcal{N}(0, 1)$ and then compute $\boldsymbol{z} = \mu(\boldsymbol{x}) + \Sigma^{\frac{1}{2}}(\boldsymbol{x}) * \epsilon$. Now, given a fixed x and a sample ϵ, this function is deterministic and continuous in the parameters ϕ and θ (Doersch 2016).

The reparameterisation trick allows us to propagate the error back from $\boldsymbol{x'}$ to $\boldsymbol{x}$ without going through the stochastic node $\epsilon \in \mathcal{N}(0, 1)$.

2.2 Graph Neural Networks (GNNs)

The need for machine learning to deal with graph structures places new requirements on the models. Many methods have been proposed in recent years. This section aims to give an insight into the different approaches and to categorise them in a meaningful way.

To do this, we use the taxonomy proposed in Wu et al. (2019). They divide the recent advances in this research field into four different categories: first, recurrent GNNs (RecGNNs). Second, convolutional GNNs (ConvGNNs). Third, graph autoencoders (GAEs). And finally spatial-temporal GNNs (STGNNs). The first two categories are strongly related to CGVAE. Therefore, we present them individually in the following subsections.

2.2.1 Recurrent Graph Neural Networks (RecGNNs)

Graphs can be interpreted as a distributed system where each node holds a piece of information. To obtain the information of the whole graph, the fragments from

each of the connected nodes have to be combined. The connectivity of the nodes can be seen as a temporal dependency. This is very similar to a time series, where each point in time provides a piece of the final information. That is why RecGNNs is based on recursive neural networks (RNNs).

The encoder and decoder of CGVAE are such RecGNNs. The specific implementation is the gated graph neural network (GGNN) by Li et al. (2015). In order to make a statement about a graph, the GGNN carries out two steps:

- **Propagation step**: The GGNN computes a message for each node. The message is also called the node embedding $h_v^{(t)}$. The message is assembled recursively by collecting messages from the neighbouring nodes. This process is known as neighbourhood aggregation.
- **Output step**: Map the node embeddings to an output. Depending on the task, this can be done either globally or for each node.

To initialise the propagation step, the embedding or hidden state of every node is annotated with a vector. In the original paper of Li et al. (2015), this vector is two dimensional. The first entry of the embedding incorporates the label of the respective node v, which is defined as $l_v = l_e | e \in \mathcal{S}$, where $\mathcal{S}$ is the set of all edges in the graph. The remaining dimensions are filled with zeros to add more capacity to the embedding of the node $h_v^{(t)}$. In Table 2.1, the GGNN propagation model of the GGNN is summarised and compared to a classical RNN propagation model.

GGNNs are trained with backpropagation over time. Because this is very expensive to compute, the training algorithm is only executed for a fixed number of previous timesteps. Classical RNNs usually suffer from the vanishing gradients problem when trained over a large number of timesteps. To tackle this problem, the propagation model is replaced by a gating mechanism like long short-term memory (LSTM) or a gated recurrent unit (GRU) (Li et al. 2015). The gates help to control the information flow during the recurrent steps. The goal is that the propagated hidden unit $h^{(t)}$ only keeps the relevant information and forgets all non-relevant aspects of the entire sequence. A comparison of the different propagation models can be seen in Figure 2.4.

Because the gating networks share parameters, changes to them also affect the first time steps in the sequence. Thus, early GRU or LSTM cells in the recurrent series do not suffer as much as RNN cells from vanishing gradients during backpropagation over time. Remember that the GGNN and RecGNNs, in general, are based on the idea of a temporal dependency between the nodes of a graph. However, RecGNNs are outperformed by other approaches like convolutional graph neural networks which are

[4]Image source: http://dprogrammer.org/rnn-lstm-gru. Information source: https://towardsdatascience.com - "Illustrated Guide to LSTMs and GRUs: A step by step explanation" by Michael Nguyen, 24.09.2018

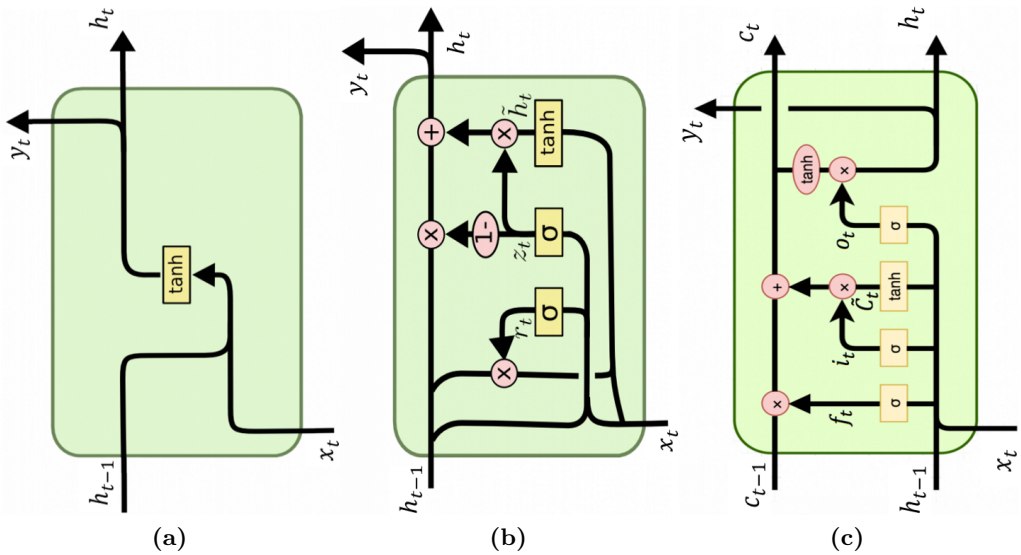


Figure 2.4: A collection of different recurrent neural network cells[4]. **(a):** A recurrent neural network unit takes a hidden state h_{t-1} and an input x_t, and maps them to the range $(-1, 1)$. Here, the $tanh()$ activation is used. This results in both a new output y_t, and a new hidden state h_t, which again serves as input to the next RNN unit. Since h_t gets passed through all timesteps, it can be interpreted as a kind of memory between the timesteps. However, the size of this memory is very limited because h_t is completely recalculated in every time step. The memory of an RNN is therefore referred to as a short-term memory. **(b):** A longer lasting memory is implemented in the GRU cells, where the inputs x_t and h_{t-1} are used to set the gates r_t and z_t. The values of these gates lie within the range $(0, 1)$ because of the sigmoid activation σ and they help to control the information flow. The reset gate (r_t) decides how much of h_{t-1} should be forgotten, while the update gate (z_t) balances the ratio of old and new information for the new hidden state h_t. The result is that important information from even the first timesteps can persist till the last timesteps. **(c):** The LSTM cell uses a separate data line (cell state c) to carry the important information of the entire sequence. The forget gate (f_t) decides whether to keep/drop information from c, the input gate i_t adds new information to c, and the output gate o_t produces a new hidden state based on h_{t-1}, x_t and the new c_t.

Table 2.1: The propagation models of two recurrent neural networks. This table shows a comparison of the GGNN propagation model and the classical RNN. The formulas are from the slides of the paper by Li et al. (2015). It can be seen that the GGNN is designed for graphs since the hidden state h_v depends on each node v. The GGNN then combines all node predictions into a graph-level output $h_{\mathcal{G}}$.

GGNN	**RNN**
$h_v^{(0)} = [l_v^\top, 0^\top]^\top$	$h^{(0)} = 0$
$a^{(t)} = A_v^\top [h_1^{(t-1)\top} ... h_{\lvert\mathcal{V}\rvert}^{(t-1)\top}]^\top + b$	$a^{(t)} = Wh^{(t-1)} + Ux^{(t)} + b$
Reset gate: $r_v^t = \sigma(W^r a_v^{(t)} + U^r h_v^{(t-1)})$	– no gate
Update gate: $z_v^t = \sigma(W^z a^{(t)} + U^z h_v^{(t-1)})$	– no gate
$\widetilde{h^{(t)}}_v = tanh(Wa^{(t)} + U(r_v^t \odot h_v^{(t-1)}))$	– no gate input
$h_v^{(t)} = (1 - z_v^t) \odot h_v^{(t-1)} + z_v^t \odot \widetilde{h^{(t)}}_v$	$h^{(t)} = tanh(a^{(t)})$
$o_v = g(h_v^{(T)}, l_v)$	$o^{(t)} = Vh^{(t)} + c$
$h_{\mathcal{G}} = \sum_{v \in \mathcal{G}} \sigma(i(h_v^{(T)}, l_v)) \odot h_v^{(T)}$	$\hat{y}^{(t)} = softmax(o^{(t)})$

introduced in the next subsection (2.2.2). RecGNNs mark the beginning of machine learning on graphs and are regarded as pioneer work in this field (Wu et al. 2019).

2.2.2 Convolutional Graph Neural Networks (ConvGNNs)

Convolutional graph neural networks are a generalisation of the convolutional neural networks (CNNs) that are used for machine learning tasks on images (Wu et al. 2019). An image can be seen as a special type of graph where each pixel represents a node and edges only exist between neighbouring nodes. Thus, CNNs can extract local features from the entire image.

In practice, this is achieved by moving a kernel over the image. The kernel size and structure define the way in which the individual pixels are combined to form a new image. This is possible because every pixel shows the same structure and, if interpreted as a graph, is only connected to all adjacent pixels. There are two ways to define convolutions on a graph. Spatial-based approaches (see Section 2.2.2.2) rely on ideas from recurrent neural networks to aggregate the node information (RecGNN). ConvGNNs with a more robust mathematical foundation are the spectral-based approaches (see Section 2.2.2.1). As in computer graphics, the theory of spectral-based approaches originated within the domain of signal processing (Wu et al. 2019).

Spectral-based approaches need a mapping function that translates from the spatial

space into the so-called graph Fourier space. To obtain this function, an eigendecomposition of a graph-specific matrix is required. We have to restrict ourselves to graphs with the same structure, e.g., images, since two different graph structures yield two different mapping functions. This restriction ensures that we end up in the same graph Fourier space for every graph and that we do not have to learn different filters for all the graphs in our dataset. Together with the need to compute an eigendecomposition, this issue results in spectral-based approaches being less attractive than spatial-based approaches. Although spectral approaches have some weaknesses, they are mathematically sound and use signal processing methods. In the next subsection, we look at spectral-based approaches.

2.2.2.1 Spectral-based Approaches

Unlike images, a graph cannot be convolved by a simple sliding window. Because of the complex and unregulated relationships between the nodes, the kernel would have to alter its shape every time it is moved. In any case, to perform a convolution on a graph, the graph is first mapped to its specific spectral domain, a process which is referred to as the graph Fourier transformation, which takes advantage of the properties of the Laplacian matrix of the respective graph.

The Laplacian of a graph with n nodes is defined as $L = I_n - D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ where I_n is the identity matrix of size n, D is the degree matrix of size n (diagonal matrix with node degrees on the diagonal), and A is the adjacency matrix of the graph. The Laplacian L is symmetric and positive definite (SPD). Therefore, it can be decomposed as $L = U\Lambda U^T$ where U is the matrix of eigenvectors, and Λ is a diagonal matrix with the eigenvalues of L as the non-zero entries. A vector x can be rotated, such that the eigenvectors of L form a basis for x. This is called a graph Fourier transformation and can be achieved by applying the functions $\mathcal{F}(x) = U^T x$ and $\mathcal{F}^{-1}(x) = Ux$ respectively. If we know the mapping functions between the spectral and the spatial domain of a graph signal x, we can define a graph convolution with a filter g as a multiplication in the frequency space. The operator $\odot$ denotes the elementwise multiplication, also called Hadamard product, which is commutative:

$$\begin{aligned} x *_G g &= \mathcal{F}^{-1}(\mathcal{F}(x) \odot \mathcal{F}(g)) \\ &= \mathcal{F}^{-1}(\mathcal{F}(g) \odot \mathcal{F}(x)). \end{aligned} \tag{2.7}$$

If we define the filter $\mathcal{F}(g)$ as a function h of Λ, the Hadamard product can be interpreted as a simple matrix multiplication: [5]

$$\begin{aligned} \mathcal{F}^{-1}(\mathcal{F}(g) \odot \mathcal{F}(x)) &= U(U^T g \odot U^T x) \\ &= U \underbrace{diag[h(\lambda_1), h(\lambda_2), ...]}_{\Theta} U^T x. \end{aligned} \tag{2.8}$$

[5] https://en.wikipedia.org/wiki/Hadamard_product_(matrices)#Properties

$$H_{:,j}^{(k)} = \sigma(\sum_{i=1}^{f(k-1)} U\Theta_{i,j}^{k-1}U^T H_{:,i}^{k-1}), (j = 1, 2, ..., f(k))$$

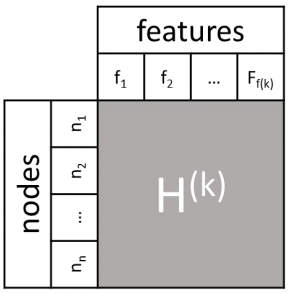


Figure 2.5: The update rule for spectral-based ConvGNN layers. Each layer of such a network represents a graph signal matrix H. The grey area on the right-hand side illustrates such a matrix. The ith row contains all the features for the ith node. Initially, $H^{(0)}$ is composed of all feature vectors (x_1, x_2, ..., x_n). $\Theta_{i,j}^{k-1}$ is a diagonal matrix that maps between the ith feature of layer $k-1$ and the jth feature in layer k. U is composed of the eigenvectors of L, σ is an activation function, and f returns the number of features for one specific layer.

All existing spectral-based ConvGGN approaches share the idea shown so far, differing mainly in the choice of the filter g. In practice, the filter g is often modelled as a diagonal matrix, where every diagonal element is a learnable parameter. This diagonal matrix can be written as $\Theta_{i,j}^{(k)}$, where k denotes the layer index, i the feature index of layer $k-1$ and j is the feature index of the current layer k. An application of a spectral-based graph convolutional neural network layer can be expressed using the formula in Figure 2.5.

Popular choices for the filter g are the polynomials of the diagonal matrix Λ. A well-known implementation is ChebNet (Defferrard et al. 2016), which uses Chebyshev polynomials to scale the diagonal elements of Λ. The filter responses for both the polynomial filter and the Chebyshev filter are shown in the following formulas:

$$h_{poly}(\lambda) = \sum_{i=0}^{K} \theta_i \lambda^i, \tag{2.9}$$

$$h_{Cheb}(\lambda) = \sum_{i=0}^{K} \theta_i T_i(\tilde{\lambda}). \tag{2.10}$$

In both formulas, λ is a placeholder for a single diagonal element of Λ (which is an eigenvalue of L), and ω is a trainable weight. In the interval (-1, 1), Chebyshev polynomials are orthogonal to each other. To use this favourable property, λ, which is always positive, is scaled onto this interval:$\tilde{\lambda} = \frac{2\lambda}{\lambda_{max}} - 1$. The function $T_i(x)$ returns the ith Chebyshev polynomial and is defined recursively as

$$T_i(x) = 2xT_{i-1}(x) - T_{i-2}(x) \text{ with } T_1(x) = x \text{ and } T_0(x) = 1. \tag{2.11}$$

Writing down the formula for a spectral GNN layer that uses many Chebyshev polynomials would result in a very complicated formula. Therefore, ChebNet (Defferrard et al. 2016) makes the following simplifications:

- The number of Chebyshev polynomials is set to two: $K = 1$.
- The weights for the two polynomials are the same: $\theta_0 = -\theta_1$.
- The value of the biggest eigenvalue is set to two: $\lambda_{max} = 2$.

If we expand the general spectral-based ConvGGN update rule with the peculiarities of ChebNet, we get:

$$\begin{aligned} H_i &= U\Theta U^T H_{i-1} \\ &= U(\Theta_0 T_0(\frac{2\Lambda}{\lambda_{max}} - I_n) + \Theta_1 T_1(\frac{2\Lambda}{\lambda_{max}} - I_n))U^T H_{i-1} \\ &= \Theta_0 H_{i-1} + \Theta_1 T_1(L - I_n)H_{i-1} \\ &= \Theta_0 H_{i-1} - \Theta_0(I_n - D^{-\frac{1}{2}}AD^{-\frac{1}{2}} - I_n)H_{i-1} \\ &= \Theta_0 H_{i-1} + \Theta_0 D^{-\frac{1}{2}}AD^{-\frac{1}{2}}H_{i-1} \\ &= \underbrace{(I_n + D^{-\frac{1}{2}}AD^{-\frac{1}{2}})}_{\tilde{A}}\Theta H_{i-1}. \end{aligned} \tag{2.12}$$

In the first step of the rearrangements, we simply replaced the filter Θ with the definition of the first two Chebyshev polynomials. To get to the next Formula, we replace: $\lambda_{max} = 2$, $T_0(x) = 1$, and $U\Lambda U^T = L$. For the next step, L is substituted by its definition, and we make use of $\theta_0 = -\theta_1$.

So, the full update rule for a single layer is

$$H_i = \sigma(\tilde{A}H_{i-1}\Theta). \tag{2.13}$$

2.2.2.2 Spatial-based Approaches

Spatial-based approaches share the same underlying idea as RecGNNs. The main objective is to combine the message of a central node with the messages of its neighbours to obtain a combined message that represents the entire graph. However, RecGNNs use recurrent functions for neighbourhood aggregation while spatial-based approaches use convolutions.

Compared with spectral-based approaches, the most significant difference is that spatial-based approaches illustrate graph convolutions in the spatial node space and not in the graph frequency space. The effect of this is that spatial-based methods are

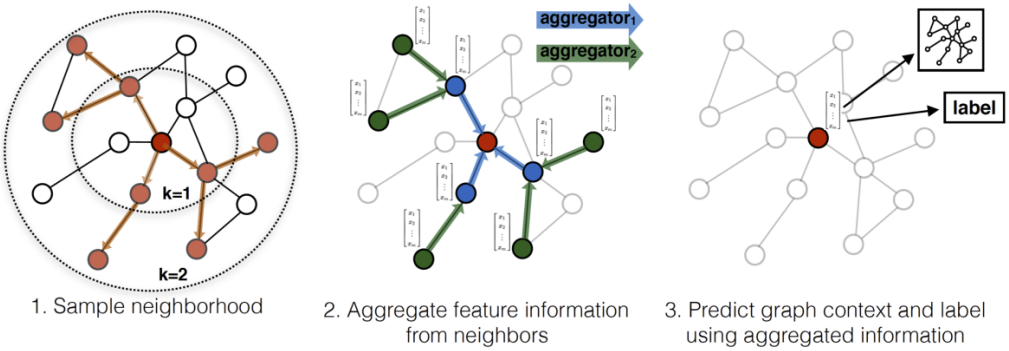


Figure 2.6: An visual illustration of the GraphSAGE approach. Figure from (Hamilton et al. 2017).

preferred to spectral-based methods. As already discussed in the introduction to this section, the mapping onto the frequency space is inefficient and makes spectral-based approaches less general and less flexible.

A prominent example of a spatial-based layer is Sample and Aggregate (GraphSAGE) proposed by Hamilton et al. (2017) at Stanford University. To tackle the issue that different graph structures have a different number of nodes, GraphSAGE samples the neighbourhood nodes that it wants to consider. The functionality of this approach is illustrated in Figure 2.6.

To obtain an embedding $h_v^{(k)}$ for a node v, Hamilton et al. (2017) define a graph convolution by means of the recursive relation

$$h_v^{(k)} = \sigma(W^{(k)} f_k(h_v^{(k-1)}, \{h_u^{(k-1)}, \forall u \in S_{\mathcal{N}(v)}\})), \tag{2.14}$$

where $h_v^{(0)}$ is initialised with the node features x_v. $W^{(k)}$ is a trainable weight matrix and the function f_k is an aggregator function which combines multiple node embeddings. $S_{\mathcal{N}(v)}$ is a set of sampled neighbours of v.

A fairly new approach uses auto-regressive moving average (ARMA) filters (Bianchi et al. 2019). Like GraphSAGE, this is a spatial-based approach which is defined in the spatial space rather than in the frequency space. However, ARMA filters are a concept of signal processing. The main components of the ARMA model are two polynomials; one models the auto-regressive (AR) part of the filter, while the other polynomial models the moving average (MA) part. The AR part tries to predict the filtered graph signal $\tilde{x}^{(t)}$ based on the known signal values, and the MA part models the actual signal value $x^{(t)}$ based on the unknown values $\tilde{x}$. To show this, we assume the dynamic process on some graph representation x given by $x^{(t+1)} = Lx^{(t)}$, where L is the Laplacian of this particular graph. The two parts of the ARMA filter can be written as **AR part**: $\tilde{x}^{(t)} = x^{(t)} + p_1 x^{(t+1)} + p_2 x^{(t+2)} + ...$, and **MA part**: $x^{(t)} = q_1 \tilde{x}^{(t+1)} + q_2 \tilde{x}^{(t+2)} + ...$

If we combine both equations, write them as a sum and solve for the filter response $\tilde{X}$, we get

$$\tilde{x}^{(t)} - q_1\tilde{x}^{(t+1)} - q_2\tilde{x}^{(t+2)} - ... = p_1x^{(t+1)} + p_2x^{(t+2)} + ... \tag{2.15}$$

$$(1 - \sum_{k=1}^{K} q_kL^k)\tilde{x}^{(t)} = (\sum_{k=0}^{K} p_kL^k)x^{(t)} \tag{2.16}$$

$$\tilde{X} = \frac{(\sum_{k=0}^{K} p_kL^k)X}{1 - \sum_{k=1}^{K} q_kL^k}. \tag{2.17}$$

The parameter K defines the number of timesteps of the dynamic process on which the filter output $\tilde{X}$ should depend. Because ARMA filters are a combination of two polynomial filters, they are more robust to noise compared to classical polynomial filters. Since polynomials are smooth, they cannot model sharp changes in the filter response. The design of ARMA filters allows them to model a lager variety of filter shapes compared to polynomial filters (Bianchi et al. 2019).

In practice, the output of an $ARMA_1$ filter is approximated by applying the graph convolutional skip (GCS) layer iteratively T times. The CGS layer is defined as

$$\tilde{X}^{(t+1)} = \sigma(\tilde{L}\tilde{X}^{(t)}W^{(t)} + X^{(0)}V^{(t)}), \tag{2.18}$$

where $W^{(t)}$ and $V^{(t)}$ are trainable matrices. $\tilde{L}$ is a modified Laplacian: $\tilde{L} = I - L$ where L is the normalised Laplacian. For the first timestep, $\tilde{X}^{(0)}$ is initialised with $X^{(0)}$. Such an iterative application of the GCS layer is called a 'GCS stack'. Remember that a GCS stack approximates a $ARMA_1$ filter. An $ARMA_K$ filter is achieved by combining K parallel GCS stacks where each stack runs for T timesteps. The output of an $ARMA_K$ layer is given by

$$\tilde{X} = \frac{1}{K}\sum_{k=1}^{K} \tilde{X}_k^{(T)}. \tag{2.19}$$

Chapter 3

Model and Implementation

The core concept of this work is CGVAE (Liu et al. 2018). As already discussed in Section 1.1.3, CGVAE follows a VAE architecture. In the first section of this chapter, we present the configuration and implementation details of CGVAE (see Chapter 3.1). The next three sections explain our approach for the experiments conducted in this thesis.

3.1 Overwiew of CGVAE's Structure

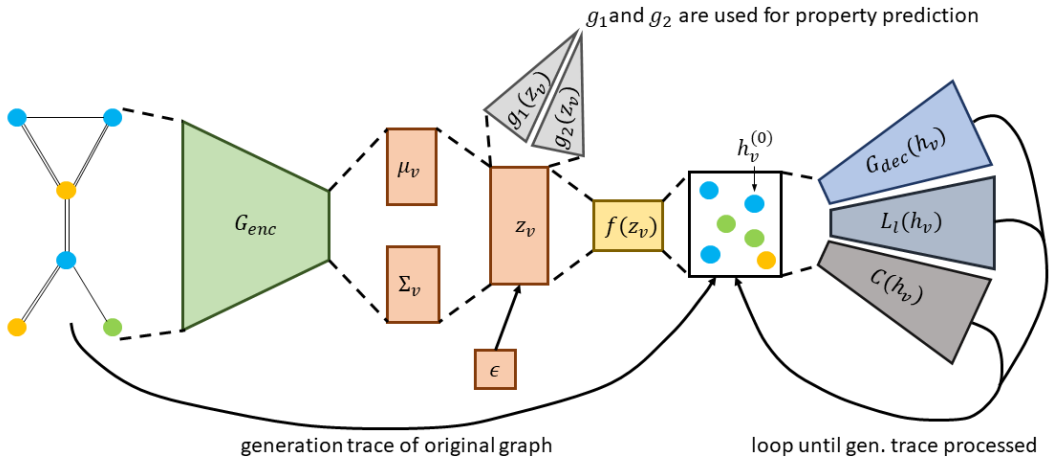


Figure 3.1: A sketch of the CGVAE model. To illustrate how a molecule is processed, we use a mock molecule with oversized and coloured nodes. The version shown here applies to the training of CGVAE. When generating new molecules, there is no conditioning with the generation trace of the original molecule. The generative procedure can be seen in Figure 3.2.

First, CGVAE uses an encoder (G_{enc}) to assemble a Gaussian parametrised by a mean μ_v and a covariance matrix Σ for each node v of the original graph. With the help of the reparametrisation trick (see Figure 2.3), latent vectors z_v are sampled. On the one hand, the sampled vectors are used as input for the property prediction models g_1 and g_2 (see Formula 3.4). On the other hand, the z_vs are used as an input for the linear classifier f, which grows the latent vectors z_v to node states $h_v^{(0)}$ (see Section 3.1.2).

Then, we enter a loop between edge selection, edge labelling, and node update (see Figure 3.2). During training, the sequential build-up of the reconstructed molecule is supervised by the generation trace of the original graphs. As a consequence, the node states h_v are arranged in a certain order. Guided by the ground truth generation trace, the networks C and L_l add bonds to the first node in the ordered set of node states $h_v^{(0)}$. After that, G_{dec} updates the states h_v of all nodes and the loop starts again. This time, bindings are added to the second node state. During training, this loop continues until each sampled node is connected. An artificial stop-state $\oslash$ is used as a termination condition when generating new molecules.

An illustration of the overview described in this section can be found in Figure 3.1. The following subsections describe the training and generation procedures of CGVAE in more detail.

3.1.1 Training of CGVAE

This VAE implementation trains the models C, L_l, G_{dec}, f, g_1, and g_2 simultaneously by optimising the following loss function, which consists of three parts:

$$\mathcal{L}_{CGVAE} = \mathcal{L}_{Recon} + \lambda_1 \mathcal{L}_{KL} + \lambda_2 \mathcal{L}_{QED}. \tag{3.1}$$

The λs are trade-off parameters. They determine the importance of the individual terms, and it is their responsibility to ensure that all terms are optimised equally. Let us have a look at each term individually: the reconstruction loss is given by

$$\mathcal{L}_{Recon} = \sum_{G \in D} log[p(G|G^{(0)})p(G^{(0)}|z)], \tag{3.2}$$

where D represents the training set which is composed of different graphs G. $G^{(0)}$ denotes the collection of unconnected nodes from which the decoder then builds a graph. The first probability factor in the logarithm can be interpreted as the likelihood to assemble the original molecular graph G from the collection of node states in $G^{(0)}$. The second factor can be understood as the probability of regenerating or resampling the nodes $h_v = (z_v, \tau_v)$ in $G^{(0)}$.

To compute the likelihood of the graph, the full generation history has to be marginalised out, because it is possible that multiple generation traces denote the same molecular graph. CGVAE sidesteps these issues by supervising its training routine with the generation traces of the molecules in the training dataset, and making reasonable assumptions about the distribution of these traces.

The second loss term

$$\mathcal{L}_{KL} = \sum_{v \in G} KL(\mathcal{N}(\mu_v, diag(\sigma_v)^2) | \mathcal{N}(0,1)), \tag{3.3}$$

is typical for VAE architectures. Its meaning is explained in section 2.1. The third loss term measures the L_2 distance between the actual quantitative estimation of drug-likeliness (QED) property of a latent vector z_v and a target property value Q:

$$\mathcal{L}_{QED} = \sum_{v \in G} d(R(z_v) - Q), \tag{3.4}$$

where R is defined as $R(z_v) = \sum_v \sigma(g_1(z_v)) * g_2(z_v)$. This loss term allows to optimise sampled vectors z_v for labelled property values Q during generation time.

3.1.2 Sequential Generation of New Molecules

To generate new molecules, N latent vectors z_v are sampled from the latent distribution, where N is an upper bound of the number of atoms in the original molecule. Since CGVAE has a smooth latent space, the sampled vectors z_v can be optimised with regard to a desired target property of the molecule. Next, a neural network f outputs a one-hot encoded vector $\tau_v = f(z_v)$ which indicates the node type of v. The concatenation of these two vectors is referred to as node state $h_v = (z_v, \tau_v)$. The decoder then reconstructs the original atom, that is, node-by-node or atom-by-atom, from the collection of available node states.

The decoder builds the molecule successively by executing two functions by turns - `focus` and `expand`. The `focus` function is implemented as a deterministic queue *focusqueue*, which follows the FIFO access principle, and centres on one of the N sampled vectors in the latent space. Once a node v is popped from the *focusqueue*, the `expand` function adds edges between the current focus node v and a non-focus node u, until an artificial stop node $\oslash$ is selected. After the addition of the new neighbours, the node representations h_v are updated by G_{dec} for each node v in the partial graph to account for the new neighbours.

While the `focus` function merely returns a node, the `expand` function is much more complex. For a given focus node v, it formulates a bond probability distribution

$v \overset{l}{\leftrightarrow} u = p(v \leftrightarrow u|\phi^t_{u,v}) * p(l|\phi^t_{u,v})$ of candidate edges l to other nodes u. The first factor specifies the softmax probability of an edge between the current focus node v and some other node u:

$$p(v \leftrightarrow u|\phi^{(t)}_{u,v}) = \frac{M^{(t)}_{v\leftrightarrow u} exp[C(\phi^{(t)}_{v,u})]}{\sum_w M^{(t)}_{v\leftrightarrow w} exp[C(\phi^{(t)}_{v,w})]}. \tag{3.5}$$

The second factor defines the softmax probability distribution of the edge type:

$$p(l|\phi^{(t)}_{u,v}) = \frac{m^{(t)_l}_{v\leftrightarrow u} exp[L_l(\phi^{(t)}_{v,u})]}{\sum_k m^{(t)_k}_{v\leftrightarrow u} exp[L_k(\phi^{(t)}_{v,u})]}. \tag{3.6}$$

To calculate the two factors, the neural networks C and L_l are used. The network C determines the node type of u, and L_l outputs the edge type of l. In order to make the distributions dependent on the current state of the partial molecule, both are conditioned on a feature vector $\phi^{(t)}_{v,u}$ for the possible connection $v \leftrightarrow u$. Chemically unstable connections are suppressed by the binary masks $M^t_{v\leftrightarrow w}$ and $m^{t\,k}_{v\leftrightarrow u}$, where $M^t_{v\leftrightarrow w}$ is for bond connections between nodes and $m^{t\,k}_{v\leftrightarrow u}$ is for the mask for the edge types. Connections from v to u are then sampled from this distribution until a connection to the stop node $\oslash$ is made. After the cancellation, the node representations or embeddings h_v are updated, a new focus node is selected by executing the `focus` function, and then, `expand` is called up again. The iterative call of the two functions runs until the *focusqueue*, which initially contains N elements, is empty.

Note that the final graph does not necessarily have to have N nodes, because during the expand function, no new connections are made to nodes that have once served as focus node. After the *focusqueue* has been processed, some unconnected nodes may remain, which are ignored. An overview of the entire generative model can be seen in Figure 3.2.

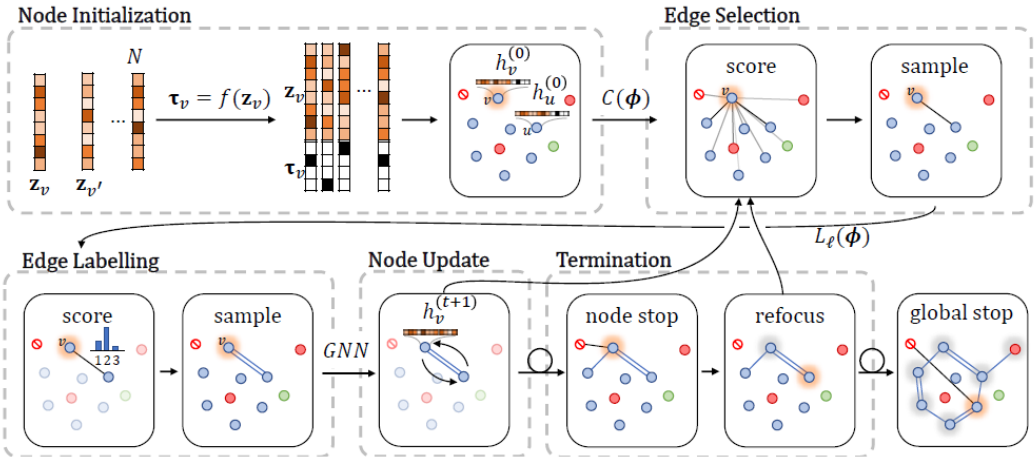


Figure 3.2: The steps used by CGVAE (Liu et al. 2018) to generate a new molecular graph. First, latent vectors z are sampled N times. Then, each of them gets a node type estimation τ, and we are left with N node representations or embeddings h. Next, a loop of edge selection, edge labelling and node updates is entered. This behaviour is repeated until an edge from the focus node (highlighted in red) to the stop node $\oslash$ is selected. Figure from (Liu et al. 2018).

The complex nature of the CGVAE molecule generation process may be better understood in pseudocode. In appendix A, the generation process, which in this chapter is explained textually, is given algorithmically.

3.2 Implementation of the Encoder and the Property Prediction Networks

CGVAE uses GGNNs (Li et al. 2015) for both encoder (G_{enc}) and decoder (G_{dec}). The networks consist of recurrent GRU cells (Figure 2.4). In the CGVAE paper, the recurrence lasts for seven steps. However, in the implementation on GitHub, which we use for our experiments, they use 12 steps. A visualisation of the procedure for a single step can be found in Figure 3.3. The Code for CGVAE uses residual connections. Such connections allow the output of a layer i not only to serve directly as input for the next layer $i+1$ but also to skip a few layers and to be used in a layer with an index $> i+1$. Residual connections can reduce the effect of the vanishing gradients problem. With this connections, the structure of the network is simplified, and fewer derivatives have to be calculated[1]. In CGVAE, the residual connections bypass the steps with an odd index. A step with an even index $2t$ takes the hidden states of all previous layers with an even index number $\{2r|\forall r < t\}$ as additional inputs. For example, in step 6, the hidden states from steps 0, 2 and 4 are used.

[1] https://en.wikipedia.org/wiki/Residual_neural_network

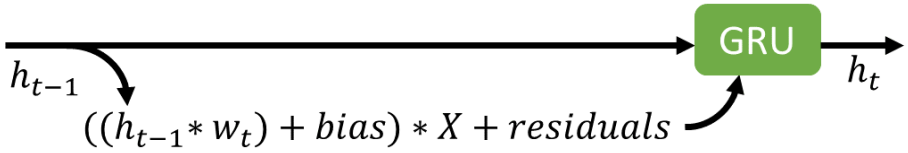


Figure 3.3: Illustration of a single recurrent step of a GGNN (Li et al. 2015). First, the hidden state of the previous layer h_{t-1} is multiplied by a weight matrix w_t. To ensure that the product is non-zero, a bias is added. In the CGVAE code, these three terms are called m and control the information flow between the vertices of the molecule representation X. Before the input is processed in the GRU cell, residuals are added. These are hidden states from any previous time steps: $Residuals = h_{t-i|i>1}$.

The hidden state of the encoder is initialised with a zero-filled matrix. During the 12 recurrent steps of the encoder, the hidden state learns the parameters of the distribution from which the latent vectors z_v are sampled. To predict quantum chemical properties of the molecule, CGVAE uses the linear transformations g_1 and g_2. These are implemented as a neural network without any hidden layers and linear activations. Both, g_1 and g_2 take a latent vector z_v as input and map to a scalar value.

In the first part of our experiments, we compare such a GGNN to other network architectures. All tested networks are trained to regress different quantum chemical properties on a subset of QM9 molecules. The preprocessing of the used dataset and detail about the architectures are described in Section 4.1.

3.3 Implementation of the Generative Procedure

The node states for the generative procedure of CGVAE are formed by the linear classifier f. Every latent vector z_v is enlarged with a one-hot vector τ that determines the node type of z_v. Like G_{enc}, a GGNN (Li et al. 2015) is used for the decoder too. The hidden state of the decoder is initialised with the concatenation $h_v = (z_v, \tau)$ (see Figure 3.2). The other networks for the molecule generation, C and L_l are fully connected three layer networks. The hidden layer uses 200 nodes with the $relu()$ activation.

For our experiments concerning molecule generation (see Section 4.2.2), we reconstruct 6400 randomly picked QM9 molecules with a trained CGVAE model. Because we just focus on the reconstruction ability of CGVAE, we used the mean μ_v for z_v. By avoiding any sampling in the latent space, we hope for more accurate reconstructions because the mean of every Gaussian is the most probable point. For the same reason, we apply an $argmax()$ Function to the two networks C and L_l. This ensures that only the most feasible edges are generated between the individual nodes.

The reconstructions are performed on multiple CGVAE models with a different amount of latent dimensions. We compare the reconstructed molecule to the original molecule in different aspects: first, if both denote the same molecule. Second, the difference in the number of nodes. And finally, we report the mean absolute error in the ϵ_{GAP} property. The goal of these experiments is to find to find the most convenient number of latent dimensions for CGVAE, which maximises its reconstruction ability.

3.4 Sampling of Molecular Graphs

The last part of the experiments discusses the question of how CGVAE structures the latent space. The authors of CGVAE (Liu et al. 2018) claim that the latent space is smooth and latent vectors z_v can be optimised for a desired quantum chemical property. We investigate if the latent space is partitioned into regions with different property values, such as Chemical VAE (Gómez-Bombarelli et al. 2018). For this experiments, we train CGVAE to predict the ϵ_{GAP} values of 6400 molecules. We multiply the parameter $\epsilon\ \mathcal{N}(0, 1)$ by different scalars to influence the variance during sampling.

To visualise the latent vectors, which were generated during training, we have to map all latent vectors z_v of a molecule to a two-dimensional vector. First, we calculate the mean vector of all latent vectors z_v. Then, we are left with a single vector. Nevertheless, the mean vector still lives in the latent space, which has many dimensions. Second, we apply PCA to the mean vector to find a two-dimensional subspace with maximal variance. The planar vectors are then coloured according to the ϵ_{GAP} property of the original molecule.

Chapter 4

Experiments

The experiments on CGVAE (Liu et al. 2018) are divided into three parts. In each part, we focus on a different piece of the CGVAE loss function $\mathcal{L}_{CGVAE}$, which is composed of three individual loss terms (see Section 3.1.1). In the first part, we map molecular graphs to molecule properties. This corresponds to the third loss term $\mathcal{L}_{QED}$. The core of the next part is the reconstruction ability of CGVAE, which coincides with $\mathcal{L}_{Recon}$. Last but not least, we try to form the latent space where $\mathcal{L}_{KL}$ is the controlling factor.

4.1 Property Prediction on Molecules

The goal of the first part (see Section 4.1) is to evaluate a machine learning model that maps a molecule descriptor to a specific property of that molecule. Similar experiments have already been performed by Nesterov et al. (2019). They focused on linear-, spatial-, and quantum-chemical descriptors of molecules. In the foreground of these models is the number, type and position of the atoms in the molecule. The bonds, especially the bonds type, are only reflected by the SMILES representation, which suffers from other shortcomings. As already mentioned in the introduction, SMILES are not suitable for generative machine learning models because it is difficult to create a semantically meaningful string. These also ignore the spatial arrangement of the atoms, which has a huge impact on the molecules properties (Eames 2019).

Because of these drawbacks, we represent the molecules as graphs and let the machine learning models predict some quantum chemical properties based on those graphs. A graph is defined by a set of nodes and a set of edges. Since our dataset contains no information about the connectivity of the atoms in the molecule ,except the SMILES strings, the required information is gathered from the open-source cheminformatics software RDKit Landrum (2010). Figure 4.1 shows a comparison between the skeletal formula and our graph representation of the molecule 1,2-oxazole-4-carbaldehyde.

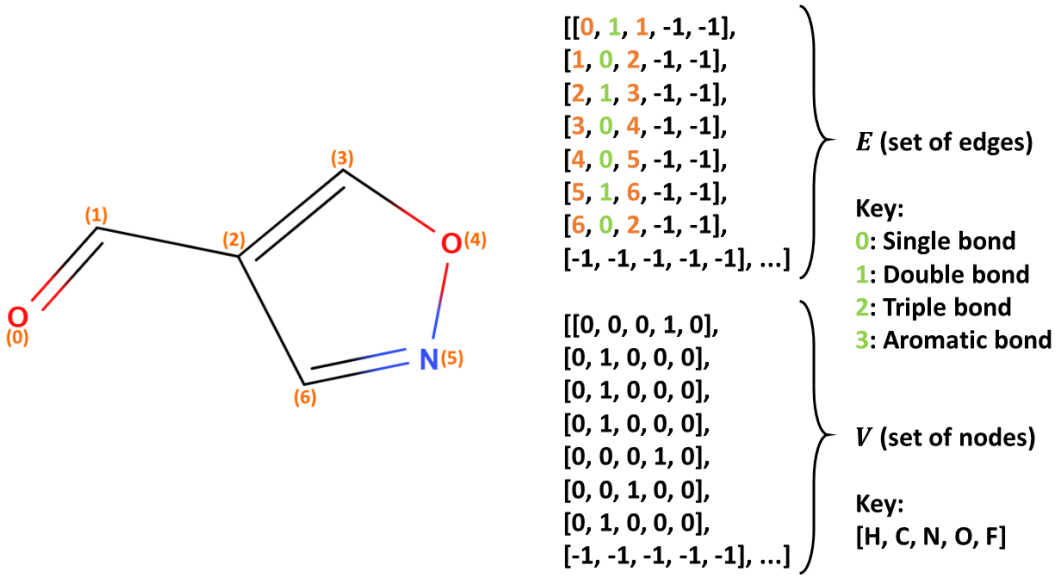


Figure 4.1: An illustration of the used graph-based molecule representation $\mathcal{G} = (V, E)$. **(a)** depicts the skeletal formula of 1,2-oxazole-4-carbaldehyde. **(b)** shows the same molecule as a combination of a set of vertices V and a set of edges E. The nodes are encoded as a one-hot vector in V. An edge is a triple where the first and third entry are indexes of the atoms involved. The second entry encodes the bond type. So that the list items of E have the same width as those of V, the elements are padded with -1. To ensure that the two sets have the same size for all molecules in the dataset, dummy entries, filled with -1, are added to the list.

The properties will be predicted by four different models. Two of them were already discussed in the previous chapter. One is the GGNN, which serves as the en- and decoder of CGVAE (introduced in Section 2.2.1), the other is the ARMA model (introduced in Section 2.2.2.2), which represents the latest developments in GNNs. To put the results into perspective, the experiment with the same descriptor is also performed on two classical types of neural networks: CNNs and RNNs. The architecture of those models follows the architectures proposed by Nesterov et al. (2019). This allows us to compare the results with their work too. In the following, the CNN and RNN are referred to as baseline models. The following subsections summarise the architectures used.

4.1.1 Architecture of the CNN and the RNN

The RNN is composed of three layers. The first two layers consist of 128 and 64 LSTM cells, while the third layer consists of a single densely connected cell. The latter uses a linear activation function, whereas both recurrent layers use the $tanh()$-activation. Between the LSTM layers, the entire output sequence is shared.

The CNN uses three 1-d convolution layers with an increasing number of filters: 8, 12, and 16. In contrast, the size of the utilised convolution kernels is decreasing: 7, 5, and 3. After the three convolution layers, two densely connected layers with a $relu()$-activation are added to the model. They consist of 256 and 128 units respectively. Last, one densely connected node with a linear activation is added as an output node.

4.1.2 Architecture of the GGNN and the ARMA Model

The GGNN that we use for property prediction differs slightly from the GGNN installed in the encoder and decoder of CGVAE (see Section 3.2). The first part of the model is a GGNN which carries out four recurrent steps. Then a single densely connected layer takes over to turn the multidimensional hidden state into a one-dimensional output. The hidden state is a matrix of size (v, 100), where v is the maximal number of nodes in a molecule. For the QM9 dataset, $v \leq 9$. This shortening, compared to the encoder of CGVAE, should make the size of the GGNN model comparable to the other models. We use the implementation from the Deep Program Understanding project at Microsoft Research, Cambridge, UK[1].

At the time of working on this thesis, one of the newest approaches for machine learning on graphs was the spatial-based ARMA layer introduced in Section 2.2.2.2. Since the research paper on this approach (Bianchi et al. 2019) shows the many advantages of the ARMA model compared to other propositions, we decided to work with the ARMA layer. Our model consists of three ARMA convolutional layers, which learn a 128-dimensional vector for each atom. The size of the filters of these layers is 32, 64 and 128. Since the final output of the encoder should be a scalar, we apply a recursive LSTM layer that aggregates all node-specific vectors into one 128 dimensional vector. The last three layers are dense layers which use the $relu()$-activation and a dropout probability of 0.1. Their sizes are 256, 128, and 1, respectively.

4.1.3 Dataset and Preprocessing

Every recent work in the field of molecule generation worked with the QM9 dataset (Ramakrishnan et al. 2014, Ruddigkeit et al. 2012). The findings in Nesterov et al. (2019), on which our experiments are based, used this record for the experiments as well. It contains 133,885 small organic molecules with 9 heavy atoms at maximum. All atom types except hydrogen are considered as heavy atoms. The basis of organic molecules consists of covalent bonds of carbon atoms. Because of its popularity, we will work with this dataset as well.

The molecules in the QM9 dataset are annotated with 15 chemical descriptors.

[1] https://github.com/microsoft/gated-graph-neural-network-samples

According to Huang et al. (2018), a good molecule representation can even model the most important properties – the energies. These includes the insensitive HOMO-LUMO gap, which typically does not correlate with the model size. In total, four energies are being predicted in this experiment. These are the energy of the highest molecular orbital (ϵ_{HOMO}), the energy of the lowest molecular orbital (ϵ_{LUMO}), and the gap between HOMO and LUMO (ϵ_{GAP}), and the internal energy at 0K (U_0). The internal energy is calculated from the kinetic energy, which is the movement of the atoms, as well as from the potential energy, which can be seen as the bond energies or strengths[2]. In the QM9 dataset, a molecule's internal energy (U_0) is given relative to the sum of the internal energies of all atoms involved. Therefore, we did not work with the specified value, but first calculated the absolute value. For this, we subtracted the U_0 value for every atom type from the specified U_0 value for the molecule. Ramakrishnan et al. (2015) mentions that the annotated molecule properties have an accuracy of $\approx 1 kcal/mol$.

Before training, the dataset was partitioned into three disjoint parts. First, we separated $30,000$, randomly selected samples from the entire dataset. Those were used to test the actual performance of the model. The samples for training and evaluation of the model were selected from the remaining $103,885$ molecules of the QM9 dataset (Ramakrishnan et al. 2014, Ruddigkeit et al. 2012).

There is another very popular dataset for quantum machine learning. In Appendix B, the same experiment is repeated using the popular ZINC dataset (Sterling & Irwin 2015). On average, the molecules of this dataset are bigger and more complex than the molecules in QM9. Although the molecules in ZINC are specifically selected to find new drugs, they are annotated with only three descriptors. These are the quantitative estimation of drug-likeness (QED), the synthetic accessibility score (SAS), and the water-octanol partition coefficient (logP).

4.1.4 Results with the Graph Descriptor

For our first experiments, we encoded all molecules of the QM9 dataset as a classical graph. The molecule representation that was used is illustrated in Figure 4.1. Then, we trained neural networks to predict quantum chemical properties on the encoded molecule. The learning curves for both baseline models that were introduced in Section 4.1.1 can be seen in Figure 4.2. A regression of the properties on other molecule descriptors has already been learned by Nesterov et al. (2019) on the same architectures of the baseline models. The found mean absolute error (MAE) values for the SMILES descriptor are summarized in Table 4.1.

[2] https://en.wikipedia.org/wiki/Internal_energy

Table 4.1: The MAE values with a molecule descriptor which is based on SMILES strings (Nesterov et al. 2019). These results can be directly compared (except U_0) to our baseline results, because they used the same CNN and RNN architectures for the experiments. For more details, see Nesterov et al. (2019).

Descriptor	$\epsilon_{\mathbf{HOMO}}$	$\epsilon_{\mathbf{LUMO}}$	$\epsilon_{\mathbf{GAP}}$	$\mathbf{U_0}$ (computed differently)
SMILES (CNN)	6.26	8.77	10.75	(9.27)
SMILES (RNN)	3.46	3.37	4.81	(8.46)

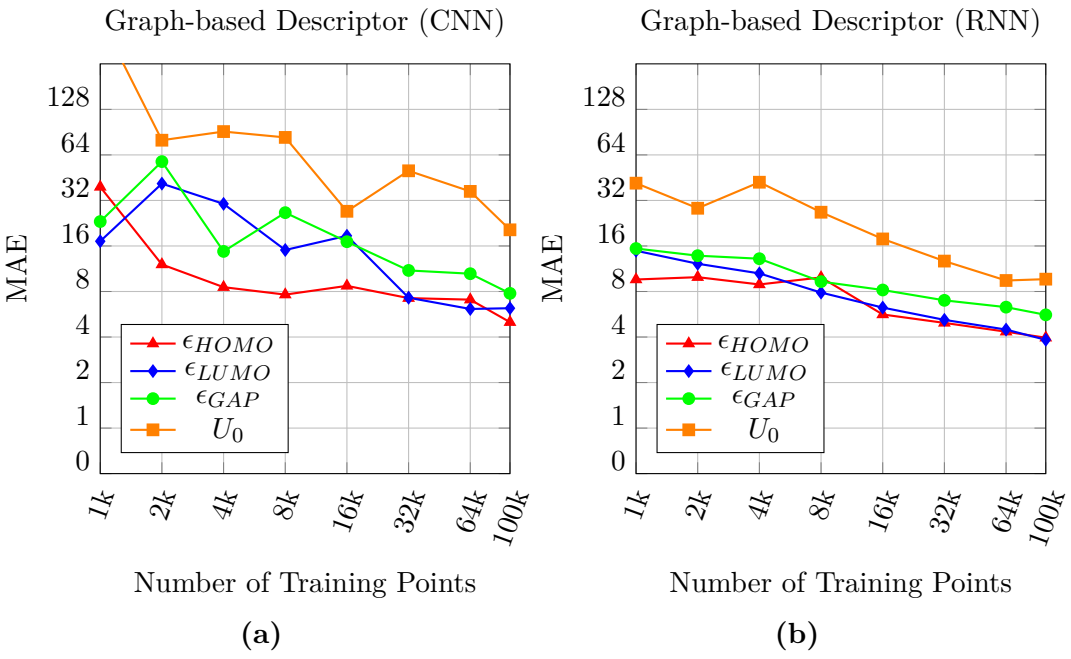


Figure 4.2: Learning curves for classical neural networks with the graph descriptor. Both plots show MAE values for four different molecule properties and a variyng number of training points. It is interesting to note that these simple models in combination with this simple descriptor achieve good prediction accuracy. For **(a)**, we used a CNN, and an RNN for subfigure **(b)**.

For the CNN, the MAE values on the graph descriptor are slightly better than with the SMILES descriptor (except for U_0). Although the RNN outperforms the CNN with the graph descriptor, it does not outrun the RNN with the SMILES descriptor. Even though we calculated the U_0 property differently, the MAEs for U_0 of both experiments are worse than the MAE values provided in Table 4.1. As our next experiment, we evaluate the graph-based descriptor on two models, which are specifically designed to learn on graph structures. Their learning curves can be seen in Figure 4.3.

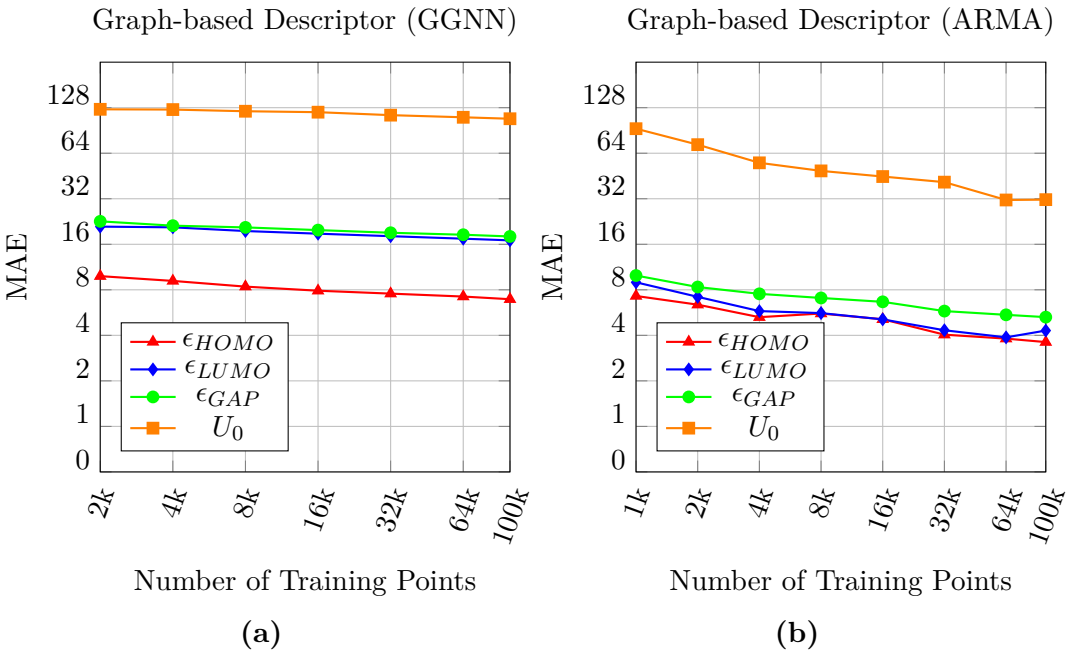


Figure 4.3: Learning curves for GNNs with the graph descriptor. Both plots show MAE values for different molecule properties. Each property is predicted with a varying number of training samples. Note that **(a)** does not show MAE values for 1k training points, since it uses a batch size of 256 and the 200 points for evaluation would not yield an entire batch. Nevertheless, it is easy to see that the learning curve is very flat and that the U_0 property can not be accurately predicted. This is also true for **(b)**, which outperforms the GGNN and the CNN.

Despite their flat learning curve, the complex GNN models, GGNN and ARMA, do not outperform the RNN model on this simple graph descriptor. We assume that either the two models were not trained sufficiently and have a low generalisation ability, or the graph-based descriptor lacks important information about the molecule. If we look at the training statistics, the accuracy on the validation set already reached its minimum after a few epochs, and as a result, the training was ended early.

As we have already seen, the bond energy or strength is an important factor for U_0. The bond energy can be directly derived from the length of the binding if it is known[3]. Since the QM9 dataset provides the 3-d coordinates of the involved atoms, the bond lengths can easily be determined. Therefore, we repeat the previous experiment with a molecule descriptor which additionally contains the bond lengths in Section 4.1.5.

4.1.5 With Bond Lengths

In this subsection, we enlarge the classical graph descriptor with the lengths of the bonds in the molecule. The fourth column of the set E is now filled with the length of the bond instead of '-1'. With this updated descriptor, the same experiments as without the bond lengths are performed. The MAE values for the two baseline models are given in Figure 4.4.

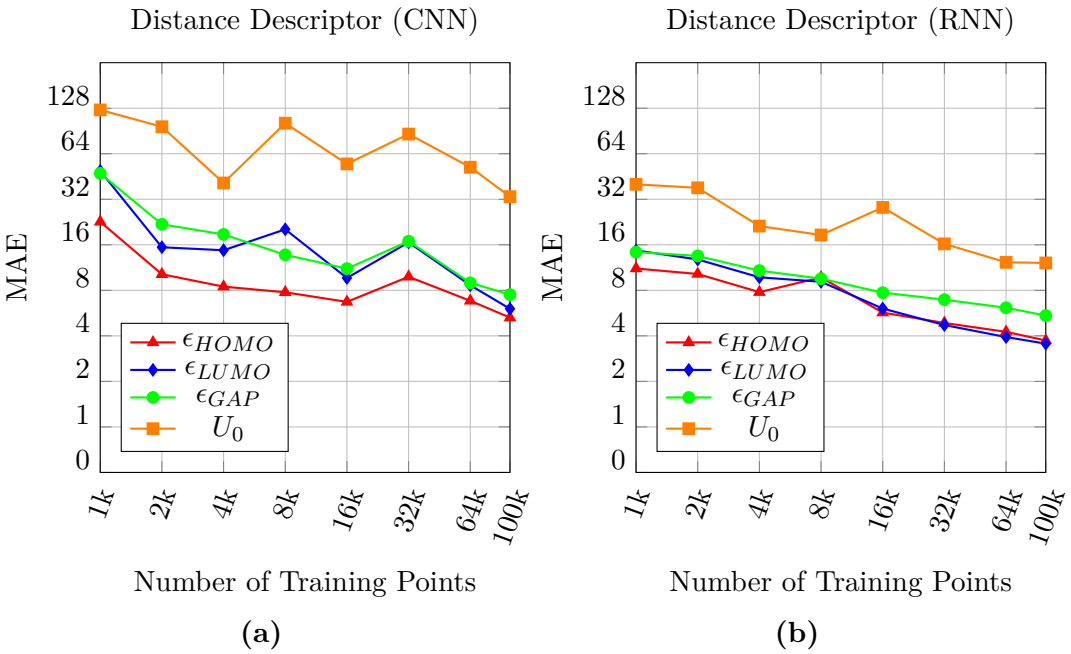


Figure 4.4: Learning curves for classical neural networks with the graph descriptor including bond lengths. Both plots shows MAE values for a varying number of training points. For **(b)**, we used a RNN, and a CNN for subfigure **(a)**.

[3]https://en.wikipedia.org/wiki/Bond_length

If we compare the learning curves for the baseline models of both descriptors, we see that that the effect of the added distance measure is minimal. In fact, the descriptor with the bond lengths performs even worse than the descriptor without bond lengths. It could be the complexity of these simple models is very low and therefore the model cannot handle this additional variable. The learning curves for the complex models on the descriptor with bond lengths can be seen in Figure 4.5.

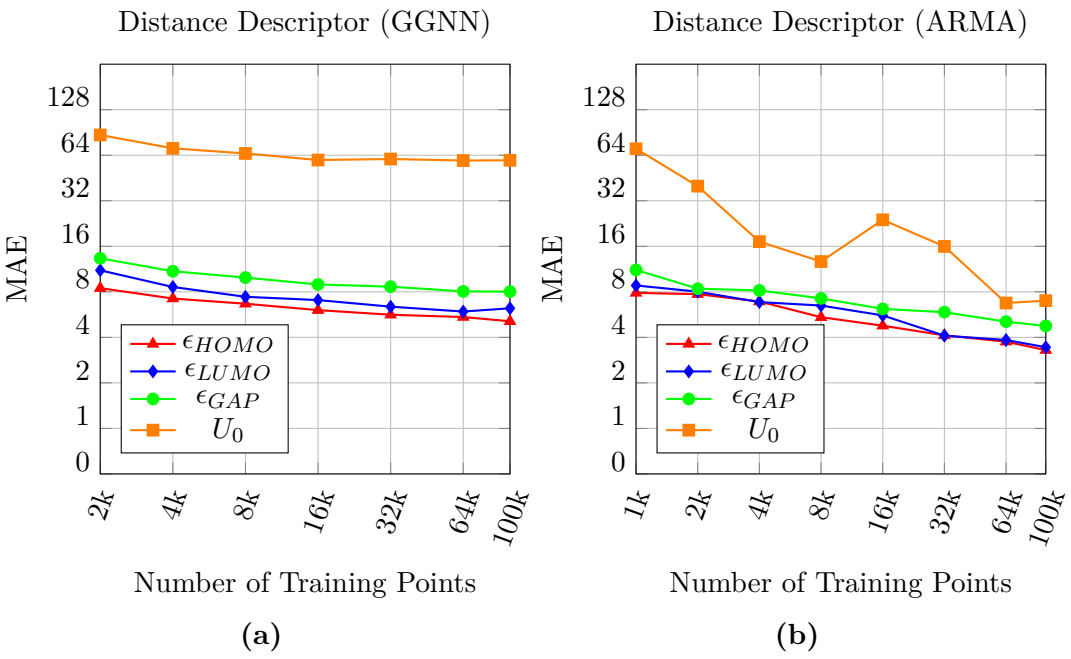


Figure 4.5: Learning curves for Graph Neural Networks with the graph descriptor including bond lengths. Both plots shows MAE values for a varying number of training points. For **(a)**, we used a GGNN, and a custom ARMA model for subfigure **(b)**.

The GNN models were able to utilise the advances of the new descriptor. The GGNN shows its ability to achieve a low error even on a few training points again, and the MAE values of the ARMA model outperforms every experiment done so far. Since the choice of the descriptor has a huge impact on the performance, we extended the graph descriptor with the angles between the bonds of the molecule. The bond angles were calculated based on the atomic coordinates given in the QM9 dataset. Learning curves for this descriptor alone and in combination with the bond distances can be found in Appendix C.2.

The exact results for all the experiments on the prediction of quantum chemical properties, including the results from the appendix, are summarised in Table 4.2. We do not report the results for $> 100k$ training points, since these are just used to show the learning progress in the learning curve plots. The exact value of the intermediate results are not essential.

Table 4.2: A summary of the property prediction results. The Table shows the MAE values in $kcal/mol$ for all combinations of descriptor and model architecture. The best results for each property (written in bold) were achieved with a graph-based descriptor including angles and lengths of the bonds. However, the graph-based representation with bond lengths only grants the best trade-off between performance and size of the descriptor. The first row shows the MAE values for them mean model. For all n molecules, the mean model outputs the average value of the respective property y: $MAE_y = \frac{1}{n}\sum_{i=1}^{n}|y_i - mean(y)|$. The MAEs of the mean model provide benchmarks for every property.

Descriptor	**Architecture**	$\epsilon_{\mathbf{HOMO}}$	$\epsilon_{\mathbf{LUMO}}$	$\epsilon_{\mathbf{GAP}}$	$\mathbf{U_0}$
Mean model	-	10.20	24.11	24.77	187.79
Graph-based	CNN	5.01	6.20	7.78	20.47
	RNN	3.96	3.84	5.65	9.66
	GGNN	6.94	16.99	18.03	108.19
	ARMA	3.61	4.30	5.28	31.58
Graph-based with bond lengths	CNN	5.38	6.04	7.50	33.45
	RNN	3.73	3.55	5.45	12.14
	GGNN	5.11	6.23	8.03	59.26
	ARMA	3.30	3.44	4.76	7.00
Graph-based with bond angles	CNN	6.00	6.33	7.54	66.79
	RNN	3.88	3.89	5.56	10.39
	GGNN	3.22	3.26	4.65	8.57
	ARMA	3.22	2.78	4.76	12.91
Graph-based with bond lengths and bond angles	CNN	5.60	6.34	7.64	18.89
	RNN	3.52	3.26	5.03	**6.43**
	GGNN	**3.15**	3.29	5.10	7.42
	ARMA	3.32	**2.95**	**4.74**	11.36

4.1.6 Discussion

To structure the latent space according to a certain property, CGVAE uses two neural networks to regresses this property from the latent vectors. In this first part of the experiments, this task is trained on different machine learning models. CGVAE's property prediction is simulated by training a GGNN, which is the model used in CGVAEs encoder. For this experiments, the GGNN is equipped with a dense one-layer neural network to generate a scalar and not a latent vector. For comparison, classical CNN and RNN architectures were also tested. Likewise, we evaluated an ARMA model that was made especially for data with graph structure.

In the first part, a molecule was represented as a graph using a set of nodes and a set of edges. Although the representation used contains no information about the spatial placement of the atoms, some of the tested models can learn very well on this graph representation. With this representation, the RNN and the ARMA model, in particular, were able to predict certain properties better than with a SMILES-based descriptor. However, there is a large variance in individual properties. While the prediction accuracy of some properties is high, it is low for other properties. To overcome this issue, we added the bond lengths to the graph descriptor. This had two effects: first, the prediction accuracy of all properties were slightly higher. And second, the variance over all the properties was reduced by the added bond lengths.

Unfortunately, the GGNN did not outperform the other models. Regardless if the graph descriptor is enlarged with bond lengths, both the classical RNN and ARMA model show a lower prediction error. Only the classical CNN performed worse than CGVAEs encoder. Although the prediction accuracy of the GGNN model was not very high, the results were resistant to a low number of training data. This is indicated by the flat training curves in the GGNN model.

4.2 Molecule Generation with Decoder

The second part of our experiments deals with the evaluation of the decoder of CGVAE (Liu et al. 2018). According to the taxonomy used for this section, we now focus on the two loss terms L_{QED} and L_{Recon}. To get reconstructed molecules, both encoder and decoder are combined for this part. Since we now use two loss terms for the training, a relevant question is how to balance them. For this section, we set $\lambda_2 = 10$. This decision is based on empirical findings. In the following experiments, we annotate each molecule in the dataset with its ϵ_{GAP} property in Hartrees instead of the QED score. Therefore, we write L_{GAP} instead of L_{QED} from now on.

Initially, we train CGVAE and report training statistics for both loss terms. The aim of this experiment is to find the numbers of both training epochs and latent dimensions needed to maximise the reconstruction accuracy. This is measured in terms of the structural similarity of the original and the reconstructed molecules, as

well as with the property difference of both molecules.

4.2.1 Training of CGVAE

With the following experiments, we want to find the number of training epochs and the number of latent dimensions that yield the most accurate reconstructions. Therefore, several CGVAE models with different parameter settings are trained. In Figure 4.6, the training losses for all different frameworks can be seen.

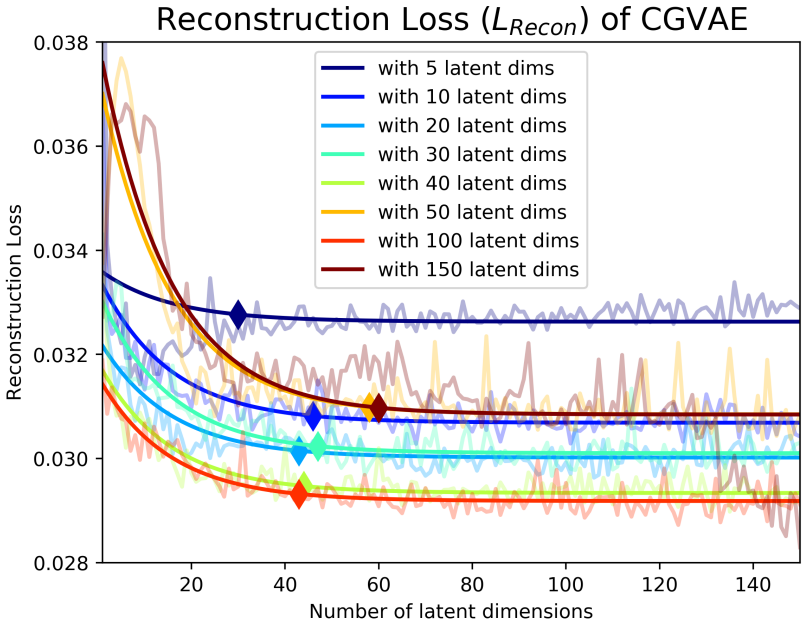
The results show that for every number of latent dimensions, the loss begins to converge after ≤ 60 epochs of training. A smaller number of latent dimensions tends to take fewer epochs to converge than a large number of latent dimensions. The vertical displacement of the curves in Figure 4.6a has no particular meaning. The displacements are due to the stochastic nature of CGVAE, and cannot be compared directly. It would be incorrect to conclude that 100 latent dimensions bring the best accuracy, and five latent dimensions the worst accuracy. Each execution of the experiment shows a different vertical arrangement of the curves. If averaged over several runs of the experiment, the curves would overlap, but this would reduce readability.

4.2.2 Reconstruction of Molecules

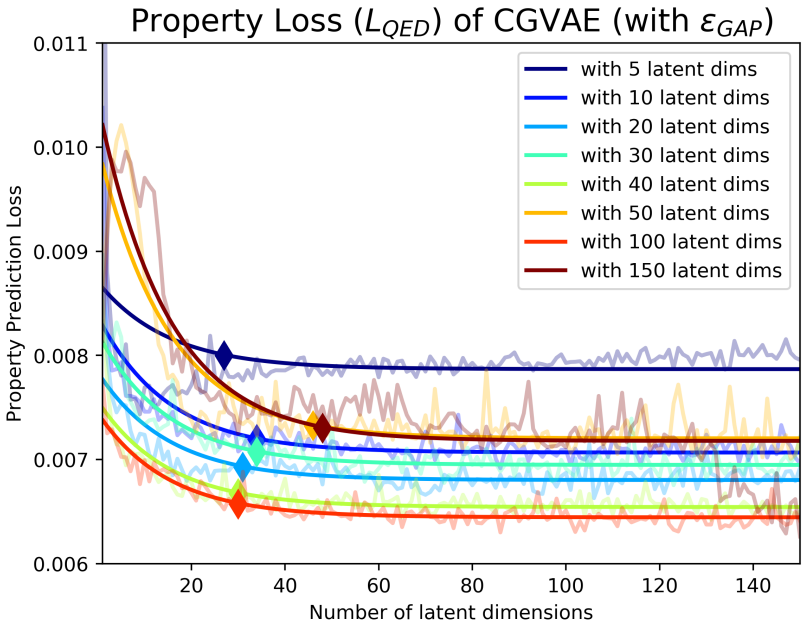
Because the vertical arrangements of the training curves in Figure 4.6 are caused by the stochastic character of the training process, they cannot be used to make a statement about the best number of latent dimensions. Instead, we let the model reconstruct molecules from a validation data set and use them to conduct some measurements. To disable any sampling during the generation process, ϵ, which is used in the reparametrisation trick (see Figure 2.3), is set to zero. The weight for $\mathcal{L}_{GAP}$, which is given by λ_2, is set to 10 for this experiment. In Figure 4.7, we compare the original to the reconstructed molecules in three different ways: first, we report the percentage of molecules, which can be perfectly reconstructed. Second, the percentage of matching nodes in both molecules is shown. Lastly, the ϵ_{GAP} property of the original and the reconstructed molecules are analysed.

4.2.3 Discussion

The second part of our experiments focused on the reconstruction ability of CGVAE, which corresponds to the $\mathcal{L}_{Recon}$ loss term. In other words, the capability to encode a given molecule into a latent vector, and decode a molecule from it which is as similar to the original as possible. In contrast to the first part, we need both en- and decoder to reconstruct molecules. The goal is to find the number of training epochs and the dimensionality of the latent vectors that yield the most accurate



(a)



(b)

Figure 4.6: CGVAE trains on a loss composed of three seperate terms: $\mathcal{L}_{CGVAE} = \mathcal{L}_{Recon} + \lambda_1\mathcal{L}_{KL} + \lambda_2\mathcal{L}_{QED}$. Both plots show how one specific loss term evolves over time for a variety of latent dimensions. For each colour, the original loss data is displayed as a transparent curve. To factor out the fluctuations in these curves, we regressed them with a polynomial of order 4, which is shown as a solid line. The corresponding diamond marks the point where the derivative vanishes for the first time. The losses were calculated using a subset of QM9 containing 13082 molecules.

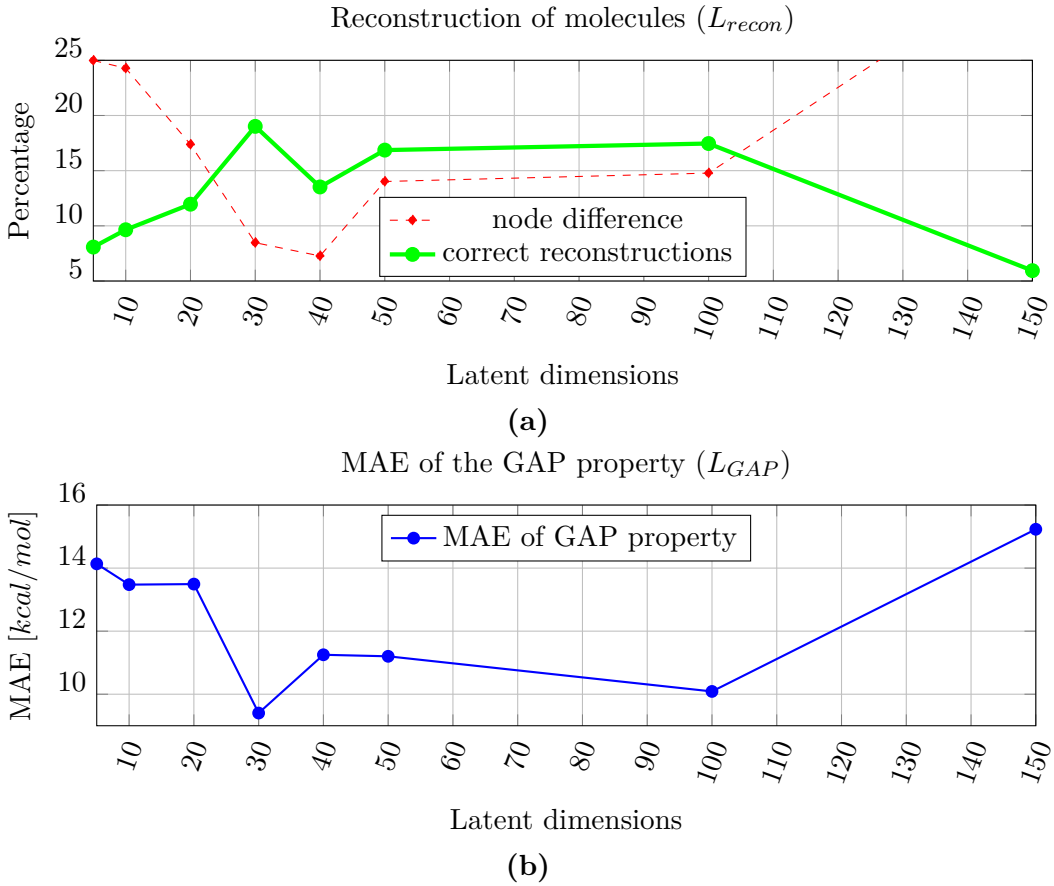
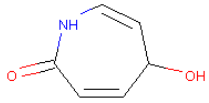
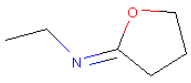
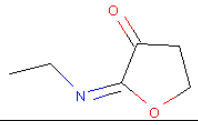
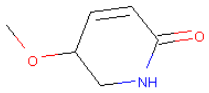
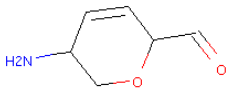
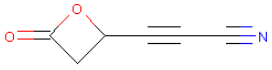
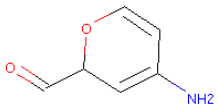
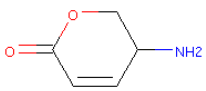
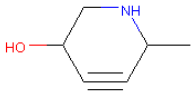


Figure 4.7: Molecule recostructions with different latent sizes. The measurements in plot **(a)** correspond to L_{recon}. With 30 latent dimensions, the reconstructed molecules have 90 % of the atoms of the original molecules (error of 8.5 %). Since the reconstructed molecule is a random configuration of the atoms in the latent space, only 19 % can be correctly reproduced (see Figure 3.2). The mean difference in GAP property, L_{GAP}, can be seen in **(b)**. 30 latent dimensions seem to be a suitable number of latent dimensions for our next experiments.

Table 4.3: Visualisation of a molecule, which is reconstructed with different numbers of latent dimensions. We also report the ϵ_{GAP} property of each reconstruction in kcal/mol. The last row shows the absolute deviation of this property from the original molecule. It can be seen that the reconstruction with 30 latent dimensions is the most accurate.

	Original	5	10
SMILES	OC1C=CNC(=O)C=C1	CCN=C1CCCO1	CCN=C1OCCC1=O
Image			
ϵ_{GAP}	20.818	12.151	80.833
MAE	0.0	8.666	60.016

	20	30	40
SMILES	COC1C=CC(=O)NC1	NC1C=CC(C=O)OC1	N#CC#CC1CC(=O)O1
Image			
ϵ_{GAP}	53.118	24.915	5.310
MAE	32.299	4.097	15.508

	50	100	150
SMILES	NC1=CC(C=O)OC=C1	NC1C=CC(=O)OC1	CC1C#CC(O)CN1
Image			
ϵ_{GAP}	16.466	54.926	51.908
MAE	4.352	34.108	31.089

reconstructions. First, we monitored both losses, $\mathcal{L}_{Recon}$ and $\mathcal{L}_{GAP}$ during the training process. Unfortunately, these numbers did not make a statement about the required number of latent dimensions, due to the stochastic nature of the training process. This behaviour of the training is mainly caused by the random split of the training data in mini-batches. However, we have found that after 70 epochs at maximum, the loss converges.

Because we could not determine all wanted numbers in the first experiment, we generated a series of test molecules and compared the reconstructions to the original molecules. To do so, we also counted the node difference in both molecules as well as their difference in the target property. All tests show that CGVAE achieves the most accurate reconstructions with 30 latent dimensions.

However, the lowest MAE value of 9.40 $kcal/mol$ is far off the results found in the fist part of the experiments (see Section 4.1). This could have multiple reasons: first, CGVAE computes its property estimates with the linear transformations g_1 and g_2 (see Section 3.2), while we mainly used nonlinear activations for our property prediction experiments (see Section 4.1). Second, CGVAE jointly optimises three loss terms (see Section 3.1.1), while there is only one loss term when predicting quantum chemical properties. However, the λs in CGVAE's loss function weight the individual terms against each other. In Appendix E, the experiments shown in Figure 4.7 are repeated with a larger value for λ_2, which controls $\mathcal{L}_{GAP}$.

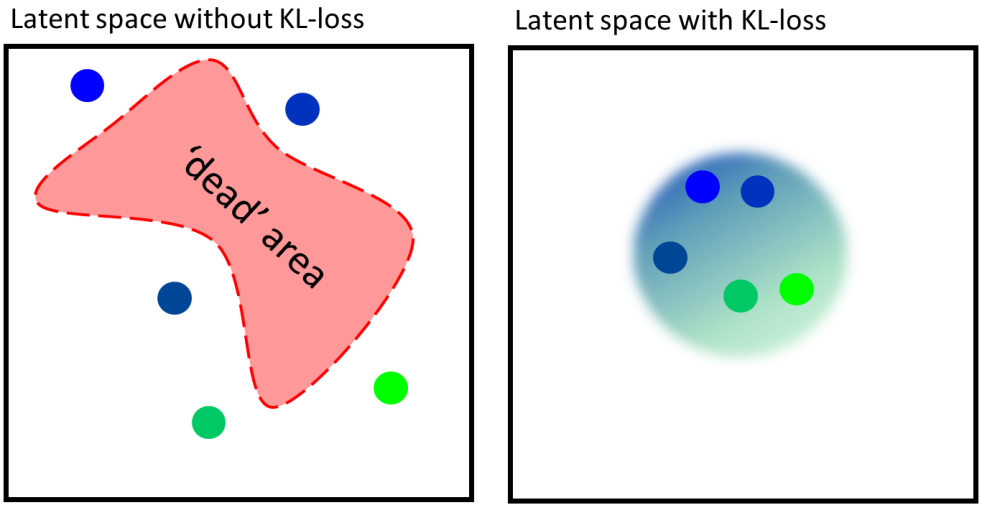
4.3 Structuring of Latent Space

In the last part of the experiments, we focus on all three loss terms together: $\mathcal{L}_{CGVAE} = \mathcal{L}_{Recon} + \lambda_1\mathcal{L}_{KL} + \lambda_2\mathcal{L}_{GAP}$. In contrast to the previous parts, we now include L_{KL}, the characteristic loss term of any VAE architecture. The individual terms are weighted by $\lambda_1 = 0.3$ and $\lambda_2 = 10$.This loss shapes the latent space $q_\phi(z|x)$ according to a predefined prior distribution $p_\theta(z) = \mathcal{N}(0, 1)$. This artificial tailoring of the latent space has both advantages and disadvantages.

On the one hand, the KL divergence causes the latent vectors to be concentrated in the same area. This has the effect that an original molecule x cannot be as accurately reconstructed as without the KL loss. Without the KL loss term, a VAE would distribute the latent vectors all over the latent space to differentiate the individual molecules as well as possible.

On the other hand, the new loss term allows sampling in the continuous latent space. Without the KL loss, the latent vectors would spread out and the area between two samples does not represent another sample x. The effect of the KL-loss on the latent space is illustrated in Figure 4.8.

An important parameter here is the sample variance which is controlled by ϵ. As already discussed, it is a result of the reparametrisation trick illustrated in Figure



(a) without $\mathcal{L}_{KL}$ **(b)** with $\mathcal{L}_{KL}$

Figure 4.8: The impact of the KL divergence loss term on the latent space. **(a)** shows a typical latent space in the absence of the KL loss term. To distinguish the samples x, the VAE placement of the latent vectors would be widespread. Unfortunately, deserted areas between the latent vectors prevent us from sampling in the latent space. In subfigure **(b)**, the additional loss term forces the VAE to learn a compact and continuous representation of the latent space. In order to reconstruct the GAP property as well as possible, the latent space is structured accordingly. The circle with the colour gradient should represent a normal distribution, which is structured based on the GAP property.

Table 4.4: Metrics for the generation of new molecules. We trained CGVAE with different values for ϵ, which affects the sample variance. The training was done using the insights gained in previous experiments. That is 60 epochs and 30 latent dimensions. Although the decoder of CGVAE always outputs valid molecules, the other metrics show clear trends: because the applied QM9 dataset does not contain the same molecule twice , which means 100 & unique molecules, a low sample variance achieves a high uniqueness value and vice versa. The novelty score increases with the rise in ϵ. According to our experiment, there is no value for ϵ with which one could reproduce the values reported in the CGVAE paper.

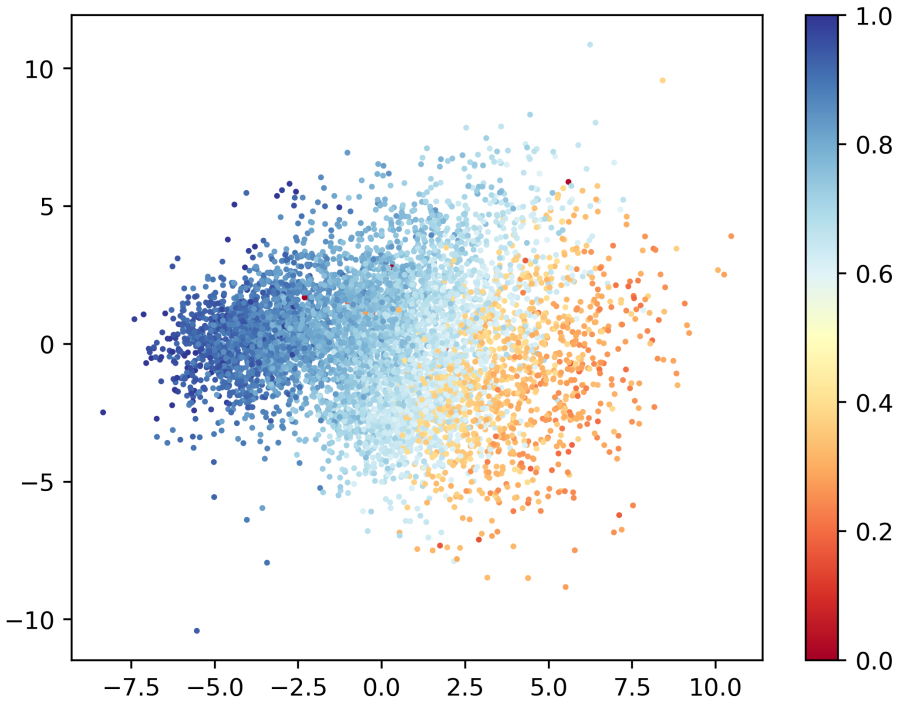
Value of ϵ	Novelty	Uniqueness	Validity
$\epsilon * 2.5$	84.84 %	89.18 %	100 %
$\epsilon * 2$	86.35 %	90.71 %	100 %
$\epsilon * 1.5$	88.75 %	96.10 %	100 %
$\epsilon * 1$	87.10 %	94.75 %	100 %
$\epsilon * 0.5$	79.71 %	96.60 %	100 %
$\epsilon * 0.1$	63.78 %	99.14 %	100 %
values reported in the paper	94.35 %	98.57 %	100 %

2.3. For the following experiment, summarised in Table 4.4, we evaluate the molecule generation ability of CGVAE for different values to scale the variance of ϵ. For each value, we report three measurements, which are very common in the research field of molecule generation with machine learning:

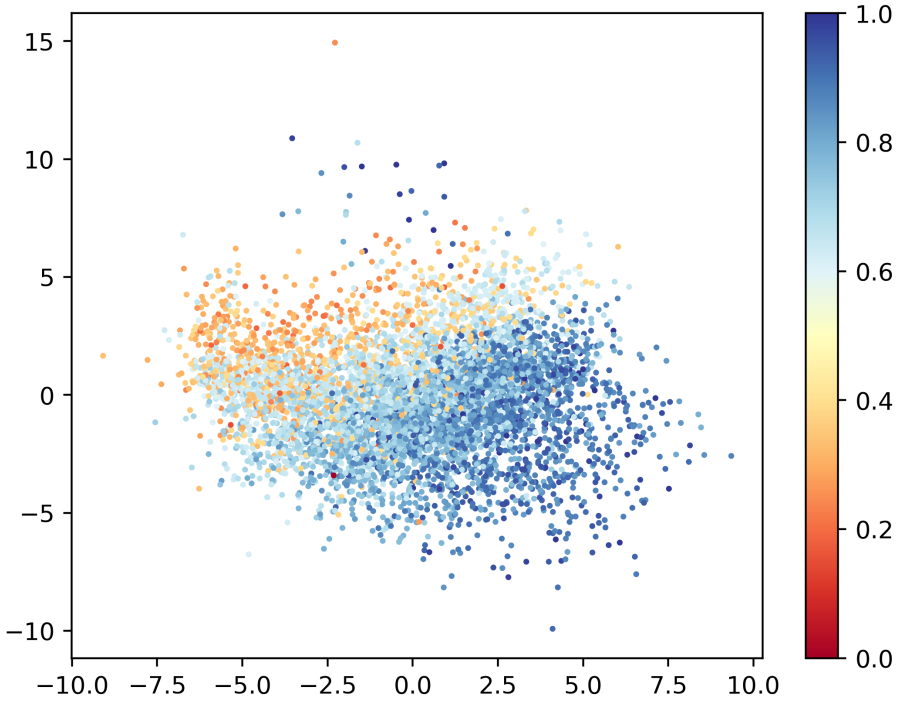
- **Novelty**: The ratio of generated molecules, that are not in the training dataset.
- **Uniqueness**: The ratio of generated molecules, that are not generated twice.
- **Validity**: The ability of the decoder to generate syntactically valid SMILES strings.

In Figure 4.9, an attempt to visualise the latent space can be seen. Since CGVAE uses more than two dimensions for the latent space, we could not plot these vectors directly. PCA was used to search for a two-dimensional subspace which best describes the latent space. For the visualisations of the latent space, we set $\lambda_2 = 1000$.

Although we characterised the latent space for every tested value of ϵ, we only show the latent spaces for the highest and the lowest value in Figure 4.9. The plots for all intermediate values for ϵ can be found in Appendix G.



(a) $\epsilon * 0.1$



(b) $\epsilon * 2.5$

Figure 4.9: Probable visualizations of the latent space, reduced to two dimensions. Latent spaces for two different sample variances can be seen. Each dot represents a node that can be used for the molecule reconstruction. For an encoded molecule, the N atoms in the latent space are coloured according to its ϵ_{GAP} property value. The larger sample variance in **(b)** shows a slightly smoother colour gradient than in **(a)**.

4.3.1 Discussion

The last part of our experiments adds the KL divergence to the loss function. The additional term shapes the latent space according to a tractable probability distribution. This ensures that the latent space is continuous, and one can decode sampled vectors from the latent space. In the case of CGVAE, the target distribution is a Gaussian parametrised by a mean μ and a variance σ^2. The idea is to use a random vector, drawn from this Gaussian, to create a molecule that has never been seen before. The variance has a huge impact on the number of unique and novel molecules. However, this poses additional requirements on the decoder. For instance, the decoder has to ensure that the molecules found are chemically stable.

Our first experiment deals with different sizes for the sample variance. The sample variance can be controlled by multiplying $\epsilon\, \mathcal{N}(0, 1)$ (see 2.3) with various numbers. The found results can be explained in an intuitive way: on the one hand, a higher sample variance leads to nodes or latent vectors, that were not present in the original molecule. The chance of combining them into a completely new molecule increases.

On the other hand, a small sample variance ensures that the sampled nodes are not far from the nodes of the original and unique molecules. Therefore, the percentage of unique molecules inversely correlates with the choice of ϵ. These explanations underline the results shown in Table 4.4. However, as a result of the opposite trends, the values reported in the original paper (Liu et al. 2018) cannot be confirmed.

In the last experiment, an attempt to visualise the latent space was made. Since there are multiple latent vectors for a molecule, we average them to get a single vector representing the encoded molecule. To visualise this vector, it must be reduced from 30 to two dimensions. We used PCA for this dimensionality reduction. Then, each vector was coloured according to the GAP property of the original molecule. In Figure 4.9, it can be seen that the latent space of CGVAE is actually structured based on these values. It can also be seen that an increased value for epsilon leads to a more diffuse gradient.

Chapter 5

Conclusion

The sheer number of potential molecules holds a great opportunity – however, classical methods for molecule discovery lack in scalability to take it. First, traditional methods for molecule generation scan a vast database with already known compounds. Although such a database is usually immense, it only represents a tiny fraction of all possible compounds. Then, the most suitable molecule is selected from the database using computationally exhausting and inefficient procedures. In recent years, attempts have been made to solve this problem by using more efficient machine learning models. The dominant model architecture used in these papers is based on the variational autoencoder (VAE) architecture. A VAE is a concatenation of two artificial neural networks called an en- and decoder, respectively. During training, the encoder learns a sparse vector of the original molecule, and the decoder then reconstructs the input molecule from the learnt vector. These vectors are called latent vectors. If the VAE is trained, the decoder can be used to translate random vectors into new molecules, which were unseen during training.

Discussion of the Model The publications that generated new molecules with a VAE differed significantly in the representation of the molecules. Attempts like the Chemical VAE (Gómez-Bombarelli et al. 2018) used the simplified molecular-input line-entry specification (SMILES) to describe the molecules used. SMILES are simple string representations of molecules, which contain information about both the atoms and bonds involved. However, the strict syntax of SMILES is a severe obstacle for machine learning models and must be enforced with hand-crafted rules. Moreover, SMILES are not invariant to permutations of the molecule and the strings do not contain spatial information about the molecule.

In this thesis, we evaluated the constraint graph variational autoencoder (Liu et al. 2018), a VAE for molecule generation, which represents molecules as graphs. By opting to work with a graph-based descriptor, CGVAE does not eliminate all the mentioned problems. However, they no longer have to worry about the complex

syntax of SMILES. Instead, CGVAE uses masks to ensure the validity of the generated molecules. Their training objective, called loss function, consists of three separate terms: $\mathcal{L}_{CGVAE} = \mathcal{L}_{Recon} + \lambda_1 \mathcal{L}_{KL} + \lambda_2 \mathcal{L}_{GAP}$, where the λs are trade-off parameters. The meaning of these terms is briefly introduced in the following list:

- $\mathcal{L}_{Recon}$: This loss term forces CGVAE to reconstruct molecules which are as similar to the input molecule as possible.
- $\mathcal{L}_{KL}$: This term shapes the latent space according to a tractable probability distribution. Hence, latent vectors can be sampled from this distribution, which then could be decoded into new molecules.
- $\mathcal{L}_{GAP}$: While the other two loss terms are typical for a VAE, this term causes CGVAE to reconstruct molecules which have similar properties to the original. Consequently, latent vectors with different property values are spatially separated. This results in the structuring of the latent space.

Discussion of Experiments The experiments in this thesis were grouped into three parts, where every piece adds a term to CGVAE's loss function: in the first part (see Section 4.1), we focused on the prediction of quantum chemical properties, which use $\mathcal{L}_{GAP}$ as a single loss term. In the first part of our experiments, we modified this term to predict four properties: ϵ_{HOMO}, ϵ_{LUMO}, ϵ_{GAP}, and the internal energy U_0. The reconstruction loss is added in the second part (see Section 4.2), and the full loss function, including $\mathcal{L}_{KL}$, is evaluated in the last part (see Section 4.3). We used the popular QM9 dataset (Ramakrishnan et al. 2014, Ruddigkeit et al. 2012) for the experiments in this work.

First, we found that the predictive accuracy of graph-based descriptors can be improved, by adding spatial information of the molecular graph to the descriptor. Furthermore, the graph-based descriptor allows traditional recurrent neural network architectures to compete with models, which were explicitly designed for learning on graph-structured data. If we incorporate both lengths and angles of the molecule into the descriptor, we achieve a prediction accuracy ranging from 6.43 to 2.95 kcal/mol. To put this into perspective, the accuracy of an *ab initio* method is $\approx$ 1kcal/mol. Although the variance in our results is large, the best result is not far from the 1 kcal/mol benchmark.

To optimise L_{Recon} together with L_{GAP}, we combined the en- and decoder of CGVAE. Models with different sizes of the latent space were trained and used to reconstruct a subset of the QM9 molecules. We compared the reconstructed molecules by various aspects to determine the dimensionality of the latent space, which maximises the reconstruction ability of CGVAE. First, we counted the atoms in both molecules: due to the complicated molecule reconstruction method of CGVAE, the number of non-hydrogen-atoms in both the reproduced and the original molecules differ by 0.64 atoms on average. For comparison, the mean molecule in the utilised dataset

has 8.82 non-hydrogen-atoms. Second, we compared both molecules in terms of the sensitive energy gap ϵ_{GAP}. The reconstruction of molecules is a more sophisticated task than the prediction of quantum chemical properties. Therefore, our results show an error of the ϵ_{GAP} property of 6.49 kcal/mol.

The last loss term, L_{KL}, forms the latent space according to a continuous distribution, which allows the sampling of latent vectors. The sampled vectors can then be decoded into new molecules. In the research field of molecule generation with machine learning, multiple metrics are used to test the generation ability of the models. In our experiments, we demonstrated a trade-off between these metrics. Therefore, it was not possible to optimise all metrics together, which prevented us from reproducing the results reported in the CGVAE paper (Liu et al. 2018). Although all of the generated molecules are valid, 87 % are not present in the QM9 training data set, and only 5 % of them are duplicates.

Overall, our results indicate that machine learning models can discover new compounds. However, much more innovation is required to optimise the molecules precisely for desired quantum chemical properties than with conventional *ab initio* methods.

Future Work CGVAE chooses to generate molecules sequentially. To generate a new molecule, the encoder first samples a pool of N nodes in the latent space, where N is an upper bound on the atoms in the original molecule. Then, the decoder randomly chooses a node and starts building a partial molecular graph, to which it adds more and more nodes from the pool until a termination condition is met. This procedure explains the small number of molecules that can be reconstructed correctly. With 30 latent dimensions, this number barely reaches 20 % (see Figure 4.7). We think that this ratio could be significantly increased if the decoder reconstructs the whole molecule in one step. Therefore, further studies should investigate in the one-shot generation of molecules.

Also, the sequential generation of a molecule takes a relatively long time. With the infrastructure used for our experiments, the generation of a single molecule took about two seconds. The time required is also the reason why the experiments in the second and third part of our experiments are performed on a rather small dataset. Further research could perform more extensive tests with the one-shot generation.

Also, we did not solve the problem of permutation symmetric molecule descriptors. Our graph descriptor puts the nodes of a graph in an arbitrary order. A clear structure is necessary to refer to the individual nodes, which in turn is essential to characterise the edges in the graph. The used descriptor can represent a molecule in several ways by selecting different node orderings. Looking forward, further research could develop a graph descriptor which is invariant to permutations of the molecule.

Bibliography

Bianchi, F. M., Grattarola, D., Alippi, C. & Livi, L. (2019), 'Graph neural networks with convolutional arma filters', *arXiv preprint arXiv:1901.01343* .

Blei, D. M., Kucukelbir, A. & McAuliffe, J. D. (2017), 'Variational inference: A review for statisticians', *Journal of the American Statistical Association* **112**(518), 859–877.

Defferrard, M., Bresson, X. & Vandergheynst, P. (2016), Convolutional neural networks on graphs with fast localized spectral filtering, *in* 'Advances in neural information processing systems', pp. 3844–3852.

Doersch, C. (2016), 'Tutorial on variational autoencoders', *arXiv preprint arXiv:1606.05908* .

Duchi, J., Hazan, E. & Singer, Y. (2011), 'Adaptive subgradient methods for online learning and stochastic optimization', *Journal of machine learning research* **12**(Jul), 2121–2159.

Eames, E. V. (2019), '3-D Structures of Molecules'.

Gómez-Bombarelli, R., Wei, J. N., Duvenaud, D., Hernández-Lobato, J. M., Sánchez-Lengeling, B., Sheberla, D., Aguilera-Iparraguirre, J., Hirzel, T. D., Adams, R. P. & Aspuru-Guzik, A. (2018), 'Automatic chemical design using a data-driven continuous representation of molecules', *ACS central science* **4**(2), 268–276.

Hamilton, W., Ying, Z. & Leskovec, J. (2017), Inductive representation learning on large graphs, *in* 'Advances in neural information processing systems', pp. 1024–1034.

Huang, B., Symonds, N. O. & von Lilienfeld, O. A. (2018), 'The fundamentals of quantum machine learning', *arXiv preprint arXiv:1807.04259* .

Kim, K., Kang, S., Yoo, J., Kwon, Y., Nam, Y., Lee, D., Kim, I., Choi, Y.-S., Jung, Y., Kim, S. et al. (2018), 'Deep-learning-based inverse design model for intelligent discovery of organic molecules', *npj Computational Materials* **4**(1), 67.

Kingma, D. P. & Ba, J. (2014), 'Adam: A method for stochastic optimization', *arXiv preprint arXiv:1412.6980* .

Kingma, D. P. & Welling, M. (2013), 'Auto-encoding variational bayes', *arXiv preprint arXiv:1312.6114* .

Kirkpatrick, P. & Ellis, C. (2004), 'Chemical space'.

Landrum, G. (2010), 'Rdkit: Open-source cheminformatics'.
URL: `http://www.rdkit.org`

Li, Y., Tarlow, D., Brockschmidt, M. & Zemel, R. (2015), 'Gated graph sequence neural networks', *arXiv preprint arXiv:1511.05493* .

Liu, Q., Allamanis, M., Brockschmidt, M. & Gaunt, A. L. (2018), 'Constrained graph variational autoencoders for molecule design', *CoRR* **abs/1805.09076**.
URL: `http://arxiv.org/abs/1805.09076`

Nesterov, V., Wieser, M., Wieczorek, A. & Roth, V. (2019), Learning deep representationsfor molecule design, Master's thesis, University of Basel, Department of Mathematics and Computer Science - Biomedical Data Analysis.

Ramakrishnan, R., Dral, P. O., Rupp, M. & von Lilienfeld, O. A. (2014), 'Quantum chemistry structures and properties of 134 kilo molecules', *Scientific Data* **1**.

Ramakrishnan, R., Dral, P. O., Rupp, M. & von Lilienfeld, O. A. (2015), 'Big data meets quantum chemistry approximations: The δ-machine learning approach', *Journal of chemical theory and computation* **11**(5), 2087–2096.

Ruddigkeit, L., Van Deursen, R., Blum, L. C. & Reymond, J.-L. (2012), 'Enumeration of 166 billion organic small molecules in the chemical universe database gdb-17', *Journal of chemical information and modeling* **52**(11), 2864–2875.

Ruder, S. (2016), 'An overview of gradient descent optimization algorithms', *arXiv preprint arXiv:1609.04747* .

Samanta, B., De, A., Ganguly, N. & Gomez-Rodriguez, M. (2018), 'Designing random graph models using variational autoencoders with applications to chemical design', *CoRR* **abs/1802.05283**.
URL: `http://arxiv.org/abs/1802.05283`

Simonovsky, M. & Komodakis, N. (2018), 'Graphvae: Towards generation of small graphs using variational autoencoders', *CoRR* **abs/1802.03480**.
URL: `http://arxiv.org/abs/1802.03480`

Sterling, T. & Irwin, J. J. (2015), 'Zinc 15 ligand discovery for everyone', *Journal of Chemical Information and Modeling* **55**(11), 2324–2337. PMID: 26479676.
URL: `https://doi.org/10.1021/acs.jcim.5b00559`

Tieleman, T. & Hinton, G. (2012), 'Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude', *COURSERA: Neural networks for machine learning* **4**(2), 26–31.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C. & Yu, P. S. (2019), 'A comprehensive survey on graph neural networks', *arXiv preprint arXiv:1901.00596* .

Appendix A

Sequential Molecule Generation of CGVAE in Pseudocode

The sequential generation procedure of CGVAEs decoder. This code assumes that the node states h_v, as well as their order in the *focusqueue*, is already known.

procedure SEQUENTIALGENERATION
  finalGraph ← [ ]
  while not *focusqueue*.empty **do** ▷ Iteration of `FOCUS` and `EXPAND`
    focusnode ← FOCUS()
    EXPAND(*focusnode*, *finalGraph*)
  discardUnconnectedNodes()
  return *finalGraph*

procedure FOCUS()
  return *focusqueue*.pop() ▷ Get a node to focus on

procedure EXPAND(v, G)
  $w \leftarrow allNodesInLatentSpace()$ ▷ Get list of all sampled nodes
  $k \leftarrow allEdgeTypes()$ ▷ Get list of all edge types
  $p(v \leftrightarrow u|\phi^t_{u,v}) = \frac{M^t_{v\leftrightarrow u} exp[C(\phi^{(t)}_{v,u})]}{\sum_w M^t_{v\leftrightarrow w} exp[C(\phi^{(t)}_{v,w})}$ ▷ Softmax distribution of u's
  $p(l|\phi^t_{u,v}) = \frac{m^{t\ l}_{v\leftrightarrow u} exp[L_l(\phi^{(t)}_{v,u})}{\sum_k m^{t\ k}_{v\leftrightarrow u} exp[L_k(\phi^{(t)}_{v,u})]}$ ▷ Softmax distribution of edge label l
  $edgeDistr = p(v \leftrightarrow u|\phi^t_{u,v}) * p(l|\phi^t_{u,v})$ ▷ Combine both distributions
  while True do
    $v \overset{l}{\leftrightarrow} u \leftarrow$drawSample($edgeDistr$) ▷ Sample an edge
    if $u == \oslash$ **or** $u \notin$ *focusqueue* **then** ▷ Check if edge is valid
      break
    G.add($v \overset{l}{\leftrightarrow} u$)
    for *node* in G **do** ▷ Update the node representations h_v
      $m_{node} \leftarrow node.h$
      $i \leftarrow 0$
      for $i < 7$ **do**
        $m^{(i+1)}_{node} = GRU[m^{(i-1)}_v, \sum_{uv} E_l(m^{(i)}_u)]$
        $i++$
      $node.h^{(t+1)} \leftarrow m^{(7)}_{node}$

Appendix B

Property Prediction on the ZINC Dataset

B.1 Baseline Models

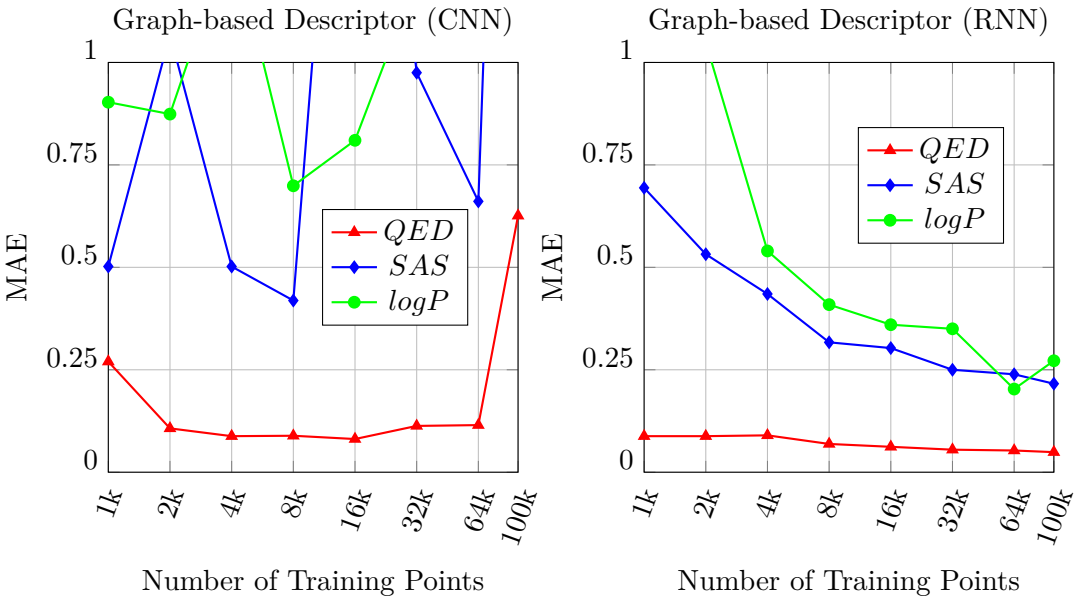
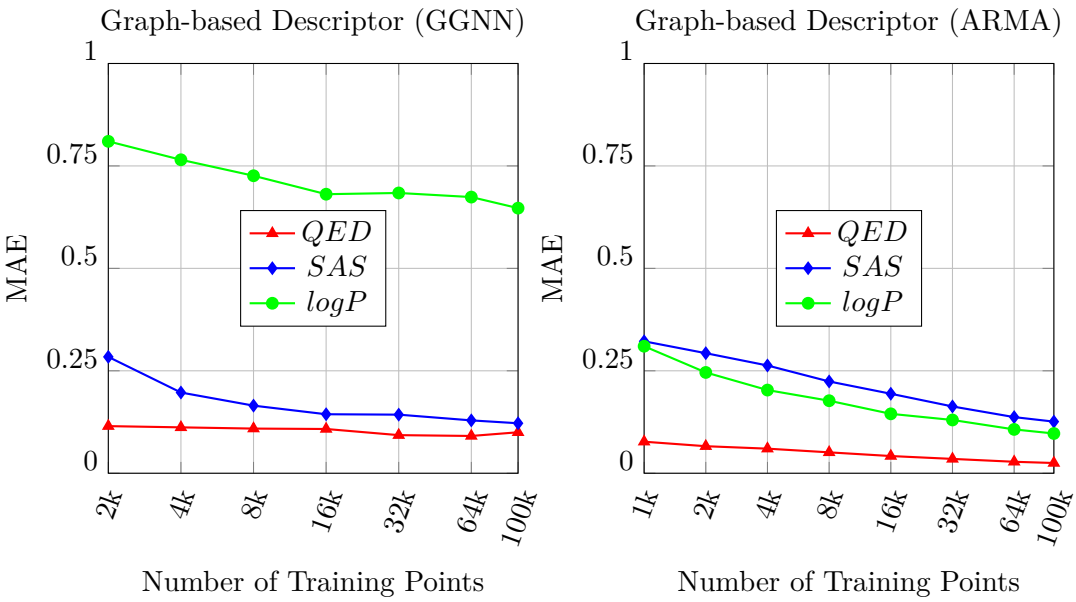


Figure B.1: Learning curves for classical neural networks on the ZINC dataset. For this experiment, we use the baseline models introduced in Section 4.1.1. Subfigure **(a)** depicts the results for the CNN. It can be seen that the CNN is overstrained with the bigger molecules of the ZINC dataset. However, the RNN shows a steep decay of the learning curve **(b)**. With enough samples for training, a good result can be achieved.

B.2 Specifically Designed Models



(a)

Figure B.2: Learning curves for graph neural networks on the ZINC dataset. The models used were introduced in Section 4.1.2. Both plots show MAE values for a varying number of training points. For **(a)**, we used a GGNN, and a custom ARMA model in **(b)**. As with the QM9 dataset, both models achieve good prediction accuracy with just a few training points.

Here, the experiments on prediction of quantum chemical properties (see Section 4.1) are repeated with the ZINC dataset (Sterling & Irwin 2015). In the experiments with the QM9 dataset, we converted the MAE into kcal/mol. The units that were used at ZINC are unknown to us. That is why the scales of the MAE plots are different compared with the experiments on QM9.

Appendix C

Property Prediction with Bond Angles

C.1 Baseline Models

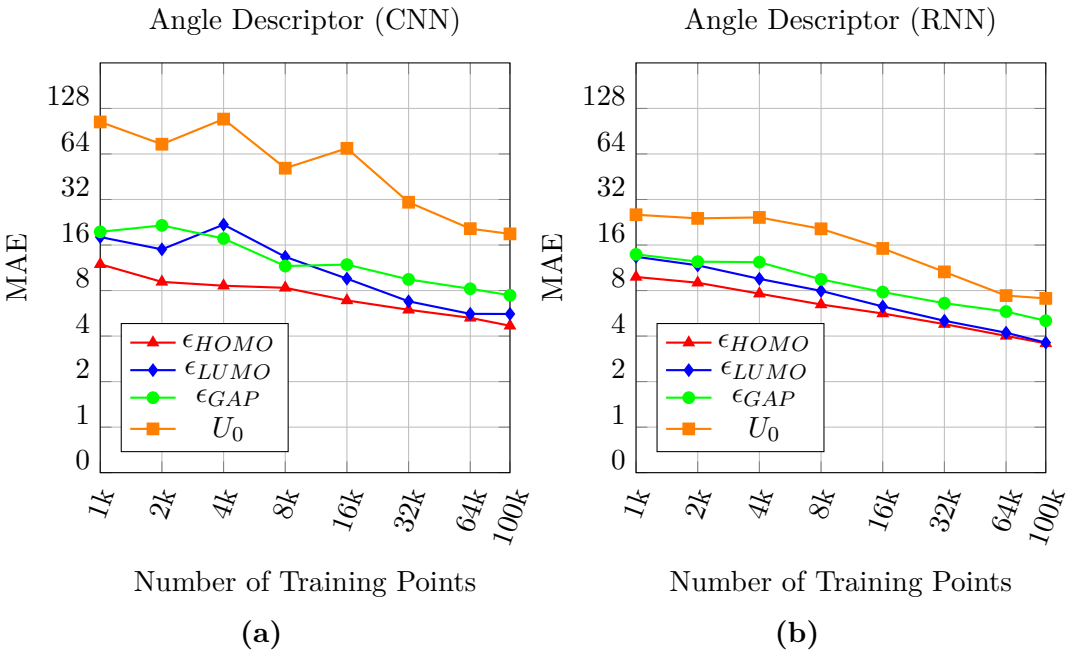


Figure C.1: Learning curves for classical neural networks with the graph descriptor, including bond angles. Subfigure **(a)** depicts the results for the CNN, while **(b)** shows the results for the RNN.

C.2 Specifically Designed Models

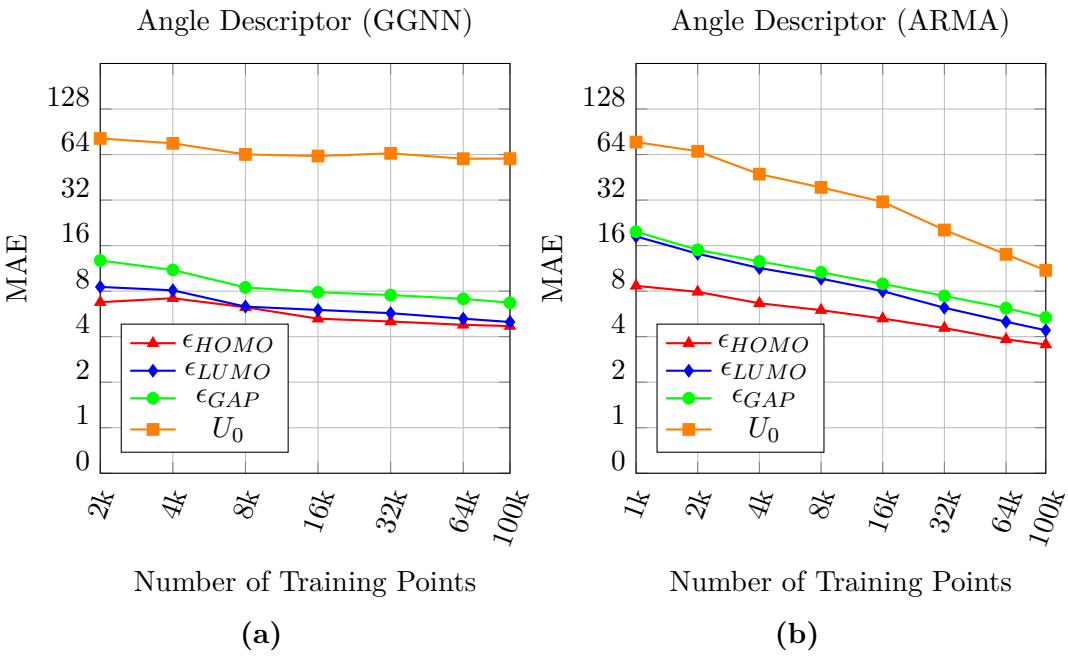


Figure C.2: Learning curves for Graph Neural Networks with the graph descriptor, including bond angles. Subfigure **(a)** depicts the results for the GGNN (en- /decoder of CGVAE), while **(b)** shows the results for the ARMA model.

To incorporate the bond angles, we extend the graph descriptor, consisting out of a set of atoms V and a set of bonds E, by an additional set of angles. It was not possible to integrate the new data into an existing set since angles are defined between two bonds or three nodes. The results are slightly worse than the results on the graph descriptor with the bond lengths.

Appendix D

Property Prediction with Combined Descriptor

D.1 Baseline Models

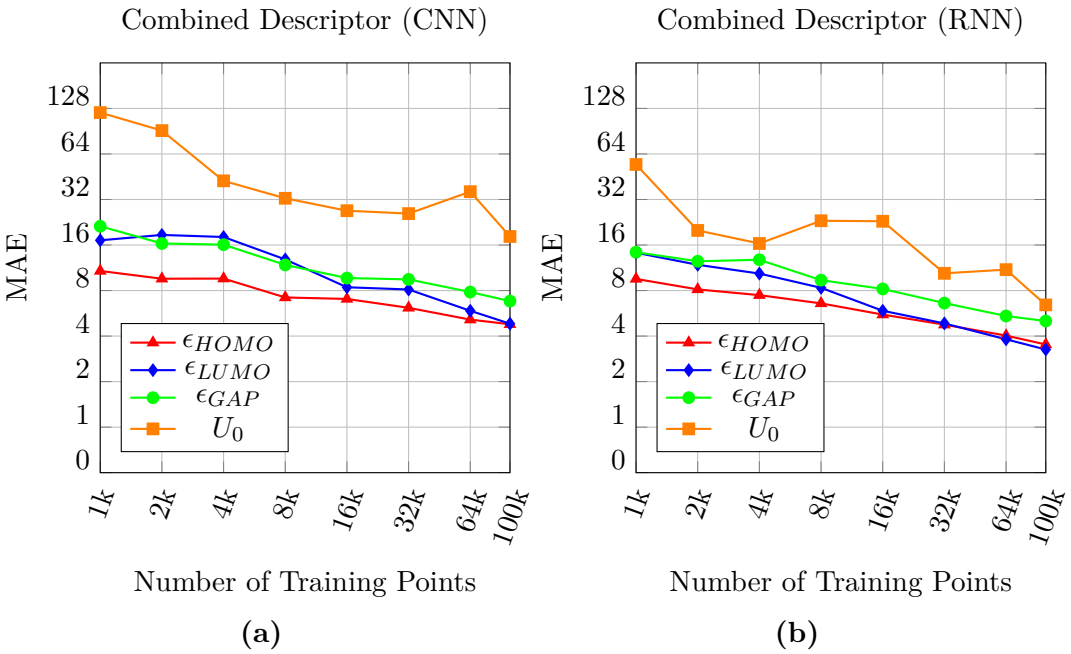


Figure D.1: Learning curves for classical neural networks with the graph descriptor, including bond lengths and angles. Subfigure **(a)** depicts the results for the CNN, while **(b)** shows the results for the RNN.

D.2 Specifically Designed Models

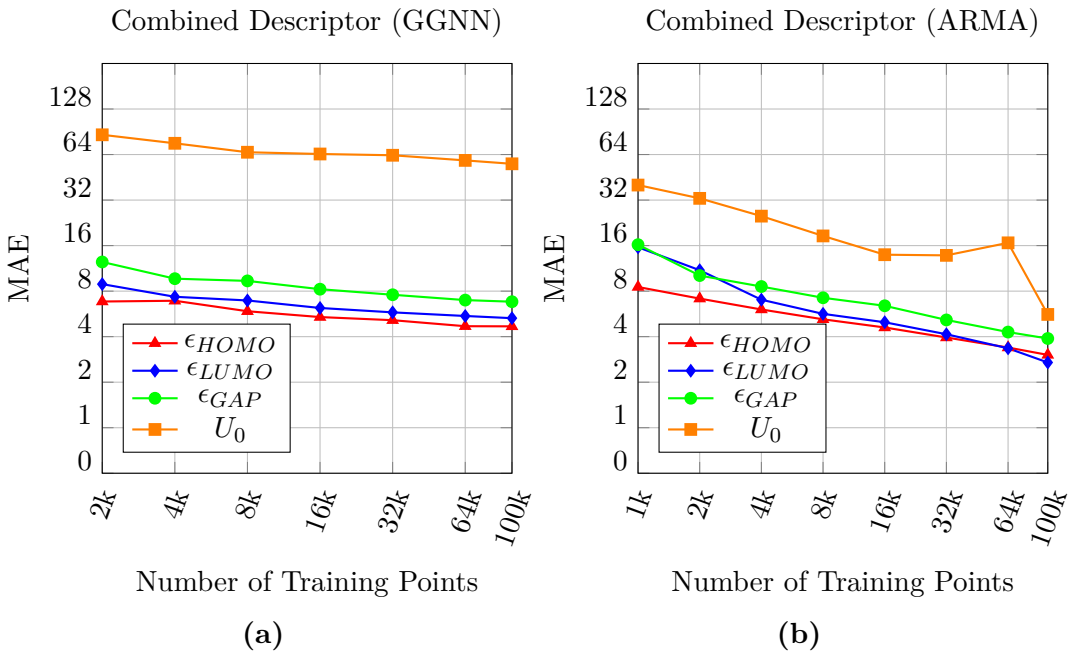


Figure D.2: Learning curves for graph neural networks with the graph descriptor, including bond lengths and angles. Subfigure **(a)** depicts the results for the GGNN (en- /decoder of CGVAE), while **(b)** shows the results for the ARMA model.

In this chapter we combine the angle descriptor with the bond lengths. We hope for a stronger descriptor that combines the predictive power of the molecular graph, the set of angles, and the bond lengths. However, the results show no great difference compared to the previous appendix.

Appendix E

Reconstruction of Molecules with modified Property Loss

Here, we repeat the experiments of Section 4.2.2 with a higher weight for the L_{GAP} loss term. The weight is determined by λ_2, which is set to 1000 for this experiment. In Section 4.2.2, we set λ_2 to 10. The aim was to evaluate both loss terms L_{GAP} and L_{Recon} with a different amount of dimensions for the latent space. The results of this experiment can be seen in Figure E.1.

It can be seen that the updated weight has a positive effect on the property loss of L_{GAP}. While we achieved a minimal MAE of 9.40 kcal/mol with $\lambda_2 = 10$, we now obtain a minimum of 6.49 kcal/mol. However, both Subfigures show ambiguous results for the preferred number of latent dimensions.

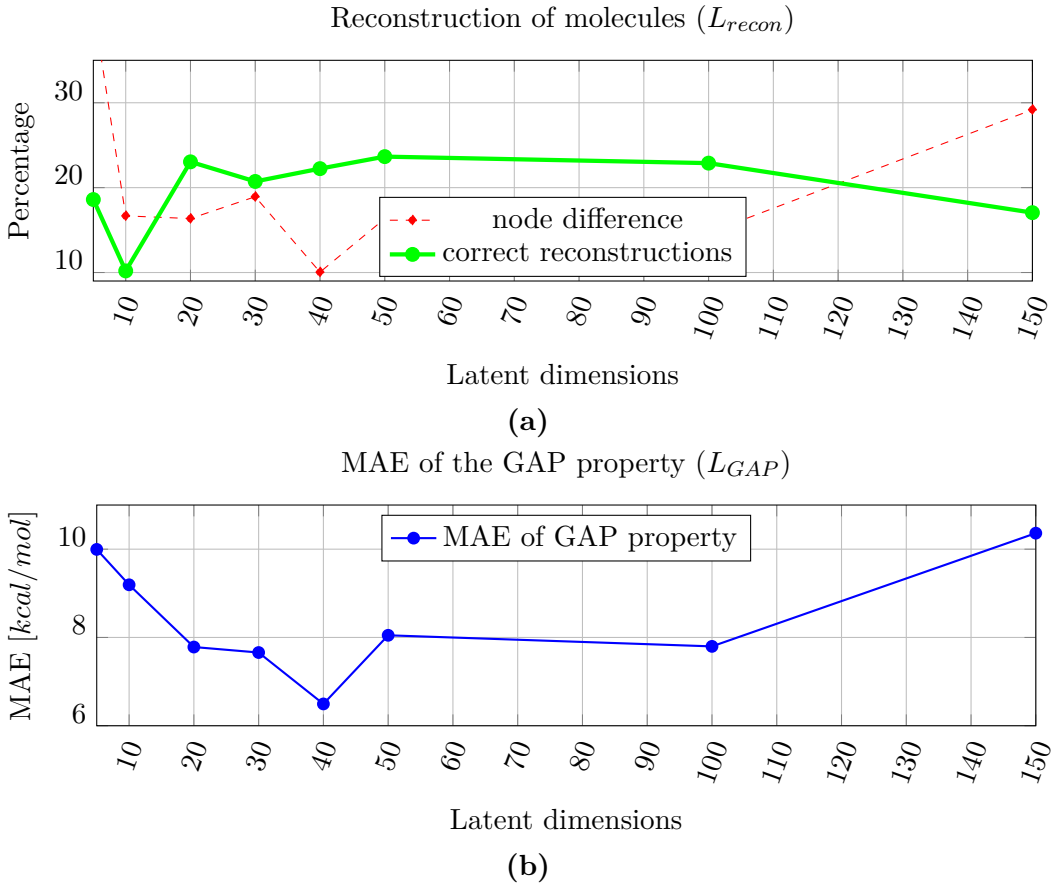


Figure E.1: Molecule reconstructions with different latent sizes. This time, we increased λ_2 to 1000, whereas we used a weight of 10 for the experiment in Section 4.2.2. Despite the good reconstructions, the plots show different values for the optimal number of latent dimensions. In **a**, most molecules can be correctly reconstructed in a with 20 latent dimensions. However, the difference between the atoms of the original and the reconstructed molecule is the smallest with 40 dimensions. Subfigure **b** shows a peak with 40 latent dimensions.

Appendix F

Optimizer

For all the experiments in this thesis, the loss function is minimised with the Adam optimiser (Kingma & Ba 2014). For our work, we use the mean absolute error (MAE) as objective function. This is an enhancement of classical stochastic gradient descent (SGD) and combines the advantages of two other SGD methods - AdaGrad (Duchi et al. 2011), which incorporates the learning rate for each dimension of the loss function separately, and RMSProp (Tieleman & Hinton 2012), which uses a series of previous gradients to compute the current optimisation step. In the classical formulation of SGD, the update rule for a parameter w, cost function J, and learning rate η is

$$w_t = w_{t-1} - \eta g_t, \tag{F.1}$$

where $g_t = \frac{\partial J}{\partial w_{t-1}}$. Instead of updating the parameters directly, which would affect the first-order momentum of the parameters, the Adam optimiser alters the rate of change for the respective parameter, which corresponds to the second-order momentum of the parameters. In the following subsections, we firstly present these related concepts, and then, the ideas introduced are combined to form the Adam optimiser.

F.0.1 Adaptive Gradient Algorithm (AdaGrad)

AdaGrad (Duchi et al. 2011) addresses the problem of multiple dimensions with gradients of different sizes. In classical SGD, a small learning rate may lead to a minimisation along one dimension of the objective function, but has almost no effect on a dimension with a very small gradient. The parameter update rule for AdaGrad becomes

$$w_t \leftarrow w_{t-1} - \frac{\eta}{\sqrt{s_t + \epsilon}} g_t, \tag{F.2}$$

where $s_t \leftarrow s_{t-1} + (g_t)^2$.The derivative of the loss error function with regard to parameter w at time step t is referred to as $g_t = \frac{\partial J}{\partial w_t}$. η is the learning rate and ϵ is just a small number to prevent division by zero.

AdaGrad sequentially updates an auxiliary variable s by adding the squared gradient at each time step t. The division of the gradient g_t with s scales the gradients to the same magnitude. Since s is monotonically increasing, the learning rate η is decreased accordingly.

F.0.2 Root Mean Square Propagation (RMSProp)

However, the build-up of the sum of squared gradients s has the effect that the updates become very small in the long run. RMSProp (Tieleman & Hinton 2012) tackles this issue by redefining s as an exponentially weighted moving average:

$$\begin{aligned} s_t &= \gamma s_{t-1} + (1-\gamma)(g_t)^2 \\ &= (1-\gamma)(g_t)^2 + \gamma s_{t-1} \\ &= (1-\gamma)(g_t)^2 + (1-\gamma)\gamma(g_{t-1})^2 + (1-\gamma)\gamma^2(g_{t-2})^2 + ... + 1-\gamma)\gamma^t(g_0)^2 \\ w_t &= w_{t-1} - \frac{\eta}{\sqrt{s_t+\epsilon}} g_t. \end{aligned} \tag{F.3}$$

where $\gamma \in (0,1)$. It is thus apparent that the coefficients sum to one. This can easily be checked because the coefficients form a Taylor expansion: $\frac{1}{1-\gamma} = 1+\gamma+\gamma^2+...+\gamma^t)$. That ensures that s_t is, in fact, a weighted average where all weights sum up to 1. Figure F.2a shows how the choice of γ affects the weight distribution.

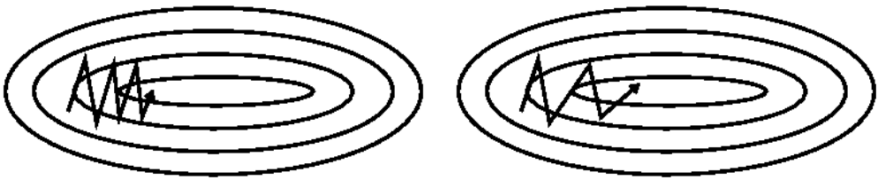
F.0.3 Momentum

Momentum is an improvement on SGD methods which is intended to accelerate the learning process. In a very complex loss function, small displacements may lead to rapid changes in the gradient. There may also be flat areas where the gradient vanishes entirely.

An SGD method with momentum overcomes this issue by smoothing the gradient using the gradients of past time steps. It does so by updating the change rate of a parameter w rather than updating the parameter directly, as shown in Figure F.2a. The change rate of a parameter is also called v, short for velocity. The formulas for SGD with momentum are as follows:

$$\begin{aligned} w_t &= w_{t-1} - v_t \\ v_t &= \gamma v_{t-1} + \eta g_t, \text{ where } 0 \leq \gamma < 1 \\ &= \eta g_t + \gamma v_{t-1} \\ &= \eta g_t + \gamma\eta g_{t-1} + \gamma^2 \eta g_{t-2} + ... + \gamma^t \eta g_0. \end{aligned} \tag{F.4}$$

By updating the change rate v instead of x, we can prohibit fast directional changes in the gradient and keep a principal update direction, even if there is no gradient.



(a) SGD without Momentum **(b)** SGD with Momentum

Figure F.1: The effect of the Momentum method for optimizers. Here, the example of a ball rolling towards the minimum of a loss function is presented. **(a)**: the trajectory a lightweight ball changes its path due to slight changes in the gradient. Subfigure **(b)** shows the route of a heavier ball, which does not fluctuate so much. Figure from Ruder (2016).

Note that the coefficients do not sum to 1 and v is monotonically increasing (like s in AdaGrad). The distribution of the coefficients can be seen in Figure F.2b.

The way the momentum method works can be illustrated using the example of a ball. Imagine a convex loss function of a model with two parameters. A ball is now dropped on the surface of this 3-d function. A heavier ball, analogous to momentum, will approach the optimum in a more direct way than a lighter ball, corresponding to classic SGD. This is illustrated in Figure F.1.

F.0.4 Adaptive Moment Estimation (Adam)

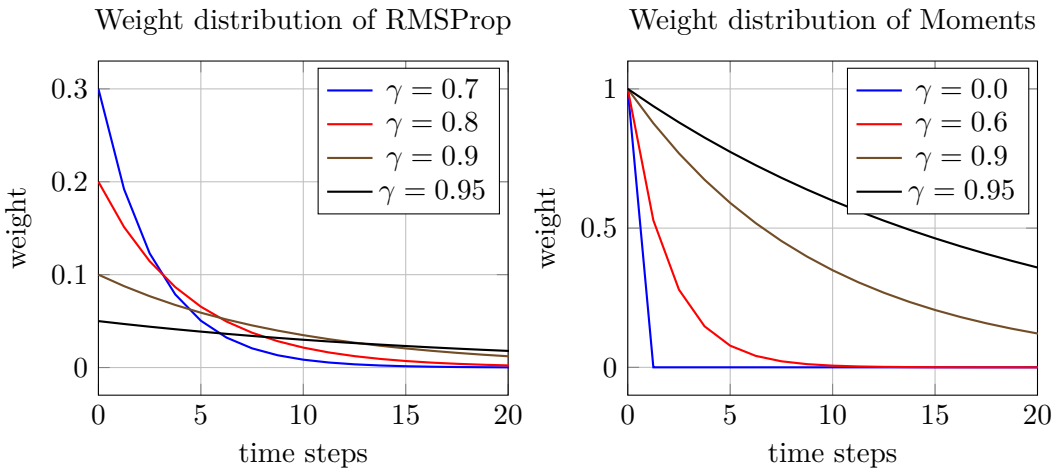
Adam combines RMSProp with the momentum method. Since both methods use a parameter γ, the Adam optimizer uses $0 \leq \beta_1 < 1$ and $0 \leq \beta_2 < 1$ to differentiate them. The learning rate η is also replaced with $(1 - \beta_1)$ and $(1 - \beta_2)$ respectively. The update rules thus become

$$\begin{aligned} v_t &= \beta_1 v_{t-1} + (1 - \beta_1) g_t \\ \hat{v_t} &= \frac{v_t}{(1 - \beta_1^t)} \\ s_t &= \beta_2 s_{t-1} + (1 - \beta_2)(g_t)^2 \\ \hat{s_t} &= \frac{s_t}{(1 - \beta_2^t)} \\ x_t &= x_{t-1} - \frac{\eta \hat{v_t}}{\sqrt{\hat{s_t}} + \epsilon}. \end{aligned} \tag{F.5}$$

For the first few time steps, v_t and s_t are very sensitive to β_1 and β_2. For example, the official TensorFlow API[2] suggests setting $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 1e - 08$. This means that $v_1 = 0.001(g_1)^2$ and $s_1 = 0.1 g_1$. However, if we use $w_2 = w_1 - \frac{\eta v_1}{\sqrt{s_1} + \epsilon}$

[1]Image source: http://courses.d2l.ai/berkeley-stat-157/units/adam.html

[2]https://www.tensorflow.org/versions/r1.15/api_docs/python/tf/train/AdamOptimizer



(a) **(b)**

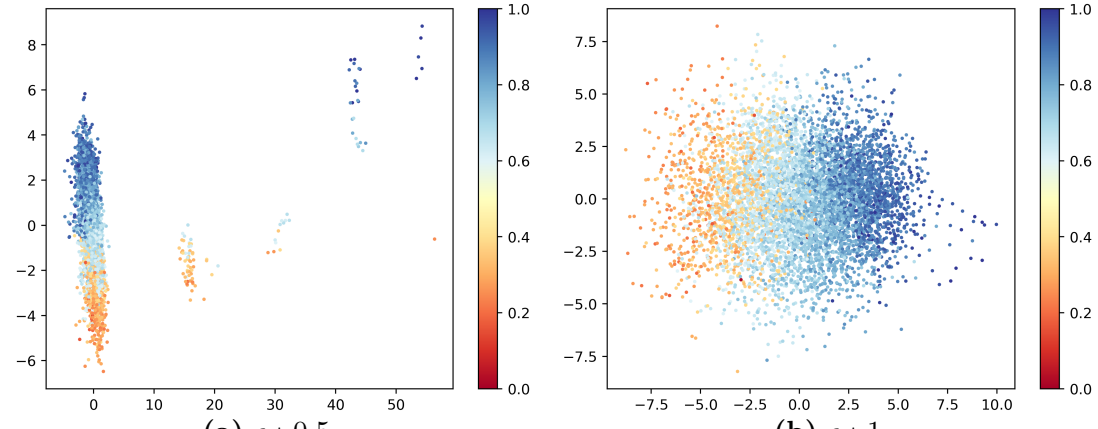
Figure F.2: The weight distributions for RMSProp (Tieleman & Hinton 2012) and the Momentum method. This Figure shows distributions over time for different values of γ. **(a)** shows the weights for the RMSProp running weighted average of squared gradients. Note that the weights sum up to 1. **(b)** depicts the weights for the velocity v of the momentum method[1].

the resulting weight update would become very large, and therefore we normalize the values to $\hat{v_1} = (g_1)^2$ and $\hat{s_1} = g_1$.

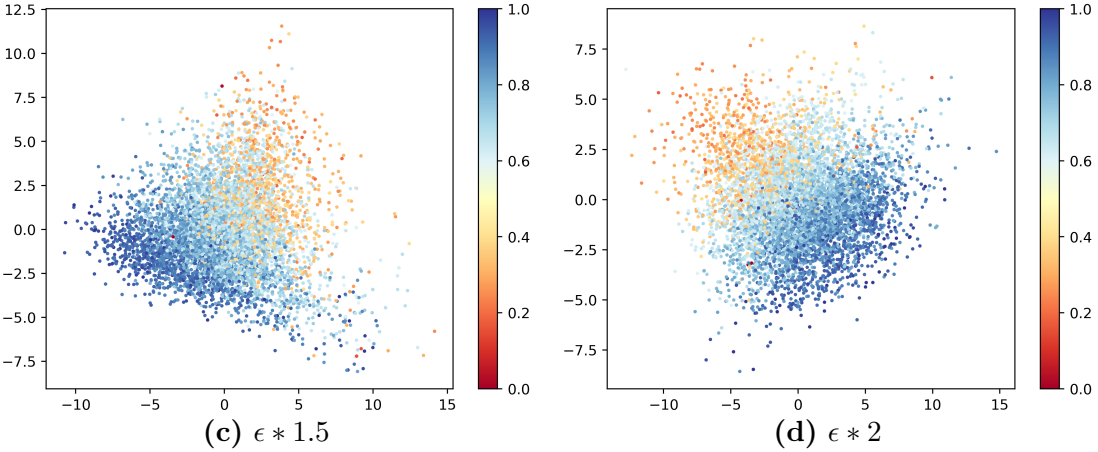
Appendix G

Latent Space Visualisations

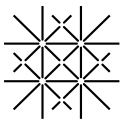
Table G.1: Visualisations of the latent space of CGVAE. The plots were generated under the same circumstances as in Figure 4.9, except for the sample variance of ϵ. The colour of the dots represents the value of the energy gap ϵ_{GAP} of the corresponding molecule. **(a)** For some values, PCA returned discontinuous dimensions. The resulting graphics do not represent the latent space as a smooth distribution.



(a) $\epsilon * 0.5$ **(b)** $\epsilon * 1$



(c) $\epsilon * 1.5$ **(d)** $\epsilon * 2$



Natural Science Faculty of the University of Basel
Department of Mathematics and Computer Science
Biomedical Data Analysis

Title of work:

Graph-based Molecule Design with Deep Latent Variable Models

Thesis type and date:

Master Thesis, 17 April, 2020

Examiner:

Prof. Dr Volker Roth

Supervision:

Vitali Nesterov and Mario Wieser

Student:

Name:	Elias Arnold
E-mail:	elias.arnold@stud.unibas.ch
Matr.-Nr.:	2014-930-770

Statement regarding plagiarism:

I hereby declare that this submission is my own work and that I have fully acknowledged the assistance received in completing this work and that it contains no material that has not been formally acknowledged. I have mentioned all source materials used and have cited these in accordance with recognised scientific rules.

Basel, 17 April, 2020: