# Evaluating Autoencoder Methods for Building a Molecule Graph Autoencoder

Amelia Woodward {ameliawd}
CS229 Project (Spring 2020)
General Machine Learning/Physical Sciences
Mentors: Keiran Thompson[1] and Todd Martinez[2]
The Martinez Group, Stanford Department of Chemistry and SLAC PULSE Institute

***Abstract*— We wish to build an autoencoder for molecules using molecule graphs (Molgraphs) as input. To begin developing this, we assess existing graphical autoencoding architectures for the purpose. By testing on organic molecules from the GDB13 database, we find that Kipf and Welling's VGAE model is the most promising model for development and can reconstruct some simple organic molecules before a large hyperparameter search. We will continue to explore VGAE as well as explore adjusting this encoder and decoder architecture going forward.**

## I. Background

**Motivation.** Automated molecular design is highly desired to accelerate chemical innovation across industries, from pharmaceutical discovery to materials design. The Martinez Group is currently developing an automated synthesis planner for this purpose which uses graphs of molecules. From hereon, let us call these molecule graphs 'Molgraphs'. For use in the synthesis planner and beyond, we want to build an autoencoder for Molgraphs (Figure 2).

This paper evaluates existing autoencoding techniques as applied to the task of autoencoding Molgraphs. Particularly, we implement existing graphical autoencoder deisgns and evaluate their graph decoder architectures. Since one can never separate the loss function from the network architecture, we also analyze reconstruction loss used. This work acts as the initial step to guide our development of an effective autoencoder for Molgraphs.

*Many thanks to the Martinez Group especially Keiran and Todd for their ongoing support and for access to GPUs. Thank you also to Stefan Seritan for his vital help with Docker container woes.

[1] *Keiran Thompson is a lead Research Associate in the Martinez Group whose impact on this project has been tremendous.*

[2] *Todd Martinez is the the head of the Martinez Group and a huge inspiration.*

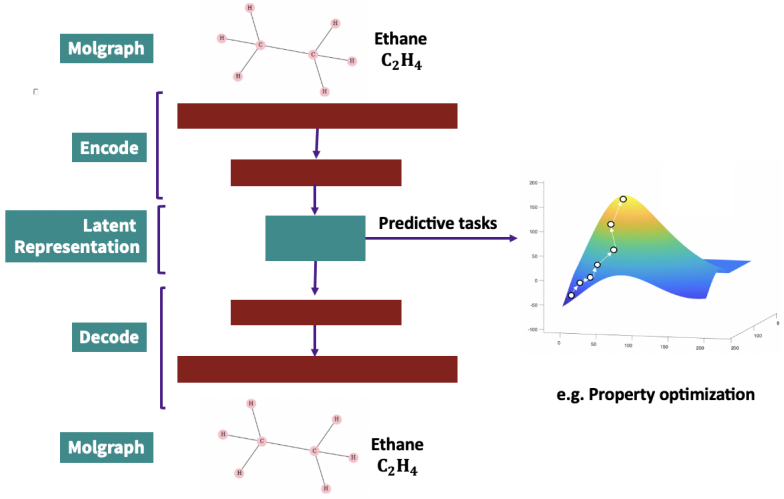


Fig. 1. Molgraph Autoenconder Design Overview [1]

**Autoencoders.** Broadly speaking, an autoencoder is a feedforward neural network which, when trained accurately, should return the same input as output. Autoencoders have an *encoder* and a *decoder*. The encoder maps the input to a *latent space*: a vectorized representation of its input. Depending on the encoding method, the latent space may be discrete or continuous. If the latent space is continuous, we call this a *variational autoencoder* (VAE). The *decoder* maps the latent space to an output, which we desire is the same as the input. In the context of encoding a Molgraph, this means that ideally it takes in a Molgraph and returns the same Molgraph (Figure 2).

Autoencoders, and particularly VAEs, are useful in that their latent space may be used for downstream machine learning tasks. In our context such tasks

include:
(1) *Link prediction* between reactions: Link prediction in the context of synthesis planning means finding possible new reaction pathways in a graphical web of known reactions. This requires having some vectorized representation of Molgraphs.

(2) *Optimization for desired chemical properties*. Imagine there is some property you wish to optimize in a drug's design (perhaps you wish for it to have strong binding affinity to a particular protein, or to be highly polar in some area). In this scenario, you would encode a known molecule (in Molgraph form) that has similar properties to that which we desire. You would then optimize the latent space for the particular property you desire, and then decode back to a Molgraph form, possibly uncovering a new molecule closer to the desired property.

**Autoencoding Molecules.** Given the wide-ranging uses of molecule autoencoders, they have been a topic of open and expanding research and development. Notably, the Aspuru-Guzik group present a chemical VAE which accepts SMILES strings: a text representation of molecules with specific rules (e.g. carbon dioxide = 'O=C=O' and ethane = 'CC')[2,3]. They encode SMILES strings to a latent space using a variational autoencoder, and decode back to SMILES strings. While this text autoencoder is a step forward, the SMILES format misses rich spatial information about molecules that may be captured by instead feeding autoencoders with graphical representations of molecules [4]. This is because molecule graphs explicitly take into account bonding and connectivity information. In general, nodes in a graph can represent atoms and edges can represent bonding information between atoms. Node and edge features can be adjusted to account for known chemical information, both at the atomic and bonding level. We expand on the particular design of Molgraphs in the Methods section.

**Autoencoding Graphs** While research into graphical deep learning has exponentialized, there remain challenges in autoencoding graphs, particularly in the decoding step. Specific to VAEs, it is challenging to take a discrete graphical structure, encode it into a continuous latent space, then accurately decode back to a discretized graphical structure.

Tackling this, Kipf and Welling pioneer a technique

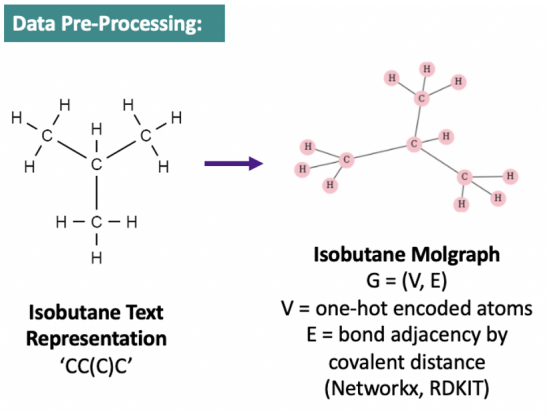


Fig. 2. Molgraph Preprocessing [7]

for variationally encoding graphs, called VGAE, initially demonstrating its use in large-graph link prediction tasks, and see improvements compared with a discretized graphical autoencoder (GAE) model[5]. They also see improvements on previously developed embedding techniques including spectral clustering and DeepWalk [5]. In addition, Simonovsky and Komadakis make suggestions for adjustments to this structure specific to small graphs, coining their method GraphVAE [6].

## II. Methods and Components

In summary, we implement GAE, VGAE and take inspiration from graphVAE and our knowledge of small molecule properties to design an adapted graphVAE. After preprocessing molecules to Molgraphs, We train the models on subdatasets taken from the publicly available GDB13 database. We first pre-process these into Molgraphs. We evaluate the model's decoding abilities to reconstruct the data by comparing reconstruction loss, L2-loss and average graph edit distance. We then draw conclusions about how each algorithm operates and where to take the direction of development of an effective autoencoder.

**Molgraphs.** We define Molgraphs to be a graph $G = (V, E)$ of a molecule with $V$ being atoms in the molecule and $E$ being edges representing bonds. For now there are no edge attributes, though these may (and will likely be) added when building a more fine-tuned model. For now, we wish to build the simplest graphical model of a molecule possible. Edges are marked to exist if the covalent distance between atoms is less than some purpose-defined threshold ($\alpha$). In the standard case this would be $\approx 1.5$ atomic radii, but

there may be cases when we would want this to be a higher threshold (ie. when intramolecular properties may be particularly important, so this is adjustable). In this paper we restrict to using Molgraphs with $\alpha = 1.5$. We define what we believe to be the simplest meaningful node attributes: a one-hot encoding of atoms where each index in the array corresponds to a particular atomic number (e.g. Hydrogen 'H'(atomic number=1) has a 1 in the 0th index, Carbon 'C' (atomic number=6) has a 1 in the 5th index).
We use Networkx and RDKit to construct the Molgraphs from a SMILES (text representation) input of the molecule.

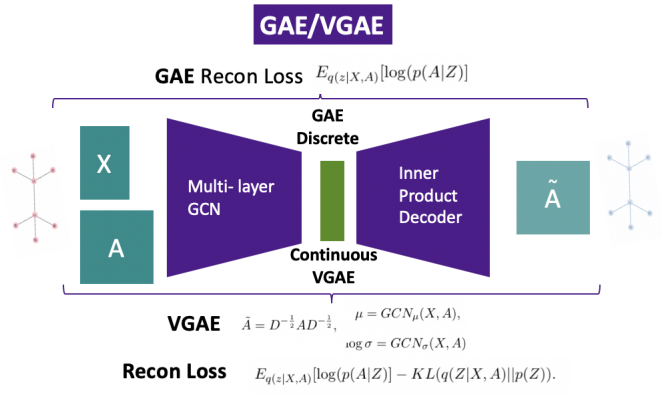


Fig. 3. GAE/VGAE Architecture Overview

**Autoencoding Methods**

**VGAE and GAE. [5]** First we implement the VGAE architecture presented by Kipf and Welling [5]. Given a graph $G = (V, E)$ with $N = |V|$ nodes, we can construct an adjacency matrix $A \in R^{n\times n}$ and diagonal degree matrix $D \in R^{n\times n}$. Node features are given in a $N \in R^{n\times d}$ matrix, where $d$ is the length of the node featurization vectors. Latent variables are summarized in matrix $Z \in R^{n\times f}$, where $f$ is the number of channels.

Kipf and Welling then define an *inference model* for $Z$ given $X$ and $A$ parameterized by a two-layer graphical convolutional network (GCN). Specifically,

$$q(Z|X, A) = \Pi_{i=1}^{N} q(Z_i|X, A),$$

$$q(Z_i|X, A) = N(z_i|\mu_i, (\sigma^2))$$

The two-layer GCN is

$$GCN(X, A) = \tilde{A}(\tilde{A}XW_0)W_1$$

with weight matrices to be updated: $W_1, W_2$. Here

$$\tilde{A} = D^{-\frac{1}{2}}AD^{-\frac{1}{2}},$$

$$\mu = GCN_\mu(X, A),$$

$$\log\sigma = GCN_\sigma(X, A).$$

Here $\text{ReLU}(.) = \max(0, .)$.

The *generative model* takes a simple inner product between latent variables:

$$p(A|Z) = \Pi_{i=1}^{N}\Pi_{j=1}^{N} p(A_{ij}|z_i, z_j),$$

$$p(A_{ij}|z_i, z_j) = (z_i^T z_j).$$

Here $\sigma(.)$ is the sigmoid function. To learn the model, they optimize the variational lower bound $\mathcal{L}$ with respect to weight matrices $W_1, W_2$:

$$E_{q(z|X,A)}[\log(p(A|Z)] - KL(q(Z|X, A)||p(Z)).$$

Here, $KL[p(.)||q(.)]]$ is the Kullback-Leibler divergence between $p(.)$ and $q(.)$ and

$$p(Z) = \Pi_{i=1}^{N} N(0, I)$$

is the Gaussian prior. The difference between GAE and VGAE is that for the GAE, instead of $\tilde{A}$, they calculate $\hat{A} = \sigma(ZZ^T)$ with $Z = GCN(X, A)$ and we do not include the KL loss in the learning step.

**Adapting graphVAE. [7]** The main difference between graphVAE and Kipf and Welling's GVAE is that graphVAE makes a restriction on the size of the graph (a maximimum of $k$ nodes, where $n$ is the original number of nodes and $n \leq k$) and output a probabilistic fully connected graph of the form $\tilde{G} = (\tilde{A}, \tilde{E}, \tilde{F})$. Here, $\tilde{A} \in R^{k\times k}$ where node probabilities exist along the diagonal and edge probabilities along the off-diagonal. $\tilde{E} \in R^{n\times n\times e}$ is the edge attributes and node attributes are in the matrix $\tilde{F} \in R^{n\times d}$. In this paper we will just adapt $\tilde{A}$ and $\tilde{F}$, since we are currently not assigning any edge attributes to Molgraphs, and so from hereon consider our adapted version.

In the decoding step, instead of using only an inner product decoder like Kipf and Welling, they use a multi-layer perceptron with a sigmoid activation function outputting $\tilde{A}$. We implement a fully connected layer with soft-max activation functions outputting $\tilde{F}$. We use a binary assignment matrix $X \in R^{k\times n}$ for

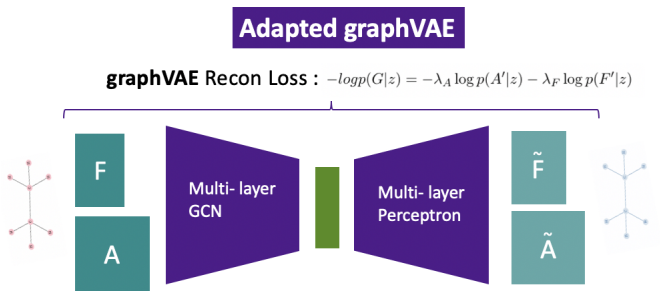


Fig. 4. graphVAE Architecture Overview

graph matching, using the Hungarian Algorithm [8], and then calculate a loss function:

$$-logp(G|z) = -\lambda_A \log p(A'|z) - \lambda_F \log p(F'|z)$$

Where $\lambda$'s are all set to 1, $A' = XAX^T$ and $\tilde{F}' = X^T\tilde{F}$.

$$\log(A'|z) = \frac{1}{k}\sum_i A'_{a,a} \log \tilde{A}_{a,a} + (1 - A'_{a,a}) \log 1 - \tilde{A}_{a,a})$$

$$+ \frac{1}{k(k-1)} \sum_{a \neq b} A'_{a,b} \log \tilde{A}_{a,b} + (1 - A'_{a,b}) \log 1 - \tilde{A}_{a,b}),$$

$$\log p(F|z) = \frac{1}{n}\sum_i \log F_i^T \tilde{F}',$$

**Dataset**

We use the open-source GDB13 Database, available from the University of Bern Department of Chemistry [7]. GDB13 is a multi-million molecule dataset containing only feasible organic molecules. The molecules in the database contain up to 13 atoms of C (Carbon), O (Oxygen), N (Nitrogen), S (Sulfur) and Cl (Chlorine). This means these molecules often have more than 13 atoms as H (hydrogen) is not included in the atom count to 13. These molecule are written in SMILES (text) format, so we pre-process them to Molgraphs and then to appropriate tensors for deep learning.

We extract subsets of molecules with particular size properties from this dataset in order to begin evaluating and understanding how well training with graph autoencoders on Molgraphs works. The details of these subsets are specifically explained in the Experiments section.

**Metrics**

We test on three main metrics: reconstruction loss (during training and testing), L2 loss between adjacency matrices and graph edit distance.

(1) The *reconstruction loss* is directly calculated during training, validation and testing from the loss functions defined in the Methods section. (Note that this means we should not compare reconstruction loss between models, but rather between traininc instances of the same model type).

(2) *L2-loss*: Since we have different reconstruction loss functions between models, we would like some consistent way of comparing the quality of decodings. Where $A$ is the input adjacency matrix and $\tilde{A}$ is the decoded adjacency matrix, we first apply a threshold to the output adjacency matrix $\tilde{A}$ such that if in row $i$ and column $j$, $\tilde{a_{ij}} < \frac{(mean(\tilde{A}) + median(\tilde{A}))}{2}$, then $\tilde{a_{ij}}$ is set to 0. We apply dynamic thresholding because the direct output of $\tilde{A}$ is continuously probabilistic and we need some way to discretize the adjacency matrix. Let us call this thresholded $\tilde{A}$ to be $\tilde{A}_{thresh}$. Then over a test set of size $n$, we calculate:

$$loss_{L2} = \frac{1}{n}\sum_{i=1}^{n} ||A_n - \tilde{A}_{thresh_n}||_2.$$

(3) *Average Graph Edit Distance (GED):* GED is a measure of similarity between graphs used in graph theory. Its formula for comparison between two graphs is:

$$GED(g_1, g_2) = \min_{e_1, e_2, \ldots, e_k \in \mathcal{P}(g_1, g_2)} \sum_{i=1}^{k} c(e_i)$$

where $\mathcal{P}(g_1, g_2)$ is the set of all edit paths transforming $g_1$ into $g_2$ and $c(e)$ calculates the cost of each graph edit operation. We then naively create an average graph edit distance in the test set of

$$GED_{average} = \frac{1}{n} GED_n.$$

We don't believe this is a perfect metric: some molecules, particularly vastly larger molecules are likely to have much larger GEDs than a smaller molecule that is coded incorrectly, so it is difficult to determine what is actually going on using this metric. However, it is a standard graph similarity metric nonetheless, so we at least attempt to include it. In addition we will examine how the models code particular molecules of interest (e.g. straight chained vs ringed structures) in attempting to understand what is going on.

| Model | Metric | ≤ 4Cs | ≤ 6Cs | ≤ 8Cs |
|---|---|---|---|---|
| GAE | $RL_{Test}$ | 1.20 | 1.13 | 1.13 |
| | L2 | 6.4 | 8.5 | 8.5 |
| | GED | 154 | 228 | 227 |
| VGAE | $RL_{Test}$ | 2.00 | 1.98 | 1.98 |
| | L2 | 6.7 | 8.5 | 8.6 |
| | GED | 154 | 228 | 227 |
| graphVAE (adapted) | $RL_{Test}$ | -2.75 | -2.95 | -2.94 |
| | L2 | 14.2 | 17.2 | 17.1 |
| | GED | 155 | 228 | 227 |

Fig. 5. Experiment 1 Results

| Model | Metric | 16 Channel 2 Conv | 32 Channel 2 Conv | 16 Channel 3 Conv | 32 Channel 3 Conv |
|---|---|---|---|---|---|
| GAE | $RL_{Test}$ | 1.04 | 1.05 | 1.09 | 1.12 |
| | L2 | 11.8 | 12.0 | 12.3 | 12.3 |
| | GED | 228 | 237 | 235 | 234 |
| VGAE | $RL_{Test}$ | 1.89 | 2.46 | 1.95 | 2.55 |
| | L2 | 11.9 | 11.9 | 12.2 | 12.4 |
| | GED | 239 | 234 | 233 | 238 |
| graphVAE (adapted) | $RL_{Test}$ | -2.95 | -2.9 | -2.95 | -2.93 |
| | L2 | 17.2 | 16.5 | 17.2 | 16.6 |
| | GED | 233 | 233 | 235 | 235 |

Fig. 6. Experiment 2 Results


Fig. 7. Training: Reconstruction per Epoch, Experiment 2; 16 Channel, 2 Conv Experiment, VGAE training had learning rate 1e-3, graphVAE training had learning rate 1e-4

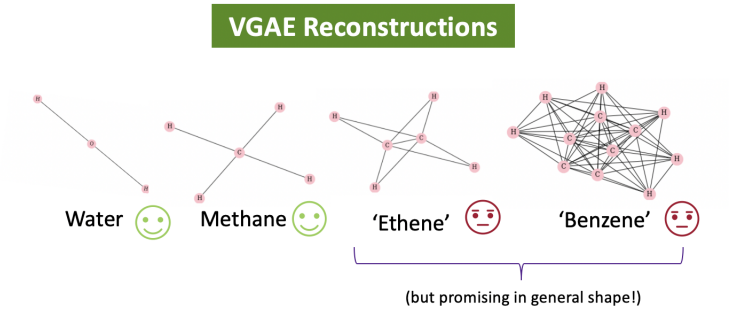


Fig. 8. VGAE Reconstructions of Simple Molecules

## III. Experiments

We conduct the following experiments to better understand how the models work and what is required for them to train well. Note that at this stage, we really don't expect the models to perform stellarly. Instead we are looking for the best step forward to effectively build a Molgraph autoencoder.

**Experiment 1.** We first attempt to understand how the models train on increasingly large molecules. We use three training/val/test subsets extracted from GDB13, each with (Train/Val/Test: 10K/1K/1K):

1. Up to 4 Carbon Molecules
2. Up to 6 Carbon Molecules
3. Up to 8 Carbon Molecules

We train on each of GAE, VGAE and adapted graphVAE. For the hyperparameters,
we set the number of channels in the latent space to 16. The number of epochs is 400. The learning rate is 0.01 for GAE/VGAE (which we have deemed appropriate from pre-trials) and the learning rate is 0.001 for the adapted graphVAE. We achieve the results seen in Fig 5.

**Experiment 2.** After analyzing small datasets in Experiment 1, we tried increasing the dataset size to 100K molecules. We restricted these experiments to 20 epochs. We then tried some simple hyperparameter changes which we thought might improve the model performance.

We tried using 16 and 32 channels in the latent space. We also tried keeping 2 graphical convolutional layers, and also adding a 3rd convolutional layer.

Plotting the reconstruction losses during training we saw that each model is converging, in both Experiments 1 and 2. GAE and VGAE take much longer to converge and need more than 20 epochs, and achieved much better results after the increase in epochs in Experiment 1. Therefore, going forward we will try using both the larger dataset and more epochs when training GAE/VGAE.

While GAE/VGAE had smooth reconstruction loss curves during training, graphVAE steeply decreased its reconstruction loss before plateauing after the second epoch. This same trend appeared in Experiment 2, even when we reduced the learning rate to 1e-4 (Figure 7). This means that we are likely in some way saturating the model and need to try extensive hyperparameter tuning (and potentially debugging) to improve the outcomes of this model.

Autoencoding a variety of standard organic molecules using the trained models, we found that the model that worked best was VGAE trained on 400 epochs for the up to 4 and 6 Carbons datasets (see Figure 8). (Actually GAE worked slightly better than VGAE with the same dataset and trainig parameters, but since we care about finding continuous representations of molecules, then we are really wishing to compare VGAE and the adapted graphVAE). We can effectively reconstruct some of the most simple molecules such as water and methane. We noticed that almost when the graph outputs are incorrect, there is a tendency for the outputs to be overly-connected graphs, and want to delve more into why this is the case and possible decoding or thresholding techniques that might help to alleviate this. This also explains why the average graph edit distance remains so high in Fig 5 and 6: if graphs are very highly connected like benzene in figure 8, then the graph edit distance of arbitrary test molecules is likely to remain high.

We did not see a significant improvement in outcomes by increasing the number of channels or adding a convolutional layer. However, we would like to try this experiment again on VGAE with more epochs of training to come to a more informed conclusion on the matter.

## IV. Outcomes and Next Steps

We come to the following main conclusions and outcomes towards the development of a molgraphVAE. VGAE appears to be more flexible and have significantly better Molgraph reconstruction outcomes (even when not fully converged) than adapted graphVAE. We will consider adding node and edge attribute features to this model to explore further development.

Positively, all models are learning during training in that their reconstruction loss is decreasing. To optimize, we will conduct comprehensive hyperparameter search to determine optimal learning rates, explore adding many more convolutional layers. Especially, we will uncover why graphVAE saturates after 1-2 epochs (may lead to significantly improved results).

All models tend to over-draw edges and require high thresholds in output adjacency matrix to recover 'molecule-like' outputs. We should explore why this occurs and what thresholding techniques make the most sense with these models.

We hypothesize that adding edge weights containing bond order information may drastically help the model learn. For instance bond order information contains tells the model whether a carbon should be connecting to 2,3 or 4 other atoms. Therefore, we will also try training with bond orders.

## References


[1] Wang, L., Titov, A., McGibbon, R., Liu, F., Pande, V., Martínez, T. (2014). Discovering chemistry with an ab initio nanoreactor. Nature Chemistry, 6(12), 1044-1048. doi: 10.1038/nchem.2099.

[2] Gomez-Bombarelli, R. et al. (2018). Automatic Chemical Design Using a Data-Driven Continuous Representation of Molecules. ACS Central Science, 4(2), pp.268-276.

[3] https://www.daylight.com/dayhtml/doc/theory/theory.smiles.html

[4] Woodward, A. (2019) Machine Learning on Chemical Reaction Networks: Summer Research Update. (My summer research last year)

[5] Kipf and Welling. (2016). Variational Graph Autoencoders. arXiv:1611.07308v1

[6] Simonovsky and Komadakis.(2018) graphVAE: Towards the Generation of Small Graphs Using Variational Autoencoders.arXiv:1802.03480v1

[7] Reymond Research Group. (2007) GDB Database. University of Bern http://gdb.unibe.ch/

[8] Peng, R. (2015) Hungarian Algorithm. https://www.cc.gatech.edu/ rpeng/ 8434S15 hungarianAlgorithm.pdf


## V. RESOURCES

**Deep Learning** We used Pytorch and the Pytorch Geometric library, as well as our own implementations of network architectures to process Molgraph data and implement GAE, VGAE and build adapted graphVAE from scratch.