Challenge 2: Molecule Generation | sub2

MICHAEL RAFFELSBERGER

ACM Reference Format:
Michael Raffelsberger. 2022. Challenge 2: Molecule Generation | sub2. 1, 1 (May 2022), 2 pages.

1 EXPLORATION

Similar to last time, I started with a brief data exploration. We have 1036643 smiles strings, most of which have a length between 30 and 70 characters. The maximum length in the training set is 101. We should be aware that all strings end with \n. Ignoring \n, the alphabet consists of 37 characters.

2 PREPARATION

I extended all smiles strings with an SOS token ("A") and an EOS token ("Z"), the later replacing the \n. Furthermore, I performed a train-validation split with 10% of the samples going into the validation set. The `Encoder` class can map from smiles strings to character indices and back. A `SmilesDataset` class is defined to supply PyTorch models with data later on.

3 MODEL

3.1 Related Work

I early on decided to use a VAE [6] since I wanted to get more familiar with this model anyway. The model is very similar to the text VAE in [1] and the molecule VAE in [4]. [1] encodes sentences to a continuous latent space using an RNN and then decodes, again using an RNN. Especially if the latent space does not contain useful information, the decoder is similar to a standard RNNLM. [4] encodes smiles strings with a fixed length using 1D convolutions and decodes with 3 GRU layers with around 500 cells each. My idea behind all this was to make sampling/generation more sophisticated with ancestral sampling in a VAE without the generated sequences collapsing to some mode.

3.2 Model Architecture

My VAE implementation uses an `EncoderLSTM` and a `DecoderLSTM`. Both share a character embedding layer. Before encoding, all smiles strings in a batch are padded to the same length. The encoder then actually consists of two LSTMs, one processing the sequences from the front and the other from the back. It is therefore similar to a bidirectional LSTM while circumventing the problem of different sequence lengths. Finally, a single linear layer maps from the concatenated outputs of the two LSTMs to the parameters μ_i and $\log(\sigma_i^2)$ of $q(z_i|x_i)$. Note that we assume an isotropic gaussian prior for z_i. The `DecoderLSTM` is similar to a NextCharLSTM. Its input at timestep t is the concatenation of the embedding of the last output $\hat{y}_i^{(t-1)}$ and the latent variables z_i. In the first timestep, we use the embedding of the SOS token. At every timestep, a linear layer outputs logits for all alphabet entries.

Author's address: Michael Raffelsberger, k11772903@students.jku.at.

4 TRAINING

When first training the above architecture, I experienced a bad posterior collapse. Essentially $q(z_i|x_i)$ was equal to the prior $p(z_i)$. This phenomenon is common when we have a powerful (autoregressive) decoder that is strong enough to ignore the latent space [9][7]. [1] suggests to use KL cost annealing, where we control the strength of the Kullback-Leibler term in the loss function with a parameter β like in [5]. In the beginning, we just focus on reconstruction (standard Autoencoder objective) and gradually increase the Kullback-Leibler regularization term, softly pushing the posterior $q(z_i|x_i)$ towards the prior. In the end, the posterior still collapsed, which is ok for our task though. However, there is no big difference to a standard NextCharLSTM now. I trained the model using teacher forcing on Kaggle. For optimization, I used Adam with batch size 256, an initial learning rate of 0.001 and learning rate decay (`StepLR`). The embeddings have dimension 16, as have the latent spaces. The `EncoderLSTM` consists of two LSTMs with 256 cells each, while the DecoderLSTM has two layers with 1024 cells[1] each.

5 EVALUATION AND FINAL FIT

Apart from comparing the errors on the training and the validation set, I regularly looked at reconstructions and the parameters of some posterior distributions to get a sense of how well the model worked. I also used the provided evaluation code to check what score I could expect every now and then.

After finding a sweet spot, I still trained the model for two more epochs on all 1036643 samples.

6 GENERATION & POSTPROCESSING

To generate new samples, I sampled from the isotropic gaussian prior distribution $p(z)$, decoded and converted the indices to smiles strings. At every timestep at decoding, a sample is drawn from a categorical distribution based on the logits, i.e. decoding is not deterministic.

While we were supposed to submit 10000 new smiles strings, I actually generated 100000. I just kept the valid (canonical), unique and novel (compared to training set) strings and again drew 10 random samples of size 10000 from the more than 80000 strings left. I evaluated the Fréchet ChemNet distance (FCD) of these 10 candidate submissions locally and submitted the best sample achieving a public leaderboard score of 0.905.

7 DISCUSSION

While I am glad to have learned a little more about VAEs, not just sticking to and optimizing a simple NextCharLSTM was certainly a mistake in retrospect. The VAE was just additional work, making everything slower and more complicated without any real benefit for the purpose of generation. I am quite confident that I could easily lower the FCD further, if I focused on the simpler LSTM model.

[1]less would also be ok

8 ACKNOWLEDGMENTS

Apart from the already mentioned references, [8], [3] and [2] were really helpful for this challenge.

REFERENCES

[1] Samuel R Bowman, Luke Vilnis, Oriol Vinyals, Andrew M Dai, Rafal Jozefowicz, and Samy Bengio. 2015. Generating sentences from a continuous space. *arXiv preprint arXiv:1511.06349* (2015).

[2] Marco Cerliani. 2020. *Time Series generation with VAE LSTM.* Retrieved May 11, 2022 from https://towardsdatascience.com/time-series-generation-with-vae-lstm-5a6426365a1c

[3] William Falcon. 2020. *Variational Autoencoder Demystified With PyTorch Implementation.* Retrieved May 11, 2022 from https://towardsdatascience.com/variational-autoencoder-demystified-with-pytorch-implementation-3a06bee395ed

[4] Rafael Gómez-Bombarelli, Jennifer N Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D Hirzel, Ryan P Adams, and Alán Aspuru-Guzik. 2018. Automatic chemical design using a data-driven continuous representation of molecules. *ACS central science* 4, 2 (2018), 268–276.

[5] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. 2016. beta-vae: Learning basic visual concepts with a constrained variational framework. (2016).

[6] Diederik P Kingma and Max Welling. 2013. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114* (2013).

[7] Ali Razavi, Aäron van den Oord, Ben Poole, and Oriol Vinyals. 2019. Preventing posterior collapse with delta-vaes. *arXiv preprint arXiv:1901.03416* (2019).

[8] Sean Robertson. 2021. *NLP from scratch: Translation with a sequence to sequence network and attention.* Retrieved May 11, 2022 from https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html

[9] Codie Marie Wild. 2018. *With Great Power Comes Poor Latent Codes: Representation Learning in VAEs (Pt. 2).* Retrieved May 11, 2022 from https://towardsdatascience.com/with-great-power-comes-poor-latent-codes-representation-learning-in-vaes-pt-2-57403690e92b