Notes: "Automatic Chemical Design Using a Data-Driven Continuous Representation of Molecules"

Sun

July 20, 2020

1 Summary:

1. You can train a variational autoencoder to be used on molecules.
2. While in the latent-space, you can perform gradient-based optimization upon desired properties, leading to generation of molecules close in the latent space, but with more desirable attributes.

1.1 High level:

This paper is mostly one of applications, but I still think it is interesting. The authors train a boilerplate text VAE on molecule SMILES strings (a string representation of a molecule), and find that the VAE actually works.

In addition, they also develop a novel (?) way of maximizing a given property in molecule-space by gradient-based optimization in latent-space, which I believe is useful (and very cool), even outside of chemistry, which I believe is a good topic of discussion.

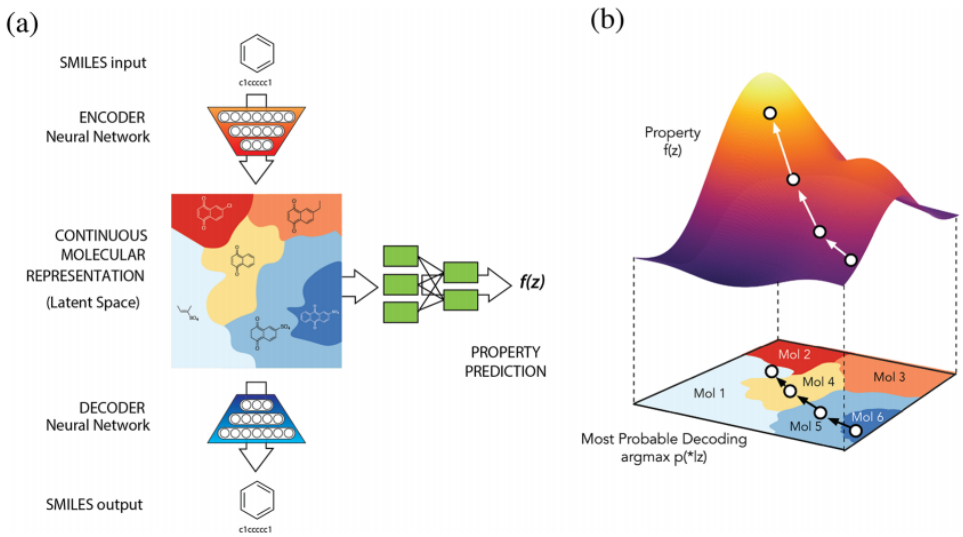


Figure 1. (a) A diagram of the autoencoder used for molecular design, including the joint property prediction model. Starting from a discrete molecular representation, such as a SMILES string, the encoder network converts each molecule into a vector in the latent space, which is effectively a continuous molecular representation. Given a point in the latent space, the decoder network produces a corresponding SMILES string. A mutilayer perceptron network estimates the value of target properties associated with each molecule. (b) Gradient-based optimization in continuous latent space. After training a surrogate model $f(z)$ to predict the properties of molecules based on their latent representation z, we can optimize $f(z)$ with respect to z to find new latent representations expected to have high values of desired properties. These new latent representations can then be decoded into SMILES strings, at which point their properties can be tested empirically.

1.2 Low level:

The high-level summary does contain the majority of the paper, but the implementation techniques are a bit interesting.

1.2.1 Text VAE:

Praise be, SMILES strings are traditionally quite small (20ish characters), so text VAEs actually work here. They use a seemingly generic text VAE that uses two RNNs, one as an encoder, one as a decoder, with some hacks to get it to work:

1. The KL term is slowly increased (to a max of 1) to prevent immediate posterior collapse.
2. Words are randomly replaced by the `[UNK]` token as a form of 'anti-teacher-forcing' which incentives the model to learn a better (and use) z.

Does it work on text, even? Sorta. Some generations are good, but none are over a sentence.

1.2.2 Property prediction:

In addition to the loss terms of the KLD and reconstruction error, they add a property prediction loss term: a simple ANN on the latent space should be able to recover given physical attributes about the molecule. They find this regularizes the latent space and groups more 'like' molecules together better.

1.2.3 Latent-space gradient-based optimization for desired properties:

The authors then approximate the ANN using a Gaussian process model, with the main motivation of creating a smooth landscape to optimize on. From there, they can simply use gradient-based optimization on a given latent code to increase a given wanted property, while staying close to the origin (the original z).

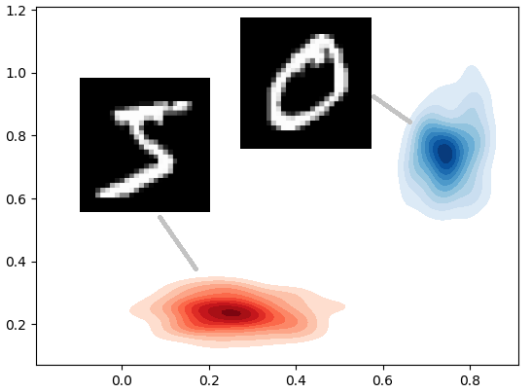
1.3 What can property optimization be used for?

It might not seem *that* useful at first, but I personally think it really could be.

Given a decoder with a smooth latent space and a discriminative model to predict properties (age, gender, chance of repaying a loan, etc..), one can optimize the output of the decoder to meet any of these. Below is an example on CelebA, optimizing from 'male' to 'female' with an l2 penalty on the perturbation:

Figure 1: From left to right: Original, recreation, optimized recreation.

But it's not quite that simple: it's not obvious what to even use as the discriminator. In order to prevent optimization from going out-of-distribution, you should probably add a loss term that is the NLL of the point, but then latent space of the decoder must be very well regularized and smooth, or else you could have disconnected modalities (and, if you're optimizing from one to another, could potentially be impossible). See below for a visual example.



If the loss term for the probability was too strong, it could be impossible to get between these two modalities. Yet, if it is too weak, it can veer off out of distribution and "fool" the discriminator. Also, you probably want a term that reduces the magnitude of the perturbation to prevent the modified latent code from being modified too much.

There's no obvious answer for this, as far as I can tell.

What do you all think?