PML project: Variational Autoencoders for protein sequences

Supervisor: Wouter Boomsma (wb@di.ku.dk)

Introduction

In this project, you will develop a deep latent variable model (aka a VAE) for amino acid preferences in protein sequences.

A protein is a chain of amino acids. The length of this chain can be anywhere between tens up to thousands of amino acids. There are 20 naturally occurring amino acids, which are typically represented in the standard Latin alphabet. For instance, the amino acid Alanine has the letter "A", Cysteine has the letter "C" and so forth. A protein sequence can therefore be represented as a sequence of letters, e.g.:

```
HPETLVKVK
```

Due to our shared evolutionary history, different species will have many of the same proteins, but over time, the amino acids sequences tend to gradually change due to random mutations. When considering similar proteins across different species, we typically call this a *protein family.* In addition to mutations, evolution will sometimes also introduce insertions and deletions, which means that proteins in a protein family will not necessarily have exactly the same length, e.g.:

Protein 1: `HPETLVKK`

Protein 2: `HTKVKVK`

For this reason, protein sequences are often preprocessed using a procedure called *multiple sequence alignment*, which identifies similar positions in the different protein sequences and places them in the same columns by inserting gaps ('-' or '.') various places in the sequences. For the sequences above, the alignment could look like this:

Protein 1: `HPETLVK-K`

Protein 2: `H--TKVKVK`

Our task in this project will be to create a probabilistic model that describes the preferences for the 20 different amino acids for an alignment of proteins from a specific protein family. In other words, given a multiple sequence alignment with n sequences and m columns, we wish to create a model that can give us the joint distribution $P(\bar{x})$, where $\bar{x} = x_1, x_2, x_3 \dots x_m$ are the positions in the alignment. Why is this an interesting model? Given such a model, we can investigate which mutations we might expect to see at a given position. This is relevant if you wish to design new proteins (e.g. drugs or industrial enzymes), or if you wish to understand why some naturally occurring variants of a protein are more or less potent than others (such as we are currently seeing with the omicron variant of COVID19). The model can also tell us which mutations we expect to be very unlikely. This is relevant for understanding human genetic diseases, where an occurrence of an unlikely amino acid in a patient might mean that a protein does not function as it should.

An example of a very simple model is to consider each of the columns in an alignment independently. We can then just model each of the columns with a categorical distribution, where we estimate the frequencies of the 20 amino acids simply by counting how often they occur in the multiple sequence alignment in that column. This is a poor model, since we know that amino acids interact with one another, so a mutation at one position in a protein will have an impact on the preferences for amino acids at other positions. Instead, we will therefore here consider a model where we describe the covariance between positions using a latent variable model. The idea is that that the positions in our alignment will now be independent *conditionally* on the latent variable z. For a given value of z, we can therefore write $p(\bar{x}|z) = p(x_1|z)p(x_2|z)\dots p(x_m|z)$.

A few years ago, such a model was published in Nature Methods, under the name DeepSequence: `https://www.nature.com/articles/s41592-018-0138-4`. We will try to recreate this model, and investigate potential extensions.

Tasks

The project is split into two tasks. In the first, you will implement a VAE to reproduce some of the results from the paper above. The second task is free-form: it consist of a number of suggestions for extensions of the basic VAE, from which you can choose if time permits - but you are also free to choose other extensions if you like.

Task 1: Implement DeepSequence VAE model

We will be working with the following dataset:

```
https://sid.erda.dk/share_redirect/a5PTfl88w0/BLAT_ECOLX_1_b0.5_labeled.fasta
```

This is an alignment of proteins from the Beta-lactamase and is the same file as in the original paper, except that I have added labels in the title field of each protein, which will be useful for plotting purposes later. The file is in FASTA format, which means that protein entries appear in blocks starting with a single title line, which begins with a > character, followed by several lines containing the amino acid sequence itself. Start by familiarizing yourself with the data file.

To avoid wasting too much time on data processing, I have created a simple notebook with code to parse the input data, and load it into a pytorch dataloader.

```
https://colab.research.google.com/github/wouterboomsma/pml_vae_project/blob/main/protein_
vae_data_processing.ipynb
```

1. Implement a standard VAE Implement a VAE for this data set. The paper describes several enhancements over a simple VAE - but please ignore these for now, and focus initially on getting a basic VAE working, using a Gaussian prior, a Gaussian distribution for z and a categorical distribution for for $p(x|z)$. I strongly recommend using the distribution objects from `torch.distributions` (e.g. `distributions.normal.Normal` and `distributions.categorical.Categorical`), which also means that you can use the built-in `distributions.kl.kl_divergence`.

2. Visualize the z-space Try to reproduce the right-hand plot in Figure 4 in the DeepSequence paper. The different colors denote which phylum (level of species taxonomy) the proteins are from. To create this plot, you should train a VAE with a latent space of dimension 2, such that it can be plotted directly. The dataloader in the provided jupyter notebook contains the phyla labels that you need to do this.

3. Quantitative assessment In addition to the qualitative assessment provided by the visualization, we would also like to have a more quantitative assessment of how well the model is doing. As I wrote in the introduction, we would expect the model to be able to determine whether mutations are likely to occur or not. Imagine we have an original, naturally occuring protein sequence[1] x_{wt}, to which we make a single mutation, leading to a x_{mt}, such that there is a single amino acid difference at one position in the sequence. We can score such a mutation by considering the probability of the new sequence, relative to the original sequence - or more precisely, the log ratio of their respective marginal likelihoods $\ln(p(x_{\mathrm{mt}})/p(x_{\mathrm{wt}})$. As you might remember, we cannot easily calculate these quanties in a VAE - but the ELBO is an approximation for $p(x)$, so you can use that instead. This is explained in detail in the paper.

We can now compare this mutation score to experimental measurements of variants of the protein, to see how they correlate. This correlation is what is reported in Figure 3.a of the paper (the left-most dots are for the Beta-lactamase dataset which we are using here). Similar to the DeepSequence paper, we will be using the data from the 2015 study by Ranganathan2015

```
https://sid.erda.dk/share_redirect/a5PTfl88w0/BLAT_ECOLX_Ranganathan2015.csv
```

Note that the jupyter notebook above also contains code to parse this file.

4. Sequence reweighting According to the paper, one of the biggest performance improvements of the model is obtained by reweighting input sequences according by the density in input space. The weights are obtained by calculating, for each sequence, how many other sequences exist that have a similarity above some threshold (typically set to be 0.8 in normalized Hamming distance) - see the section "Sequence weights" in the paper for details. The weight calculation itself is included in the jupyter notebook above (remember to turn on the GPU in colab, otherwise this calculation is quite slow). Implement a reweighing strategy during training, and assess whether it has an impact on the performance of the model. Hint: you can use `torch.utils.data.sampler.WeightedRandomSampler` in combination combination with your dataloader.

Task 2: Model extensions and/or more elaborate analysis

In this second part, you can either extend the model to obtain better performance, or make a more rigorous analysis of the original model. Below are a number of suggestions. Feel free to **pick one or more** topics from this list, or come up with your own extensions.

- **Compare to baseline model** I mentioned above that one could have constructed a simpler model, which assumed complete independence between positions, which could be estimated through simple counting. Implement this model, and investigate how much worse its correlation to experiment is compared to your VAE model.

[1]the naturally occurring variant is called the *wild type*

- **Importance Weighted Autoencoder** As described during the lecture on VAEs, you can get a tigher bound on the evidence by using an Importance Weighted Autoencoder. Implement this, and discuss whether it improves performance - both visually in terms of the separation of phyla, and with respect to the correlation to experiment. How does it depend on the number of samples?

- **Using a different prior** VAEs typically use a standard Normal distribution as a prior, $p(z)$. Figure 4 in the paper suggests that we might obtain better performance by choosing a more complex prior. Recent literature suggests that a Dirichlet prior might be a fruitful approach to obtain much clearer separation in latent space. Note that for this particular choice of prior, we actually restrict z to be between zero and one, which means you also have to change the output distribution of your encoder. Details on how to implement this are found in this reference: https://arxiv.org/abs/1901.02739.

- **Double variational VAE** In a standard VAE, the weights of the decoder are estimated as point estimates, using maximum likelihood. In the Nature Methods paper, the authors suggest modelling the weights as probability distributions as well, which leads to a model where we have variational distributions over both the latent variable z, and the weights of the decoder. Implement this strategy, and investigate whether it has an impact on the performance.