FACULTY OF SCIENCE
UNIVERSITY OF COPENHAGEN

Deep Unsupervised Representation Learning on Protein Sequences

Master's Thesis

Victor Nordam Suadicani* **[swl460@alumni.ku.dk]**

Emil Petersen* **[empe@di.ku.dk]**

2020/06/04

Supervisor: Wouter Krogh Boomsma

* Equal contribution

Abstract

Proteins are complex macro-molecules used extensively in living organisms and in the biological industries. Their ubiquity necessitates their exploitation – the synthesis of useful proteins can lead to advances in medicine and other organic chemical products. However, this synthesis requires a thorough understanding, leading to the field of protein engineering.

Representation learning is an interesting avenue for protein engineering, as good representations facilitate protein analysis and may aid an exploration of the space of proteins. We perform experiments that evaluate and compare three recent protein deep learning architectures: (1) UniRep, a recurrent model, (2) the variational autoencoder, a latent space model, and (3) WaveNet, a convolutional model.

We show that the variational autoencoder achieves superior performance on mutation effect prediction for local protein families and argue that its representations display desirable properties for exploration, not exerted by the other models. Representations produced by UniRep outperform WaveNet on general protein analysis tasks, while WaveNet shows promise on mutation effect prediction.

Acknowledgements

We would like to thank our supervisor on this thesis, Wouter Krogh Boomsma. This thesis (like most, we imagine) was not without its ups and downs, but Wouter always managed to steer us in the right direction, both with regards to theoretical considerations and our own stress levels. Wouter also suggested that we should apply for the Novo Scholarship 2020, which we would not have received otherwise.

We thank Novo Nordisk and Novozymes, who supported this thesis with 84.000 DKK through the Novo Scholarship Programme, a scholarship given to exceptional students to allow them to fully dedicate their time to their theses. In addition to the financial aid, we were put in contact with mentors from both companies: Daniele Granata and Alexander Junge from Novo Nordisk and Dennis Pultz from Novozymes, all of whom we thank for their guidance and advice.

Building and reproducing models from papers can be a nontrivial task in itself. We thank Aaron Kollasch and Deborah S. Marks at the Debora Marks Lab, Harvard Medical School, for kindly answering our questions in times of need and for making their data and work open source.

Finally, we would like to thank the many people behind The Universal Protein Resource (UniProt) consortium. Without their dedication and initiative to maintain an open and accessible protein resource database, any of the experiments in this thesis would have been unlikely to exist.

Contents

1
Introduction

1.1 Motivation

Proteins are complex macro-molecules used by all living organisms in their biological processes and as such, proteins can be considered one of the building blocks of life. They are crucial to cell functions and so has widespread application to biological industries such as agriculture, food, medicine, cosmetics, as well as countless chemical products. Besides industrialism, the increased understanding of proteins and their complex behavior informs our perpetual development of better, more useful tools against human suffering and sickness. Protein function and properties lie at the heart of almost any biological process within us, and is thus a key to not just saving lives, but also their improvement. The synthesis of useful proteins is critical for these applications. However, finding useful proteins is a daunting task, especially when such proteins deviate significantly from those found in nature. The abundance of proteins makes laboratory experiments fall short at the scale at which proteins are discovered and stored. Other approaches are needed to adress this, and bioinformatics is the use of computational tools to aid the search and investigation of proteins.

Due to the extreme complexity of proteins, their analysis and optimization becomes difficult. In recent years, machine learning methods have been applied to a wide variety of problems with success in many fields. The successful application of machine learning on proteins could allow the discovery of new, useful proteins. Not only that, but an explosion in the amount of available raw protein data has happened in recent years, reaching hundreds of millions of protein sequences [1]. The disparity in raw sequences versus sequences with associated properties necessitates the use of learning methods, which solely analyze the raw sequence and does not rely on manual, human annotation. Such unsupervised methods would have to be tractable on a large dataset, and should ideally facilitate the exploration of new, useful proteins. For these requirements, representation learning seems fitting.

In the ideal case, representation learning enables the transformation from arbitrary protein sequences into compact representations, and vice versa, living in a smooth space. Such a learned representation should contain the essential features which describe a protein, and the discovery of new proteins should be feasible by exploring the smooth space, rather than exploring the space of proteins directly. Due to the complexity of proteins, such transformation onto proteins involves some degree of uncertainty. A probabilistic representation learning model would mitigate this by explicitly incorporating such uncertainty into the transformation by producing a distribution over representations. This distribution would inform the application of the representations and be of great use in settings where the degree of confidence in predictions are of consequence. A distribution over representations also affords sampling of representations, and so proteins can be generated from such models.

Unfortunately, such ideal representations are currently beyond the reach of modern representation learning methods. There are several issues that hinder the production of the above ideal representations. It is not obvious how a model is supposed to learn how to produce “good” representations that live in a smooth space (if a smooth space of protein representations is even sensible). Proteins are inherently sequential in nature – how should a model reduce a protein sequence of arbitrary length into a fixed-size compact representation? Similarly, how should a model decode a representation back into a protein sequence, and how is it supposed to handle the inherent uncertainty that this involves? Considering the size of the entire (global) protein space, it may also be far too optimistic to represent all proteins in a compact representation space, and smaller, local regions of the protein space (protein families, for instance) may have to suffice, leading to smaller, less generally useful representation models.

The above issues raises several interesting fundamental questions, some of which we would like to explore in this thesis:

- To what extent do recent advances in protein deep learning reach such ideal representations as outlined above?
- What are the advantages and disadvantages, if such exist, of global versus local representations?
- How does the choice of the representation affect the performance on select downstream tasks?

1.2 Outline and delimitations

This thesis is focused on applying representation learning to protein sequences. We examine the performance of different representations and their implications for protein informatics by developing three select models capable of working on protein sequences. Each model approaches the task in its own particular way and is chosen based on shown merits. All the models are trained to reconstruct their input, an unsupervised task. A key goal is, if possible, to systematically discern the design choices of the model architectures that yield superior performance on a chosen set of protein analysis tasks.

Suitable background theory is provided before the models-, experiments- and discussion chapters. In chapter 2, the task of learning representations is introduced, providing a foundation for the notion of a "good" representation. Chapter 3 on probabilistic modelling gives much of the background needed for the probabilistic model, the variational autoencoder, relying on approximation and inference. In a similar fashion, chapter 4 introduces sequence learning, which is the basis of the two remaining models. After these chapters, the reader should be ready to understand the concepts used to apply the model architectures. These are the focus of chapter 5. We start by presenting a single strictly local model, before presenting two different approaches to global models.

In section 5.1 we present UniRep [2], a recurrent model. UniRep utilizes recurrent neural networks to process protein sequences, which allows it to function on the global protein space and has previously been shown to work well on this space [2].

In section 5.2 we present a Variational Autoencoder (VAE), which uses protein sequences aligned to a fixed-length to allow the use of simple linear neural networks, and to take advantage of regularities that can be related to absolute positions within such an alignment. Since the VAE only works with aligned protein families, it is a strictly local protein model – as such, the VAE will serve as the main local model in our comparison of local and global models. The VAE constructs a representation by encoding a protein sequence into a fixed-size, latent, probabilistic vector, which through a Bayesian neural network can be decoded back into a protein sequence.

Finally, in section 5.3, we present a convolutional model inspired by the WaveNet [3] architecture – for lack of a better name, we will simply refer to this model as "WaveNet" in this thesis. WaveNet uses layers of causal convolutions, with progressively larger dilation to analyze the protein sequence. It can, in principle, process proteins of arbitrary length, allowing it to train on the global protein space, but as far as we are aware, this has not been attempted yet. It has however shown promise when trained on local protein families [4]. The model does not produce an explicit representation, but a representation may be obtained similarly to UniRep.

These models represent different approaches to the protein representation learning problem. The VAE is a model restricted to aligned protein families (local), but this brings with it the advantage of lower complexity. The other models work with unaligned sequences (global) and are thus more general, but this also requires a more complex model, capable of finding patterns among variable-length sequences.

To summarize, our contribution in this thesis initially includes an explanation and implementation of three conceptually different protein representations models. We subsequently provide comparable benchmark performances on benchmark protein analysis tasks suitable for evaluating the quality

of the produced protein representations. In addition, we perform a detailed examination of the relationship between global and local protein models, providing a systematic foundation for representation selection. UniRep has previously been evaluated on general protein tasks with favor. We provide an extended investigation into its representations and their merits on local protein space. Likewise, WaveNet has been applied to local mutation effect prediction; we reproduce these experiments, and additionally provide protein representations directly extracted from this model. We measure and compare the performances of these representations with UniRep on general protein analysis tasks.

1.3 Introduction to Proteins

Proteins are large molecules (macromolecules) that are used by organisms in virtually every part of their function, such as enzymes or hormones. However, proteins are also widely used in the biological industries for a variety of purposes, including medical purposes, food processing and biological detergents.

1.3.1 Structure

Proteins consists of long chains of *amino acids*. Amino acids are a class of organic compounds, of which about 20 different acids are used in proteins. Proteins commonly consists of hundreds of amino acids (sometimes referred to as *residues*), connected together in a sequence, making proteins molecules of potentially thousands of atoms. In nature, proteins are constructed by organisms by reading their genetic code, in the form of DNA. The genetic code (genes) of the DNA describes which amino acids to put together in a sequence, in order to construct the desired protein.

A protein is in theory defined entirely by its amino acid sequence. However, when proteins are assembled, the chain of amino acids folds up into a complex 3-dimensional structure. This structure is what largely determines the function and properties of the protein. When speaking of protein structure, we concern ourselves with three main levels of structure:

Primary Structure: Purely the 1-dimensional amino acid sequence itself, with no additional information about the 3-dimensional structure of the protein.

Secondary Structure: 3-dimensional structure of local parts of the protein. When folding, local parts of the amino acid sequence arranges itself into certain structures, most commonly α*-helices* and β*-sheets*. Secondary structure is knowing at each position of the protein, how the amino acid participates in local structure at that position.

Tertiary Structure: The full specification of the folded protein, that is, the 3-dimensional coordinates of all the amino acids.

See figure 1.1 for an illustration of the three levels of protein structure.

A central task in protein machine learning is prediction of secondary or tertiary structure, given the primary structure. Many models have some success when predicting secondary structure, as it is not a very difficult task. Comparatively, predicting tertiary structure from primary structure alone is an incredibly difficult task, as the folding of a protein is such a complicated process.

In this work, we concern ourselves with useful representations derived from the primary structure alone. In doing this, we consider the protein's primary structure as merely a sequence of the common amino acids. Since there are not that many, each amino acid can be assigned a letter of the alphabet, resulting in a protein being represented as merely a string of letters (i.e., "LDRSIEKFMK..."). This actually makes representation learning on proteins rather similar to representation learning on natural language, at least in its technical specification (to produce a representation given a list of tokens).

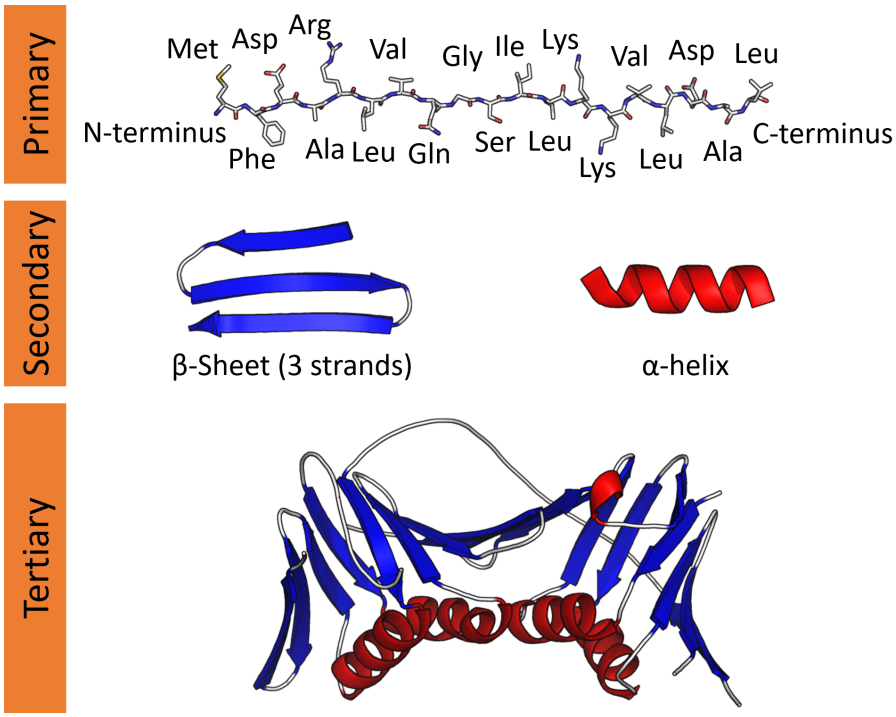


Figure 1.1: Depiction of the levels of protein structure – primary, secondary and tertiary. Note that the primary structure only consists of the sequence of amino acids, not any of the atoms' coordinates. Figure provided by Shafee [5].

1.3.2 Protein Families

In addition to the structure of a single protein, proteins also partially share more abstract structure between them, due to their connected origins. This causes proteins to be clustered into what we call *protein families*.

As organisms reproduce, their genetic material experiences random mutations – these mutations in turn cause mutations in the organisms' proteins, which may change the proteins' function. Through the process of natural selection, the organisms which experience beneficial protein mutations continue reproducing, while organisms with detrimental protein mutations die out.

Hence, just as different organisms are related in species or families, so are proteins related. This causes proteins to be structured in a phylogenetic tree, just as organisms are. We may use this information to our advantage to construct what is known as *sequence alignments*, which we will discuss in section 4.3.

1.3.3 Methodological Relation to Natural Language Processing

In natural language processing (NLP), most often data is in the form of written text, usually represented as a sequence of words. We can view our protein data as a similar sequence, except that the words of proteins are amino acids, and while natural languages has hundreds of thousands of words, proteins have just 20 amino acids. Proteins can be thousands of amino acids long, and similarly, documents may be thousands (if not tens or hundreds of thousands) of words long. It is clear that, despite some differences, representation learning in NLP and protein machine learning are related.

With this relation in mind, we look to NLP for inspiration for methods in machine learning to apply to proteins. As it turns out, many of the ideas in NLP can also be used in protein machine learning. In NLP there is the idea of a "language model", that is, a model which is able to predict (and hence produce) natural language following a given piece of text. This requires the model to learn the fundamentals of natural language, its interconnected dependencies and the probabilities of certain words following

other words. Often a model will produce a representation of the text seen so far in order to inform the language generating process.

With proteins, a similar model would be very useful. Such a model could be trained to predict the following amino acids, given a part of a protein. As part of this training, it would have to learn a good representation of the protein in order to inform the generation of the amino acids. The hope is that such a model would have learned the fundamentals of the "language of proteins".

1.4 Related Work

Machine learning has been an active research topic for some time. Protein machine learning has been a part of this development and many different approaches have been explored – however, the results are not as dramatic as in some other machine learning fields, such as computer vision. Proteins seem to present a more substantial challenge for learning models, which is not too surprising given their complexity.

As computer scientists, we have had to familiarize ourselves with this research field. We have primarily studied fairly recent papers on novel model architectures that show promise on the protein representation learning problem.

In 2018, Riesselman, Shin, Kollasch et al. presented a Variational Autoencoder (VAE) architecture capable of encoding and decoding protein sequences [6]. The model was shown to perform well on predicting mutation effects, but also requires an aligned protein family. Our exploration of VAE's on proteins is largely inspired by their efforts and we use their aligned protein family datasets for training our models.

In 2019, Alley et al. presented the UniRep ("**Uni**fied **Rep**resentation") model, which allows processing of variable-length proteins [2]. This enables training on the global protein space, after which a representation of a protein can be obtained from the model. The UniRep representation showed promising results on standard protein prediction metrics. However, the model does not include a decoder so the representation cannot be transformed back into a protein.

In 2019, Riesselman, Shin, Kollasch et al. proposed a new convolutional model [4]. It was inspired by the WaveNet architecture, a kind of causal convolutional neural network originally used for audio processing [3]. The model achieved similar performance to the VAE on local protein families.

2
Representation Learning

In machine learning, algorithms and learning methods are applied to datasets. Producing useful representations is a crucial step in almost any application that seeks to utilize such datasets quantitatively. Depending on the representation, patterns and explaining regularities in the dataset might be more or less obscured to the model [7]. For this reason, finding a representation that makes such regularities more prominent is often a nontrivial learning task itself. Usually, this involves various preprocessing and cleaning steps of the raw data samples, as well as engineering features that capture the crucial information well. In protein engineering, such cleaning steps may include multiple sequence alignments of protein families and removal of certain noisy/insignificant parts of the protein sequence, while feature engineering could be the step from low-level representation of protein sequences to more abstract features containing fundamental protein properties such as stability and secondary structure. Manually engineering features does not scale well and throttles the application of machine learning models. For this reason, automatically learning suitable representations is a task of major importance in machine learning, often applied internally in a model to transform the input data before passing it to a predicting component such as a classifier. In other cases a machine learning model is built with the sole task of figuring out good representations that can then subsequently be applied to downstream learning tasks as input. Of course, *good* representations depend on the task at hand, but often they will capture abstract, general features of the data. Such representations can often be captured without a task in mind, i.e. by learning from *unsupervised* data. Such representations are discussed in section 2.1.

Section 2.2 focuses on the task of representing data in machine learning. In probabilistic settings, representations are distributed in a feature space, or *latent space*, such that a given data sample has an associated uncertainty. These representations are discussed in section 2.2.1.

2.1 Supervised vs. Unsupervised Learning

A *supervised* learning setting consists of a data set $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ of n pairs of a data sample x_i and its target y_i (also called a label of the data sample). The learning task is to infer the general function that maps from samples to targets, based on the available data set. This includes unseen samples, and so a successfully learned function generalizes to all samples, not just the observed. In these settings, learned representations has a clear objective; they should aid the supervised learning task. Supervised learning usually requires some processing of data to obtain its associated targets (a supervision), requiring human interaction. This dependency has the undesirable effect that labelled data sets are in many cases expensive and small. Therefore, in many practical settings, supervised learning is a constraint on the size of the available data set, effectively excluding any available data which does not have associated targets. Such constraints can lead to extreme data inefficiency, as in the case of protein sequences shown in figure 2.1. Here the Protein Data Bank (PDB) contains the labeled data, in this case the protein sequence and its 3D structure, and UniParc contain raw sequences only. The amount of available unlabeled data is staggering compared to the amount of labeled available data. This difference alone is a strong incentive to explore what we can do in unsupervised learning settings. Even more importantly, the growth of unlabeled protein sequences has so far behaved exponentially, and so we might expect the difference to be even larger in the future.

In an *unsupervised* learning setting, data samples have no associated targets. The data set is simply $x_1, x_2, \ldots, x_n$ and the learning task is to find underlying structure or pattern in the data samples that is shared across that dataset. Basically, we desire any commonalities or general information in the data

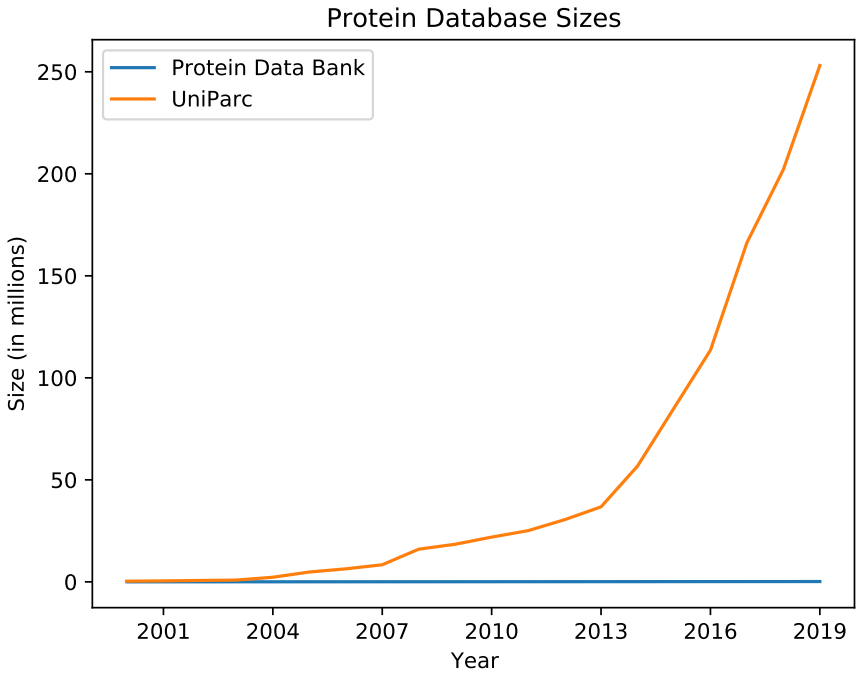


Figure 2.1: The development of protein database sizes over the last decade. While there is an increase in the size of the Protein Data Bank (from 16,403 in 2001 to 158,985 in 2019), this increase is almost unnoticeable compared to the exponential increase of raw protein sequences in the UniParc database.

set to be reflected in the produced representations. This is typically information such as the distribution of the data set or detection of clusters within the data. This is not trivial, as there is no obvious learning objective, i.e. how does the machine know if it has found anything meaningful when there is no labels to guide the process. In the autoencoding approach, adopted in this thesis and discussed in section 3.2.1, the unsupervised learning task is simulated by a supervised setting by reconstructing an input data sample from a compressed representation. That is, the data sample is its own target. In the context of representation learning, the underlying assumption is that in order for the model to learn to reconstruct its input from a compressed representation, the representation must necessarily retain properties of the sample that identify it the most. Making the representation compressed is essential, as the model could otherwise naively pass the sample unaltered through its internal components, effectively learning the identity function $f(x) = x$. The model learns to reconstruct the input samples, having the side-effect that it produces internal representations of the data samples.

2.2 Representations

A fundamental assumption in machine learning settings is that hierarchical modelling is possible: explaining aspects of the learning problem can be broken into less and less abstract sub-factors that are ultimately founded in the raw sample input. That is, when we use a machine learning model on a learning task, it represents a hierarchical representation of how the explaining factors of the data is related, starting from concrete, low-level features of the input and becoming gradually more abstract as we pass through the model toward its output. Explaining factors are what a good representation contains, implying that such representations can be learned using neural networks that captures the representation hierarchy.

Another desired property we have on the representation space is that it is smooth: if data samples x_1, and x_2 are close (by some distance metric), then so are their representations $r(x_1)$ and $r(x_2)$. This is central to the idea that similar samples end up having similar representations. In exploratory set-

tings, this implies that we should look in the neighborhood of a sample if we wish to observe close modifications of that sample.

In addition, any data set we are likely to use is a product of many, complex processes that interact in a shared context. Explanatory factors of the data set are therefore often *entangled*, meaning that they are highly dependent and thus hard to separate (look at independently). Because the data is generated from such complex factors, learning good representations ideally involves *disentangling* them to features that still captures the variation of the data, but are not dependent on each other. Such disentangled representations are however a fundamental impossibility without biases, as shown by Locatello et al. [8], indicating that we should not expect to separate causes into neat and separate dimensions in a representation space. Still, representations that contain features that correspond to causes of the observed data may very well generalize to several tasks as these causes are what is assumed to explain the factors of the observed data. The idea is thus that the generality of good representations can be exploited in different supervised learning settings on the same data.

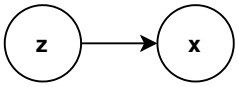
This does not however mean that the learned representation features are easily understood by humans. Usually, representations learned by the deep learning model are complex and hard to interpret. Often, the model is like a black box that somehow figures out how to transform the data into appropriate abstract representations, and those representations are then evaluated based on their metrics/performance, rather than interpretation of what the representations are. Ideally, a good representation also provides this aspect of interpretability for humans.

2.2.1 Latent Spaces

Here we briefly touch on the idea of modelling explanatory factors using random variables, an idea which is explored in detail in Chapter 3 on probabilistic modelling. As discussed above, if a data sample $\boldsymbol{x}$ is generated from a mixture of causes, we can conceptually think of $\boldsymbol{x}$ being generated from a mixture model, where each component of the model being a cause for the generating process. As we do not know the underlying mixture model, a way round this is to represent all the explanatory factors by a single (vector) variable. We call this variable $\boldsymbol{z}$ a *latent variable*, as we only obtain information about it indirectly through the data, and not from direct observation. The causal generative model for the observed data can then be expressed in terms of $\boldsymbol{z}$ as

$$p(\boldsymbol{x}, \boldsymbol{z}) = p(\boldsymbol{x} \mid \boldsymbol{z})p(\boldsymbol{z}),$$

where we regard the joint distribution as structured in terms of $\boldsymbol{z}$ causing $\boldsymbol{x}$, which can be viewed as a simple probabilistic graph model:



The marginal distribution $p(\boldsymbol{x})$ can be recovered by integrating over the latent space. The idea is that a good representation is one which recovers the underlying causes represented by $\boldsymbol{z}$. Specifically, if we are interested in some property $\boldsymbol{y}$ of $\boldsymbol{x}$ that is either itself a cause of $\boldsymbol{x}$ or related to a cause, the knowledge of $\boldsymbol{z}$ can help inferring $p(\boldsymbol{y} \mid \boldsymbol{x})$, i.e. how likely it is that we have the property $\boldsymbol{y}$ given $\boldsymbol{x}$.

2.2.2 Protein Representations: Global and Local Representations

The sample space of proteins is very large and governed by a long and arduous evolutionary process. The protein samples observable today in life are distributed in accordance with this process. Making good representations for all proteins is therefore a global representation, i.e. the representation should capture well the core properties and causes of any protein likely to be generated by evolution. A model that can produce such representations would need to be be sufficiently complex to capture the variance

of such a vast input space, and the learning task of training such a model is an active research field. The motivation is clear: such representations would be broadly applicable in any setting where proteins are used, regardless of the type of protein. It would also be able to take full advantage of the entire sample space of proteins, as any protein informs such a representation. For much the same reason, the model would be left in the dark for large areas of protein space, as the number of available protein sequences do not evenly cover existing organisms on Earth. It is also questionable whether such a representation would be able to learn truly general properties given that the available protein data is sparse. In addition, most practical protein tasks focus on small sub-regions of protein space, and it is questionable whether a general representation would perform sufficiently well in such settings, compared to a less ambitious, local representation.

A *local* representation of protein sequences is a representation that is applicable to a subset of proteins, e.g. a protein family, and not the entire global protein space. Such specialized representations have the advantage that they do not need to learn about every single natural protein sequence; this smaller learning task is much more feasible and requires data from the protein subset space alone. The downside is that they do not exploit available data outside of the local space and thus its representations cannot be efficiently used for tasks that rely on data outside the local protein space. In comparison with global representation learning, local representation learning can, all other things being equal, make do with less complex models as the learning task is simpler.

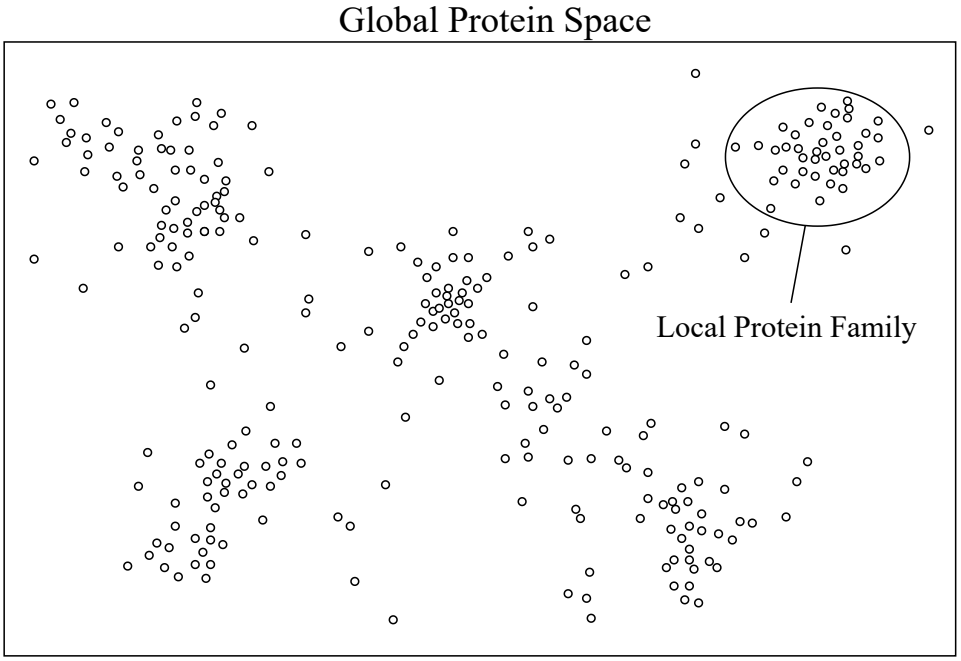


Figure 2.2: A sketch of a space of protein representations in 2 dimensions. The global protein representations (dots) encapsulates all natural protein sequences, while local protein representations are only concerned with a clustered subset, often a protein family.

Any natural[1] protein share commonalities and causal factors that should help a learner to infer useful information about one protein from another protein. This is irrespective of any global or local distinction, and so a model might preferably in any case first look at all the available protein data (pretrain on global protein space) and then, under task specific conditions finetune on local protein space. In practical settings, the available data under a local space might be sparse, e.g. small protein families with few members. Global pretraining can be the important factor that allow learning to take place from such small datasets.

[1] a protein likely to exist in our evolutionary environment, or synthesized proteins derived from occurring proteins

The usefulness of global versus local representations is an open question, especially in protein representation learning. Global representations seem more generally useful, but also require more training data and more complex models to sufficiently capture the essential features of any given protein. It is also questionable whether global representations for data like proteins are even feasible – consider this back-of-the-napkin calculation: Let's say that all proteins are approximately 1000 amino acids long, and there are 20 different amino acids. That gives about $1000^{20} = 10^{60}$ potentially possible proteins. Probably only a very small fraction of these are actually possible/useful proteins, so let's just round it down to 10^{20} (as a comparison, it is estimated that there are about 10^{80} atoms in the universe).

Even with this reduced number of proteins, it seems infeasible for a reasonably sized model to accurately describe so many proteins. That is, a single "small" representation should be able to capture all the essential features of all those proteins. If the representation is unable to do this, it may only capture large-scale trends across the global protein space, but fail to capture detail in any local protein space.

In comparison, the local protein space is minuscule. A protein family still contains a lot of proteins, but probably no more than, say, 10^{10}, and often no more than a few tens of thousand of proteins. Additionally, the proteins in a protein family have more similarities than the proteins of the global space, which means the representation can be simpler, as it only has to concern itself with the remaining differences. Thus, a model utilizing a local representation seems intuitively much more feasible than one using a global representation.

3
Probabilistic Modelling

The world is complicated. Consequently, learning to navigate in such a world is hard for any living species, let alone computers. One difficulty is that the governing processes that determine much of the information we observe are vastly more complex than what we are capable of understanding. This introduces uncertainty about what we think and know and how it affects our decisions. The field of *probabilistic modelling* is a way to incorporate this uncertainty into the way we think about the things we observe and how we act on them. Namely, observed data is assumed to be sampling from an underlying distribution, and the modelling task is to capture this distribution.

In this chapter we relate probabilistic modelling to the procedures that structure the machine learning models we use.

3.1 Inference

In statistics, *inference* is the process of finding out information or even determining the underlying distribution of data. In our settings, we care about inferring the distribution that governs the observed data. This is intractable to compute directly in many cases of interest, so usually the best alternative is to approximate.

There is at least one kind of inference type that we desire in this project context: the marginal inference, i.e. the marginal distribution of a variable. In our case, we care about the marginal distribution

$$p(\boldsymbol{x}) = \int_{\boldsymbol{z}} p(\boldsymbol{x} \mid \boldsymbol{z}) p(\boldsymbol{z}) \, \mathrm{d}\boldsymbol{z} \tag{3.1}$$

for samples $\boldsymbol{x}$ and possible representations $\boldsymbol{z}$. This expression of $p(\boldsymbol{x})$ is also called a mixture model, as we integrate over all possible values of $\boldsymbol{z}$, weighing the conditional distribution $p(\boldsymbol{x} \mid \boldsymbol{z})$ according to $p(\boldsymbol{z})$. Defining $p(\boldsymbol{x})$ in these terms also links the type of inference to *Bayesian inference*, that incorporates proposed knowledge about the system as a prior distribution using Bayes' theorem

$$p(\boldsymbol{z} \mid \boldsymbol{x}) = \frac{p(\boldsymbol{x} \mid \boldsymbol{z}) p(\boldsymbol{z})}{p(\boldsymbol{x})}. \tag{3.2}$$

This relation will become more apparent in section 3.2.3.

3.1.1 Variational Inference

In *variational inference*, an intractable distribution p is approximated by inspecting a set of candidate distributions $\mathcal{Q}$ and choosing a distribution $q \in \mathcal{Q}$ which is the most similar to p. This way, we can use q as a substitute approximation for p, using it for whatever task that required p. In our settings, we constrain the set of candidate solutions $\mathcal{Q}$ to the set of diagonal Gaussian distributions.

Similarity in our case is determined by the *Kullback-Leibler divergence* (KL), which is roughly a measure of distance between two distributions: KL divergence is zero when the distributions are identical and positive otherwise, but not necessarily symmetric (which one would expect from a distance measure) [9]. It is defined as follows:

$$\mathrm{KL}\left(q \parallel p\right) = \mathop{\mathbb{E}}_{\boldsymbol{x} \sim q} \left[\log q(\boldsymbol{x}) - \log p(\boldsymbol{x})\right].$$

Note that the expectation is with respect to the first input of the divergence. We introduce variational inference, as it is specifically used in variational autoencoders (VAEs), to simultaneously (1) derive an approximation $q(\boldsymbol{z} \mid \boldsymbol{x})$ to the true conditional distribution $p(\boldsymbol{z} \mid \boldsymbol{x})$, where $\boldsymbol{x}$ is a sample from the observed data space, and $\boldsymbol{z}$ is a representation given that sample and (2) increase the loglikelihood of the observed data. The first is achieved by minimizing the KL divergence. Here the procedure is explained, but not how the divergence measure is used to find the most similar candiate distribution for p. The actual variational inference process maximizes the *evidence lower bound* (ELBO). This key component of the VAE is discussed further in section 3.2.3.

3.2 The Variational Autoencoder

3.2.1 The Autoencoder

An *autoencoder* network is reconstructing its input from an internal representation of the input. To make the network non-trivial (and useful), it is constrained in its capacity to fully represent inputs. This has the effect that it forces the network to compress inputs into its most important properties in order for the network to make the most accurate reconstruction. As a result of this important side effect, meaningful representations of the inputs can be learned by the autoencoder, making it useful in unsupervised settings where only the raw inputs are available.

Such a model is typically constructed by two networks: the *encoder* and the *decoder* network. The task of the encoder is to convert the input into its compressed internal representation (as the name suggests, encoding the input). We occasionally refer to this as the "bottleneck" representation of the model input. The output of the encoder is then passed as input to the decoder, tasked to reconstruct the original input of the encoder.

This idea can be expanded to probabilistic settings, in partial by approximating a distribution over the encoded space and the decoded space, and more fully by incorporating distribution over the model parameters themselves. The former is discussed in the following section 3.2 and the latter is discussed in section 3.3 on Bayesian networks.

An important consequence of the architecture is that the autoencoder only have to learn to represent samples from the input data distributions well. This is important as it relaxes the needed power of the network, only requiring it to accurately represent samples that are likely under the data distribution and not the much larger space of unlikely samples as well.

3.2.2 The Variational Autoencoder

One way to utilize the autoencoding setup is to combine it with variational inference: Representations are latent variables $\boldsymbol{z}$ from a latent space of dimension d. Given a prior distribution $p(\boldsymbol{z})$ and a dataset $\boldsymbol{X}$ containing samples $\boldsymbol{x}$ from some sample space, we seek to build an *encoder* neural network that can transform $\boldsymbol{x}$ into a conditionally distributed latent variable $\boldsymbol{z}$. Another *decoder* neural network then learns to transform latent variables back to sample-space by inference. This entire setup is collectively called a *Variational Autoencoder* (VAE). For our purposes VAE's have at least two major useful properties: (1) they learn latent representations of its inputs and (2) they contain a generative model. The generative model is a joint distribution on $\boldsymbol{x}$ and $\boldsymbol{z}$ that can be expressed as $p(\boldsymbol{x}, \boldsymbol{z}) = p(\boldsymbol{x} \mid \boldsymbol{z})p(\boldsymbol{z})$. The decoder correspond to the conditional distribution $p(\boldsymbol{x} \mid \boldsymbol{z})$. The encoder is the part of the model that approximates $p(\boldsymbol{z} \mid \boldsymbol{x})$ by a distribution $q(\boldsymbol{z} \mid \boldsymbol{x})$. This model architecture is further discussed in section 5.2, and diagrammed figure 6.2.

The VAE is by now a popular and examined framework for various machine learning tasks, with established results in the domain of protein prediction [6], that we have used throughout our experiments as a basis and guide. But there are other more general reasons why the VAE framework is well suited for our task instead of other approximation methods. Specifically, the VAE assumes no tractability

on any of the involved integrals, and can thus be applied in very general settings where the involved distributions are complicated, which is what is present in our case. In addition it allows for gradient based optimization using batches, which mean that we practically avoid more time-consuming sampling schemes. This is critical for how well the model scales, which is important as the unlabeled protein data available is very large. Finally, as mentioned, the VAE allows for explicitly learned representations and for protein exploration in the representation space, both of which is at the core of our focus, and will be discussed throughout this and subsequent chapters.

In order to make the generative model $p(\boldsymbol{x} \mid \boldsymbol{z})$ likely to generate samples that are similar to the ones observed in $\boldsymbol{X}$, the task is to maximize the probability $p(\boldsymbol{x})$ of the observed samples $\boldsymbol{x} \in \boldsymbol{X}$. That is, if the model has high probability on the observed samples, then samples that are similar should be probable as well; conversely, samples that are unlike what has been observed are unlikely to be generated by the model. This is achieved by integrating out the latent variables $\boldsymbol{z}$ from the generative model $p(\boldsymbol{x}, \boldsymbol{z})$ to get the marginal distribution $p(\boldsymbol{x})$ as expressed in equation (3.1).

This requires two things, namely making a suitable latent space, and integration over that space to get $p(\boldsymbol{x})$. The former is learned by the network (how this is done is discussed in section 3.2.3 and 3.2.4). The latter is usually intractable, so the solution is to instead approximate the integral by independently sampling a number of latent variables $(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n)$ from $p(\boldsymbol{z})$, the distribution over the latent space, and use $\frac{1}{n}\sum_{i=1}^{n} p(\boldsymbol{x} \mid \boldsymbol{z}_i)$ as an approximation for $p(\boldsymbol{x})$. This is an unbiased estimator of $p(\boldsymbol{x})$ as its expectation is $p(\boldsymbol{x})$:

$$\mathop{\mathbb{E}}_{\boldsymbol{z}_1,\ldots,\boldsymbol{z}_n}\left[\frac{1}{n}\sum_{i=1}^{n} p(\boldsymbol{x} \mid \boldsymbol{z}_i)\right] = \frac{1}{n}\sum_{i=1}^{n} \mathop{\mathbb{E}}_{\boldsymbol{z}_i}\left[p(\boldsymbol{x} \mid \boldsymbol{z}_i)\right] = \mathop{\mathbb{E}}_{\boldsymbol{z}}\left[p(\boldsymbol{x} \mid \boldsymbol{z})\right] = \int_{\boldsymbol{z}} p(\boldsymbol{x} \mid \boldsymbol{z}) p(\boldsymbol{z}) \, \mathrm{d}\boldsymbol{z} = p(\boldsymbol{x}),$$

where we use that the samples are independently and identically distributed. The problem with this approach is that if we sample from $p(\boldsymbol{z})$ we might need many samples to accurately approximate $p(\boldsymbol{x})$ because for almost all $\boldsymbol{z}$, the probability $p(\boldsymbol{x} \mid \boldsymbol{z})$ is very small. The intuition here is that there are many latent space candidates that do not represent protein $\boldsymbol{x}$ very well. We preferably want to sample only values of $\boldsymbol{z}$ for which $p(\boldsymbol{x} \mid \boldsymbol{z})$ is large, as these are the values most significant to the approximation. Such a distribution over $\boldsymbol{z}$ would allow us to skip all the insignificant samples, effectively reducing the amount of samples needed to approximate $p(\boldsymbol{x})$. In fact, the distribution we want to sample $\boldsymbol{z}$ from is $p(\boldsymbol{z} \mid \boldsymbol{x})$, determining the most likely values of $\boldsymbol{z}$ given a specific sample $\boldsymbol{x}$. This leads us to $q(\boldsymbol{z} \mid \boldsymbol{x})$, the encoding part of the VAE, as this is the model we use to find candidate distributions similar to $p(\boldsymbol{z} \mid \boldsymbol{x})$. Figure 3.1 shows the relationship between the sample space and latent space, where $p(\boldsymbol{x} \mid \boldsymbol{z})$ and $q(\boldsymbol{z} \mid \boldsymbol{x})$ facilitates the transformations between the spaces.

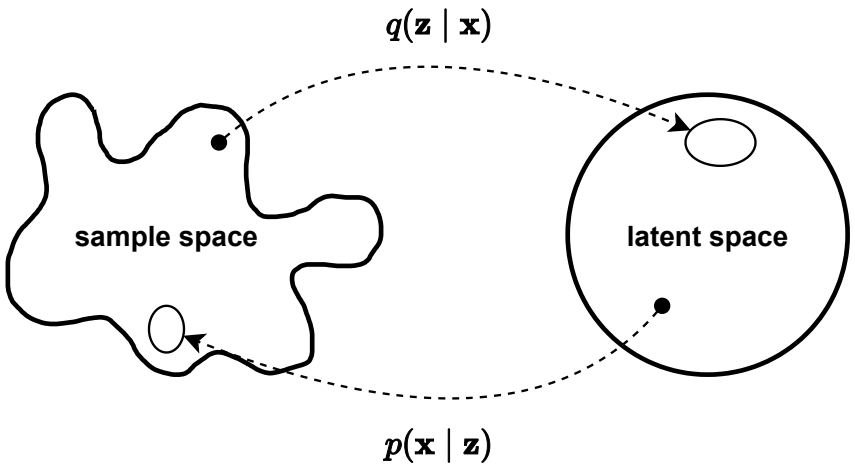


Figure 3.1: An observed sample $\boldsymbol{x}$ from the sample space distribution is encoded to a latent variable $\boldsymbol{z}$ in latent space by the approximating distribution $q(\boldsymbol{z} \mid \boldsymbol{x})$. This is the encoder of the VAE. Latent variables are decoded back to sample space with $p(\boldsymbol{x} \mid \boldsymbol{z})$. This is the decoder of the VAE. The distribution over sample space $p(\boldsymbol{z})$ is in our settings assumed to be a unit Gaussian, and so this space is spherical, whereas the distribution over the sample space is more complex.

In order to determine a suitable approximation distribution $q(\boldsymbol{z} \mid \boldsymbol{x})$ and to train the autoencoder network, an objective function is needed with respect to the network parameters. In alignment with existing literature, we denote $\boldsymbol{\theta}$ as the parameters of the generative model (decoder), and $\boldsymbol{\phi}$ as the parameters of the variational model (encoder). As discussed in section 3.1.1, the ELBO is what the network will maximize.

3.2.3 Evidence Lower Bound

The evidence lower bound (ELBO) is the measure used to train the variational autoencoder. Without further explanation the ELBO is

$$\mathcal{L}(\boldsymbol{x}) = \mathop{\mathbb{E}}_{\boldsymbol{z}\sim q}[\log p(\boldsymbol{x} \mid \boldsymbol{z})] - \mathrm{KL}\left(q(\boldsymbol{z} \mid \boldsymbol{x}) \parallel p(\boldsymbol{z})\right), \tag{3.3}$$

where the expectation subnotation $\boldsymbol{z} \sim q$ is shorthand for $\boldsymbol{z} \sim q(\boldsymbol{z} \mid \boldsymbol{x})$ (used throughout this section). It contains two expressions, an expectation of $\log p(\boldsymbol{x} \mid \boldsymbol{z})$ and a divergence between $q(\boldsymbol{z} \mid \boldsymbol{x})$ and $p(\boldsymbol{z})$. Note that if the prior distribution on $\boldsymbol{z}$ is assumed unit Gaussian, $p(\boldsymbol{z}) = \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ and if $q(\boldsymbol{z} \mid \boldsymbol{x})$ is also Gaussian, then the divergence term can be computed analytically.

The expression in equation (3.3) can be derived in several ways; one is to regard the goal of the variational inference, namely to find a suitable distribution $q(\boldsymbol{z} \mid \boldsymbol{x})$ that approximates $p(\boldsymbol{z} \mid \boldsymbol{x})$. The KL divergence is the metric used to determine the most suitable candidate. From this expression, we get the following derivation:

$$\begin{aligned}
\mathrm{KL}\left(q(\boldsymbol{z} \mid \boldsymbol{x}) \parallel p(\boldsymbol{z} \mid \boldsymbol{x})\right) &= \mathop{\mathbb{E}}_{\boldsymbol{z}\sim q}[\log q(\boldsymbol{z} \mid \boldsymbol{x}) - \log p(\boldsymbol{z} \mid \boldsymbol{x})] && (3.4)\\
&= \mathop{\mathbb{E}}_{\boldsymbol{z}\sim q}\left[\log q(\boldsymbol{z} \mid \boldsymbol{x}) - \log\left(\frac{p(\boldsymbol{x} \mid \boldsymbol{z})p(\boldsymbol{z})}{p(\boldsymbol{x})}\right)\right] && (3.5)\\
&= \mathop{\mathbb{E}}_{\boldsymbol{z}\sim q}[\log q(\boldsymbol{z} \mid \boldsymbol{x}) - \log p(\boldsymbol{x} \mid \boldsymbol{z}) - \log p(\boldsymbol{z}) + \log p(\boldsymbol{x})] && (3.6)\\
&= \mathop{\mathbb{E}}_{\boldsymbol{z}\sim q}[\log q(\boldsymbol{z} \mid \boldsymbol{x}) - \log p(\boldsymbol{z})] - \mathop{\mathbb{E}}_{\boldsymbol{z}\sim q}[\log p(\boldsymbol{x} \mid \boldsymbol{z})] + \log p(\boldsymbol{x}) && (3.7)\\
&= \mathrm{KL}\left(q(\boldsymbol{z} \mid \boldsymbol{x}) \parallel p(\boldsymbol{z})\right) - \mathop{\mathbb{E}}_{\boldsymbol{z}\sim q}[\log p(\boldsymbol{x} \mid \boldsymbol{z})] + \log p(\boldsymbol{x}), && (3.8)
\end{aligned}$$

from which it follows that

$$\log p(\boldsymbol{x}) \geq \log p(\boldsymbol{x}) - \mathrm{KL}\left(q(\boldsymbol{z} \mid \boldsymbol{x}) \parallel p(\boldsymbol{z} \mid \boldsymbol{x})\right) \tag{3.9}$$

$$= \mathop{\mathbb{E}}_{\boldsymbol{z}\sim q}\left[\log p(\boldsymbol{x} \mid \boldsymbol{z})\right] - \mathrm{KL}\left(q(\boldsymbol{z} \mid \boldsymbol{x}) \parallel p(\boldsymbol{z})\right) \tag{3.10}$$

$$= \mathcal{L}(\boldsymbol{x}). \tag{3.11}$$

The derivation uses Bayes' Theorem in equation (3.5) and the definition of KL in (3.8). The inequality in (3.9) is due to the fact that KL is always nonnegative. From equations 3.9 - 3.11 we can see why the expression is named a lower bound; it bounds the logarithm of the data (evidence) distribution from below. Because the logarithm is monotonically increasing function, maximizing $\log p(\boldsymbol{x})$ will also maximize $p(\boldsymbol{x})$. Since $\mathcal{L}(\boldsymbol{x})$ is a lower bound, a larger ELBO ensures a better guarantee on how small $p(\boldsymbol{x})$ can be. It is also worth noting that the gap between $p(\boldsymbol{x})$ and $\mathcal{L}(\boldsymbol{x})$ is $\mathrm{KL}\left(q(\boldsymbol{z} \mid \boldsymbol{x}) \parallel p(\boldsymbol{z} \mid \boldsymbol{x})\right)$, indicating that increasing the ELBO effectively decreases the divergence between $q(\boldsymbol{z} \mid \boldsymbol{x})$ and $p(\boldsymbol{z} \mid \boldsymbol{x})$, which is what we desire in the first place.

In some places the ELBO is denoted as

$$\mathcal{L}(\boldsymbol{x}) = \mathop{\mathbb{E}}_{\boldsymbol{z}\sim q}\left[\log \frac{p(\boldsymbol{x}, \boldsymbol{z})}{q(\boldsymbol{z} \mid \boldsymbol{x})}\right],$$

which is equivalent to (3.3), as

$$\begin{aligned}
\mathop{\mathbb{E}}_{\boldsymbol{z}\sim q}\left[\log \frac{p(\boldsymbol{x}, \boldsymbol{z})}{q(\boldsymbol{z} \mid \boldsymbol{x})}\right] &= \mathop{\mathbb{E}}_{\boldsymbol{z}\sim q}\left[\log p(\boldsymbol{x}, \boldsymbol{z}) - \log q(\boldsymbol{z} \mid \boldsymbol{x})\right] \\
&= \mathop{\mathbb{E}}_{\boldsymbol{z}\sim q}\left[\log p(\boldsymbol{z} \mid \boldsymbol{x}) + \log p(\boldsymbol{z}) - \log q(\boldsymbol{z} \mid \boldsymbol{x})\right] \\
&= \mathop{\mathbb{E}}_{\boldsymbol{z}\sim q}\left[\log p(\boldsymbol{z} \mid \boldsymbol{x})\right] + \mathop{\mathbb{E}}_{\boldsymbol{z}\sim q}\left[\log \frac{p(\boldsymbol{z})}{q(\boldsymbol{z} \mid \boldsymbol{x})}\right] \\
&= \mathop{\mathbb{E}}_{\boldsymbol{z}\sim q}\left[\log p(\boldsymbol{z} \mid \boldsymbol{x})\right] - \mathop{\mathbb{E}}_{\boldsymbol{z}\sim q}\left[\log \frac{q(\boldsymbol{z} \mid \boldsymbol{x})}{p(\boldsymbol{z})}\right] \\
&= \mathop{\mathbb{E}}_{\boldsymbol{z}\sim q}\left[\log p(\boldsymbol{z} \mid \boldsymbol{x})\right] - \mathrm{KL}\left(q(\boldsymbol{z} \mid \boldsymbol{x}) \parallel p(\boldsymbol{z})\right) \\
&= \mathcal{L}(\boldsymbol{x}),
\end{aligned}$$

where it is used that $\log a = -\log \frac{1}{a}$ in the fourth line.

The ELBO objective function is desirable not just because of its maximization leading to a good approximation. In addition, under the i.i.d. assumption, it can be optimized with mini-batching as the probability of observing the entire dataset $\boldsymbol{X}$ can be split into a sum of observing each data point $\boldsymbol{x} \in \boldsymbol{X}$. Also, as explained below in section 3.2.4, we can compute an unbiased estimate of the gradient of the batch using Monte Carlo sampling, which the reparameterization trick allows us to do effectively. Finally, the ELBO is nicely divided into two interpretable terms: the reconstruction loss and the regularizing KL term. Such interpretable terms can help guide the training process as it is helpful to understand what the model tries to improve at various steps in the training procedure.

3.2.4 Gradient Based Optimization

By far the most common way to optimize artificial neural networks is to apply a gradient-based algorithm to iteratively optimize the parameters of the network. This implies that the objective must be differentiable with respect to the optimized parameters and that the gradient can be computed. In the VAE setting, the optimized parameters are those of the encoder, $\boldsymbol{\phi}$ and the decoder $\boldsymbol{\theta}$. That is, for each input $\boldsymbol{x}$ of the network, we wish to compute the gradient $\nabla_{\boldsymbol{\phi},\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{x})$ of the ELBO with respect to the model parameters $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$. The objective function of a batch of samples is the sum/average over all the individual ELBOs of the batch samples.

It turns out that computing the gradient of the ELBO is in general intractable [10], potentially posing an issue with gradient based optimization. There are however unbiased approximations that can substitute the gradient, which allows us to train the model using those approximations and still achieve meaningful results. The gradient $\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{x})$ with respect to $\boldsymbol{\theta}$ can be approximated by $\nabla_{\boldsymbol{\theta}}(\log p(\boldsymbol{x}, \boldsymbol{z}))$ by sampling a value $\boldsymbol{z}$ from $q(\boldsymbol{z} \mid \boldsymbol{x})$ and use this as an approximation of the expected value. This is basically because the expectation is dependent on q, which is parameterized by $\boldsymbol{\phi}$, so the gradient w.r.t $\boldsymbol{\theta}$ can safely be moved inside the expectation. This is however not something that can be done when taking the gradient w.r.t. $\boldsymbol{\phi}$, as the expectation is dependent on $\boldsymbol{\phi}$. That is, we have

$$\nabla_{\boldsymbol{\phi}} \mathcal{L}(\boldsymbol{x}) = \nabla_{\boldsymbol{\phi}} \mathop{\mathbb{E}}_{\boldsymbol{z} \sim q} \left[\log \frac{p(\boldsymbol{x}, \boldsymbol{z})}{q(\boldsymbol{z} \mid \boldsymbol{x})} \right] \neq \mathop{\mathbb{E}}_{\boldsymbol{z} \sim q} \left[\nabla_{\boldsymbol{\phi}} \log \frac{p(\boldsymbol{x}, \boldsymbol{z})}{q(\boldsymbol{z} \mid \boldsymbol{x})} \right].$$

This means that we cannot directly sample a $\boldsymbol{z}$ value and compute a gradient approximation as we did in the gradient approximation w.r.t. $\boldsymbol{\theta}$. Nor can we backpropagate through the sampling of $\boldsymbol{z}$ because this is a random variable. The way to overcome this is to use a *reparameterization trick* to make the expectation and $\boldsymbol{z}$ depend on a noise parameter $\boldsymbol{\epsilon}$ instead of $q(\boldsymbol{z} \mid \boldsymbol{x})$, which depends on $\boldsymbol{\phi}$. This parameter can be moved outside the backward gradient pass, allowing for backpropagation through the model.

Reparameterization Trick

Instead of sampling $\boldsymbol{z}$ from $q(\boldsymbol{z} \mid \boldsymbol{x})$, we sample an independent random variable $\boldsymbol{\epsilon}$ from which we compute $\boldsymbol{z}$ as a (deterministic) function f of $\boldsymbol{x}$, $\boldsymbol{\phi}$ and $\boldsymbol{\epsilon}$. This way, the randomness has been moved "outside" of $\boldsymbol{z}$ to an input parameter $\boldsymbol{\epsilon}$. Computing $\boldsymbol{z}$ is now an evaluation of a deterministic function given its inputs. If f can also be differentiated, we can compute gradients through $\boldsymbol{z}$ w.r.t $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$. With this reparameterization we have that $\boldsymbol{z} = f(\boldsymbol{x}, \boldsymbol{\phi}, \boldsymbol{\epsilon})$. This allows us to write the expectation in $\mathcal{L}(\boldsymbol{x})$ over $\boldsymbol{\epsilon}$ instead of $\boldsymbol{z}' \sim q(\boldsymbol{z} \mid \boldsymbol{x})$:

$$\begin{aligned}
\nabla_{\boldsymbol{\phi}} \mathcal{L}(\boldsymbol{x}) &= \nabla_{\boldsymbol{\phi}} \mathop{\mathbb{E}}_{\boldsymbol{z}' \sim q} \left[\log \frac{p(\boldsymbol{x}, \boldsymbol{z}')}{q(\boldsymbol{z}' \mid \boldsymbol{x})} \right] \\
&= \nabla_{\boldsymbol{\phi}} \mathop{\mathbb{E}}_{\boldsymbol{\epsilon}} \left[\log \frac{p(\boldsymbol{x}, \boldsymbol{z})}{q(\boldsymbol{z} \mid \boldsymbol{x})} \right] && \text{by reparameterization and } \boldsymbol{z} = f(\boldsymbol{x}, \boldsymbol{\phi}, \boldsymbol{\epsilon}) \\
&= \mathop{\mathbb{E}}_{\boldsymbol{\epsilon}} \left[\nabla_{\boldsymbol{\phi}} \log \frac{p(\boldsymbol{x}, \boldsymbol{z})}{q(\boldsymbol{z} \mid \boldsymbol{x})} \right] && \text{as the expectation does not depend on } \boldsymbol{\phi} \\
&\simeq \nabla_{\boldsymbol{\phi}} \log \frac{p(\boldsymbol{x}, \boldsymbol{z})}{q(\boldsymbol{z} \mid \boldsymbol{x})} && \text{by sampling } \boldsymbol{\epsilon}.
\end{aligned}$$

The important takeaway is that this reparameterization trick makes it tractable to approximate the gradient of the ELBO w.r.t. the model parameters and thus do gradient-based optimization. In our settings, the function f is simply $f(\boldsymbol{\mu}, \boldsymbol{\sigma}, \boldsymbol{\epsilon}) = \boldsymbol{\mu} + \boldsymbol{\epsilon} * \boldsymbol{\sigma}$, where $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are distribution parameters of $q(\boldsymbol{z} \mid \boldsymbol{x})$ that are determined by $\boldsymbol{\phi}$. The operator $*$ indicates element-wise multiplication. We sample $\boldsymbol{\epsilon}$ from a unit Gaussian $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$, effectively making $\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}\boldsymbol{I}) = q(\boldsymbol{z} \mid \boldsymbol{x})$, as we desire. This sampling of $\boldsymbol{\epsilon}$ implies that this trick is only viable for a continuous latent space (which is our case), as the sampling is continuous. It also reveals the distribution class of $q(\boldsymbol{z} \mid \boldsymbol{x})$ in our settings to be the set of diagonal Gaussian distributions. This is an important aspect, as we can see from equation (3.1) that this decision makes $p(\boldsymbol{x})$ a mixture model over Gaussians, if we use $q(\boldsymbol{z} \mid \boldsymbol{x})$ in place of $p(\boldsymbol{z} \mid \boldsymbol{x})$.

3.3 Bayesian Neural Networks

In the above gradient-based optimization learning procedure, the optimization is applied directly to the encoder and decoder parameters $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$. This application yields *maximum a posteriori* (MAP) estimates of the optimal parameters, if regularization is applied to the *maximum likelihood estimate* in

form of a Gaussian prior on the model parameters [11]. These are point estimates to the parameters that maximize the ELBO, but the modelling procedure does not capture the uncertainty inherent in the estimation process. Ideally it is desirable to capture this uncertainty during learning as this information tells us how much we should rely on the estimates.

In *Bayesian Neural Networks* (BNN), the parameters $\boldsymbol{\theta}$ of the model are not fixed, but distributed according to some distribution $p(\boldsymbol{\theta})$, which can be conditioned on the observed data $\boldsymbol{X} = \boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ to yield a posterior distribution $p(\boldsymbol{\theta} \mid \boldsymbol{X})$ that captures both the best estimate (the mean of the posterior) and the uncertainty involved in the estimate (the deviation of the distribution from the mean). This allows us not only to ascertain how precise the estimates are, but also what likely alternatives would be, as we can sample parameters from the distribution. We already applied this idea to the latent variable $\boldsymbol{z}$ and there is no significant conceptual barrier that prevents us from extending this idea to the parameters of the model itself. Figure 3.2 shows a sketch of a classical linear neural network layer and the same layer with Bayesian weights.

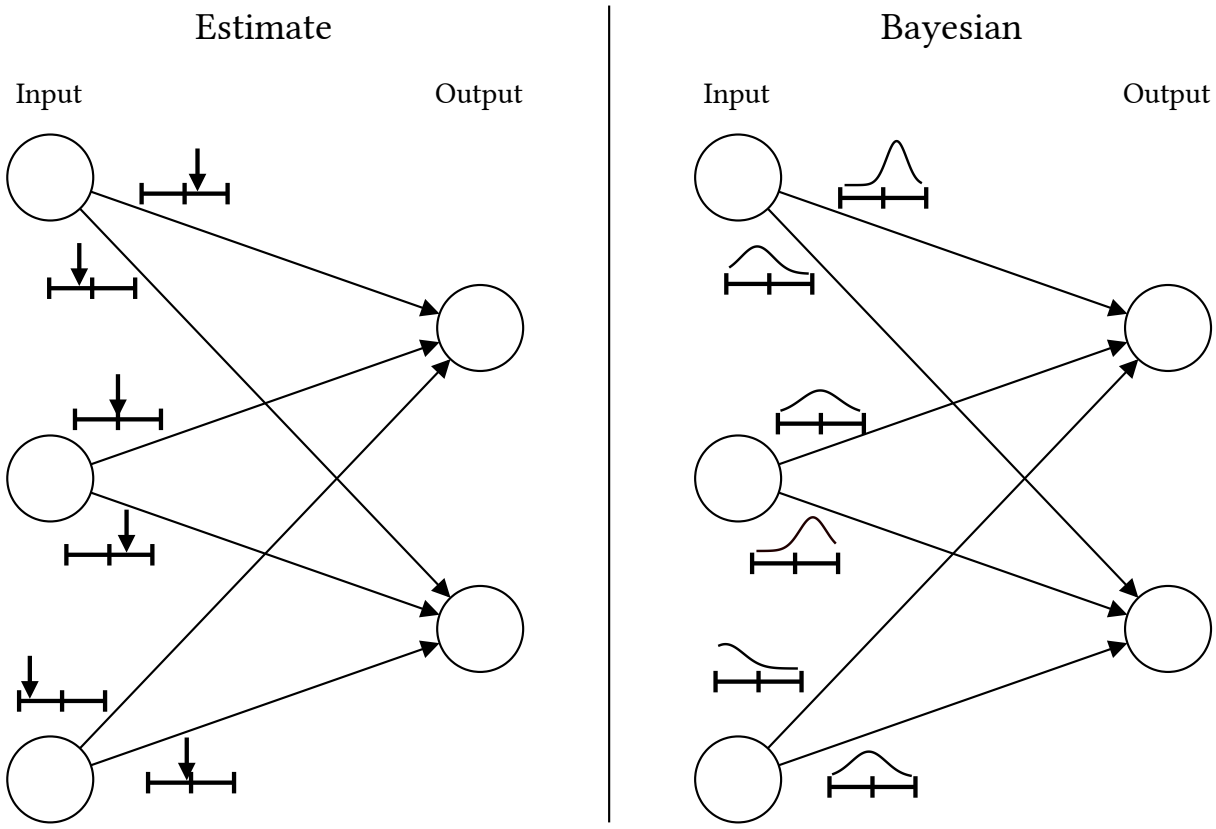


Figure 3.2: A linear neural network layer from a dimension of 3 to 2. The left section shows the classical linear layer, where the weights given to each neuron are point estimates (indicated by an arrow on a number line). The right section shows the Bayesian counterpart of the same linear layer, where the weights are given by a normal distribution (indicated by the probability density function), instead of an estimate. The weight distributions inform an uncertainty on each weight that was lacking from the estimate view.

The networks are Bayesian in the sense that the prior and the posterior of the parameters interact with each other as expressed by Bayes' Theorem (3.2). In our case of neural network parameters this computation is intractable, and so variational inference is applied once again to approximate the posterior. So the augmentation of the neural network into a BNN is simply a use of the same probabilistic modelling and inference to a broader set of parameters. Effectively, this means that the model learns the distribution over the parameters and not the parameters themselves, and it learns these distributions by approximation from samples. This way, an ensemble of point-estimate parameter networks is available

through sampling from the BNN [11].

In our modelling, we applied this extension to the parameters $\boldsymbol{\theta}$ of the decoder of our VAE model [12, Appendix F]. This inclusion of the decoder parameters changes the ELBO on the log probability of a given sample $\boldsymbol{x}$, as we now need to incorporate the approximation of the decoder parameters.

One key difference is that the decoder parameters $\boldsymbol{\theta}$ affects not just single sample $\boldsymbol{x}$, but the entire dataset $\boldsymbol{X}$. For that reason, we consider the marginal distribution $p(\boldsymbol{X})$ over the entire dataset. Let $\boldsymbol{Z}$ denote the set of corresponding latent variables for the samples in $\boldsymbol{X}$.

In order to obtain the marginal distribution $p(\boldsymbol{X})$, we now have to integrate not only over the latent space variables $\boldsymbol{Z}$, but also the parameters of the decoder $\boldsymbol{\theta}$:

$$p(\boldsymbol{X}) = \int_{\boldsymbol{\theta}} \int_{\boldsymbol{Z}} p(\boldsymbol{X}, \boldsymbol{Z}, \boldsymbol{\theta}) \, \mathrm{d}\boldsymbol{Z} \, \mathrm{d}\boldsymbol{\theta} = \int_{\boldsymbol{\theta}} \int_{\boldsymbol{Z}} p(\boldsymbol{X} \mid \boldsymbol{Z}, \boldsymbol{\theta}) p(\boldsymbol{Z} \mid \boldsymbol{\theta}) p(\boldsymbol{\theta}) \, \mathrm{d}\boldsymbol{Z} \, \mathrm{d}\boldsymbol{\theta}. \tag{3.12}$$

This integral is intractable but can be approximated using sampling. The difference from the standard VAE is that we have to sample $\boldsymbol{\theta}$ to do the approximation. We wish to sample from $p(\boldsymbol{\theta} \mid \boldsymbol{X})$ instead of sampling $\boldsymbol{\theta}$ from $p(\boldsymbol{\theta})$. However, as $p(\boldsymbol{\theta} \mid \boldsymbol{X})$ is unknown, we wish to use variational inference to approximate it. We do this by a candidate distribution $q(\boldsymbol{\theta} \mid \boldsymbol{X})$ by minimizing their Kullback-Leibler divergence. This leads us to the following lower bound on $p(\boldsymbol{X})$:

$$\begin{aligned}
\mathrm{KL}\left(q(\boldsymbol{\theta} \mid \boldsymbol{X}) \parallel p(\boldsymbol{\theta} \mid \boldsymbol{X})\right) &= \mathop{\mathbb{E}}_{q}\left[\log \frac{q(\boldsymbol{\theta} \mid \boldsymbol{X})}{p(\boldsymbol{\theta} \mid \boldsymbol{X})}\right] \\
&= \mathop{\mathbb{E}}_{q}\left[\log q(\boldsymbol{\theta} \mid \boldsymbol{X}) - \log p(\boldsymbol{X} \mid \boldsymbol{\theta}) - \log p(\boldsymbol{\theta}) + \log p(\boldsymbol{X})\right] \\
&= \log p(\boldsymbol{X}) + \mathop{\mathbb{E}}_{q}\left[\log q(\boldsymbol{\theta} \mid \boldsymbol{X}) - \log p(\boldsymbol{\theta})\right] - \mathop{\mathbb{E}}_{q}\left[\log p(\boldsymbol{X} \mid \boldsymbol{\theta})\right] \\
&= \log p(\boldsymbol{X}) + \mathrm{KL}\left(q(\boldsymbol{\theta} \mid \boldsymbol{X}) \parallel p(\boldsymbol{\theta})\right) - \mathop{\mathbb{E}}_{q}\left[\log p(\boldsymbol{X} \mid \boldsymbol{\theta})\right]
\end{aligned}$$

Moving terms around we get the lower bound

$$\log p(\boldsymbol{X}) \geq \log p(\boldsymbol{X}) - \mathrm{KL}\left(q(\boldsymbol{\theta} \mid \boldsymbol{X}) \parallel p(\boldsymbol{\theta} \mid \boldsymbol{X})\right) = \mathop{\mathbb{E}}_{\boldsymbol{\theta} \sim q}\left[\log p(\boldsymbol{X} \mid \boldsymbol{\theta})\right] - \mathrm{KL}\left(q(\boldsymbol{\theta} \mid \boldsymbol{X}) \parallel p(\boldsymbol{\theta})\right)$$

The log probability $\log p(\boldsymbol{X} \mid \boldsymbol{\theta})$ can be written in terms depending on each data sample $\boldsymbol{x} \in \boldsymbol{X}$, such that $\log p(\boldsymbol{X} \mid \boldsymbol{\theta}) = \sum_{\boldsymbol{x} \in \boldsymbol{X}} \log p(\boldsymbol{x})$. Here $p(\boldsymbol{x}) = p(\boldsymbol{x} \mid \boldsymbol{\theta})$ is the standard VAE marginal likelihood where decoder parameters are determined. Thus we can lower bound this using the evidence lover bound as described in section 3.2.3.

Thus the main difference between standard VAE and full Bayesian VAE is the collective divergence term $\mathrm{KL}\left(q(\boldsymbol{\theta} \mid \boldsymbol{X}) \parallel p(\boldsymbol{\theta})\right)$ over the entire dataset $\boldsymbol{X}$. In order to determine the ELBO on individual or batched samples, we have to distribute this term onto each sample of the datset. This can be done unevenly weighted as long as weights sum to 1 and the batches are chosen uniformly at random [11], but it can also simply be done uniformly.

4
Sequence Learning

In this chapter we delve into the models used when data is in the form of sequences of variable length. Sequential variable-length data require models that do not make assumptions on the length of the inputs. For example, standard dense linear layers cannot work on variable-length inputs, since the linear layer has a set number of input neurons.

In the realm of unsupervised learning, the goal of learning on sequences is often to reproduce the input sequence. This is usually done by making the model predict the next element of a sequence, given the sequence up to that point. In natural language modelling, this task is known as "language modelling" since it allows the generation of natural language. In our case, we're interested in similar sequential models, but working on proteins rather than sentences or documents.

In section 4.1 we discuss Recurrent Neural Networks (RNNs), which are sequential models that can be applied to variable-length input data. We cover one of the most popular and practical kinds of RNN, the Long Short-Term Memory (LSTM) model, and briefly mention some alternatives.

In section 4.3 we address how sequential protein data within a single protein family can be manipulated to avoid the problem of variable-length data altogether, with some other disadvantages as a trade-off.

4.1 Recurrent Neural Networks

In regular neural networks, we usually think of the input being given to the model, going through a layer which produces an output, which is then fed to the *next* layer of the model. This continues until the data reaches the final layer, which produces the final output.

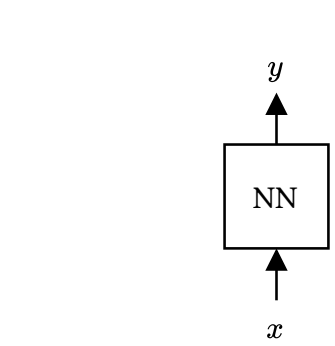
A recurrent neural network (RNN) is a layer that *recurs* in the model – for the purposes of RNNs, this refers to the input being processed by the RNN, producing an output, which is then fed into the RNN again, usually in combination with more data. For sequential data, this "more data" is usually the next element of the sequence. This enables processing of variable-length data, by making the RNN produce a "hidden state" as it processes the sequence. The RNN starts with an initial hidden state, typically initialized to a zero vector. It then takes this hidden state and the first element of the sequence and processes it in some way, to produce the next hidden state. It repeats this process with the new hidden state and the next element of the sequence, until there are no elements in the sequence left. The intermediate hidden states can be thought of as the RNN's "memory" about the sequence seen so far. The final hidden state can then be considered the output of the RNN, or one can use the sequence of the intermediate hidden states as an output, depending on the application.

More formally, consider a sequential datapoint $x_1, x_2, \ldots, x_n$. An RNN can be seen as a function $\mathrm{RNN}\colon \left(\mathbb{R}^h, \mathbb{R}^l\right) \to \mathbb{R}^h$, where h is the size of the hidden state (a hyperparameter) and l is the size of each x_i of the datapoint. To apply the RNN on the entire sequence as described above would mean evaluating the following:

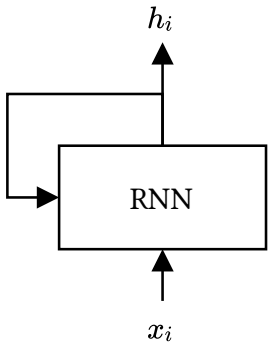
$$h_n = \mathrm{RNN}(\mathrm{RNN}(\ldots \mathrm{RNN}(\mathrm{RNN}(h_0, x_1), x_2) \ldots, x_{n-1}), x_n)$$

Where h_0 is the initial hidden state (usually simply $\mathbf{0}$) and h_n is the final hidden state of the RNN after processing the entire sequence. The RNN function can be defined in many, many ways, but a few popular options have become commonplace. This includes RNNs like the Long Short-Term Memory (LSTM) proposed by Hochreiter et al. [13] or the Gated Recurrent Unit (GRU) proposed by Cho et al. [14].

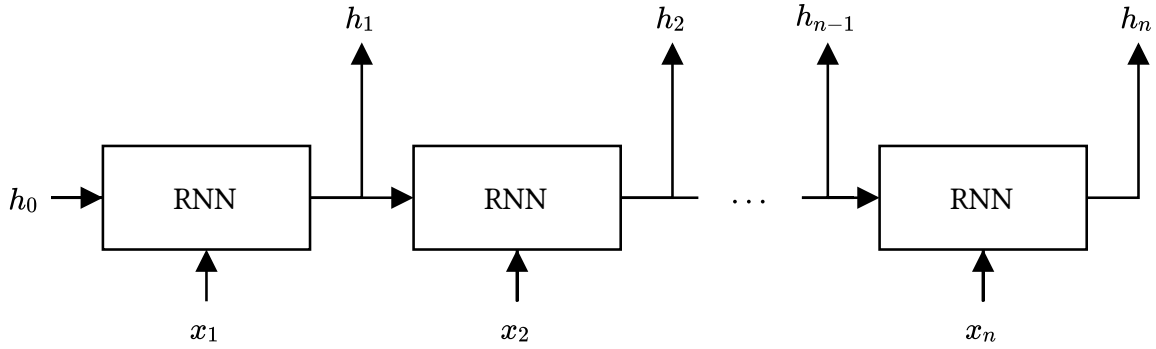
Figure 4.1 shows a comparison of regular neural networks and RNNs. It also illustrates the concept of "unrolling" an RNN. The RNN as shown in figure 4.1b is a recurrent model, that takes its own output as an input. However, if you unwind the recursion over the entirety of the input sequence, you reach figure 4.1c, which more accurately and explicitly shows how an RNN functions. Note that even though the RNN is drawn multiple times in the unrolled version, it is the same model used again and again.



(a) Regular neural network.



(b) Recurrent neural network.



(c) Unrolled recurrent neural network.

Figure 4.1: (a) A regular neural network usually takes a single input and produces a single output. (b) A recurrent neural network takes a sequence as input, processing each element one at a time, feeding the intermediate hidden states as input to the next recursive call of the RNN. The RNN's output can be considered the last hidden state (h_n) or the entire sequence of hidden states. (c) An "unrolled" RNN, more explicitly showing the sequential inputs and the intermediate hidden states.

4.1.1 Long Short-Term Memory

The Long Short-Term Memory (LSTM) is a recurrent neural network (RNN) which uses several *gates* to control how to manipulate the input hidden state based on the input sequence element.

In contrast to some other RNNs, the LSTM actually utilizes two inner states of the same size, where one is known as the "cell" state and another is the hidden state. Thus the LSTM can be explained as the

function $\mathrm{LSTM}\colon \left(\left(\mathbb{R}^h, \mathbb{R}^h\right), \mathbb{R}^l\right) \to \left(\mathbb{R}^h, \mathbb{R}^h\right)$ which has the definition [15]

$$\begin{aligned}
\mathrm{LSTM}((c_{i-1}, h_{i-1}), x_i) &= (c_i, h_i) \\
c_i &= F_i * c_{i-1} + I_i * G_t \\
h_i &= O_i * \tanh(c_i) \\
F_i &= \sigma(\mathrm{L}_1(x_i) + \mathrm{L}_2(h_{i-1})) \\
I_i &= \sigma(\mathrm{L}_3(x_i) + \mathrm{L}_4(h_{i-1})) \\
G_i &= \tanh(\mathrm{L}_5(x_i) + \mathrm{L}_6(h_{i-1})) \\
O_i &= \sigma(\mathrm{L}_7(x_i) + \mathrm{L}_8(h_{i-1}))
\end{aligned}$$

where O_i, G_i, F_i and I_i are the "gates", known as the output gate, cell gate, forget gate and input gate respectively. σ is the sigmoid function while $*$ denotes element-wise multiplication. Each $\mathrm{L_i}$ denotes a linear transformation of the form

$$\mathrm{L_i}(x) = W_i x + b_i$$

where W_i and b_i are learnable parameters of the model, denoted as weight and bias respectively – thus each $\mathrm{L_i}$ denotes what is otherwise known as a dense linear layer with bias.

The above formal definition is perhaps not the most instructive. Figure 4.2 shows a diagram of the structure of the LSTM. Intuitively, the gates can be understood as operations on the internal "memory" of the LSTM (its states). The forget gate F_i can be thought of as deciding what to remove or "forget" from the cell state. The input gate I_i and the cell gate G_i then decide what to add to the "memory" of the cell state. Finally, the output gate O_i decides how to combine the resulting new cell state c_i and the previous hidden state h_{i-1} into a new hidden state h_i.

The LSTM was designed to avoid the common problem of vanishing gradients in neural networks. This problem is especially prevalent in RNNs, because the repeated application of the RNN leads to a very deep computational graph, in which gradients have to be backpropogated. Due to the depth of this graph, the gradient has a tendency to either vanish or explode. Note how the LSTM actually does not perform a lot of operations on the cell state c_{i-1}. This helps prevent the vanishing gradient problem.

4.1.2 Other Recurrent Neural Networks

There are many other types of RNNs. One of the earliest RNNs was proposed by Jeffrey Elman [16], and consists simply of the following computation [15]:

$$\mathrm{Elman}(h_{i-1}, x_i) = h_i = \tanh(\mathrm{L}_1(h_{i-1}) + \mathrm{L}_2(x_i))$$

Where each $\mathrm{L_i}$ again denotes a linear layer, as above.

Another common RNN is the Gated Recurrent Unit (GRU) which, like the LSTM, uses gates to regulate the internal state of the RNN. The GRU however has a somewhat simpler architecture and uses only a single state. The simpler architecture makes the GRU less computationally expensive, and further helps to prevent vanishing or exploding gradients, in comparison to the LSTM. The GRU consists of the following computations [15]:

$$\begin{aligned}
\mathrm{GRU}(h_{i-1}, x_i) &= h_i \\
h_i &= z_i * h_{i-1} + (1 - z_i) * n_i \\
z_i &= \sigma(\mathrm{L}_1(x_i) + \mathrm{L}_2(h_{i-1})) \\
n_i &= \tanh(\mathrm{L}_3(x_i) + r_i * \mathrm{L}_4(h_{i-1})) \\
r_i &= \sigma(\mathrm{L}_5(x_i) + \mathrm{L}_6(h_{i-1}))
\end{aligned}$$

In the above, r_i, z_i and n_i are referred to as the *reset*, *update* and *new* gates, respectively. Intuitively, the gates can be thought of as a method for the GRU to cleverly interpolate between a newly constructed

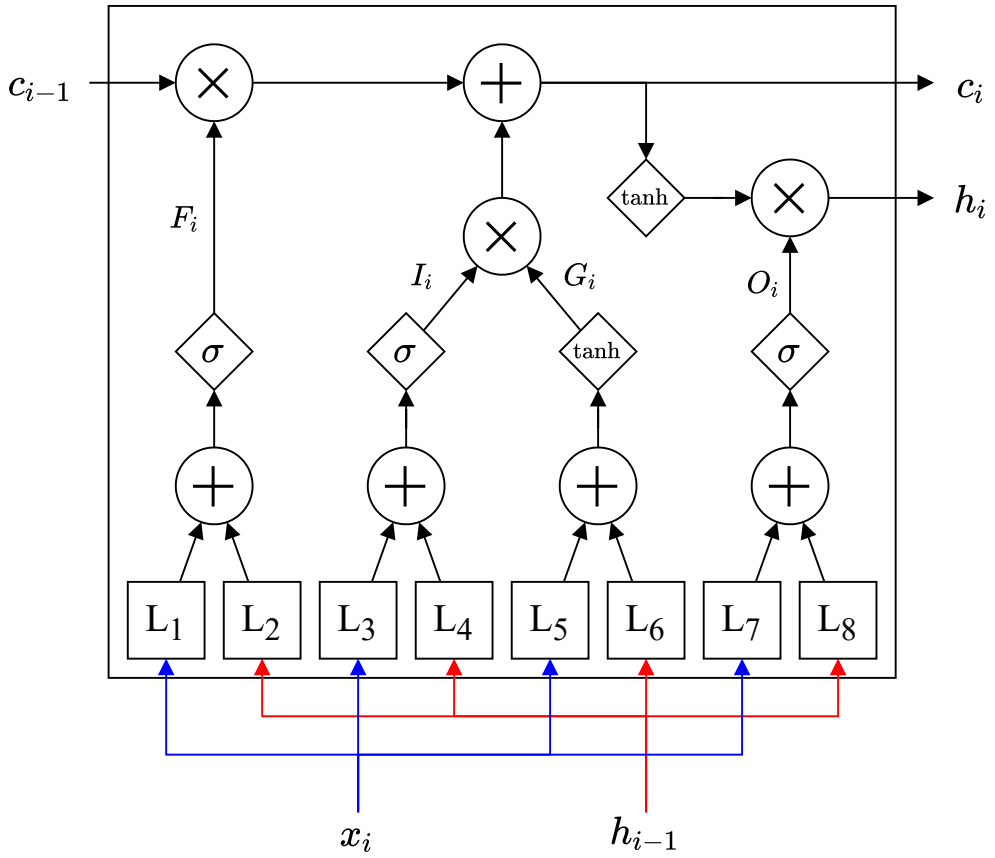


Figure 4.2: The structure of the LSTM, taking c_{i-1}, h_{i-1} and x_i as inputs, and returning c_i and h_i. Squares indicate dense linear layers, diamonds indicate element-wise non-linear activation functions and circles denote binary element-wise operators. Connections from x_i and h_{i-1} coloured for clarity.

hidden state (n_i) and the old hidden state. It makes this interpolation using the update gate's result, z_i. The reset gate roughly decides what to keep/forget from the old hidden state when constructing the new one.

There are many more RNNs, of which many are variations of the above. For example, the multiplicative LSTM (mLSTM) [17] extracts certain information from the previous hidden state, which it then uses in the LSTMs gates, instead of the previous hidden state. That is, it calculates

$$m_i = \mathrm{L}_{\mathrm{m}_1}(x_i) * \mathrm{L}_{\mathrm{m}_2}(h_{i-1})$$

and then uses m_i instead of h_{i-1} in the gates of the LSTM.

4.2 Sequence-to-Sequence Models and Representations

As mentioned above, an RNN's output may be considered either its last hidden state h_n, or the entire sequence of resulting hidden states $(h_1, \ldots, h_n)$.

In the case of the last hidden state, the model produces a "bottleneck"; It forces the sequence through a representation of a fixed size, which is small in the sense that it does not depend on the length of the input sequence. For such a model, a compact representation is readily available in the form of the bottleneck layer. We say that the model up to the bottleneck layer encodes the sequence to the bottleneck and is as such known as the encoder. In order to facilitate conversion from the bottleneck representation back to a sequence (which is useful if you want to explore the space of representations)

and in order to ensure that the representation holds the most essential features of the sequence, a model (the decoder) is trained to decode the representation back to the input sequence – thus we reach the concept of an autoencoder. The decoding of a representation into a variable-length sequence is however a more complex task for a model to learn and perform, as the output length is unknown at the reconstruction. In section 4.3 we will see one way to circumvent this problem.

In the case of the entire sequence of hidden states as the output, we say that the RNN is a "sequence-to-sequence" model, since it takes one sequence of some length n and produces another sequence of the same length[1]. A compact, fixed-size representation is not as easily obtained from the resulting sequence as in the case of bottleneck models, since there is no obvious canonical way to reduce the sequence to such a representation. The sequence could be reduced in many different ways, but this is not always desired, as the translation back to a sequence becomes more complicated or impractical, depending on the way the reduction is done.

It's not obvious which of the above two approaches should be preferred. Bottleneck models easily provide a compact representation, but such a representation is hard to decode back to a variable-length sequence. Sequence-to-sequence models do not require a decoder, since they produce a sequence out of the box – however, obtaining a compact representation is more difficult. We will explore both bottleneck models, in the form of the variational autoencoder, and sequence-to-sequence models in the form of models based on RNNs or other methods.

RNNs are just one type of sequence-to-sequence models. There are other types of models that can be used to process sequences into new sequences. For example, convolutional neural networks (CNNs) can also do this. CNNs are known for their ability to process images, but sequences work just as well – one could after all just consider a sequence as a 1-dimensional image. Making a sequence-to-sequence model based on CNNs simply involves appropriately padding the input, so that the resulting output remains the same length as the original sequence. Then it is only a question of how exactly the convolution is applied to the sequence. We will explore one such method in section 5.3.

4.3 Sequence Alignments

As proteins are sequential in nature, RNNs can be used to integrate them into deep learning models. However, despite RNNs being suited for sequential data, it is not always the best model to use for proteins. While RNNs are able to process proteins, there are certain disadvantages in the way that RNNs do processing. For example, when an RNN processes a single amino acid, it cannot take into account any remaining amino acids of the protein, because it has not reached those amino acids yet, even though it may be worthwhile to consider both previous and future amino acids in the sequence. Also, the RNN cannot directly take into account the previous positions of the sequence – it can only use the hidden state up to a certain point, which is only a compact representation of the sequence so far.

A model which could directly take into account the entire rest of the sequence when making a prediction could potentially perform better. It would be difficult to make such a model work with variable-length data however. Fortunately, it is possible to convert a family of proteins into a fixed-length number of sequences, by constructing a *sequence alignment*.

Recall the introduction on protein families in section 1.3.2. Through evolution, proteins become related in families. Consider training a model on a single protein family. Because the proteins are related, they have certain similarities. As it turns out, we may use these similarities to our advantage when training our model. A sequence alignment attempts to align the parts of the proteins in a family which appear similar, while placing gaps to accommodate for the required "stretching" of the proteins.

[1] It is possible to have sequence-to-sequence models which produce sequences of another length than the input, but this is more relevant to natural language processing and not as relevant to protein processing.

As a toy example, consider the top part of figure 4.3. The original sequence `ABCDEFGH` evolves to 3 other sequences, where one has the `B` changed to an `L`, another has the `E` removed and the last has a `K` inserted between the `F` and `G`. These changes are supposed to represent random mutations. One of the sequences then evolves further, resulting in a total of 6 different sequences.

An example of a sequence alignment of this toy protein family can be seen in the bottom part of figure 4.3. As can be seen, the sequence alignment attempts to line up the parts of the sequences that match the evolutionary history – that is, the insertion and removal of amino acids causes gaps to be inserted into the other sequences, to keep the similar parts of the proteins at the same position.

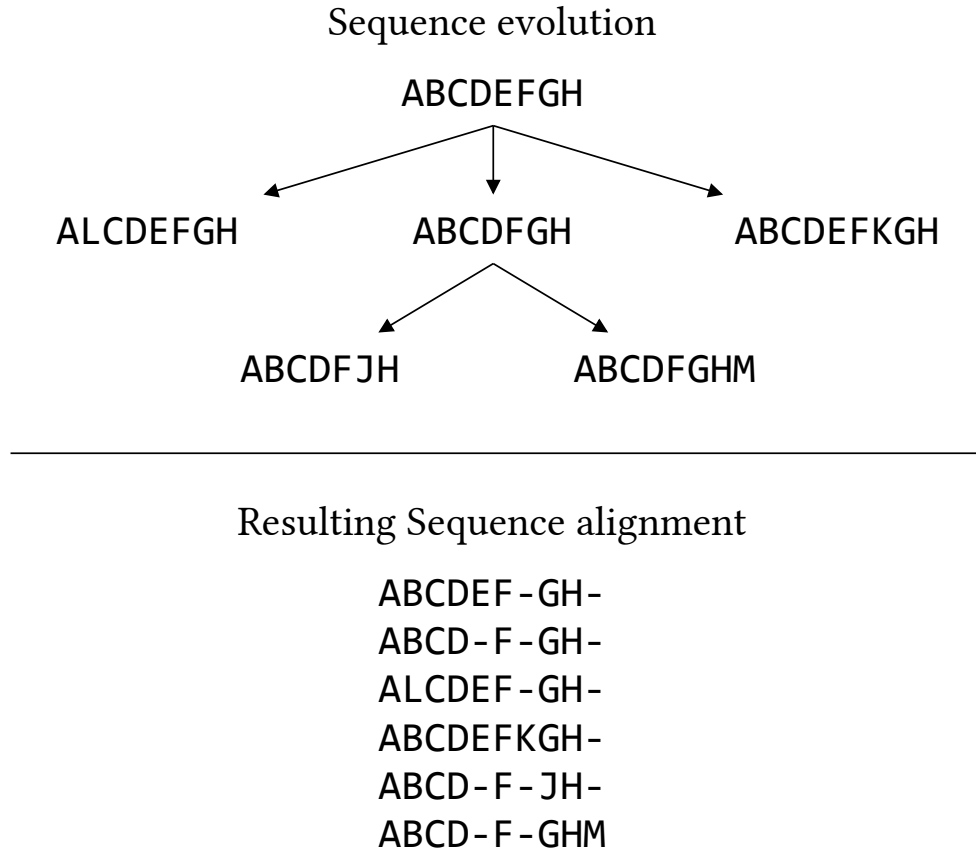


Figure 4.3: A toy example of a tiny sequence and its evolution and an example sequence alignment. The sequence used here is purely an instructive example and does not correspond to actual amino acids or any real protein.

Notice that the resulting sequence alignment causes all the sequences to be of the same length. Essentially, the sequence alignment provides absolute positions within the proteins, a luxury we did not have before. This is highly desirable, since it makes the choice of model much more flexible. We can now include models that only work on fixed-length data in our considerations.

For example, we can now apply densely connected linear layers to the sequences, since we now know the length of our data beforehand. This will prove to be very useful when we implement variational autoencoders in section 6.3.

However, sequence alignments are not perfect. There are significant disadvantages to using sequence alignments. First of all, the production of a sequence alignments requires the definition of a sort of scoring function, which basically optimizes the alignment based on how much an insertion of a gap or substitution costs. How exactly to decide on the specifics of the scoring function is not clear, making each alignment produced slightly different based on the parameters chosen. The toy example in figure

4.3 is simple and the resulting alignment is obvious, but given thousands of sequences, the problem becomes much more complicated. Also, you may not have the evolutionary tree which resulted in the protein family (like we do in the toy example), which makes the analysis of the proteins much more difficult. As if this was not enough, multiple sequence alignment is also NP-hard [18], which means that an alignment working on a large amount of sequences must rely on heuristics and sub-optimal solutions in order to be tractable.

To actually produce an alignment, many parameters must be decided upon. There are many different methods and heuristics to choose from and the resulting alignment could be different depending on these choices. This makes the training of models on alignments cumbersome – some alignments may be well-suited for training, while others may not, and it is difficult to tell the two apart.

Another problem with alignments is the way that they modify the underlying data. Fundamentally, protein data is variable-length. Changing this assumption involves the insertion of gaps. These gaps may change the proteins in ways that make it difficult for models to learn patterns that would otherwise have been easy if the gaps were not inserted.

Even worse, some alignments do not just insert gaps, they also cut away parts of the protein, or only preserve a certain section of the sequences. This is a much more destructive operation to the data, which may simply remove necessary sequence information.

5
Models

In the last two chapters we introduced probabilistic modelling and sequence learning. In this chapter we present in detail three concrete models used to learn representations from proteins, based on the this theory. In section 5.1 we present the UniRep model, a sequential model based on recurrent neural networks. In section 5.2 we present a Variational Autoencoder working on alignments of protein families. Finally, in section 5.3, we present the WaveNet model, an autoregressive model which uses convolutional layers to process a protein sequence.

5.1 UniRep: A Recurrent Global Model

In 2019, Alley et al. [2] proposed UniRep, a protein model that utilizes a recurrent neural network (RNN) to process proteins. Because it uses an RNN, it is capable of handling the variable lengths of proteins without needing a sequence alignment. This makes it able to work on global protein space, leading to what the authors of UniRep refers to as a "unified representation" – hence the name.

UniRep works by first embedding each amino acid input sequence to a high-dimensional vector. This sequence of vectors is then fed through a recurrent neural network, producing a sequence of hidden states. The hidden states are then transformed by a linear layer into probabilities for the amino acid following the position of the hidden state. This is illustrated in figure 5.1.

The canonical UniRep model presented in Alley et al. [2] uses an mLSTM (multiplicative variant of the common LSTM RNN) with a hidden state size of 1900, as the RNN inside the model. Additionally, it utilizes truncated backpropogation during training.

We have previously worked with UniRep and explored the significance of the above hyperparameters [19]. We found that one can achieve nearly equivalent performance with a medium-sized LSTM as with the large canonical mLSTM and that the choice of RNN (mLSTM, LSTM or GRU) and the hidden state size is less significant as long as the hidden state is sufficiently large. The choice of RNN and hidden state size can however have a major impact on the computational complexity of the model. We found that truncated backpropogation only made the model perform marginally worse. A diagram of the UniRep model can be seen in figure 5.1.

As can be seen in figure 5.1, UniRep does not produce a compact, fixed-size representation at any point in its training. This is because UniRep is a sequence-to-sequence model. As such, its representation must be captured externally, after the training is finished. There are many ways one could do this – in this work, we will use a mean over the RNN's hidden states, which is also what the original UniRep authors proposed.

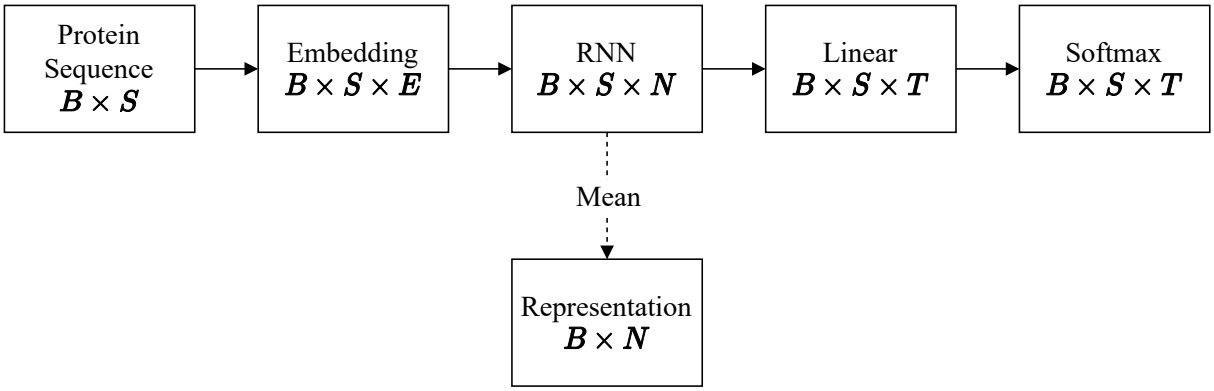


Figure 5.1: Architecture of the UniRep model. B is batch size, S is sequence length, E is the dimension of the embedding layer, N is the size of the RNN's hidden state and T is the number of tokens. UniRep's representation can be obtained by a mean over the RNN's output, but is not used directly in training.

5.2 Variational Autoencoder on Aligned Protein Families

As previously discussed in section 2.2.2, producing global representations, meant to be utilized for the entire space of all proteins, may be too optimistic and performance may suffer in the local protein family landscapes. A model constrained to capture representations of a single protein family could be more feasible, because the space of possibilities and variance in the sequences are vastly reduced. Training a model on such a smaller data-set is also simpler, requiring less memory and less time before convergence.

5.2.1 Model Architecture

In reality, protein sequence positions have a complex interrelated dependency: a single position or groups of positions may have a high determining factor for both specific functionality of the protein and other regions of the sequence. As briefly mentioned in chapter 2, one way to model this otherwise difficult process is to introduce a latent variable and transfer the dependency to that variable. This has the positive effect that it is much easier to model and compute, but also the negative effect that the model is simplified; accounting for all dependencies in a single multidimensional latent variable can be done, but is implausible that this can fully reflect the true complexity of the dependencies.

The evolutionary process of protein generation is modelled by the generative distribution $p(\boldsymbol{x}, \boldsymbol{z})$ over protein sequences $\boldsymbol{x}$ from the space of proteins, and a latent variable $\boldsymbol{z}$ that captures the underlying explanatory factors, as described above, and in section 3.2. This includes any dependency between sequence positions, which is important in the calculation of $p(\boldsymbol{x} \mid \boldsymbol{z})$. Specifically, for a protein sequence $\boldsymbol{x} = x_1, x_2, \ldots, x_n$ we have $p(\boldsymbol{x} \mid \boldsymbol{z}) = \prod_{i=1}^{n} p(x_i \mid \boldsymbol{z})$, as any dependency between positions have been taken into account by the conditional on $\boldsymbol{z}$. Conceptually, this can be thought of as a probabilistic dependency graph, where $\boldsymbol{x}$ is separated into its positional dimensions, as shown in figure 5.2.

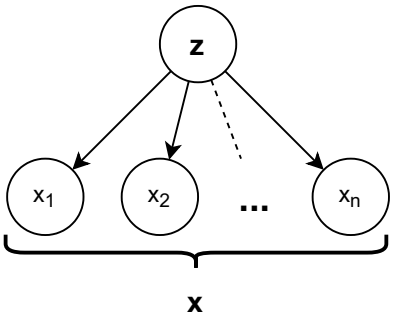


Figure 5.2: The positions $x_1, x_2, \ldots, x_n$ of the sequence $\boldsymbol{x}$ are conditionally independent given $\boldsymbol{z}$.

We model the generating process as $\boldsymbol{z}$ causing $\boldsymbol{x}$, such that $p(\boldsymbol{x}, \boldsymbol{z}) = p(\boldsymbol{x} \mid \boldsymbol{z})p(\boldsymbol{z})$. However, because the involved distributions are unknown, the marginal likelihood $p(\boldsymbol{x})$ in equation 3.1 is inferred, where the average over all latent variables $\boldsymbol{z}$ is considered. The encoder produces a likely latent variable $\boldsymbol{z}$ given the protein sequence $\boldsymbol{x}$ in question. As explained in section 3.2.3, training the VAE involves maximizing the ELBO, a lower bound on the log-probability of $\boldsymbol{x}$. The VAE is trained to maximize the ELBO; it follows that the ELBO becomes a better approximation of $\log p(\boldsymbol{x})$ as the model improves. Thus we can reasonably apply the ELBO as an approximation of $\log p(\boldsymbol{x})$.

The general VAE architecture overview can be seen in figure 5.3. The encoder network produces a distribution in latent space, from which latent variables can be sampled. The decoder then reconstructs the input from this latent space sampling. For our experimental settings, a more detailed version is provided in 6.2.

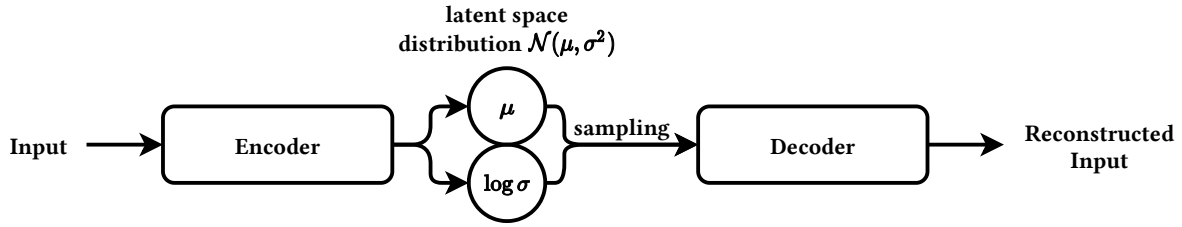


Figure 5.3: The general VAE architecture.

We modified this model, to utilize some of the ideas from Riesselman et al. [6], most importantly Bayesian weights in the decoder, but also weight sampling and sparse interactions.

Weighted samples We added weights on each sequence, in order to reduce human sampling bias and evolutionary bias. These weights influences how the model weighs each sequence in its optimization. Each sequence's weight is calculated as the reciprocal of the number of neighbours the sequence has, where "neighbour" means a sequence within 0.2 Hamming distance. This is explained in more detail in section 6.1.1.

Sparse interaction We also implemented a feature that encourages sparse interactions between weights and positions in the protein, as it has been observed that models which are constrained to pairwise interactions between protein sequence positions still perform well [20]. That is, the predicted residue at each position will be influenced by a sparse number of weights in the final linear decoder layer. Effectively this is done by introducing weights W_i for each position i of the protein sequence, and apply these to the output of the final linear layer of the decoder. The sparsity comes from the way W_i are chosen, which involves tiling (repeating) another parameter matrix S and gate those depending

on the position. The details are spared here, but can be found in Riesselman et al. [6, methods section] and in the implementation of the VAE architecture (see appendix A.1). Effectively, this also increases the capacity of the decoder significantly.

Distributed decoder weights A significant deviation from the standard VAE is the addition of Bayesian weights on the linear layers of the decoder. Rather than the decoder using standard deterministic linear layers, it uses probabilistic linear layers. The model samples a weight and bias for the layers every time they are used[1], instead of having a single deterministic weight and bias. This means that the model optimizes not just a singular linear layer – instead, it optimizes a distribution of linear layers.

This distribution of layers is particularly useful at evaluation time. By running the model multiple times on the evaluation data, the model samples different linear layers every time. Effectively, this constructs an ensemble of models of an unlimited size, because the distribution is continuous. Ensembles of models are known to, in general, perform better than single models in isolation. The intuition behind this, is that different models can compensate for each others' weaknesses and shortcomings, thus providing a model that performs better than the mean of the individual models' performances.

5.3 WaveNet: A Convolutional Model

The *WaveNet* model architechture is an *autoregressive* modelling approach similar to UniRep in the sense that it is adaptive to unaligned protein sequences [21, 3]. Unlike the variational autoencoder described in section 3.2 and 6.3, the probabilistic model representing the dependencies between amino acid residues is not incorporating a latent variable. Rather, for a protein sequence $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$, the probability of any residue $\boldsymbol{x}_i$ is conditioned on the previous residues on the protein chain:

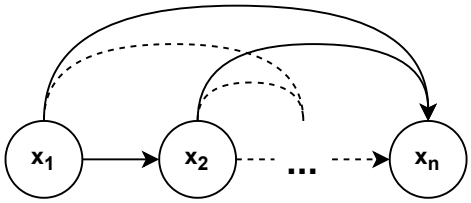


Figure 5.4: The autoregressive model graph. At each position i, the probability of x_i is conditioned on all previous positions $x_1, \ldots, x_{i-1}$.

This is the autoregressive property of the model: each residue is conditioned on the previous residues in the chain. It follows that the probability of the entire protein sequence $p(\boldsymbol{x})$ is expressible as a product of the conditioned residue probabilities:

$$p(\boldsymbol{x}) = p(\boldsymbol{x}_1) \prod_{i=2}^{n} p(\boldsymbol{x}_i \mid \boldsymbol{x}_{i-1}, \ldots, \boldsymbol{x}_1)$$

With this setting, the task becomes to model each residue probability, with the restriction that only the previous residues can affect the modelling process. Unlike UniRep, the WaveNet model is not based on recurrent neural networks, but achieves its context by gradually dilating convolution layers, grouped into blocks. A model architecture overview can be seen in figure 5.5.

[1] For performance reasons, the model actually samples a single weight and bias for an entire batch.

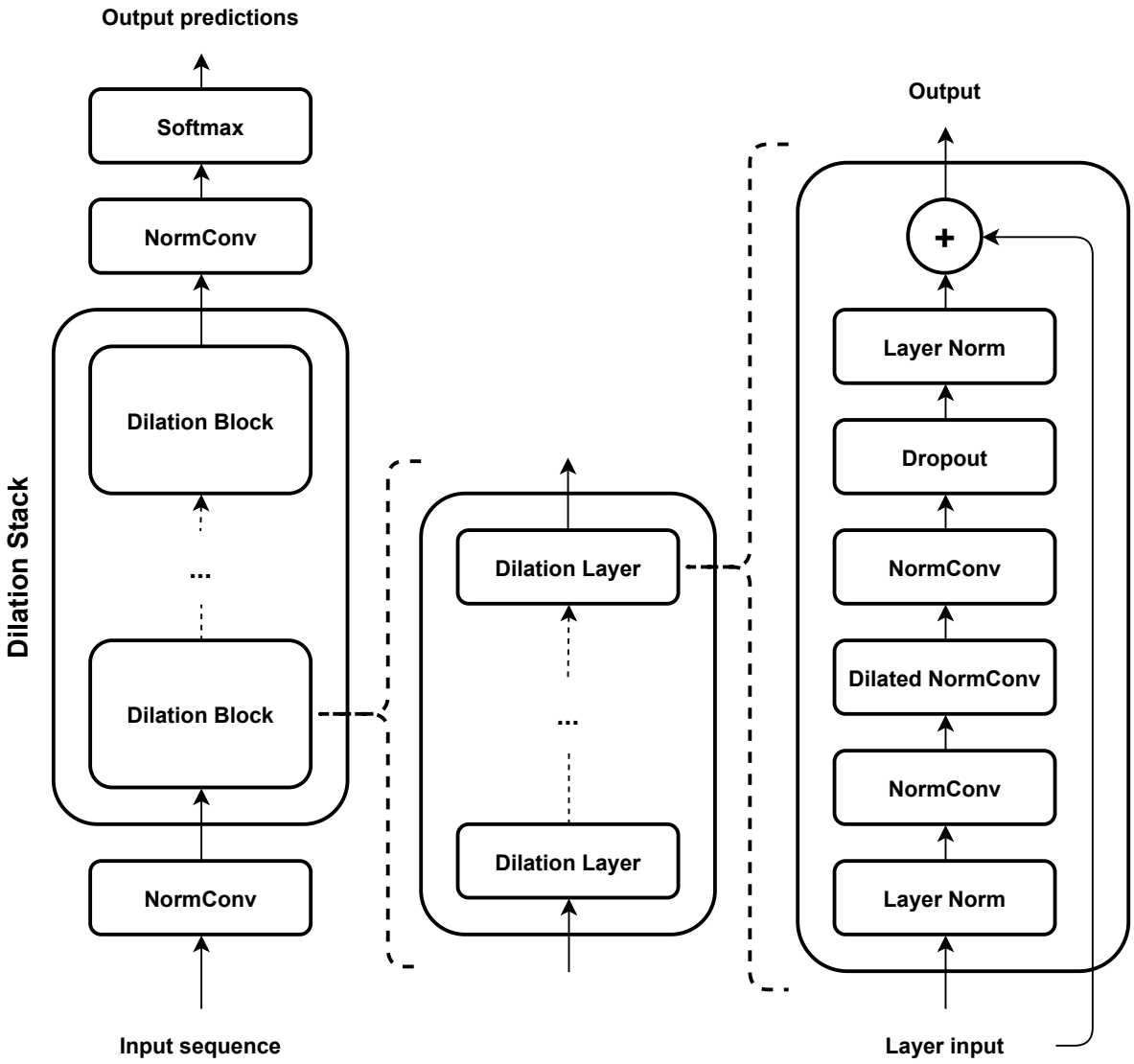


Figure 5.5: A WaveNet architecture overview. Model input is passed through a dilation stack (*left*) consisting of dilation blocks. Each dilation block (*middle*) consists of dilation layers. Each dilation layer (*right*) passes its input through a sequence of internal layers.

The model consists of some core components: the dilation stack, the dilation block and the dilation layer. The dilation stack consists of dilation blocks, which again consists of dilation layers. They all depend on a slightly expanded convolutional layer which we denote as *NormConv*, a contraction of *normalized* and *convolution*. The standard convolutional layer is parameterized by, among other things, the number of input and output channels, kernel size and a dilation, as discussed in section 5.3.1. The NormConv is a standard convolutional layer, with weight normalization applied[2] [22] and an additional scale, bias and potential activation. It returns an output $\boldsymbol{o}$ from an input sample $\boldsymbol{x}$ as follows:

$$\boldsymbol{o} = \sigma(\boldsymbol{\gamma} * \texttt{convolutional}(\boldsymbol{x}) + \boldsymbol{\beta}),$$

where σ is some activation function, `convolution` is a standard convolutional layer applied to the input $\boldsymbol{x}$, and $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ are learnable parameters that element-wise scale and shift the outcome of the convolution.

[2]Weight normalization eases the optimization of a weight by breaking it up into a scale and direction component.

The *dilation layer* consists of several internal cells that feed into each other. First, in the *Layer Norm* cell, the input is normalized w.r.t. its empirical mean and variance, and rescaled, as described by Ba et al. [23]. Then three Norm Conv layers are applied, where the middle layer is dilated. The dilation depends on the position of the dilation layer in the *dilation block*. For a dilation block of n dilation layers, the dilations are $1, 2, 4, \ldots, 2^{n-1}$, so the ith dilation layer will have dilation 2^{i-1}. All dilated NormConv layers have kernel size 2, which in combination with the dilation give rise to the expanding effect of the dilation block, as seen in figure 5.6. In figure 5.9, three convolution filters that differ in dilation are shown.

After the NormConv layers of the dilation layer, dropout is applied before another layer normalization. The original input to the dilation layer is added to the output of this procedure via a skip connection. This is the output of the dilation layer. The dilation block is many such layers that feed into each other sequentially, and the *dilation stack* is simply many such dilation blocks in sequence. The final output of the dilation stack is passed through a NormConv layer that reduces the number of output channels to the number of prediction categories, in our case, the number of candidate amino acids.

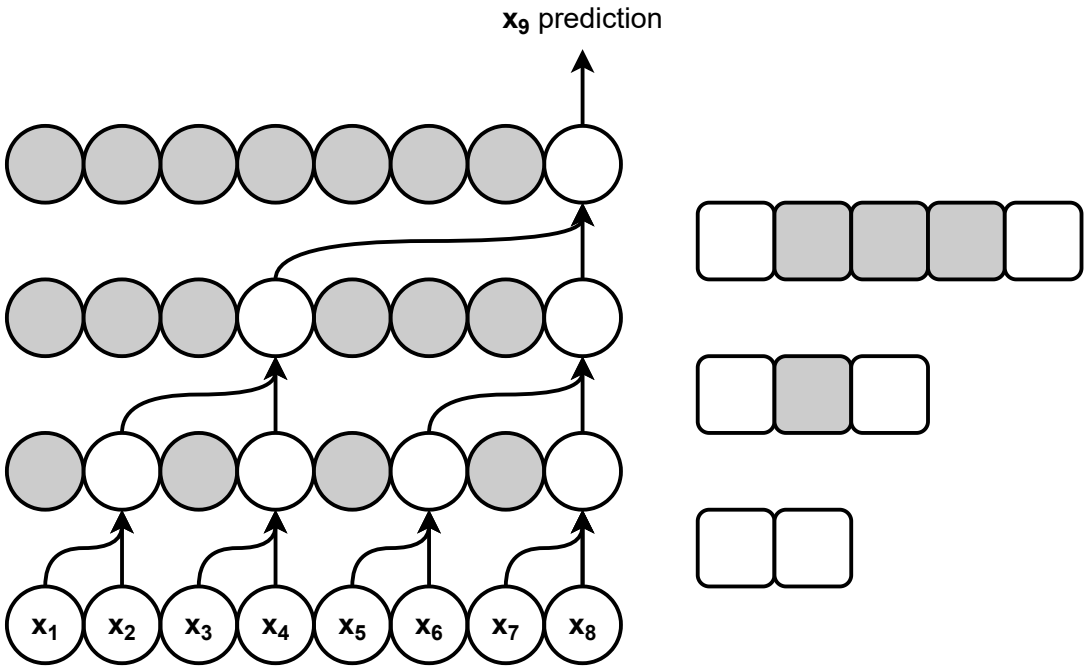


Figure 5.6: A dilation block with 3 dilation layers with dilation 1, 2 and 4 respectively, in that order, and kernel size 2. The corresponding convolution kernels are shown to the right. To the left, the connected nodes show the flow of information from the input sequence, through the dilation layers, to the prediction of the ensuing residual x_9.

Like the VAE and the UniRep model, WaveNet produces a softmax output over each residue position in the input sequence. In order to to maintain the ordered dependency between residues, any ensuing residues are masked out at each prediction step. That is, for a prediction of the residue at position i, the model only observes outcomes from the positions $1, \ldots, i-1$. This is effectively obtained by shifting the convolution an appropriate number of positions depending on the dilation, and left-padding the sequences.

5.3.1 The Causal, Dilated Convolution

A convolutional layer passes its input sequence through a *convolution matrix*, or simply a *kernel*, and produces output features in size depending on the kernel. The kernel is basically the learnable parameters of the convolution. In addition, the convolution has an additive bias parameter. The convolution

operation itself is a weighted sum over the input sequence positions under the kernel. A *causal* convolution is a convolution which at any position i of the input sequence $x_1, \ldots, x_n$, only considers the k previous positions, where k is the size of the kernel. That is, at position i, the convolution operation is $b + \sum_{m=1}^{k} f_m \cdot x_{i-k+m}$, where f_m is the mth element of the kernel and b is a bias parameter. See figure 5.7 for an example.

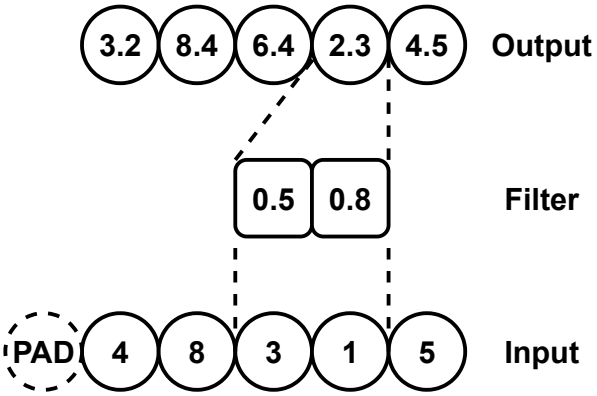


Figure 5.7: Example of a one-dimensional causal convolution using a kernel of size 2 and no bias. The input sequence is right-padded to maintain the same output length as the input. The causal relation is from left to right. Here, $f_1 = 0.5$ and $f_2 = 0.8$. At the shown output node, the convolution calculation is $0.5 \cdot 3 + 0.8 \cdot 1 = 2.3$.

In addition, a convolution can be *dilated*, meaning that the kernel is not applied to adjacent positions in the input sequence, but separated by a number of intermediate positions. An example operation similar to figure 5.7 is shown in figure 5.8. Three different lengths of dilations are shown in figure 5.9.

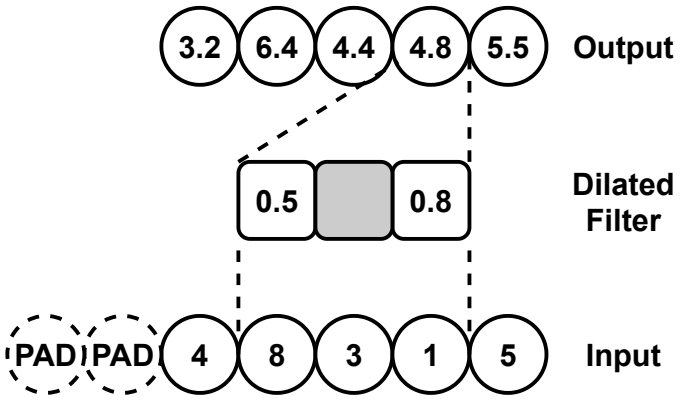


Figure 5.8: A one-dimensional *dilated* causal convolution using a kernel of size 2, dilation 2 and no bias. The input sequence is right-padded to maintain the same output length as the input. The causal relation is from left to right. At the shown output node, the convolution calculation is $0.5 \cdot 8 + 0 \cdot 3 + 0.8 \cdot 1 = 4.8$.

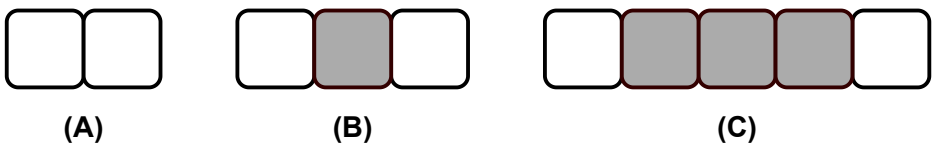


Figure 5.9: Three one-dimensional convolution kernels of size 2. White boxes denote positions that are weighted in the convolution. Shaded boxes are skipped/masked. (A) dilation of 1. (B) dilation of 2. (C) dilation of 4.

6
Experiments

In chapter 1 we asked how well good protein representations are achieved by machine learning models, how they can be implemented, what the advantages and disadvantages of global versus local representations are, and how the properties of the representation affect the performance on downstream tasks.

In order to find answers, we have carried out a number of experiments, applied to each model. There are broadly two experimental settings: (1) mutation effect prediction and (2) tasks assessing protein embeddings, otherwise known as TAPE [24]. The first setting evaluates the mutation effect prediction of the models, based on the learned distribution over proteins. This is of interest, as mutation effect prediction is tightly connected to the underlying evolutionary generating process, and what properties the mutated protein possesses. The second setting is a benchmark assessment of the performance of the protein representations on downstream tasks.

In this chapter we report how we did it and what the results were. In section 6.1 the experimental settings are accounted for, including commonalities between the experiments such as what language framework we used to implement the models, and hardware used for training and testing.

There are three conceptually different model architectures that we have tested. Each model architecture is described in chapter 5 and has its own section here with implementation details, results and discussion.

6.1 Experimental Setup

For all of our experiments, we utilized PyTorch [25], a modern machine learning framework for Python. PyTorch provides, among other things, automatic gradient computation, optimized modules of frequently used modelling architectures and low-level GPU integration for efficient training. These are fundamental dependencies in our modelling methods that has allowed a much faster development process. We preferred PyTorch over its alternatives due to its flexible yet concise usage.

In terms of hardware, we used a cluster of NVIDIA TITAN RTX graphics cards to perform both training and testing. In most settings, a single GPU is used, which contains 24 GB of memory. Pretraining of the WaveNet model was done on 4 such GPUs in parallel, in order to train faster.

6.1.1 Mutation Effect Prediction

The models have in common that they all produce a distribution over proteins $p(\boldsymbol{x})$, each in their own way. The interpretation of $p(\boldsymbol{x})$ is that it is the probability of the protein sequence $\boldsymbol{x}$ being generated by the evolutionary process. This probability allows for an informed comparison of protein in terms of how likely they are to exist; if the probability of one protein is higher than another, then the first should be expected to have higher measures of properties related to persistence, such as stability and other functional constraints. Specifically, it allows for the comparison of some protein with its mutations; if a mutation of a protein increases its probability, then this increase is likely to be associated with an increase in favorable properties of the protein due to the mutation. This is called *mutation effect prediction*, and it has been shown that the metric

$$\log \frac{p(\boldsymbol{x}_{\text{wild-type}})}{p(\boldsymbol{x}_{\text{mutant}})} = \log p(\boldsymbol{x}_{\text{wild-type}}) - \log p(\boldsymbol{x}_{\text{mutant}}) \tag{6.1}$$

correlates with many different effect predictions of proteins [20]. Here, $\boldsymbol{x}_{\text{wild-type}}$ is a wild-type protein, i.e. as it occurs in nature, and $\boldsymbol{x}_{\text{mutant}}$ is a mutated version of the wild-type protein. This ratio can be produced by the trained models, and is what we use to calculate correlation with experimental data. The intuition behind the metric is that a higher ratio (above 0) means that the mutation was beneficial for the protein, as it is now more likely than the wild-type, while a negative ratio means that the mutation is likely to be detrimental. Predicting the effect of protein mutations is a relevant task, because examination of mutations is how protein engineering is often performed (such as optimizing a known protein by examining its mutations).

The final outcome in this experimental setting is thus a correlation coefficient $\rho \in [-1, 1]$, in our case the Spearman's ρ coefficient, which in short is a nonlinear dependency measure between two datasets. In short, how well can you fit a monotonically increasing function to describe the datasets; this is more general correlation than Pearson correlation, which fits a linear function. The closer the correlation approaches one, the more the model predictions comply with experiments, which is desirable.

Data

The data used for training the various models presented in this thesis is invariably derived from the Universal Protein Resource database (UniProt) [1]; protein families and alignments are curated as specified in Riesselman et al. [6]: the UniRef100 protein dataset is iteratively traversed with the JACKHMMER protein analysis search tool [26] for a given query protein.

A protein family is a set of proteins that share functionality or structure across organisms because of a shared evolutionary history. In the complex process of protein generation it is, simply put, genes that determine the protein outcome. If proteins share a gene ancestor, we say that the proteins are *homologous* and assumes this is the case when the proteins share similar structure and function. Families induce a hierarchical structure onto protein space, collecting proteins into groups and subgroups that share a common protein ancestor. Determining this hierarchy is nontrivial and usually require probabilistic models to uncover. Likely protein families can be decided using *profile hidden markov models* (HMMs). These models captures changes between a set of similar proteins, i.e. deletions and insertions of amino acids by casting the changes into transitions and states in a Markov model with associated probabilities for each transition [27]. This model is then called the profile and can be used to iteratively search a given protein database for similar proteins. These can then be included in the family if their similarity is above some threshold. A new profile can then be modelled based on the new set of proteins. This can be repeated a fixed number of iterations, or until no new proteins are added to the collection (i.e. the search has converged on a set of proteins). This procedure is what the JACKHMMER search tool accomplishes.

The protein family datasets we have used are the sequence alignments, and their associated experimental data, made available by Riesselman et al. [6]. We have chosen a subset of 5 families to assert the diversity of the models:

BLAT ECOLX: E.coli β-lactamase.

CALM1 HUMAN: Human Calmodulin-1.

GAL4: Yeast regulatory protein GAL4.

HSP82: Yeast ATP-dependent molecular chaperone HSP82.

RASH HUMAN: Human GTPase HRas.

The protein families are chosen on the basis of the expected performance on the sets, based on the results achieved in Riesselman et al. [6]: We have tried to select families the captures variance in the performance of the models, i.e. we want families that are both easy and difficult for the models to predict, and in between. Table 6.1 below displays some metadata of the datasets.

	BLAT ECOLX	CALM1 HUMAN	GAL4	HSP82	RASH HUMAN
Family Size	8403	36224	22985	23447	84762
Effective Size	2225	4754	3542	1321	5954
# Mutations	4788	1868	1104	4313	3078
Aligned Size	263	149	75	240	189
DeepSequence ρ	0.78	0.23	0.45	0.53	0.47

Table 6.1: Datasets used for for each protein family. The last row shows the DeepSequence prediction correlations, as reported in Riesselman et al. [6].

Sample weights A protein family dataset consists of protein sequences that share a common ancestor. However, such datasets are not guaranteed to be evenly distributed among sequences from descendants. For this reason, some protein sequences might be overly represented, while others might not. In order to balance the dataset, protein sequences have be reweighted such that similar samples are grouped together and weight in inverse proportion to the size of the group. For a specific protein family with n protein sequences $p_1, p_2, \ldots, p_n$, this leads to an effective number of samples, $n_{\texttt{eff}} = \sum_{i=1}^{n} \pi_i$, where

$$\pi_i = \left(\sum_{j=1}^{n} [\ p_i \text{ is similar to } p_j\ ] \right)^{-1}.$$

Brackets denote the indicator function of the statement inside the brackets. We follow the steps of Riesselman et al. [6] and Hopf et al. [20] and measure similarity by normalized Hamming distance, with a threshold of 0.2, meaning that if two sequences share at least 80% of their positions, they are similar:

$$[\ p_i \text{ is similar to } p_j\ ] = [\ \texttt{hamming}(p_i, p_j) \leq 0.2\ ].$$

The effective dataset sizes are shown in table 6.1 in the effective size row.

Experimental data used to correlate our predictions are not necessarily the same metric across protein families, but each experiment measure levels of various fitness-related properties for a single mutation. E.g. the experimental results for β-lactamase is derived from Stiffler et al. [28] and is, to our best understanding as computer scientists, a study of fitness effects of single amino acid mutations by observing growth in E. coli cells under 2500 μg/ml ampicillin.

In the mutation effect prediction setting, we train each model on 80% of the sequences, using the rest for validation, usually with 128 batch size (in some cases a smaller batch size is used due to memory constraints). Training continues until we see no improvement on the validation score, and keep the model that performed best on the validation set.

6.1.2 TAPE

Unlike the variational autoencoder, the UniRep and WaveNet models do not include a compact representation in their training, since they are inherently sequence-to-sequence models. As discussed in section 4.2, a compact representation can however be extracted from a sequence-to-sequence model, after training has been completed. Judging the usefulness of these representations cannot be done by the training or validation procedure, since it is not part of the model's training process. Hence, the representations must be evaluated externally.

In order to assess the quality of these representations, we turn to a standard set of downstream tasks associated with protein representation learning – these are the *Tasks Assessing Protein Embeddings*, oth-

erwise known as TAPE [29]. TAPE includes a diverse set of protein-related tasks that are considered to be useful for a protein representation. Using TAPE, we can quantitatively compare the representations of UniRep and WaveNet to each other, as well as to the benchmarks already provided by TAPE on other models. Note that we cannot evaluate the variational autoencoder on TAPE, since TAPE requires the processing of arbitrary-length sequences.

Because the TAPE tasks are so diverse, they can show if a model's representation is generally useful, or reveal the strengths and weaknesses of different models, which may inform an application on which model is the most suitable to use for a given problem.

Aside from the definition of the tasks themselves, TAPE also provides a general training framework[1] which makes it easy to adapt an existing model to the TAPE task evaluation process. This framework also provides general "top-models", which are used in the training for each task. The top models learn on the outputs of the underlying model, converting the representations produced into predictions for the specific task. By using the same top-model on all the different models evaluated, TAPE provides a fair "competition" between the underlying models.

Concretely, TAPE consists of the following 5 tasks:

Secondary Structure Prediction: This task involves classification of secondary structure (see section 1.3.1) for each of the amino acids of the input sequence. The resulting score is an accuracy on the three classes "Helix", "Strand" and "Other". Secondary structure prediction is a useful first step to figuring out the function of a protein. Data is provided by Berman et al. [30], Moult et al. [31], and Klausen et al. [32].

Contact Prediction: In this task, the model must predict whether a pair of amino acids in a protein are in contact (< 8Å distance) or not. The resulting score is hence an accuracy on two classes. This information is highly useful when analyzing the dependencies between the amino acids, or when trying to figure out the tertiary structure. Data is provided by Fox et al. [33], Moult et al. [31], AlQuraishi [34], and Berman et al. [30].

Remote Homology Detection: This task asks the model to predict the protein fold (a superfamily) of an input sequence. The score is an accuracy on the 1195 possible folds. Data is provided by Fox et al. [33].

Fluorescence: In this task, the model must predict the fluorescence of the input protein. This can be useful when trying to optimize a protein's function by introducing mutations. This is a regression task, hence the score is given as a Spearman correlation. Data is provided by [35].

Stability: The final task asks the model to predict the stability of the input protein. Stability is a highly desirable property, as a protein must be stable for long enough for its purpose to be fulfilled. Predicting stability is therefore very useful. Like the previous task, this is a regression problem and the score is thus given as a Spearman correlation. Data is provided by [36].

Of the 5 tasks above, we evaluate the global models on all but the contact prediction task, due to computational complexity. The contact prediction tasks involves a prediction for every pair of amino acid in the input sequence, resulting in a square factor on memory usage. This has prevented us from being able to evaluate our models on this task. We therefore only use the four remaining tasks.

[1] Note that we use the updated PyTorch-compatible version of this framework, found here: `https://github.com/songlab-cal/tape`

6.2 UniRep

As described in section 5.1, the choice of RNN and hidden state size of the UniRep model is less significant within a certain range of choices. In accordance with this and the practical limitations of our hardware, we decided not to use the canonical UniRep model (a 1900-dimensional mLSTM) and instead opted to use an LSTM with a 512 hidden state size, without truncated backpropogation. Our previous work on UniRep suggests that the performance of this variant should not be significantly worse than the canonical UniRep model [19]. At the same time, our model is smaller and can make use of PyTorch's inbuilt LSTM module, both of which contribute to a much more computationally cheap model.

We train UniRep in two ways; with and without global pretraining. Pretraining gives the model a chance to learn general protein properties from the global protein space. When using pretraining, we first train an UniRep model on the UniRef50 dataset of sequences, which contains roughly 39 million protein sequences at the time of writing. We use 1% of the UniRef data as validation.

When evaluating the mutation effect prediction task, we use three configurations for each protein family:

Finetuned: Trained on the protein family.

Pretrained: Pretrained on UniRef50.

Pretrained + Finetuned: First pretrained on UniRef50, then trained on the protein family.

In addition to the above configurations, we also take ensembles over the models using finetuning.

When evaluating the TAPE tasks, we also use three configurations for each task.

Finetuned: Trained the entire model on the TAPE task.

Pretrained: Pretrained on UniRef50, then only training the top model in TAPE.

Pretrained + Finetuned: First pretrained on UniRef50, then trained the entire model on the TAPE task.

We must extract UniRep's representation in order to evaluate on TAPE. For a given input protein, we construct UniRep's representation by taking the mean of the hidden states produced by the LSTM. It is worth noting that this is *not* the same representation TAPE uses for their UniRep model, which they display scores of in their leaderboards. TAPE uses a special attention-weighted mean of the UniRep hidden states instead.

6.2.1 Results

Mutation Effect Prediction

We evaluate UniRep on the mutation effect prediction task by feeding the sequences through UniRep's entire architecture, as illustrated in figure 5.1. This gives us a log-probability for each amino acid, given the sequence up to that point. We obtain the log-probability for the entire sequence by summing the log-probabilities of the amino acids. While training for this task, we use 20% of the data as validation.

Table 6.2 shows UniRep's performance on the mutation effect prediction task.

We also tried visualizing UniRep's representation of the β-lactamase phylums. Since the representations are 512-dimensional, we need to perform dimensionality reduction in order to visualize them in 2D. In this case, we use the t-SNE [37] dimensionality reduction. The result can be seen in figure 6.1.

Configuration	BLAT ECOLX	CALM1 HUMAN	GAL4	HSP82	RASH HUMAN
Finetuned	0.44	0.26	0.46	0.38	0.28
Pretrained	0.02	0.21	0.16	0.04	0.27
Pre. + Fine.	0.44	0.29	0.37	0.38	0.30
Finetuned (E)	**0.50**	0.30	**0.54**	**0.45**	**0.35**
Pre. + Fine. (E)	**0.50**	**0.33**	0.49	0.42	0.34

Table 6.2: Mutation effect prediction correlation for the UniRep model, with and without pretraining. Each measure is an average over 5 runs, except Pretrained, which is only done once, with the same resulting model used for all the datasets. The bottom two rows marked with (E) are not an average over the 5 runs, but instead an ensemble over the 5 models.

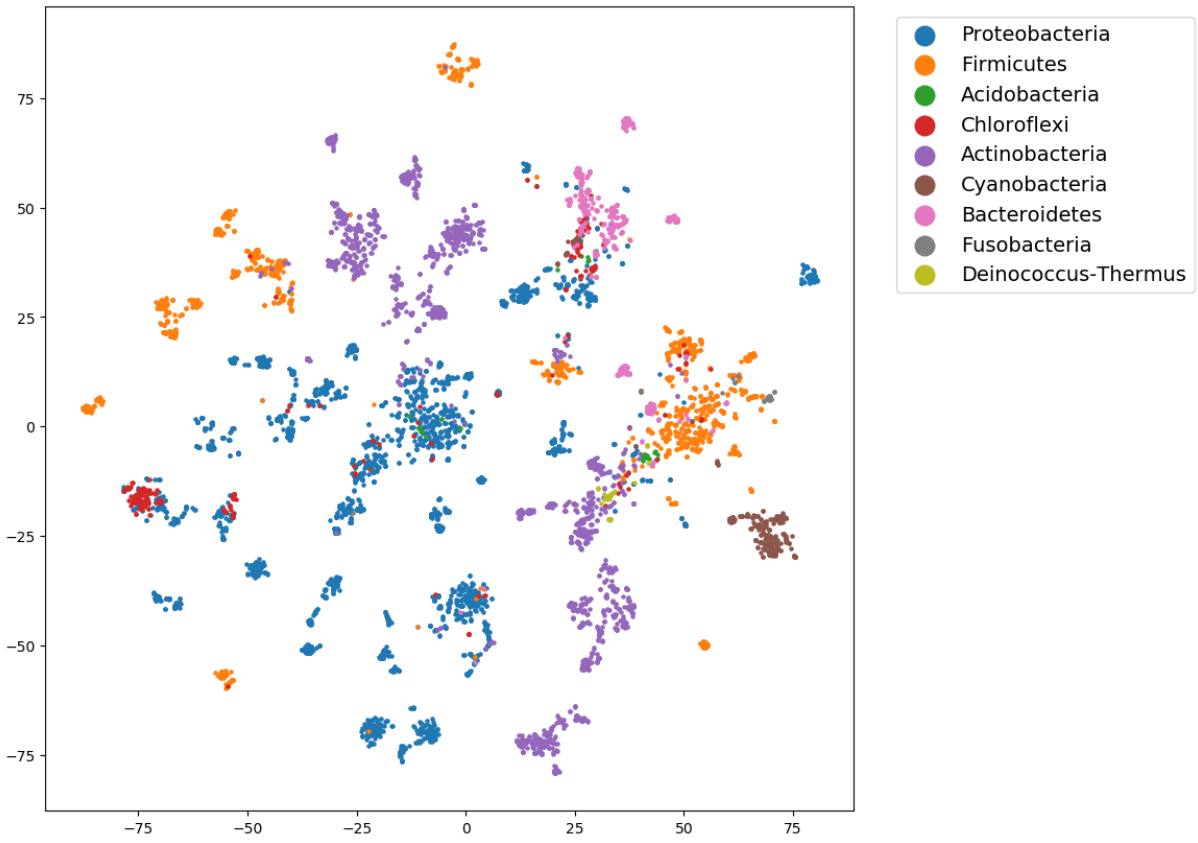


Figure 6.1: A t-SNE visualization of UniRep's representation on selected phylums of β-lactamase.

Finally, in order to assess the significance of weighted sampling, we applied a weighted UniRep ensemble over β-lactamase, using alignment weights during training. The resulting correlation with experiments jumped to 0.594 under these settings.

TAPE

We train using TAPE's own training framework and TAPE's own validation sets. We then evaluate on TAPE's testing sets. The resulting scores can be seen in table 6.3.

Configuration	Secondary Structure	Remote Homology	Fluorescence	Stability
Finetuned	0.67	0.07	0.34	0.70
Pretrained	**0.69**	**0.18**	0.45	0.66
Pre. + Fine.	0.68	0.12	**0.67**	**0.74**

Table 6.3: UniRep's performance on the TAPE tasks.

6.2.2 Discussion

Mutation Effect Prediction

The effect of pretraining UniRep seems to depend heavily on the specific protein family. Pretraining alone is insufficient to achieve good performance on all the datasets. Pretraining alone fails miraculously on the BLAT ECOLX (0.02) and HSP82 (0.04) datasets – we are unsure why pretraining is especially insufficient for these datasets. We hypothesize that the UniRef50 dataset used for pretraining somehow does not cover these families well or maybe it is covered but the rest of the data is counterproductive to learning on these families.

However, we are not surprised by the fact that pretraining alone does not lead to good performance on this task. Pretraining on the global scale may lead to the model learning only coarse features, which do not translate well onto the local protein family scale. This is supported by how dramatically the performance increases when finetuning on the individual families.

Using both pretraining and finetuning on the mutation effect prediction task seems to have inconsistent performance in comparison to just finetuning. Even when pretraining and finetuning leads to an increased performance, the difference is insignificant. In contrast, pretraining and finetuning seems to have a significantly bad effect on performance on the GAL4 family, in comparison to just finetuning (from 0.46 to 0.37). This is remedied by the ensemble however. The ensembles perform better than any of the 5 models alone. This is consistent with our expectation that ensemble models perform better in general.

Overall, the finetuned UniRep achieves a performance on the mutation effect prediction task comparable to the reference correlation (table 6.1) on 3 out of 5 datasets, while falling short on BLAT ECOLX and RASH HUMAN. It performs better than the reference on the CALM1 HUMAN and GAL4 datasets, which we find surprising since we did not expect UniRep to be able to capture the local landscape very well. Still, UniRep seems unable to achieve truly high (>0.6) correlation coefficients.

The t-SNE visualization of the UniRep representations in figure 6.1 shows a clear separation of the phylums into clusters, with only few overlaps. We found this result somewhat surprising, as we did not believe the UniRep representation to be so nuanced, considering that the mean over the hidden states of the LSTM is a somewhat destructive operation. Using a linear dimensionality reduction (like principal components analysis) does not result in neatly clustered points. The t-SNE is a non-linear dimensionality reduction, and so the non-linear part of the transformation seems significant. This suggests that while the representations indeed carry the potential to distinguish classes of proteins, extra (non-linear) capacity is needed to achieve this. If these representations were to be used in practical settings, it might be advisable to use sufficiently powerful models on top, as one would otherwise risk that the representations are not "fully understood" by the model. This also raises the question of whether such non-linear transformations could be integrated into the representations themselves, instead of being an external requirement of the model using the representation. This is in correspondence with the primitive construction of the UniRep representation – it is, after all, simply a mean over the LSTM's hidden states. This construction is a destructive operation and does not inherently have any properties that would make it easily linearly separable.

TAPE

On the TAPE tasks, pretraining seems beneficial, leading to significant performance boosts in the remote homology task (from 0.07 to 0.12) and the fluorescence task (from 0.34 to 0.67). For the remote homology task especially, this fits with expectations, as a global perspective may help in dividing proteins among folds, which are essentially a larger scale protein family. The pretrained and finetuned model achieves a performance on the stability task (0.74) which is slightly higher than the state-of-the-art models on TAPE's leaderboard[2] (0.73). However, the performance on the other tasks are not as impressive.

Surprisingly, finetuning after pretraining seems to lead to significantly worse performance on the remote homology task (from 0.18 to 0.12) – it may be that the model is more prone to overfitting when given the opportunity to train the top model as well as the inner UniRep model. Across tasks, global pretraining provided by the UniRef50 dataset seems beneficial for tasks like TAPE, which is not too surprising as the tasks use data across protein families. UniRep's performance can be significantly better on the TAPE tasks using a smarter representation, as evidenced by TAPE's own UniRep implementation which uses an attention-weighted mean as its representation. This suggests that the mean representation we use is not ideal. The other change that could play a role is the down-scaling of the representation size. We do not know how these two interrelate with respect to the observed performance. This could be investigated by simply either changing how the representation is performed, or changing the size of the representation.

6.3 Variational Autoencoder

We trained the VAE on the five protein families, varying whether the models apply MAP weight estimates or full Bayesian weight estimates. In addition we also test with- and without sparse interaction. Finally, we noticed that it is significant whether samples are batch sampled according to their weights and evenly weighted in their contribution to the loss function, or whether samples are batched uniformly at random without replacement, and then scaled in the loss function according to their weights. This difference is explicit in our results as well. We briefly explain the difference here, for a protein family $\boldsymbol{P} = p_1, \ldots, p_n$ of n samples, with weights $\boldsymbol{\pi} = \pi_1, \ldots, \pi_n$, where $\sum_{i=1}^{n} \pi_i = 1$. With random weighted sampling (rws), the batch and loss calculation is as follows:

1. Sample a set $\boldsymbol{T}$ of n training candidates from $\boldsymbol{P}$ according to $\boldsymbol{\pi}$, with replacement.
2. Train the model f on $\boldsymbol{T}$ using mini-batches. For a batch $\boldsymbol{B} = b_1, \ldots, b_m$, the loss is the mean of the individual batch sample losses, i.e. for a loss function $\mathcal{L}$ we have a batch loss of

$$\frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(f(b_i), b_i).$$

Without random weighted sampling, the batch and loss calculation is slightly different:

1. Let $\boldsymbol{T} = \boldsymbol{P}$.
2. Train the model f on $\boldsymbol{T}$ using mini-batches. For a batch $\boldsymbol{B} = b_1, \ldots, b_m$, the loss is the weighted mean of the individual batch sample losses according to their weights in $\boldsymbol{\pi}$, i.e. for a loss function $\mathcal{L}$ we have a batch loss of

$$\sum_{i=1}^{m} \pi_{b_i} \mathcal{L}(f(b_i), b_i).$$

It may seem that these two procedures are equivalent, but in practical settings the two yield different performance results, likely because of the variation on weight updates.

[2] See `https://github.com/songlab-cal/tape`

The training process uses both the encoder and the decoder. The encoder compresses the sequence from its original size (that is, a flattened one-hot encoding) into two 1500-dimensional layers, before reaching the 30-dimensional latent representation distribution, represented by two 30-dimensional linear layers, respectively denoting the mean and log-variance of the latent representation. Samples are drawn from the representation, which are sent into the decoder. The decoder has a 100-dimensional layer, followed by a 2000-dimensional layer, before reaching the original sequence size again. The architecture is illustrated in figure 6.2.

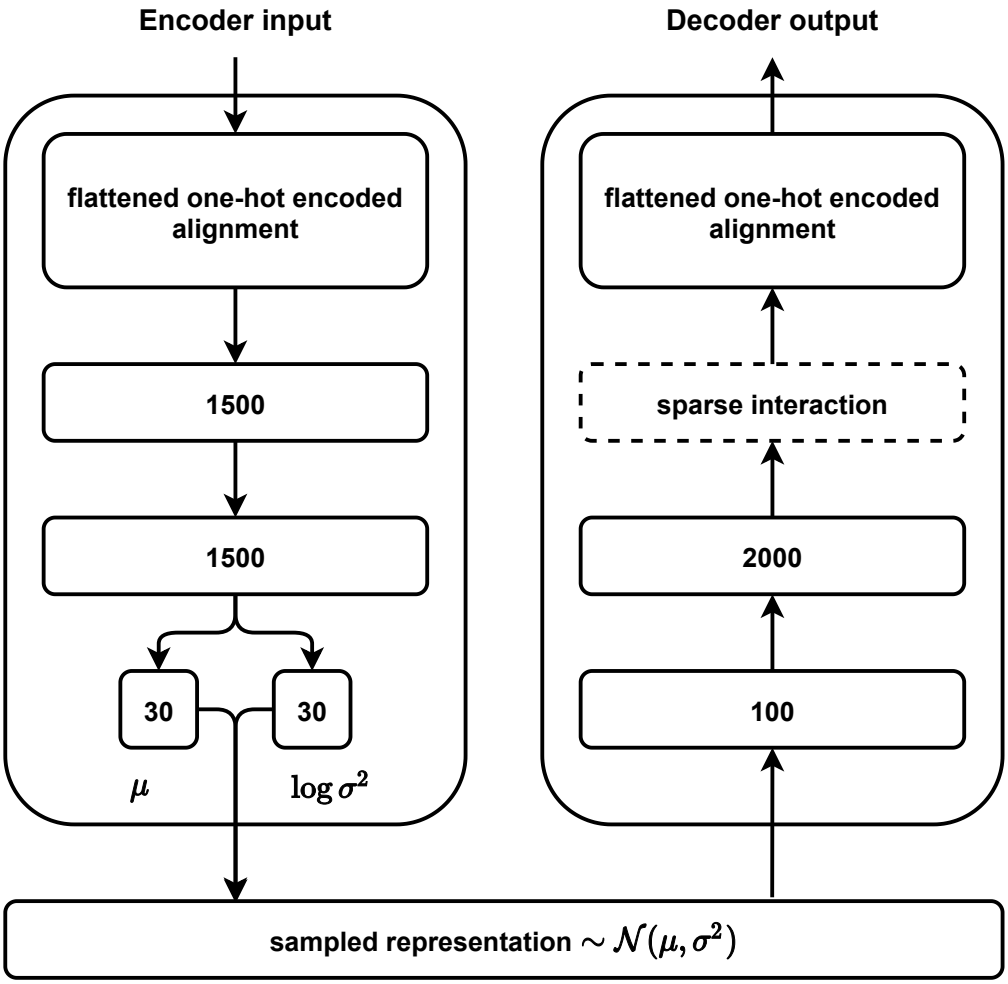


Figure 6.2: Architecture of the VAE model.

6.3.1 Results

As an initial sanity check, we wanted to inspect the representations produced by the VAE. To do this, we constructed a VAE model with a 2-dimensional latent space and plotted the resulting representations. Even in two dimensions, a plot of desirable protein representations should show similar proteins clustered together (and hence dissimilar proteins well at distance). Figure 6.3 shows the mean of the representations of proteins from the β-lactamase protein family, colored by phylum, as produced by this model. The structure of the representations resemble that of a phylogenetic tree, suggesting that the model correctly identifies the evolutionary similarities between the different variants of β-lactamase. A similar phylogenetic tree-like structure is seen when using the model on other protein families. In addition, small mutations in a protein should yield representations that lie relatively close to the unmutated protein they come from. This is shown in the zoomed subregion of figure 6.3, where the wild type (i.e. unmutated) protein is shown together with its mutations.

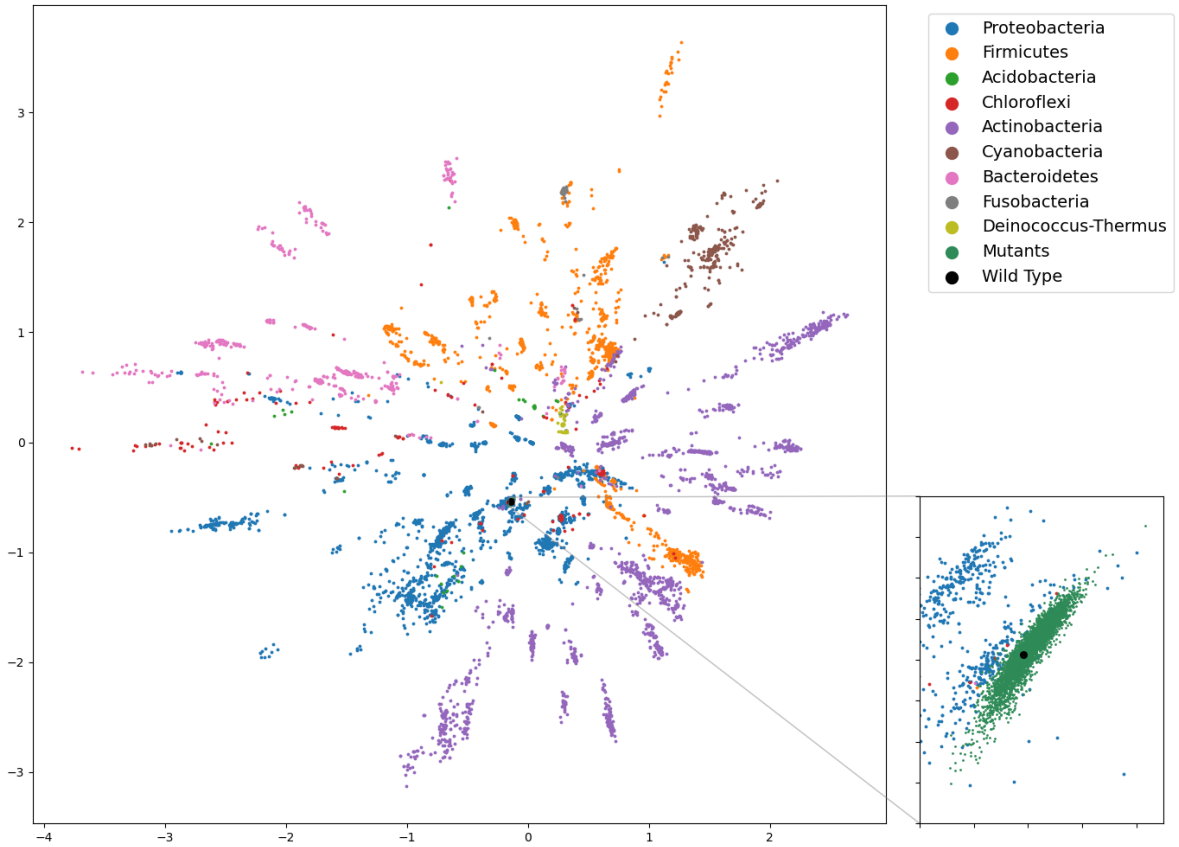


Figure 6.3: Mean of the protein representations of selected phylums of β-lactamase, produced by a VAE model with a bottleneck layer size of 2. The subregion to the right is a zoom of the representations of the wild type mutants (shown in green).

In table 6.4 we display the collected mutation effect prediction correlation results across models and protein families. Each result is the average over 5 models trained with the same settings to adjust for random initialization and other factors contributing to variance in the performance. The full results of single training and testing sessions can be found in appendix A.1.

Configuration	BLAT ECOLX	CALM1 HUMAN	GAL4	HSP82	RASH HUMAN
MAP	0.57	0.28	0.39	0.47	0.49
MAP + sparse	0.66	**0.29**	0.47	0.49	0.47
MAP + sparse + rws	0.69	**0.29**	0.48	0.51	0.40
Bayes	0.69	0.27	0.37	0.47	**0.50**
Bayes + sparse	0.71	0.27	0.37	0.51	0.38
Bayes + sparse + rws	0.73	0.26	0.61	**0.55**	0.49
Bayes ensemble	0.77	0.27	**0.63**	**0.55**	0.48
DeepSequence [6]	**0.78**	0.23	0.45	0.53	0.47

Table 6.4: Mutation effect prediction correlation using variants of the VAE model. Each measure is averaged over five sessions. DeepSequence prediction correlations are provided as a reference. *Bayes ensemble* refers to the ensemble over the 5 sessions of Bayes + sparse + rws, rather than the average.

Overall, the full Bayesian weight treatment seem to improve the performance across the datasets. The results are discussed in detail in section 6.3.2.

The model is explicitly trained to reconstruct its input protein sequence from its encoded representation. In figure 6.4 the β-lactamase query protein is shown along with its reconstruction.

```
                                        1-80
HPETLVKVKDAEDQLGARVGYIELDLNSGKILESFRPEERFPMMSTFKVLLCGAVLSRVDAGQEQLGRRIHYSQNDLVEY
...LAETVKQAEKRLGARVGYAELDLASGKLLESYRADERFPMMSTFKVLLCGAVLARVDAGEEQLDRRITYRKSDLVEY
                                        81-160
SPVTEKHLTDGMTVRELCSAAITMSDNTAANLLLTTIGGPKELTAFLHNMGDHVTRLDRWEPELNEAIPNDERDTTMPAA
SPVTEKHLADGMTVAELCEAAITMSDNTAANLLLASIGGPAGLTAFLRSIGDTVTRLDRWEPELNEALPGDERDTTTPAA
                                        161-240
MATTLRKLLTGELLTLASRQQLIDWMEADKVAGPLLRSALPAGWFIADKSGAGERGSRGIIAALGPDGKPSRIVVIYTTG
MAATLRKLLTGDVLSPASRQQLIDWMVADKVAGPLLRSVLPAGWFIADKTGAGERGSRGIIAVLGPDGKPSRIVVIYLTE
                                        241-263
                                    SQATMDERNRQIAEIGASLIKHW
                                    TEATMDERNAQIAEIGAALIKHW
```

Figure 6.4: Reconstruction of the β-lactamase query protein by the ensembled VAE model. The reconstruction is by the ensemble of models with decoder weights sampled from the learned weight distributions. The top sequence is the original, while the bottom is the reconstructed sequence. Red and green positions denote incorrect and correct reconstruction, respectively. The accuracy is $\approx 80\%$.

In a similar fashion, we found it important to inspect the likelihood predictions produced by the model. The positional predictions of a random subsample of 4 protein sequences from β-lactamase is shown in figure 6.5.

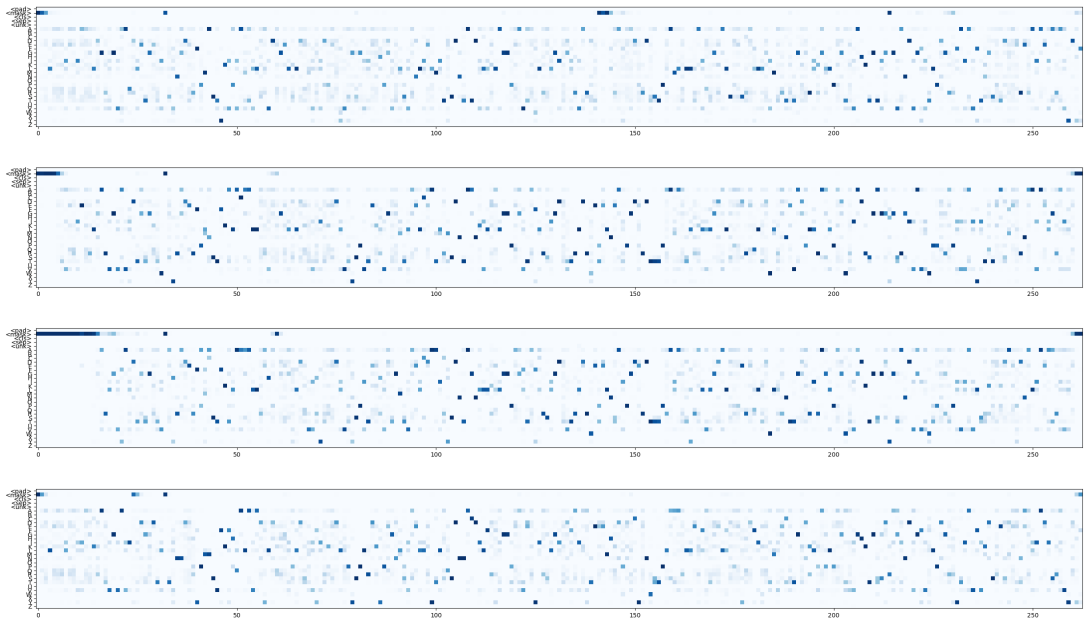


Figure 6.5: Softmax predictions on 4 protein sequences from the β-lactamase protein family. The figure does not afford close inspection, but it suffices as a sanity check that the model does indeed produce complex distributions over the potential amino acids at each position. The intensity of blue denotes more likely predictions.

6.3.2 Discussion

This section contains discussion mainly focused on the experiments related to the VAE and its produced representation. The general discussion across experiments can be found in chapter 6.

Because the VAE is constrained to aligned, fixed-length protein sequences, they do not generalize to the tasks in the TAPE benchmark, which unfortunately means that we will currently not be able to assess how well this approach perform on these tasks. This is entirely because the current setup does not allow inputs of variable lengths. In section 7.1.1 we sketch potential future work that might resolve these limitations.

Mutation Effect Predictions

This part of the discussion refers to figure 6.4. In our experiments, yielding a correlation coefficient above 0.70 on the β-lactamase protein family required Bayesian weight estimates, while MAP estimates with sparse interaction and random weighted samples is close at 0.69. The fact that Bayesian networks perform better than networks that produce MAP estimates, is in accordance with our expectations: the Bayesian model is an ensemble over networks, and so gain the benefit of ensembles. Specifically, it should reduce bias in the predictions, as these tend to be averaged out across the ensemble networks. It should also reduce variance in predictions as the number of models in the ensemble increases. That is, if you have a single protein sequence and ask many individual models for predictions, the variance of these predictions will be larger than the variance in predictions of many ensemble models. Sparse interactions seem to make a difference for MAP estimates on β-lactamase and GAL4, but there is no clear-cut conclusion to the merits of this feature. E.g. for RASH HUMAN, Bayesian predictions decline from 0.50 to 0.38 when sparse interactions are applied.

Across protein families, the correlation coefficient varies significantly, from 0.77 to 0.26. We have not been able to identify explanatory factors for this; it is likely that such explanations can be found in the relationship between the experimental metrics and the prediction metric. That is, equation 6.1 might simply correlate better with the specific β-lactamase experimental measures and less so with CALM1 experimental data. On the surface, there is no clear relationship between results and the sizes of the families, nor their mutations.

In general our produced results are in accordance with what is achieved by Riesselman et al. [6]. For all protein families but β-lactamase, we observe a relative increase in performance. The increase does however not seem to be significant, and might simply be due to variance. One noteable difference is on GAL4, where Riesselman et al. [6] achieve 0.45 and our Bayesian VAE yields 0.61. The results should be directly comparable and so this difference is a little surprising. Note that every other VAE variant than random weighted sampling, sparse, Bayesian VAE are much closer to the reference predictions by Riesselman et al. [6]. These are however minor observations to the general trend that the VAE model architecture seem to capture protein characteristics that correlate with experimental observations.

Explained Variance of representation principal components

A notable observation is that if principal component analysis is applied to the representations of the protein family, one can observe that the variance of the representations can be fully explained by 12 components, as seen in figure 6.6. This raises some questions about the size of the latent space, and whether one could get similar performance with a smaller dimensionality. Additionally, this might also indicate that the model is not fully utilizing the capacity of the representation or that the signal from the data is not strong enough, as it seems implausible that we can accurately and compactly represent proteins in 12 dimensions. It may also be that the model does utilize the remaining 18 dimensions, but that the values the model uses in those dimensions are much smaller.

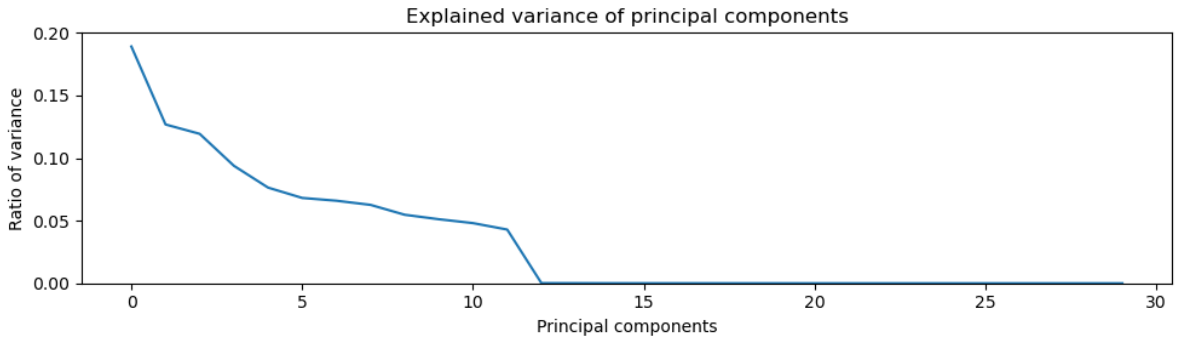


Figure 6.6: Explained variance of the principal components of the 30-dimensional representations of the BLAT ECOLX protein family.

Bayesian Weight Distributions

Because the Bayesian VAE seem to improve performance, we thought it made sense to inspect the learned distributions to see if there was anything worth noting. In figure 6.7, learned weight distributions of two decoder neurons are shown. Some neurons have weights diversely distributed, indicating that the neuron has learned useful weight distributions in relation to the task of reconstructing its input sequences. However, other neurons seem to carry weights that are all closely distributed as a unit Gaussian. Such neurons are the most common. This is not really surprising, as the model is explicitly trained to minimize the divergence between its weight distributions and the weight prior, a unit Gaussian. However, this behavior suggests that a significant part of the model capacity is used to fulfill the pull of the prior distribution. The prior was chosen solely for convenience, and this choice seems to affect the weight distributions more than desired. It is unlikely that the model effectively utilizes weights that are in expectation zero, and thus almost equally likely to be positive or negative. For these reasons, the architecture might benefit from choosing other weight priors. An initial choice could be the scale mixture as suggested by Blundell et al. [11].

This is also an indication that the network could be compressed by a procedure that eliminates weights that are distributed sufficiently close to a unit Gaussian distribution, as these weights are unlikely to contribute to the prediction behavior of the network. This might be related to some of the ideas proposed in Molchanov et al. [38], Kingma et al. [39], and Gal et al. [40], where model compression, dropout and the Bayesian approach are shown to be connected – this is outside our scope however, so we leave this as an interesting observation and potentially the foundation for future work.

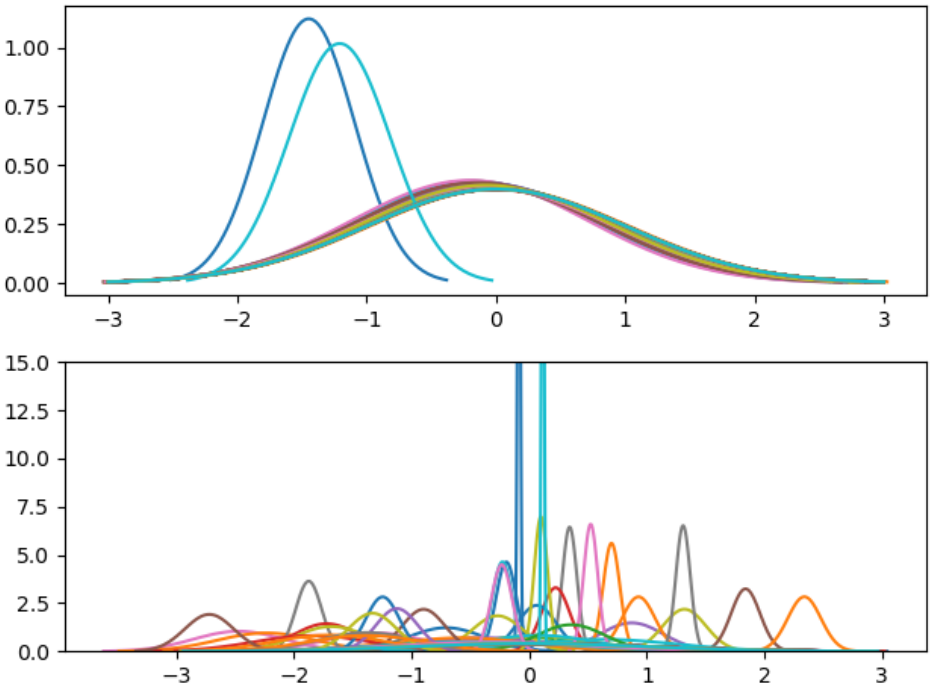


Figure 6.7: *Top:* Weight distributions of the first neuron of the 2000-sized linear layer of the decoder. Almost all its weights are closely shaped to a unit Gaussian distribution, likely an effect of the prior being unit Gaussian and a training objective that minimizes the KL divergence to the prior. *Bottom:* The 32nd neuron in the same layer. Here weight distributions are much more diverse, suggesting that this neuron is critical. Neurons similar to the first neuron are most commonly observed in the layer. Each neuron has 100 weights.

6.4 WaveNet

Like UniRep and the VAE, the WaveNet model architecture was trained on each of the five chosen protein family datasets (see section 6.1.1) for mutation effect prediction. The mutation effect experiments are conducted by training the model to reconstruct sequence inputs, compelling the model to learn factors that drive protein characteristics. For a given input sequence, the model predicts each residue position based on the sequence observed so far at that position. We trained a total of 6 models: 3 observing sequences in a "forward" direction (N-terminus to C-terminus), and 3 observing sequences in a "backward" direction (C-terminus to N-terminus). This ensemble of models determine the mutation effect predictions. We have adopted hyperparameter settings as described in Riesselman et al. [4], using L2 regularization of model parameters with coefficient $\lambda = 1$, a clipping of gradient norms to 100, and an aggressive dropout probability of $1/2$ on model components. The WaveNet architecture was instantiated with a dilation stack of 6 dilation blocks, each consisting of 9 dilation layers (that is, dilation in the range $1, 2, 4, \ldots, 256$). The number of channels used in the NormConv layers is 48 (with the exception of final and initial convolutions which map to and from the number of candidate residues).

In the case study of the β-lactamase protein family, we have two variants of the experiment: one where each protein sequence input is equally weighted in the loss calculation, and one with sequence weights according to sequence similarity (see 6.1.1 for details). This is in order to determine the significance of weighted samples on model performance. The weights are currently indirectly dependent on an alignment, and so cannot be assumed to be freely available in a general setting.

In addition another model instance was pretrained and/or finetuned for TAPE, in the same manner as the UniRep model. In this setting, the model architecture provides the representation for a top model.

Collectively, this setup is trained on labeled TAPE training datasets and evaluated provided test sets, for each task.

For pretraining, we applied the model to the full UniRef50 dataset, without any regularization, for at least a full epoch. We did not use any regularization in the pretraining since the UniRef50 dataset is so huge that we believe that overfitting will not be a big problem. We did however use 1% as validation, just like with UniRep.

6.4.1 Results

Mutation Effect Prediction

For this task, we train 6 models independently, 3 forwards and 3 backwards. We report both the mean of the models' results but we also combine them into an ensemble model to achieve stronger results. We used 20% of the data samples as a validation set. The results can be seen in table 6.5.

Configuration	BLAT ECOLX	CALM1 HUMAN	GAL4	HSP82	RASH HUMAN
Finetuned	0.56	0.27	0.50	0.45	0.39
Pretrained	0.08	0.16	0.04	0.06	0.33
Pre. + Fine.	0.50	0.27	0.46	0.45	0.35
Finetuned (E)	**0.61**	**0.29**	**0.54**	**0.51**	**0.43**
Pre. + Fine. (E)	0.55	**0.29**	0.49	0.50	0.40

Table 6.5: Mutation effect prediction Spearman's ρ correlation scores for the WaveNet model on different datasets, on the different configurations. The bottom two rows indicate ensembles over the 6 models rather than averages.

The pretrained only WaveNet variant achieve near to no correlation on most datasets, the exception being RASH HUMAN, where it performs comparably to the finetuned model. The best performing model is the finetuned ensemble, achieving 0.61 on BLAT, 0.29 on CALM1, 0.54 on GAL, 0.51 on HSP82 and 0.43 on RASH. Across the models, these are the highest correlations.

We also tried training a model on weighted samples of the β-lactamase family. If samples are weighted, the model performance increases significantly, from 0.61 to 0.72, suggesting that this is an important factor.

Just as with the other models, we have visualized WaveNet's representations of the β-lactamase family. This is a high-dimensional representation, in this case 48 dimensions, which requires us to use a dimensionality reduction. We use t-SNE, just as we did with UniRep. The result can be seen in figure 6.8.

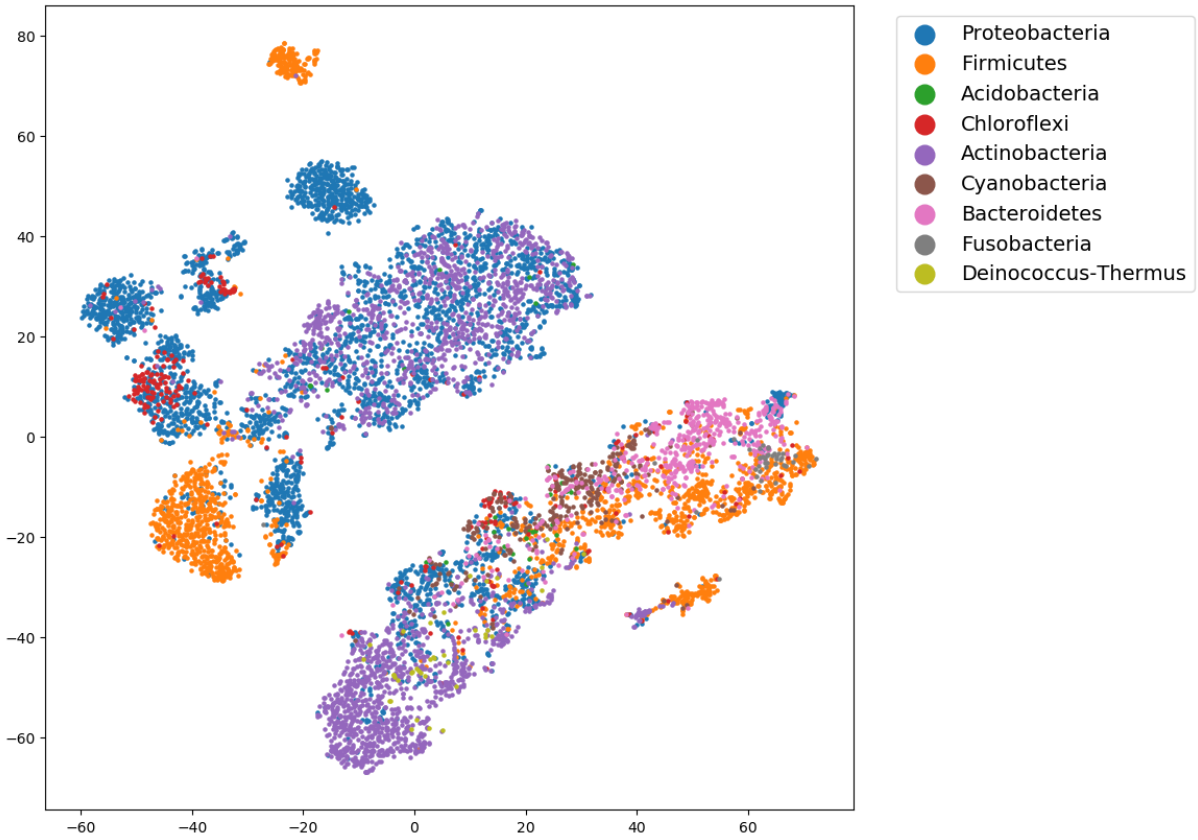


Figure 6.8: t-SNE visualization of one of the forwards WaveNet model's representations of the β-lactamase family.

TAPE

When training WaveNet on TAPE, we only use a single forward model. Otherwise, we trained WaveNet on the TAPE tasks in the same way the UniRep model was trained. The results can be seen in figure 6.6.

Configuration	Secondary Structure	Remote Homology	Fluorescence	Stability
Finetune	0.35	0.09	0.07	**0.78**
Pretrain	**0.38**	**0.17**	**0.47**	0.58
Pre. + Fine.	0.35	0.04	0.32	0.66

Table 6.6: WaveNet's performance on the TAPE tasks.

6.4.2 Discussion

Mutation Effect Prediction

When sample-weights are applied, the WaveNet model performs comparatively to the VAE and DeepSequence on the β-lactamase protein family (0.72 compared to 0.78). This confirms that the model has the potential to perform well on mutation effect prediction. It does however degrade in performance once samples are uniformly weighted, as shown in table 6.5, which shows that there is still room for improvement. Because of the significance of weights, finding a weighting scheme that does not rely on an alignment is a reasonable next step. One such approach could be to apply unsupervised clustering

algorithms to the dataset and distribute weight inside clusters. This is a common problem associated with the data and not the model, and so we discuss this separately in section 7.2.

In general, the ensemble models once again significantly outperforms the other individual models. The ensemble model without pretraining achieves the best performance on all the datasets, suggesting that pretraining is only detrimental to this task. This is similar to what we saw in the UniRep results. However, the general trend is that the performance is very similar once finetuning is applied. It is interesting that pretraining actively hurts performance, as this goes against our expectations.

Across datasets, the finetuned architecture seem to capture signal from the entire protein sequence by the gradually expanding convolutions, alleviating the need for recurrence in the network, while maintaining performance. This is a desirable property as, in contrast to RNNs, convolutions can be performed in parallel, and so one should expect an improvement in runtime on GPU accelerated training. In addition, this architecture still afford the desirable length-invariant property of the recurrent neural network (as long as all sequences are at most a specified maximum length that can be set sufficiently large to accommodate any practical setting). However, in our practical settings, WaveNet is considerably slower per epoch in comparison with both UniRep and the VAE model. It also consumes more memory, especially for pretraining on longer sequences. For this reason, multiple GPU's were used for pretraining, to speed up convergence. These practical downsides might however be purely related to implementation details and not an inherent property of the architecture.

In terms of trainable parameters, the WaveNet architecture is also significantly smaller, requiring gradients for about 600,000 parameters. This is a small model in comparison with most other models, usually requiring millions of parameters. The VAE model in our experiments require about 50 million parameters and is thus significantly larger.

There are currently at least two major drawbacks using WaveNet for mutation effect prediction: The performance loss due to unweighted samples, and the fact that the model produces no explicit, learnable, fixed-length representation for its inputs. The issue of extracting fixed-length representations for variable-length inputs is shared between both WaveNet and UniRep, and so we discuss this separately in section 7.1.1.

TAPE

In general, WaveNet performs significantly worse than UniRep on the TAPE tasks. The Finetune configuration has especially bad performance on the secondary structure task (0.35) and fluorescence task (0.07). Despite this, the same model achieves a performance on the stability task (0.78) 5 percentage points above the best models on the TAPE leaderboard[3] (0.73). Aside from this surprising result, the pretrained model performs best in general, but still not nearly as good as UniRep. Just like with UniRep, we see a significant improvement on remote homology as a result of pretraining.

The poor performance of WaveNet comes as a bit of a surprise, as it had promising performance on the mutation effect prediction tasks. We see two major possible causes for the bad performance. Firstly, the model used for the mutation effect prediction was an ensemble of three forwards and three backwards WaveNet models, while the one used in TAPE is a single forward model. This was done because the combination of the top models into an ensemble is non-trivial, and it gives the WaveNet an unfairly powerful top model in comparison to the other models. Even then, we doubt that an ensemble would be enough to bring WaveNet up to UniRep's performance. Secondly, the WaveNet model may not have sufficient capacity for a global setting. Its representation is, after all, only 48 dimensions, in comparison to UniRep's 512, and its number of parameters is approximately half the number of parameters in UniRep. Finally, the dilation of the WaveNet has a max width of 256, which might affect the performance on sequences in global space, as they are up to 2000 positions (in our settings; we imposed this limitation). That is, the network might not reach significant long range dependencies for this reason.

[3] See `https://github.com/songlab-cal/tape`

Figure 6.8 might also explain the poor performance in the TAPE tasks, as the representations are clearly not separated as well as UniRep's representations, as seen in figure 6.1. This could easily explain why the top models used in TAPE have problems achieving good performance. On the other hand, it raises the question of why WaveNet is still good at mutation effect prediction – but keep in mind that the mean representation visualized in figure 6.8 is only used by the TAPE tasks, and not by the mutation effect prediction task.

7
Discussion

In this chapter we will discuss the results from chapter 6 as a whole, comparing the models to each other. In section 7.1, we compare the representations and the results that the models achieved in the experiments. We also discuss what may cause the differences in performance between the models and suggest alternative ways to extract the representation from UniRep or WaveNet. In section 7.2, we discuss the role that the data may have in the good performance of the VAE, as well as the limits that the data imposes onto the significance of our experiments. In section 7.3, we discuss how the latent space of representations may be used for exploration in the protein space and how this may require additional tinkering. Finally, in section 8.2, we consider avenues for further work that build on the ideas that we have worked on.

7.1 Comparison of Experimental Representations

Based on experimental results alone, there are some clear guidelines to which models produce the most ideal protein representations. For convenience, figure 7.1 and 7.2 compare models across the two experimental settings.

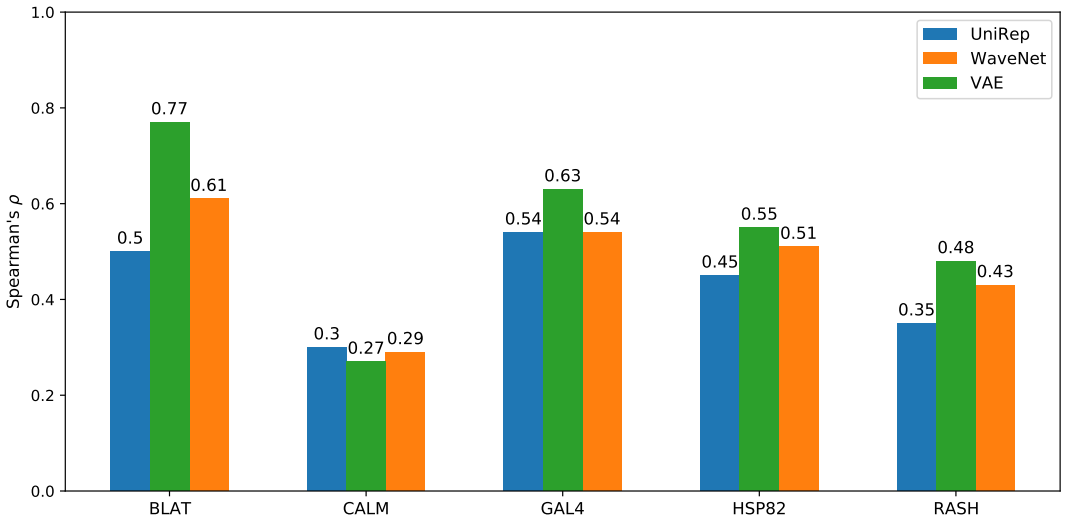


Figure 7.1: Bar chart comparing each of the three model variants on the five protein families tested. The generally best configurations for each model are shown. Almost invariably, VAE is best with WaveNet at second place. The exception is CALM, but even then the difference is hardly significant.

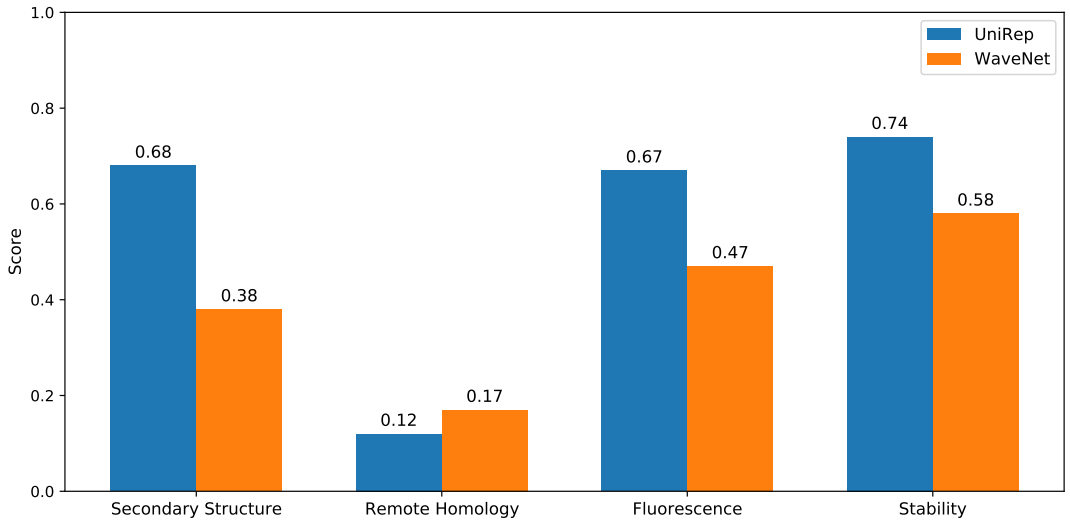


Figure 7.2: Bar chart comparing UniRep and WaveNet on the TAPE tasks. The generally best configurations for each model are shown. UniRep outperforms WaveNet on all but the remote homology task.

For family-specific, local representations, the VAE outperforms both the WaveNet and UniRep across the five protein families tested. There are several reasons why this might be the case.

One reason is that the aligned protein sequences is a strong signal to the model that is not preserved otherwise. However, the performance of WaveNet and UniRep on β-lactamase with sample weights, taken from the aligned dataset, suggests that it is not the alignment itself that is important. That is, the significant performance increase is not from the insertion of gaps, but rather from the similarity measure and reweighing of samples according to their uniqueness, that such an alignment affords. Data- and preprocessing related aspects are discussed in section 7.2.

Another reason might be the fact that the representation is explicitly trained by the model, allowing for the model to fully decide, within its capacity, what should be retained and what should be discarded in the representation.

In addition, the underlying probabilistic setting is different for the VAE. Specifically, when making predictions, the model can consider all positions indirectly through the latent space representation. In contrast, UniRep and WaveNet are constrained to the positions observed so far. This is somewhat mitigated by the bidirectional ensemble of the WaveNet, and a similar extension can be made to UniRep, but it still raises the practical issue of concatenating a "forward"– and a "backward" representation of the same protein in such cases. The VAE setting has no such issues.

Finally, the space of the representations itself affords several advantages in exploratory protein settings, as discussed further in section 7.3.

While latent spaced representations have some favorable advantages in comparison with the representations of UniRep and WaveNet, the major downside is that they cannot directly be applied to a broader, more general set of proteins. If the desired protein representation is to be broadly applicable, our experiments suggest that this will likely compromise the performance on local subsets of proteins. WaveNet's relatively good results on mutation effect prediction and comparatively bad results on TAPE, and vice versa with UniRep may suggest that there is a trade-off in a model's capacity in the local and global protein scale. It seems clear at least that, if one is constrained to work with unaligned proteins, UniRep is to be favoured in the global scale while WaveNet is favoured on the local scale. WaveNet is at a huge disadvantage on all tasks but stability on TAPE, whereas UniRep is only marginally worse

than WaveNet on mutation effect prediction. This might indicate that the WaveNet architecture is fundamentally worse at producing representations that work well across diverse tasks, whereas UniRep seems to be able to handle both diverse, general tasks and local finetuning. It does not seem impossible that UniRep can perform comparably to WaveNet on mutation effect prediction if properly augmented, whereas it might require serious rethinking of WaveNet to yield comparable results to UniRep on TAPE.

7.1.1 Extracting Representations

The representations that our models produce are quite different. The VAE inherently includes a bottlenecked representation, which allows the representation to become part of the training procedure. The UniRep and WaveNet models do not have this luxury, as the representations must be extracted after training. The task of extracting a suitable representation from models that take variable length sequence inputs is nontrivial. In our settings, we have applied the naive approach of collapsing the position-wise representations (i.e. the hidden state for UniRep and residual channels for WaveNet) in the models by taking the mean. This has the positive effect of being effective and simple, but it also has significant downsides. Specifically, such an operation is hard to reverse, and as a consequence a protein sequence cannot easily be recovered from its representation. In terms of protein exploration this is critical, as the representation space cannot be explored in search of new proteins, as you would not be able to recover a protein sequence from a representation of interest.

A mean representation does not preserve any position-related properties of the protein. Any feature that denotes local characteristics of the protein is made less relevant by the mean across the entire sequence. Imagine a single feature dimension that encodes if the current position is part of an α-helix or not. The mean of such a dimension would override such information, yielding instead the α-helix proportion of the entire protein sequence. In this sense, the mean is a crude operation with no explicit mechanism to retain such important information present in the position-encoded representations.

On the other hand, fixed-length representations of proteins with variable length inevitably need to be position-agnostic, as should be encouraged if fundamental protein properties are to be captured. Ideally, a suitable representation should be learned explicitly during model training, as in the VAE architecture, making manually engineered extraction schemes superfluous. A line of work that should at least be explored is to combine the built-in representation of the autoencoder with the variable-length models, as sketched in figure 7.3. A major problem with this approach is that the decoder model has to reconstruct a variable length-sequence from a fixed representation, which is a daunting task. It can be mitigated by teacher-forcing or by producing a beam search instead of a single prediction path, but only the latter will be viable outside training settings. Such a model would be able to explicitly train on the task of autoencoding arbitrary-length proteins into fixed-size representations.

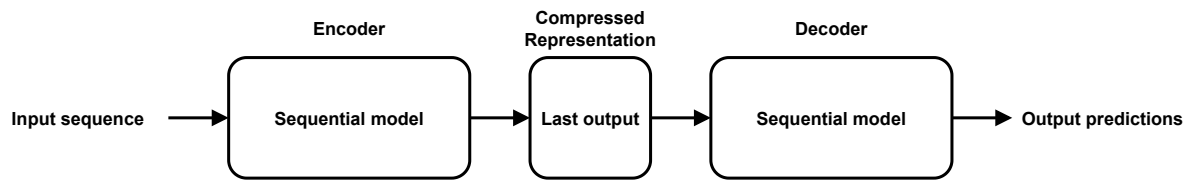


Figure 7.3: A sketch of an autoencoding model that produces fixed-length representations over variable-length sequence input. The input is passed through an encoding architecture that is able to process sequential inputs, such as the LSTM (like UniRep) or a convolution-based model (like WaveNet). The last output of the encoder could function as the compressed representation of the input, as this is explicitly trained by the model, making the architecture itself figure out what should be retained. Then, a similar architecture functions as a decoder whose task it is to reconstruct the input from the representation.

Models like the one sketched in figure 7.3 have been created before within natural language processing, but we have yet to see this method be truly effective in protein machine learning, especially on

arbitrary, non-aligned proteins. Other things being equal, explicitly training the network to construct representations should be preferred to manual engineering.

7.2 Data and preprocessing

Our findings show that the VAE is generally superior on the mutation effect prediction task, when compared to the other models. However, it is worth noting that the VAE does not only have the advantage of an explicit trained encoder-representation-decoder network, but also the advantage of a fixed-size protein alignment as data. As discussed previously in section 4.3, sequence alignments can be made in many different ways, consequentially with varying performance on downstream tasks. This is also true for the VAE model. The question is, how much of the VAEs better performance, in comparison to the competition, comes from the architecture versus the preprocessing of the data through an alignment? Ideally, we'd like to be able to attribute the performance of the VAE to its architecture, but this does not seem to be the entire influence. As evidenced by WaveNet's improved performance when given weights based on an alignment, preprocessing of the data into a neater structure can significantly improve performance. This suggests that a significant portion of the VAE's greater performance comes from the preprocessing of the proteins into an alignment, rather than the VAE's own architecture.

There is also a potential issue about the experimental data that we compare our predictions with. Each protein family in the mutation effect prediction task uses a separate set of experimental data to calculate the correlation coefficients. However, this correlation coefficient is oblivious about the other experimental data gathered on the mutations and thus the whole effect of the mutation on the protein cannot be determined by our models alone. This problem is also brought up by Riesselman et al. [6], where they note that a "mutation may be damaging with regard to some measurable protein feature – for example, enzyme efficiency – but harmless for stability or even organism fitness". That is, we need to remember that our results only highlight the correlation with one specific experimental data measure and we have no data on our models' performances on other experimental measures.

7.3 Protein Exploration

Figure 6.3 shows an interesting result about the landscape of the VAE's protein representations. Figure 6.1 shows a similar picture, although one must remember that this is from a non-linear transformation of the data and hence not directly comparable to the points' distribution in 512 dimensions. These figures suggest that the protein representation landscape resembles a phylogenetic tree at a high level. The zoomed-in view in figure 6.3 suggests that the landscape is (at least locally) smooth – that is, small changes in the protein lead to small changes in the representation. This is an especially important property when one's goal is to explore the representation space (and by extension, the space of proteins).

However, it does not seem like the smoothness of the representation space at the small scale translates into a regular euclidean distance measure at larger scales. There are large areas between the phylum clusters where there are no representations provided by the data – in principle, these empty areas contain representations just like the phylum clusters do, but the question is whether these representations are meaningful, when it is clear that the data has not informed these areas. For example, one way to explore the representation space may be to interpolate between two proteins that inhabit separate phylum clusters in the representation space. A naive linear interpolation would likely not be effective, as this would frequently cross areas where the data is not present and thus likely to be noisy or otherwise meaningless. Such a linear interpolation is shown in figure 7.4.

An effective exploration of the protein representation space might require a non-euclidean distance measure, which takes into account the phylogenetic tree-like structure. An interpolation between two protein representations using such a distance measure may be able to interpolate along data-populated areas, rather than traversing empty, high-uncertainty areas.

Figure 7.4: Naive linear interpolation in the two-dimensional latent space (shown in figure 6.3) between two protein representations in the β-lactamase protein family results in the above proteins once decoded. Only the first 80 positions of the alignment are shown. The 25 representations are evenly spaced. A dot (“.”) indicates a gap in the aligned sequence.

8
Conclusion

In chapter 1, we introduced the motivation for the work in this project and outlined its scope. We also introduced the essential terminology of proteins and related the protein engineering task to natural language processing.

In chapter 2, we explained the need for unsupervised learning and presented the concept of representations living in a latent space. We also introduced the concept of global and local scales in the protein space, as well as some desirable properties such representations should contain.

In chapter 3, we presented variational inference and the theory behind the variational autoencoder. We derived the evidence lower bound (ELBO) and showed how the reparameterization trick may be used for gradient-based optimization. This chapter concluded by presenting Bayesian neural networks and how these can be integrated with the variational autoencoder.

In chapter 4, we presented various ways of learning on sequential data, including recurrent neural networks (RNNs). We discussed sequence-to-sequence models and the issues these present with extracting their representations, in contrast to models that explicitly produce compressed representations. Finally, we explained the concept of sequence alignments, which can be used to apply classical linear neural networks to sequential protein data.

In chapter 5, we presented the UniRep, variational autoencoder and WaveNet models and explained their architectures and different ways of processing protein sequences. These models were put to the test in chapter 6, where we conducted experiments on downstream tasks, including mutation effect prediction and prediction of protein properties, like stability. In the same chapter, we discussed the individual results, while in chapter 7 we compared the results and discussed the overall implications. We show that the variational autoencoder achieves superior performance on mutation effect prediction for local protein families and argue that its representations display desirable properties for exploration, not exerted by the other models. Representations produced by UniRep outperform WaveNet on general protein analysis tasks. The tested models all indicate areas of limitation. We discuss why this is so, and potential ways to overcome these weaknesses.

8.1 Research Questions

In chapter 1, we asked a series of fundamental questions, on which we now conclude.

Our first question asked how close protein deep learning is to reaching an ideal model; a probabilistic model which transforms arbitrary protein sequences into compact representations living in a smooth, explorable space. Our findings show that there are still significant advances to be made towards this ideal. The variational autoencoder does indeed produce probabilistic, compact representations, but it cannot transform arbitrary protein sequences and the space of representations is only smooth on a small scale, impairing exploration. The UniRep and WaveNet models are able to transform arbitrary sequences, but they trade this power for a weaker representation which is neither probabilistic nor able to be decoded back to a protein. In section 7.1.1, we suggest a way to possibly remedy this for future work.

Next we asked what advantages and disadvantages exist between global and local representation. Our experiments indicate that the local representations provided by the variational autoencoder are better in general, but they do of course require training on each specific protein family. They also suggests that

global representations may have a tendency to only learn coarse features that do not translate well to the local scale. One should take all this into account when considering global and local representations and the models that produce them.

Finally, we asked how the choice of the representation affects the performance on downstream tasks. Foremost in this choice is what model to use in order to produce the representations. We can say now that the variational autoencoder is a strong choice in this case. The VAE's ability to decode its representation is an especially important property as it allows exploration, but it also forces the model to train on its representation. This is in contrast to UniRep's and WaveNet's representations which are not trained and only constructed externally. We believe this severely negatively affects the potential performance of UniRep and WaveNet.

8.2 Further Work

8.2.1 Obtaining weighted sampling without an alignment

One key observation has been that the weighted sampling procedure that is applied to aligned sequences yield a significant increase in prediction performance. This is likely because good protein representations are guided by the uniqueness of protein sequences rather than their quantity. In our settings such weighting requires an alignment, but there seem to be existing similarity searching schemes that are not constrained to fixed-length sequences. It might be possible to recover some of the lost performance by using an algorithm such as the **MMseqs2** algorithm [41], which greedily clusters proteins based on identity and overlap between protein sequences (this is also the algorithm used to Cluster the UniRef100 dataset into UniRef90 and UniRef50). Each cluster has a representative protein sequence. Such clusters can be used to weigh sequences by giving each cluster a fixed weight and distributing that weight evenly across the sequences of each cluster. Otherwise, one could simply reduce the dataset to the cluster representative sequences, as these are unique in comparison, but similar to the sequences in the clusters they represent, within some threshold. The effective number of protein sequences $n_{\texttt{eff}}$ would then be the number of clusters found by this procedure.

8.2.2 Choice of prior

In the variational approximation setting, it seems that the prior distribution on both the latent variables and weights play a significant role on the performance. Interesting further work would be to experiment with other priors, or with the VAE setup to scale the significance of the KL divergence from the prior. A good place to start could be the β-VAE [42], which allows for this kind of scaling. The β-VAE proposes a scale factor $\beta \geq 1$ as they claim this yield more disentangled representations in their experimental settings, but we think that a scaling factor $\beta < 1$ is more interesting, as this makes the deviance of the approximation to the prior less significant in the objective function.

8.2.3 Non-euclidean distance measure in the representation space

As discussed in section 7.3 above, developing a non-euclidean distance measure may be necessary for effective exploration in the representation space on the large scale. One venture to develop such a measure may be manifold learning – that is, learning a transformation from the representation space to another space in which linear interpolation *does* follow the tree-like data. We have not looked into this in depth, but this may be one way to achieve the effective interpolation in the representation space.

8.2.4 Transformer-based models

During the initial phases of this project, we considered various types of models to explore, arriving at the three models examined above. However, we also toyed with the idea of a model based on the

Transformer architecture [43]. The Transformer is based on the idea of attention, which is a technique used with sequential models (classically recurrent neural networks).

The Transformer model has been shown to be incredibly effective in natural language processing, giving rise to models which can produce text that appears as though it was written by a human [44]. We attempted to construct a Transformer-based model which could process protein sequences, but we did not have enough time to polish the model to a point where it achieved any interesting results. The architecture of the Transformer is also quite complex, and the models in NLP that achieve good performance are often obscenely large.

However, we still believe that a Transformer-based model may be a fruitful exploration, as the Transformer includes some interesting ideas, such as a method to handle the positions in a variable-length sequence, using a special positional encoding.

8.2.5 Semi-supervised Learning

In the introduction we advocated unsupervised modelling, such that the vast amount of unlabeled data can be put to use. In our experiments, the models have however been trained solely on such data and have no direct mechanism for handling protein sequences where additional information is present. Effectively, this means that practitioners would have to discard labels in order to feed the data samples to the models. This is undesirable, and further work could look at merging supervised and unsupervised learning settings such that any labeled protein sequences can be used to inform the produced representations.

8.2.6 Training Objective in Representation Space

In order to teach the models, we use a learning objective that measures how well the model reconstructs the input. The problems with this is that it is a function on the "sequence space" and not in some more abstract space (like the representation space or, for mutation effect predictions, on the correlation measure itself). What we really want is to learn a good mapping from sample space to representation space, which is the assumed side-effect when learning to reconstruct the input. However, we could preferably measure loss based on the representation space as well. This could force additional characteristics onto the representation space, similar to the Kullback-Leibler term in the VAE which pulls the distribution on the space toward the prior distribution.

References

[1] UniProt Consortium. "The universal protein resource (UniProt)". In: *Nucleic acids research* 36.suppl_1 (2007), pp. D190–D195.

[2] Ethan C Alley et al. "Unified rational protein engineering with sequence-based deep representation learning". In: *Nature methods* 16.12 (2019), pp. 1315–1322.

[3] Aaron van den Oord et al. "Wavenet: A generative model for raw audio". In: *arXiv preprint arXiv:1609.03499* (2016).

[4] Adam J Riesselman et al. "Accelerating Protein Design Using Autoregressive Generative Models". In: *bioRxiv* (2019), p. 757252.

[5] Thomas Shafee. Figure altered to only include primary, secondary and tertiary structure. Permitted use as by the Creative Commons Attribution 4.0 International license `https://creativecommons.org/licenses/by/4.0/deed.en`. 2016. URL: `https://en.wikipedia.org/wiki/File:Protein_structure_(full).png` (visited on 02/09/2020).

[6] Adam J Riesselman, John B Ingraham, and Debora S Marks. "Deep generative models of genetic variation capture the effects of mutations". In: *Nature methods* 15.10 (2018), pp. 816–822.

[7] Yoshua Bengio, Aaron Courville, and Pascal Vincent. "Representation learning: A review and new perspectives". In: *IEEE transactions on pattern analysis and machine intelligence* 35.8 (2013), pp. 1798–1828.

[8] Francesco Locatello et al. "Challenging Common Assumptions in the Unsupervised Learning of Disentangled Representations". In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. Long Beach, California, USA: PMLR, June 2019, pp. 4114–4124. URL: `http://proceedings.mlr.press/v97/locatello19a.html`.

[9] V. Kuleshov and S. Ermon. *Variational Inference*. 2020. URL: `https://ermongroup.github.io/cs228-notes/inference/variational/` (visited on 01/31/2020).

[10] Diederik P Kingma. "Variational inference & deep learning: A new synthesis". In: (2017). PhD Thesis. URL: `https://dare.uva.nl/search?identifier=8e55e07f-e4be-458f-a929-2f9bc2d169e8` (visited on 06/02/2020).

[11] Charles Blundell et al. "Weight uncertainty in neural networks". In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning-Volume 37*. 2015, pp. 1613–1622.

[12] P Kingma Diederik, Max Welling, et al. "Auto-encoding variational bayes". In: *Proceedings of the International Conference on Learning Representations (ICLR)*. Vol. 1. 2014.

[13] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[14] Kyunghyun Cho et al. "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2014, pp. 1724–1734.

[15] Torch Contributors. *torch.nn documentation*. 2020. URL: `https://pytorch.org/docs/stable/nn.html` (visited on 03/06/2020).

[16] Jeffrey L Elman. "Finding structure in time". In: *Cognitive science* 14.2 (1990), pp. 179–211.

[17] Ben Krause et al. "Multiplicative LSTM for sequence modelling". In: *ICLR Workshop track* (2017). URL: `https://openreview.net/forum?id=SJCS5rXFl`.

[18] Lusheng Wang and Tao Jiang. "On the complexity of multiple sequence alignment". In: *Journal of computational biology* 1.4 (1994), pp. 337–348.

[19] Victor Nordam Suadicani and Emil Petersen. *Reproduction and Evaluation of UniRep, a Deep Learning Representation of Protein Sequences*. 2020. URL: https://github.com/leetemil/unirep_project (visited on 05/14/2020).

[20] Thomas A Hopf et al. "Mutation effects predicted from sequence co-variation". In: *Nature biotechnology* 35.2 (2017), p. 128.

[21] Mohammed AlQuraishi. *The Future of Protein Science will not be Supervised*. 2015. URL: https://moalquraishi.wordpress.com/2019/04/01/the-future-of-protein-science-will-not-be-supervised/ (visited on 01/31/2020).

[22] Tim Salimans and Durk P Kingma. "Weight normalization: A simple reparameterization to accelerate training of deep neural networks". In: *Advances in neural information processing systems*. 2016, pp. 901–909.

[23] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. "Layer Normalization". In: *arXiv preprint arXiv:1607.06450* (2016).

[24] Roshan Rao et al. "Evaluating protein transfer learning with TAPE". In: *Advances in Neural Information Processing Systems*. 2019, pp. 9686–9698.

[25] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[26] Sean Eddy. "HMMER user's guide". In: *Department of Genetics, Washington University School of Medicine* 2.1 (1992), p. 13.

[27] Sean R. Eddy. "Profile hidden Markov models." In: *Bioinformatics (Oxford, England)* 14.9 (1998), pp. 755–763.

[28] Michael A Stiffler, Doeke R Hekstra, and Rama Ranganathan. "Evolvability as a function of purifying selection in TEM-1 β-lactamase". In: *Cell* 160.5 (2015), pp. 882–892.

[29] Roshan Rao et al. "Evaluating Protein Transfer Learning with TAPE". In: *Advances in Neural Information Processing Systems*. 2019.

[30] Helen M Berman et al. "The protein data bank". In: *Nucleic acids research* 28.1 (2000), pp. 235–242.

[31] John Moult et al. "Critical assessment of methods of protein structure prediction (CASP)-Round XII". In: *Proteins: Structure, Function, and Bioinformatics* 86 (2018), pp. 7–15. ISSN: 08873585. DOI: 10.1002/prot.25415. URL: http://doi.wiley.com/10.1002/prot.25415.

[32] Michael Schantz Klausen et al. "NetSurfP-2.0: Improved prediction of protein structural features by integrated deep learning". In: *Proteins: Structure, Function, and Bioinformatics* (2019).

[33] Naomi K Fox, Steven E Brenner, and John-Marc Chandonia. "SCOPe: Structural Classification of Proteins—extended, integrating SCOP and ASTRAL data and classification of new structures". In: *Nucleic acids research* 42.D1 (2013), pp. D304–D309.

[34] Mohammed AlQuraishi. "ProteinNet: a standardized data set for machine learning of protein structure". In: *BMC bioinformatics* 20.1 (2019), p. 311.

[35] Karen S Sarkisyan et al. "Local fitness landscape of the green fluorescent protein". In: *Nature* 533.7603 (2016), p. 397.

[36] Gabriel J Rocklin et al. "Global analysis of protein folding using massively parallel design, synthesis, and testing". In: *Science* 357.6347 (2017), pp. 168–175.

[37] Laurens van der Maaten and Geoffrey Hinton. "Visualizing data using t-SNE". In: *Journal of machine learning research* 9.Nov (2008), pp. 2579–2605.

[38] Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. “Variational dropout sparsifies deep neural networks”. In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, pp. 2498–2507.

[39] Durk P Kingma, Tim Salimans, and Max Welling. “Variational dropout and the local reparameterization trick”. In: *Advances in neural information processing systems*. 2015, pp. 2575–2583.

[40] Yarin Gal and Zoubin Ghahramani. “Dropout as a bayesian approximation: Representing model uncertainty in deep learning”. In: *international conference on machine learning*. 2016, pp. 1050–1059.

[41] Martin Steinegger and Johannes Söding. “Clustering huge protein sequence sets in linear time”. In: *Nature communications* 9.1 (2018), pp. 1–8.

[42] Irina Higgins et al. “beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework.” In: *Iclr* 2.5 (2017), p. 6.

[43] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems*. 2017, pp. 5998–6008.

[44] Alec Radford et al. “Language models are unsupervised multitask learners”. In: *OpenAI Blog* 1.8 (2019), p. 9.

A
Appendix

A.1 Code Availability. Results and Structure

The implementation of the models and the code used to train them can be found at the following GitHub repository: `https://github.com/leetemil/thesis`. This repository will be made public soon after the submission deadline of this thesis report (2020-06-05T12:00+0200).

At the top level, the repository is divided into the `report` directory (containing the LaTeXcode and figures for this report) and the `src` directory (containing the Python source code for the implementations of the model and their training code). The top level also contains the thesis project contract. The report directory should be self-explanatory.

The source code directory is divided into multiple directories:

args: Code for extracting command-line arguments, and collections of standard arguments for the mutation effect prediction datasets.

data: Code for protein data handling, as well as the data itself.

models: Code for each model presented in the thesis.

slurm: Scripts for running slurm jobs on the DIKU compute cluster.

training: Generalized training infrastructure.

visualization: Code for figure generation and visualization of results.

At the top level, there are `run_*.py` scripts, which run the training procedure for the various models. The `tape_train.py` and `tape_eval.py` files are used to train and evaluate the TAPE tasks.

A.2 Results

The full results can be found here: `https://bit.ly/2XsoN4C`

A.3 Learning Objectives

In the project contract for this thesis, we outlined the following learning objectives:

1. Present the theory behind variational autoencoders and deep representation learning.
2. Survey similar approaches (such as [2]) within the field of representation learning on protein sequences and discuss how they relate to the presented theory.
3. Explore the theoretical strengths and weaknesses of model architectures. In addition, discuss the trade-offs between the latent spaces of different models.
4. Design, implement and evaluate representation learning models on protein sequences, using variational autoencoders. Argue for the underlying design and implementation choices and analyze the performance.
5. Discuss how a well-performing representation learning model on protein sequences can be used for exploring new proteins and their properties, and other potential applications, if any.

Since the contract was written, the project has evolved to include more models than just the variational autoencoder, but suffice to say that we believe that we have lived up to the learning objectives.

A.4 External obstacles

During this thesis, we ran into some unfortunate external obstacles which, while not relating to the thesis directly, had an impact on the amount and quality of work that we could put into it.

A.4.1 Changing thesis rules at the UCPH

The rules on start and end dates of theses at the University of Copenhagen were in the process of being changed when we started our thesis. Traditionally, theses would run for 6 uninterrupted months. In the future, this will change to 4 months. As far as we have gathered, this is to make the theses end before the summer vacation, because this means the students will finish within the ministry's guidelines, thus prompting a funding bonus (this is our understanding, but it seems rather complicated).

During a transition period from the old rules to the new rules, the theses duration would stay at 6 months, but be moved 2 months back, thus ending before the summer vacation. However, this meant that the first two months of our thesis was spent while we still had other university courses. Suffice to say that we were not as productive on the thesis during these 2 months as we would otherwise have been. We did however complete a project outside of course scope about the UniRep model [19], which did help us prepare for the thesis regardless of this rule change.

A.4.2 COVID-19 pandemic

On 13th of March 2020, about half-way into our thesis period, the Danish government shut down most non-essential public institutions, including the University of Copenhagen. This meant that we could not meet physically at the university with each other or to have meetings with our supervisor. Other potential meeting places, such as libraries, were also closed, forcing us to work from home, often separately with communication only over (occasionally video) calls.

While a thesis such as ours (where the "product" just requires a computer) is probably least impacted by the pandemic in comparison to other theses (which may require a high-grade laboratory for example), we found that working in this manner decreased our productivity and motivation, and we will not be able to return to our normal working schedule before the thesis is over.

The university offered to extend the thesis with up to a month, but we did not wish to delay the completion of our education.