12-2018

Exploring the Impact of Pretrained Bidirectional Language Models on Protein Secondary Structure Prediction

Dillon G. Daudert

EXPLORING THE IMPACT OF PRETRAINED BIDIRECTIONAL LANGUAGE MODELS ON PROTEIN SECONDARY STRUCTURE PREDICTION

by

Dillon G. Daudert

A thesis submitted to the Graduate College
in partial fulfillment of the requirements
for the degree of Master of Science
Computer Science
Western Michigan University
December 2018

Thesis Committee:

Ajay Gupta, Ph.D., Chair
James Springstead, Ph.D.
Jinxia Deng, Ph.D.

EXPLORING THE IMPACT OF PRETRAINED BIDIRECTIONAL LANGUAGE MODELS ON PROTEIN SECONDARY STRUCTURE PREDICTION

Dillon G. Daudert, M.S.

Western Michigan University, 2018

Protein secondary structure prediction (PSSP) involves determining the local conformations of the peptide backbone in a folded protein, and is often the first step in resolving a protein's global folded structure. Accurate structure prediction has important implications for understanding protein function and de novo protein design, with progress in recent years being driven by the application of deep learning methods such as convolutional and recurrent neural networks. Language models pretrained on large text corpora have been shown to learn useful representations for feature extraction and transfer learning across problem domains in natural language processing, most notably in instances where the amount of labeled data for supervised learning is limited. This presents the possibility that pretrained language models can have a positive impact on PSSP, as sequenced proteins vastly outnumber those proteins whose folded structures are known.

In this work, we pretrain a large bidirectional language model (BDLM) on a nonredundant dataset of one million protein sequences gathered from UniRef50. The outputs and intermediate layer activations of this BDLM are incorporated into a bidirectional recurrent neural network (RNN) trained to predict secondary structure. We provide an empirical assessment of the impact the representations learned by our pretrained model have on PSSP and analyze the mutual information of these representations with secondary structure labels.

Contents

List of Tables

List of Figures

Chapter 1

Overview and Background

1.1 Introduction

The behavior and function of a protein in its environment is determined by its 3D structure, so knowing that structure is a first step toward learning the function of a newly discovered protein [2]. Furthermore, having a robust understanding of the relationship between structure and function is a prerequisite for designing novel protein-based medications and therapies. [3].

Experimental techniques for resolving 3D structure, such as X-ray crystallography and nuclear magnetic resonance (NMR) imaging, are orders of magnitude more expensive than sequencing entire genomes ($100,000 [4] vs. $1000 [5]). As such, the number of proteins whose sequences are known but whose 3D structures are not is increasing rapidly, demonstrating the value of predicting structure from sequence information alone.

Recently, the incorporation of deep learning methods has led to advances in many areas of bioinformatics, including subcellular localization [6], toxicity prediction [7], as well as contact map prediction [8], and has had a notable impact on protein secondary structure prediction [9].

Determining structure involves a pipeline of computational tasks, and predicting secondary structure is a critical first step [10]. Techniques such as generative stochastic networks [1], convolutional neural fields [11], and long short-term memory recurrent neural networks [12] [13] have all been shown to improve performance on protein secondary structure prediction.

1.1.1 Research objectives

This thesis is motivated by three observations:

- the vast amount of protein sequence data now available represents an opportunity to leverage it to improve tasks that have more limited data (namely protein structures),
- deep neural networks, having already been established as effective at predicting secondary structure, scale well with the amount of data on which they are trained [14], and
- recent work suggests that pretrained language models are effective for representation learning in natural language processing.

With these in mind, we propose the following objectives:

1. Determine if unsupervised pretraining is suitable for protein structure prediction
2. Explore how one unsupervised pretraining method for recurrent neural networks using language models impacts secondary structure prediction

1.1.2 Thesis structure

In the rest of Chapter 1, we cover the basics of structural biology, structure prediction, machine learning, and deep learning. Chapter 2 states the protein secondary structure prediction problem formally and presents our model architecture, experiments, results, and discussion. Chapter 3 concludes the thesis.

1.2 Structural Biology

1.2.1 Proteins: primary, secondary, and tertiary structure

Proteins are an important class of biological macromolecules that occupy a wide range of critical roles in biological processes, including catalyzing reactions, cell signaling, and immune responses. They are comprised of linear chains of smaller molecules called amino acids, ranging in length from only a few residues to several thousands.

Amino acids consist of an amine group (N terminal), a carboxyl group (C terminal), and an R group, or side-chain, that is specific to each amino acid. There are 20 common proteinogenic

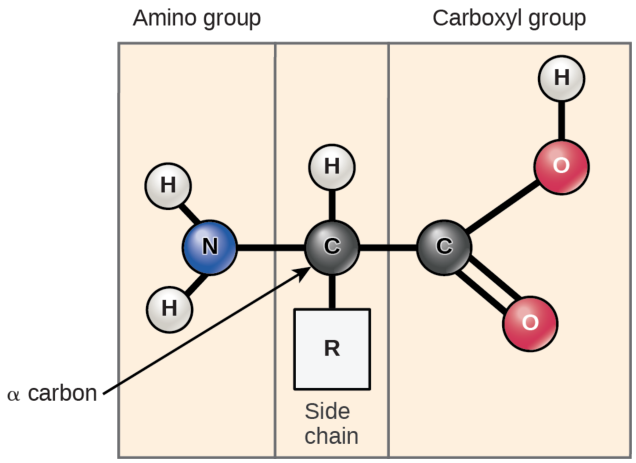


Figure 1.1: Molecular representation of an amino acid. Image from OpenStax Biology 2e / CC BY 4.0

Table 1.1: Secondary structure assignments as described by DSSP.

Q8 Label	Description
H	α-helix
E	β-strand
T	Turn
S	Bend
G	3_{10} helix
B	β-bridge
I	π-helix
L	Unknown or loop / irregular

amino acids that form peptide bonds between their N and C terminals, and a chain of multiple amino acids is called a polypeptide chain. The unique string of amino acid residues of a protein is referred to as the primary, or 1D, structure of that protein. The majority of known proteins undergo a process whereby the polypeptide backbone folds into a stable 3-dimensional conformation, known as the protein's native state. During this process, local subregions along the chain organize in regular patterns called secondary structures, two of the most common of which being alpha-helices and beta-sheets.

Secondary structures are classified by their hydrogen bonds; for example, the helical shape characterizing α-helices arises from the side chains of amino acids 4 residues apart forming a hydrogen bond. One standard classification of secondary structures is given by the dictionary of secondary structures of proteins (DSSP) , which defines 8 structural patterns [15]:

These local structures further organize into a global conformation, referred to as the tertiary structure. Understanding the folding mechanisms that determine what the tertiary structure for

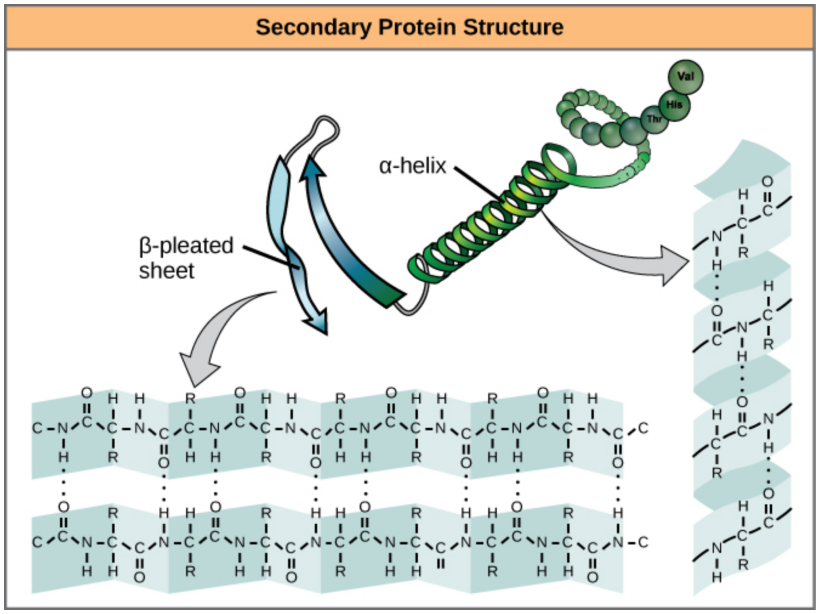


Figure 1.2: The two most common types of protein secondary structure, α-helices and β-sheets. The dots indicate hydrogen bonds between amino acid side chains. Image from OpenStax Biology 2e / CC BY 4.0

a protein is, as well as the temporal process of going from unorganized to folded, has been a fundamental challenge in biology for nearly 60 years.

1.2.2 Function and homology

Homology refers to the evolutionary relationships between biological structures - both on the macroscale and the molecular scale. For proteins, two or more proteins are homologous if they evolved from a common ancestor. Homology can be divided into two categories depending on the nature of the evolutionary relationship: *orthologs* are homologous proteins in two different species that diverged from a common ancestor (speciation), whereas *paralogs* are proteins that diverged from a common ancestor within the same species. It is also possible for two proteins in different species to have similar functions but no evolutionary relationship, in which case these are called *analogs*. Homology relationships are important when determining structure and function as evolutionarily related proteins often have similar 3D structures, even when their sequences are very dissimilar. This is due to the fact that evolutionary processes change the structure of orthologous proteins much more slowly than the amino acid sequence [16]; selective pressure on the descendant protein in each species to fulfill the function of the original. The situation for paralogs is more complicated, as only one descendant must retain its original function, whereas the other may evolve to

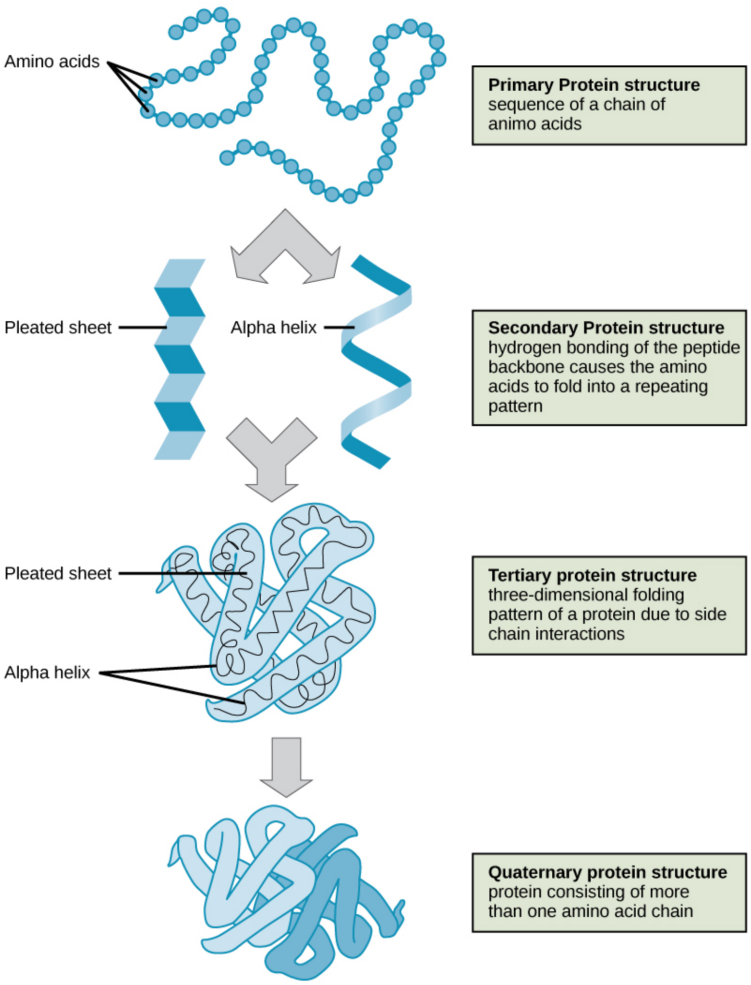


Figure 1.3: The hierarchy of protein structure. Quaternary structure has to do with the structural organization of collections of disjoint proteins. Image from OpenStax Biology 2e / CC BY 4.0

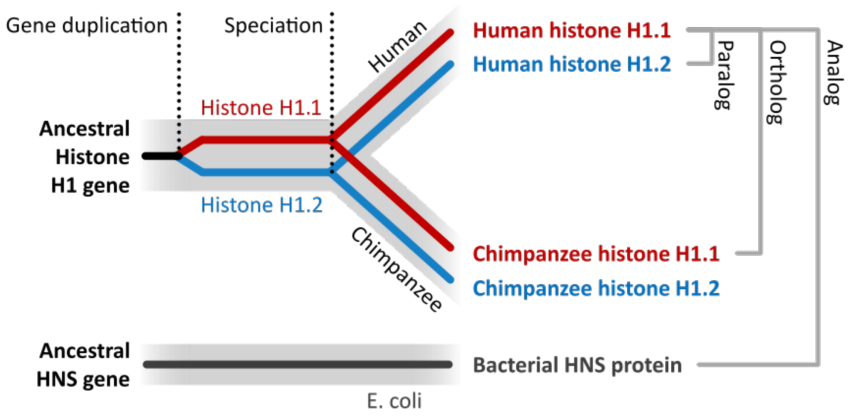


Figure 1.4: The evolution of the H1 histone gene in humans (red and blue). Two events split the gene into homologs: gene duplication within the same species creates two genes which are paralogs; speciation results in two copies of each gene, so these are orthologs. The HNS gene in E. Coli has no evolutionary relationship with H1, but serves a similar function as the H1 gene, so this is an analog. Sequence homology by Thomas Shafee / CC BY 4.0

A	C	C	A	T	G	-	T	T
	|	|	|		|		|	|
-	C	C	A	A	G	T	T	T

Figure 1.5: Pairwise alignment of two DNA sequences. Magenta columns indicate an insertion or deletion, and the cyan column indicates a mismatch.

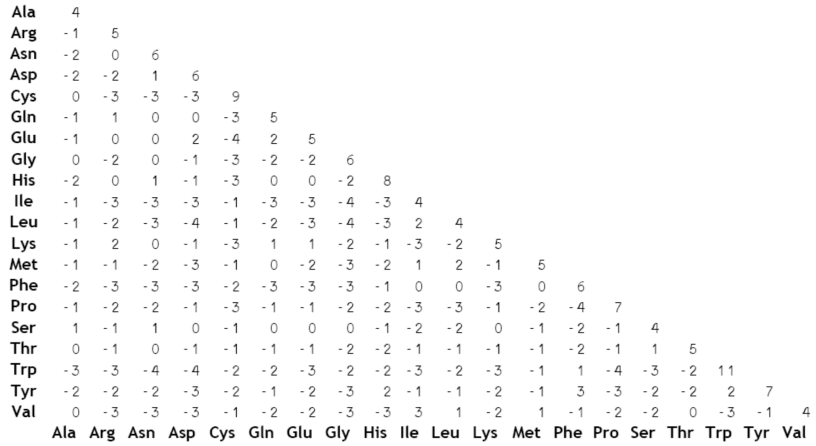
have entirely new functions.

Determining homology involves establishing that two proteins are evolutionarily related, however the reconstruction of this evolutionary tree is impossible in most cases. Instead, homology is approximated by calculating the similarity of two or more protein sequences via global or local alignment methods. These methods can be a powerful tool for discovering related sequences and structures, but also have some limitations.

1.2.3 Sequence alignment and identity

Approximating homology consists of finding an alignment of two sequences of characters that maximizes the number of matching positions, and then calculating the likelihood that those sequences are related.

Sequence alignment is closely related to the idea of string similarity. One way to measure how similar two strings are is through the Levenshtein distance [17], which counts the number of insertions, deletions, and mismatches between two strings. This is a kind of edit distance, as it



	Ala	Arg	Asn	Asp	Cys	Gln	Glu	Gly	His	Ile	Leu	Lys	Met	Phe	Pro	Ser	Thr	Trp	Tyr	Val
Ala	4																			
Arg	-1	5																		
Asn	-2	0	6																	
Asp	-2	-2	1	6																
Cys	0	-3	-3	-3	9															
Gln	-1	1	0	0	-3	5														
Glu	-1	0	0	2	-4	2	5													
Gly	0	-2	0	-1	-3	-2	-2	6												
His	-2	0	1	-1	-3	0	0	-2	8											
Ile	-1	-3	-3	-3	-1	-3	-3	-4	-3	4										
Leu	-1	-2	-3	-4	-1	-2	-3	-4	-3	2	4									
Lys	-1	2	0	-1	-3	1	1	-2	-1	-3	-2	5								
Met	-1	-1	-2	-3	-1	0	-2	-3	-2	1	2	-1	5							
Phe	-2	-3	-3	-3	-2	-3	-3	-3	-1	0	0	-3	0	6						
Pro	-1	-2	-2	-1	-3	-1	-1	-2	-2	-3	-3	-1	-2	-4	7					
Ser	1	-1	1	0	-1	0	0	0	-1	-2	-2	0	-1	-2	-1	4				
Thr	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	1	5			
Trp	-3	-3	-4	-4	-2	-2	-3	-2	-2	-3	-2	-3	-1	1	-4	-3	-2	11		
Tyr	-2	-2	-2	-3	-2	-1	-2	-3	2	-1	-1	-2	-1	3	-3	-2	-2	2	7	
Val	0	-3	-3	-3	-1	-2	-2	-3	-3	3	1	-2	1	-1	-2	-2	0	-3	-1	4

Figure 1.6: BLOSUM62 by Hannes Röst / CC BY-SA 3.0

captures how much one string must be edited to be turned into another. Typical sequence alignment algorithms are more complex than the Levenshtein distance, however, and weight edits to a protein or DNA string differently based on statistical measures as well as a priori knowledge about the evolutionary and mutation process.

An example of statistical weighting is the BLOSUM matrix (meaning BLOcks SUbstitution Matrix), which weights the cost of pairwise mutations of amino acids in protein sequences. BLOSUM is calculated by counting the frequency of mutations between proteins in highly conserved regions (regions where alignments do not have gaps (insertions/deletions)).

Note that unlike the Levenshtein distance, for which higher values mean more dissimilar strings, alignments are usually interpreted as measures of similarity, with a higher alignment score indicated a higher degree of similarity. This is the case for BLOSUM, where higher values indicate higher likelihood of a particular pairwise mutation.

Sequence identity is calculated by counting the number of exact matches in the optimal alignment of two sequences. For protein sequences, high sequence identity (>40%) unambiguously determines similar structure. However, as identity decreases below this threshold, identifying homology relationships becomes increasingly unreliable, up until below the 20% threshold, where homology is practically impossible to determine without direct analysis of the protein structure[18].

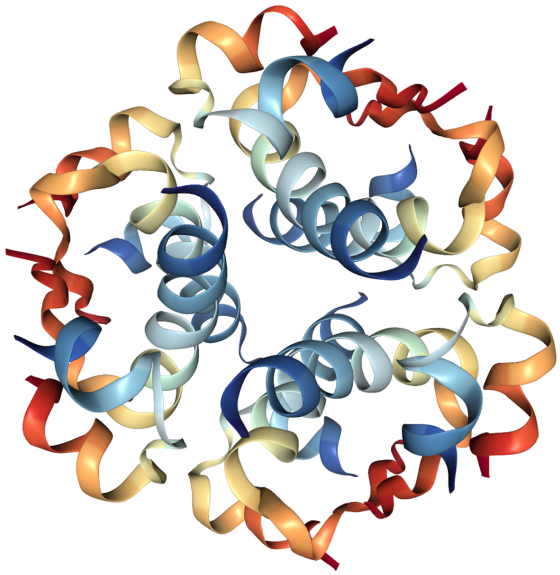


Figure 1.7: Image from the RSCB PDB of PDB ID: IZNJ (DOI 10.2210/pdb1ZNJ/pdb)

1.2.4 Folding and structure prediction

Determining the native structure of a protein in solution is equivalent to solving for how that protein will fold. Research into the protein folding problem spanning over 50 years has brought a considerable amount of insight into the mechanisms that dictate how this process takes place, which physical and chemical interactions have the most influence, as well as how information about the structure of a protein is encoded in its amino acid sequence. Extensive overviews of the progress that has been made can be found in [19] and [2]. In the following paragraphs, we share some major ideas that are relevant to our work.

In general, proteins fold quickly to their native state, although environmental factors can prevent or undo this process (protein denaturing), such as extreme temperatures or pH. Furthermore, some proteins utilize other molecules called chaperones in order to avoid premature misfolding during synthesis or undesirable aggregation [20]. Folding is thought to be hierarchical to some degree: secondary structures fold to transient local conformations which are then stabilized by the coalescing tertiary structure [21].

Early on, it was suspected that the native state of a protein was that state with the lowest

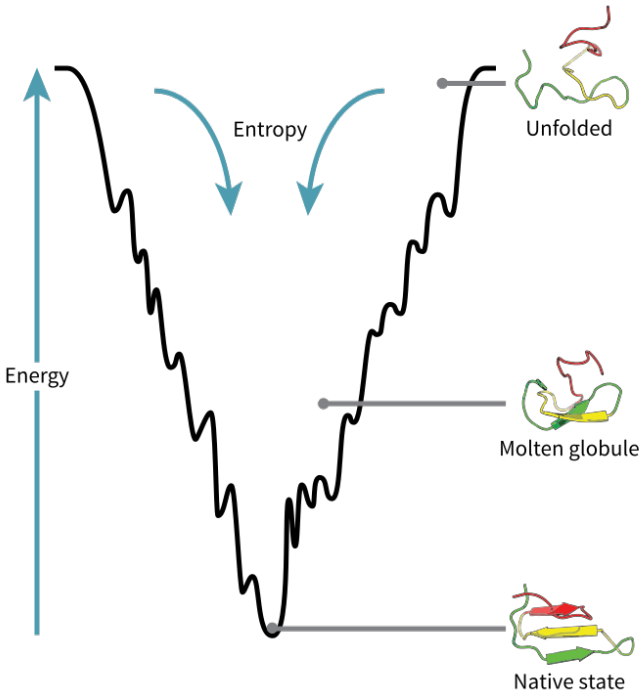


Figure 1.8: Visualization of a protein energy landscape. Image by Thomas Splettstoesser / CC BY-SA 3.0

free energy. The question of how proteins explore the exponentially large space of possible conformations to find the state with the lowest energy so quickly is referred to as Levinthal's paradox. While originally it was thought that proteins might fold along specific pathways to their native state, the common belief now is that folding takes place along funnel-shaped energy surfaces [22].

As described previously, the particular arrangement of a polypeptide backbone in its secondary and tertiary structures arises due to hydrogen bonds between the amino acid residue side chains. In light of this, one might reasonably ask if all of the information for how a protein will fold is contained in these side chains; indeed, Anfinsen's dogma[23] [1] states that protein native structure is determined solely by its amino acid sequence, although the original hypothesis only applied to globular proteins.

Our understanding of how sequence encodes structure, sometimes referred to as the protein folding code, has shown that many physical and chemical interactions play a role during folding, among these are hydrogen bonds, hydrophobic and electrostatic interactions, and van der Waals forces.

[1] Anfinsen's dogma, also called the *thermodynamic hypothesis*, states more specifically that the native structure of a polypeptide chain, when found in its typical environment, is the conformation with the lowest free energy, where the energy is a function of the interactions between amino acid side chains.

One important challenge of structural biology has been the development of computational methods able to predict protein structure. The next section covers the major existing methods, and provides context for the role of secondary structure prediction, the method on which this thesis focuses.

1.2.5 Methods of structure prediction

Protein structure prediction methods can be divided into two areas. The first involves simulating the dynamics of the folding process explicitly. The second attempts to reduce the degrees of freedom for the final conformation either by exploiting similarities to known structures when they exist - homology-based methods - or by calculating likely local conformations of the protein backbone with statistical techniques - *ab initio* methods.

Simulation

The most straightforward way for computing a protein's folded structure is to simulate the molecular dynamics of the protein as it folds directly. As mentioned before, proteins fold to the conformation with the least free energy. This energy can be modeled as an energy function, and the protein is considered folded when this energy function is minimized.

Classical physics theory models atoms as discrete masses that interact via forces. Thus, the energy of a molecular system can be described as a function of all the forces interacting on the atoms of that system. This function can be more or less complicated, but generally consists of bonded and unbonded forces:

$$E_{total} = E_{bonded} + E_{unbonded}$$

.

Bonded forces are calculated between atoms that share a covalent bond and nonbonded forces between pairs of atoms lacking a covalent bond but close enough to interact. The total energy of the system is minimized when this energy function is minimized. It is clear that if nonbonded forces are considered between all pairs of atoms in the system, then the overall complexity of calculating this energy function is of order $\mathcal{O}(n^2)$, where n indicates the number of atoms. Although

this runs in polynomial time, in general, it must be recalculated for each time step of a simulation, resulting in an overall complexity of $\mathcal{O}(t * n^2)$, where t might be in the billions. Furthermore, since these calculations typically simulate molecular dynamics within some medium, e.g. water, then the number of atoms n grows proportionally to the volume of the simulation.

Large-scale structure determination, such as the kind necessary for *de novo* protein design, might involve solving the structures of over 10,000 candidate sequences, making direct simulation of molecular dynamics impractical. However, simulation can be incorporated at the end of a structure prediction pipeline as a refinement step, where the positions of the atoms are ideally already close to their lowest energy states.[10]

Homology-based methods

Homology-based methods rely on similarities to proteins with known sequences and structures to infer the likely structure of a new protein. The power of these methods has improved considerably with the wealth of sequence and structure information contained in public repositories such as UniProt[24] and the Protein Data Bank [25, 26].

While such methods demonstrate high degrees of accuracy when similar sequences are available, their predictive power suffers when encountering a protein that has no known homologs. The usefulness of homology-guided structure prediction is particularly limited in the context of *de novo* protein design, where sequences and structures that bear no relationship to naturally occuring proteins may be of interest. This leads to the category of *ab initio* prediction methods.

Ab initio methods

Both homology-based and *ab initio* methods aim to reduce the search space of possible peptide backbone conformations. In the absence of structural information of related sequences, the search space can be reduced by finding backbone fragments with similar local sequences even if the global similarity is relatively low[27].

Since incorrect local structures lead to poor sampling, improving the accuracy of predicted secondary structure would directly improve the ability to predict native structure. As a case in point, in CASP 11, an annual competition for evaluating structure prediction methods, the greatest source of error in the top performing method came from inaccurate SS prediction and incorrect

domain parsing[10].

Role of structure prediction in *de novo* protein design

Structure prediction also serves a fundamental role in *de novo* protein design. Designing a protein with a specific structure involves computing a large number of potential backbone conformations, and for each predicting the sequence(s) with the lowest energy for that structure. Once these are found, *ab initio* structure prediction methods are used to find the structure with the lowest energy for that sequence. These predicted structures are an important validation of the designed protein, as a particular design typically won't be experimentally validated unless the computed structures converge to the designed structure[28].

Secondary structure prediction impacts other problems

Accurate secondary structure prediction also improves the prediction of a number of other problems related to protein structure, including protein function[29], solvent exposure of amino acid residues [30], and distinguishing between structured and intrinsically disordered proteins[31].

With the important role of PSSP well established, we now focus on recent methods and results that have been applied to this problem.

1.2.6 Relevant work

In this section, we give a brief description of recent work in protein secondary structure prediction that is relevant to our research.

Recurrent neural networks have become ubiquitous for structure prediction, in particular the variant long short-term memory cell. [12] used bidirectional LSTM RNNs with dense layers inserted at the hidden-to-hidden connections to predict secondary structure directly. Our replication of the architecture and experiments of this paper are included in the appendix.

[13] also employed LSTM RNNs, building on several prior works, including [32] and [33], to demonstrate that multi-task learning, and iterative learning improve the prediction of 3-state protein structure as well as solvent accessibility and torsion angles. Here, LSTM RNNs were trained to predict secondary structure, solvent accessibility, and torsion angles simultaneously, and notably showed improvements in predicting structures that arise due to non-local interactions

Table 1.2: Recent performance on protein secondary structure prediction

Paper	**Year**	**Q3 accuracy single / ensemble**	**Q3 accuracy single / ensemble**
Busia and Jaitly	2017	- / -	71.4 / -
Fang et al.	2018	- / -	70.7 / -
Heffernan et al.	2017	84.0 / -	- / -
Li and Yu	2016	- / -	69.4 / 69.7
Wang et al.	2016	82.3 / -	68.3 / -
Heffernan et al.	2015	82.0 / -	- / -
Sonderby and Winther	2015	- / -	67.4 / -
Zhou and Troyanskaya	2014	- / -	66.4 / -

between residues. This process was repeated four times: each iteration, the model was trained to convergence. Models after the first were additionally supplied with the outputs of the model trained in the previous iteration, whose weights were fixed. Both multi-task learning and iterative learning share many of the same theoretical motivations as unsupervised pretraining. Whereas unsupervised pretraining aims to learn hidden features that effectively represent the input variables x, multi-task learning finds a distributed representation of the inputs that can jointly predict different but related target variables, y and z. The similarity between these two methods is apparent once it is recognized that the auxiliary target variable, y, could also be used as an element of the input features, x. Iterative learning is even more closely related to unsupervised pretraining, in that it is nearly exactly the same procedure as greedy layer-wise pretraining [34]. In the latter case, the weights are not fixed during the final training step of the full network, whereas in the former (in this instance) an entire model is pretrained and then held fixed, not just a single layer.

Although models based on recurrent architectures handle sequence data most naturally, convolutional neural networks have also shown impressive results for predicting protein structures. Two notable examples include the inception-inside-inception network from [35] and the next-step conditioned model proposed by [36], where a CNN was trained as a language model over secondary structures. The generative stochastic network proposed in [1] also included convolutional layers, and [37] combined both recurrent and convolutional layers in their architecture. Table 1.2 lists the Q3 and/or Q8 accuracy reported in these works.

1.3 Machine Learning

Machine learning methods acquire knowledge directly from raw data, in contrast with rule-based and formal systems where knowledge is hard-coded by humans. Deep learning is a subdomain of machine learning that represents knowledge as a hierarchy of concepts, with complex high-level concepts being composed of simple, low-level concepts. The combination of these two capabilities - the ability to learn deep representations from data without direct human intervention - has led to breakthroughs in a diverse range of tasks such as image recognition [38], speech recognition [39], and machine translation [40]. There have also been many successful applications of deep learning to problems in biology and chemistry, including for predicting the activity of drug molecules [41], compound toxicity prediction [7], subcellular localization [6], and protein structure prediction.

In this section, we provide an introduction to the main ideas in machine learning and deep learning, as well as overviews of the major techniques relevant to this thesis.

1.3.1 Supervised learning

For many real-world problems, we are interested in predicting the target or outcome of an event given some value or values we can observe. This might be identifying whether an image contains a house or an airplane, or whether or not a particular protein mutation will cause disease. Supervised learning is so named because the data used to train a model to make such predictions contains information about both the inputs and outputs for each example in the dataset. By convention, we use $\mathbf{x} \in \mathbb{X}$ to indicate samples of inputs and $\mathbf{y} \in \mathbb{Y}$ to indicate the associated outputs of those samples. Interpreted probabilistically, we say that a supervised learning algorithm models the conditional distribution $p(\mathbf{y} \mid \mathbf{x})$.

Supervised learning can be subdivided into two categories, depending on the type of the target variable. When the target is categorical, supervised learning is referred to as *classification*, and the targets as class labels. For continuous targets, supervised learning is called *regression*.

1.3.2 Unsupervised learning

Unsupervised learning on the other hand encompasses tasks for which there are no targets known, either because adding that information would be prohibitively expensive or because no target is

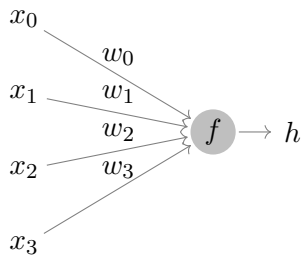


Figure 1.9: A single artificial neuron. The inputs x_i are multiplied by weights w_i, and their sum is passed through the activation function f, producing the output h.

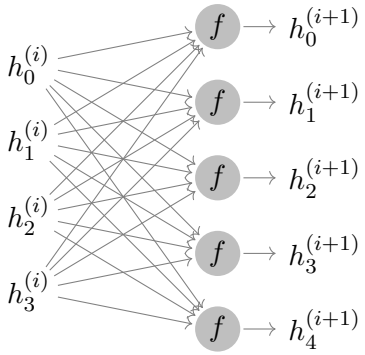


Figure 1.10: Two layers of neurons. The labels $h_j^{(i)}$ signify that these layers could be found at any level of a neural network, not necessarily just the first.

known *a priori*. An example of this latter case is clustering, where the distribution of the samples in the dataset is used to partition $\mathbb{X}$ into different categories. Commonly, the goal of unsupervised learning is to learn interesting properties of the distribution of the inputs, $p(\mathbf{x})$, and another name for unsupervised learning is knowledge discovery.

1.3.3 Deep learning

Deep learning is a subfield of machine learning wherein the primary modeling tools consists of deep artificial neural networks. Artificial neural networks consist of neurons which receive input from other neurons in previous layers and in turn produce their own output. The output of a neuron is also called that neuron's activation, or activity.

In mathematical notation, a neuron is a sum of inputs $\mathbf{x}$ multiplied by weights $\mathbf{w}$. This weighted sum is passed through an activation function, f, which is typically a nonlinear function such as tanh or the rectified linear unit (ReLU):

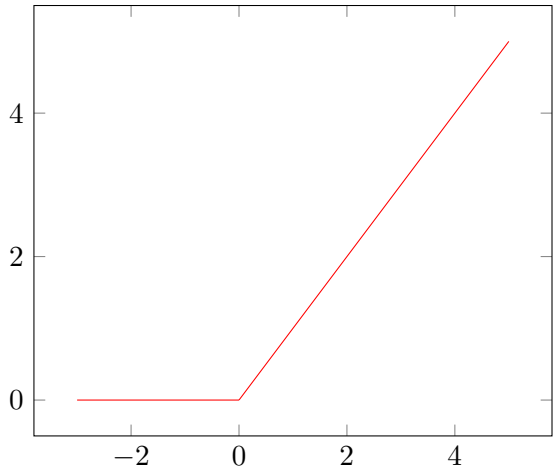


Figure 1.11: Rectified linear unit (ReLU) activation function.

$$ReLU(x) = max(0, x).$$

For a single layer neural network with n inputs and m neurons, the input-weight product can be represented by vector-matrix multiplication,

$$h = f(W\mathbf{x}),$$

where $\mathbf{x} \in \mathbb{R}^n$, $h \in \mathbb{R}^m$, and $W \in \mathbb{R}^{mxn}$.

In deep neural networks (DNNs), these neurons are stacked into multiple layers (hence "deep"), and it is precisely these layers of nonlinearities that gives DNNs their representational power.

Convolutional Neural Networks

Convolutional neural networks (CNNs) can be thought of as a restriction of "vanilla"/dense neural networks, where the input to each neuron covers only a locally connected region of the outputs of the previous layer. This *inductive bias* forces the neurons of a layer to focus on inputs that are spatially related, and ignore everything else, and is particularly well suited for inputs that consist of multiple arrays, such as images or sequences. The convolution operation also imposes a second bias on the network, translation invariance, via weight sharing. The convolution operator takes two functions f, g and returns a third function, h, which is the result of convolving f with g. If f

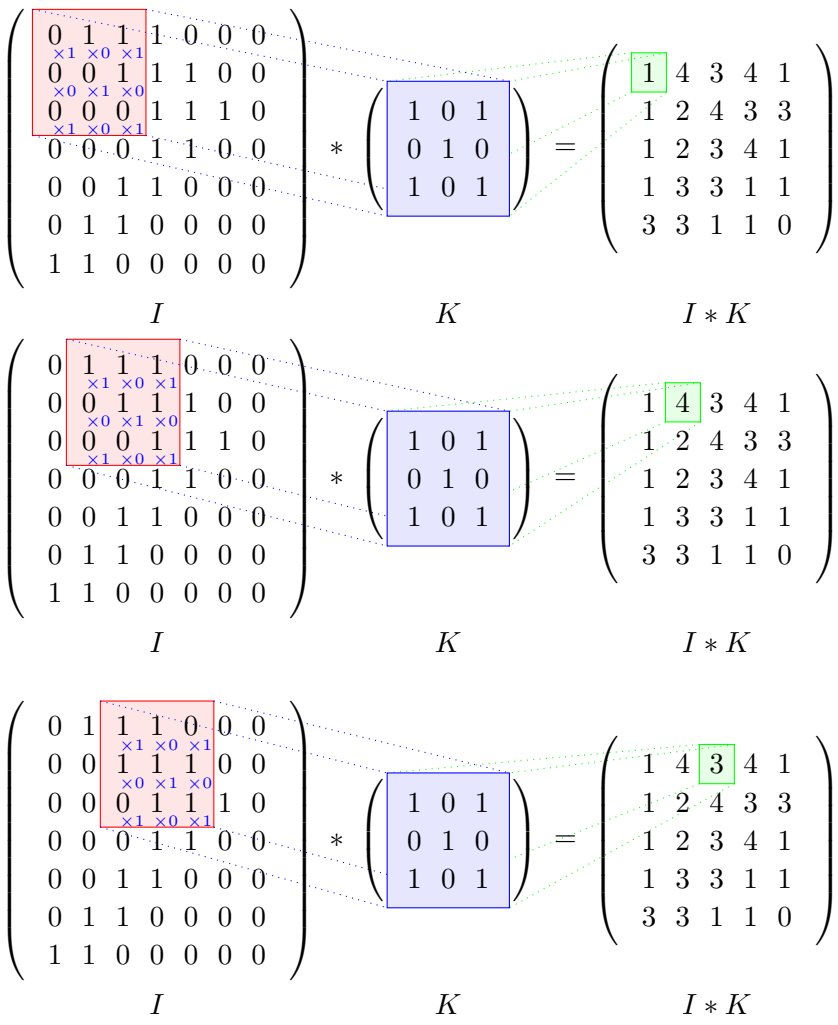


Figure 1.12: Convolution of two matrices I and K.

and g are defined over the set of integers, then h is given by

$$h(t) == \sum_{\tau=-\infty}^{\infty} f(\tau)g(t-\tau).$$

We can think of t as the offset of g, and the summation calculates the global similarity of the two functions at that offset. One way to visualize what this operation does is to imagine g "sliding" over f. When f and g resemble one another, then the magnitude of $h(t)$ will increase. Commonly, g is referred to as a filter in computer vision, which comes from the idea that a filter will pick out some important feature when convolved with an input image.

In CNNs, g is instead a matrix of trainable parameters rather than fixed *a priori*. The

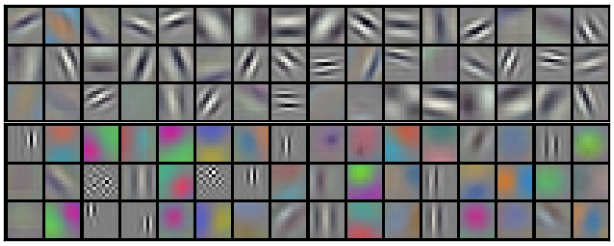


Figure 1.13: Visualization of the first layer of filters learned by AlexNet [38]. This layer learns to distinguish various kinds of edges and color transitions.

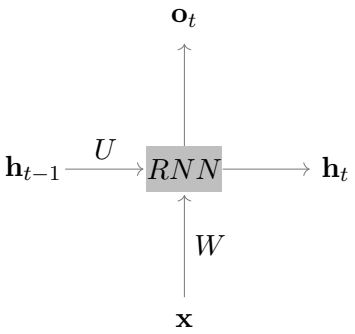


Figure 1.14: The graph showing how the inputs and previous state are related for a recurrent neural network. Note that in the basic RNN, $\mathbf{h}_t$ and $\mathbf{o}_t$ are the same.

idea is that, during training, the filters will learn a set of weights that recognize features of their inputs that are important for predicting the correct output. When multiple convolutional layers are stacked together, they also learn to compose these features into higher-level representations.

For images, the features learned by the initial CNN layers may be simple edge detectors. For 1D sequences, these filters might learn to recognize k-grams (where k-grams might be of words, letters, amino acids, etc.).

Recurrent Neural Networks

Another variation of the dense neural network that is particularly well-suited to sequence data is the recurrent neural network (RNN). The basic RNN resembles a dense neural network with an added connection that links its output to the input of the next time step. The advantage of RNNs over non-recurrent architectures is that this self-connection lets the network remember inputs it has seen previously.

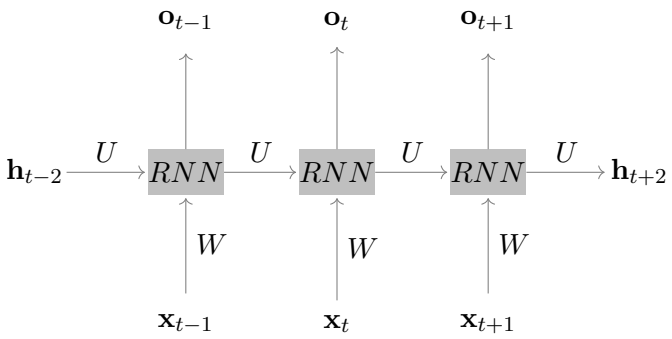


Figure 1.15: An unrolled RNN, showing computation across multiple time steps.

The algebraic formulation of the RNN is as follows:

$$\mathbf{h}_t = \phi(W\mathbf{h}_{t-1} + U\mathbf{x}_t + b),$$

where ϕ is the activation function (logistic sigmoid, tanh, ReLU, etc), $\mathbf{h}_t \in \mathbb{R}^n$ is the current state (and output), $\mathbf{h}_{t-1} \in \mathbb{R}^n$ is the previous state, $\mathbf{x}_t \in \mathbb{R}^m$ is the current input, $W \in \mathbb{R}^{n\times n}$, $U \in R^{m\times n}$,and $\mathbf{b} \in \mathbb{R}^n$ are the weights and biases, and n and m are the state and input sizes.

Theoretically, RNNs are very powerful computational devices, and can learn any computable function [42]. However, in practice training RNNs has proved to be very difficult due to the way that the information flow changes over time [43] [44].

One prominent example is that of *information morphing*. This idea has to do with the fact that at each step, the recurrent state must undergo a nonlinear transformation from one representation to another. Consider an RNN that wishes to maintain its state when it receives no new inputs at a given time step; or $F(x) = (W\mathbf{h}_{t-1} + \mathbf{b})$ is the identity function with respect to $\mathbf{h}_{t-1}$. This is impossible, as is a nonlinear function. Residual connections in deep neural networks were motivated by a similar observation in [45].

Another problem when training RNNs via gradient descent is that of exploding and vanishing gradients. From the formulation of the basic RNN, we can see that the state at each time step is multiplied by the weight matrix W. As the number of time steps grows, the gradient of the error with respect to this weight matrix will either shrink or grow without bound (vanish or explode, respectively). This is due to the fact that the gradient of output $\mathbf{h}_t$ with respect to output $\mathbf{h}_{t-k}$ includes a term of W^k. If W is square with eigenvalues $\lambda > 1$, it is easy to see that the norm

of the gradient will grow as k increases; similarly, if the eigenvalues of W are <1, the gradient will vanish [43]. Both cases pose problems for training RNNs, because when gradients explode, we can't train our models, and when gradients vanish, we can't learn long-range dependencies as the model becomes too sensitive to recent distractions.

The long short-term memory cell (LSTM) [46] was devised to address both the issue of information morphing and gradient magnitudes.

Long short-term memory

Much of the difficulty in training RNNs arises from the nonlinear matrix multiply that occurs at every time step. The LSTM avoids this by keeping a separate *memory cell*, C_t, that flows directly from one step to the next, with only element-wise updates to its state. For a particular input $\mathbf{x}_t$ and previous output $\mathbf{h}_{t-1}$, the LSTM calculates four gates:

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \tag{1.1}$$

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \tag{1.2}$$

$$\hat{C}_t = \phi(W_C[h_{t-1}, x_t] + b_C) \tag{1.3}$$

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{1.4}$$

$$C_t = f_t \otimes C_{t-1} + i_t \otimes C_t, \tag{1.5}$$

where σ is the logistic sigmoid function, ϕ is the hyperbolic tangent function (tanh), $h_t \in \mathbb{R}^n$ is the current hidden state and output, $h_{t-1} \in \mathbb{R}^n$ is the previous hidden state, $C_t \in \mathbb{R}^n$ is the cell state, $x_t \in \mathbb{R}^m$ is the current input, $W \in \mathbb{R}^{n+m \times n}$ and $b \in \mathbb{R}^n$ are the weights and biases of the gates, n and m are the state and input sizes, and $\otimes$ is the Hadamard product.

The forget gate, f_t, determines how much of the previous cell state should be forgotten the input gate, i_t, weights how much current input (called the candidate write, $\hat{C}$) should be added to the state, and the output gate, o_t, determines how much of the cell state will be "revealed" to produce the next output, h_t:

$$h_t = o_t \otimes \phi(C_t). \tag{1.6}$$

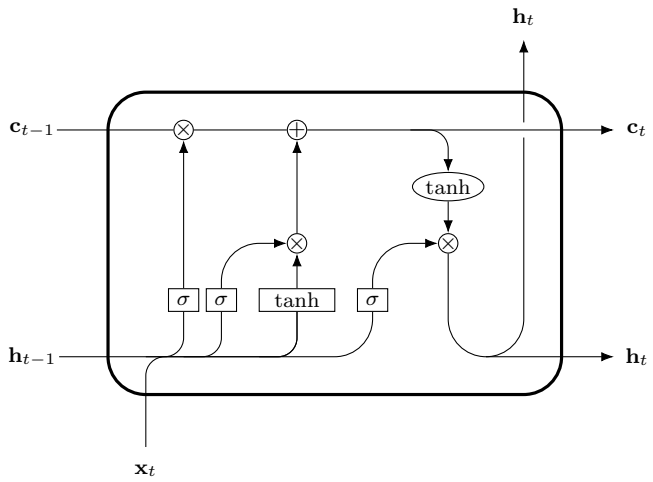


Figure 1.16: Long short-term memory cell.

LSTMs have become the *de facto* standard RNN architecture due to their ability to learn long-term dependencies. In point of fact, all of the models based on RNNs in our review of the PSSP literature used the LSTM cell, with the exception of one. [2]

1.3.4 Unsupervised pretraining

Unsupervised learning tries to learn something meaningful about the shape of the input distribution, often with the goal of leveraging unlabeled data to improve generalization on a supervised learning task [48]. In some instances, this procedure has been shown to act as a regularizer and a parameter initializer, and scales well with the amount of unlabeled data [49]. Regularization by unsupervised pretraining is also effective when combined with other regularization techniques [50]. One hypothesis for how unsupervised pretraining acts as a regularizer is that it forces the network to discover features that are relevant to the underlying causes for the observed data. Another way to state this is to say that we make the assumption that the *causal factors* of the input variables $\mathbf{x}$ are meaningfully related to the output variables $\mathbf{y}$, and so training a model to learn $p(\mathbf{x})$ will also help when learning $p(\mathbf{y} \mid \mathbf{x})$.

Supervised learning using deep neural networks can be powerful in the presence of very large amounts of labeled data. However, for many tasks the amount of unlabeled data far exceeds the amount with labels. In this latter case, unsupervised pretraining offers a pathway to improve

[2] [37] used a simplified variant of the LSTM called the Gated Recurrent Unit (GRU) [47] which ties the input and forget gates in a single "update" gate and merges the cell and hidden states

performance on supervised tasks. The next section covers several recent methods for pretraining recurrent neural networks.

Pretraining RNNs

Recurrent neural networks are a natural choice for sequence data, and several different unsupervised learning techniques for recurrent networks have been shown to improve performance on sequence prediction tasks.

[51] used next-step prediction and sequence autoencoding to pretrain long short-term memory recurrent networks in NLP, where this improved stability and generalization when later fine-tuned on various text classification tasks. Here, pretraining LSTM RNNs on a large corpus of unlabeled data allowed the authors to train models that outperformed a number of other models that had access to more labeled data.

Next-step prediction is commonly used in language modeling (LM), either at the character or word level. A character-level language model models the probability of the character x_{i+1} given inputs x_1, x_2, ..., x_i, that is, $p(x_{i+1} \mid x_{<=i})$ for an input string $x = [x_1, x_2, \ldots, x_T]$.

An autoencoder consists of an encoder $f(.)$ and a decoder $g(.)$ whose composition approaches the identity function: $g(f(x)) \approx x$. A sequence autoencoder is a type of *seq2seq* model [52] [47], where the input and output sequences are the same.

Skip-thought vectors [53] are an unsupervised learning algorithm in NLP where a recurrent network is given an input sentence and learns to predict both the preceding and following sentences. This objective forces the model's representation of a sentence to capture important semantic content by relating sentences to the context in which they're found.

[54] used pretrained bidirectional language models to learn context-sensitive word embeddings. Their bidirectional language model consisted of two unidirectional LMs that were trained separately, one in the forward direction and one in the backward direction, and then combined. These embeddings were then shown to improve performance on two natural language sequence labeling tasks. We take inspiration from this application of language modeling to sequence prediction for our own experiments, discussed in the next chapter.

Chapter 2

Methodology and Experiments

2.1 Problem Description

We express protein secondary structure prediction as a sequence transduction problem. Let $\mathbb{X}$ be the set of amino acid labels and $\mathbb{Y}$ a set of secondary structures labels, for example as determined by DSSP. The sets of all possible strings of amino acids and structures are then $\mathbb{X}^\star$ and $\mathbb{Y}^\star$, respectively. We assume each residue in the protein backbone corresponds to only one secondary structure, that is if $\mathbf{x} = [x_1, x_2, \ldots, x_n] \in \mathbb{X}^\star$, then there exist $\mathbf{y} = [y_1, y_2, \ldots, y_n] \in \mathbb{Y}^\star$ and a pairing $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ that together uniquely determine the secondary structure of $\mathbf{x}$. The goal is thus to find a map $f : \mathbb{X}^\star \mapsto \mathbb{Y}^\star$ such that $f(\mathbf{x}) = \mathbf{y}$. This can be seen as a search over the set of all possible maps; for a protein of length n, there are 8^n possible assignments of secondary structure labels.

2.2 Methods

2.2.1 Bidirectional RNNs

Secondary structure is influenced by interactions between all residues of a protein [55]. To properly capture information on both sides of each residue, we model f with a bidirectional recurrent neural network (BDRNN), $\mathcal{K}$, where

$$\mathcal{K}(\mathbf{x}) = \text{argmax}_{\hat{\mathbf{y}}} p(\hat{\mathbf{y}} \mid \mathbf{x}) \approx \mathbf{y}.$$

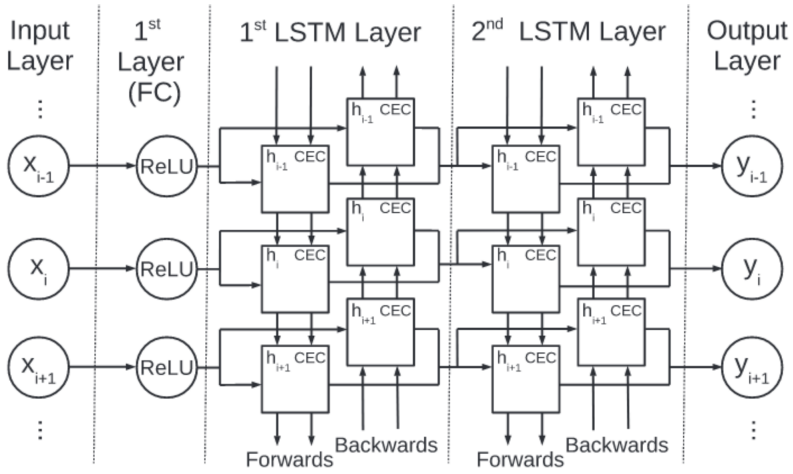


Figure 2.1: A bidirectional RNN. Image from [56] obtained with permission.

2.2.2 Language models

To investigate the impact that unsupervised pretraining has on this task, we pretrain a separate RNN as a language model on amino acids. A unidirectional language model estimates the probability of a sequence as the product of the conditional probabilities of each element of the sequence conditioned on the preceding elements. That is,

$$p(\mathbf{x}) = p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_1, x_2) \dots p(x_n \mid x_1, x_2, \dots, x_{n-1}) = \prod_{i=1}^{n} p(x_i \mid x_{j<i}).$$

2.2.3 Bidirectional language models

Our language model, $\mathcal{L}, \mathcal{L}(\mathbf{x}) = \text{argmax}_{\hat{\mathbf{x}}} p(\hat{\mathbf{x}}) \approx \mathbf{x}$, is also bidirectional in order to incorporate both the preceding and following contexts of the protein:

$$p(\mathbf{x}) = p(x_1 \mid x_2, \dots, x_n)p(x_2 \mid x_1, x_3, \dots, x_n) \dots p(x_n \mid x_1, \dots, x_{n-1}) = \prod_{i=1}^{n} p(x_i \mid x_{j \neq i}).$$

2.2.4 Objective function

Since both the language model $\mathcal{L}$ and the structure model $\mathcal{K}$ are predicting sequences of discrete categories, we measure their performance using the categorical cross entropy loss:

$$H(\mathcal{L}(x), x) = -\sum_{i=1}^{n} p(x_i) \log \mathcal{L}(x)_i, \text{ and}$$
$$H(\mathcal{K}(x), y) = -\sum_{i=1}^{n} p(y_i) \log \mathcal{K}(x)_i.$$

2.3 Representation and Features

Both amino acids and secondary structures can be represented using a one-hot encoding, which is a binary vector of n elements where exactly 1 element, indicating the discrete category of that vector, takes the value 1 and the rest are 0. There are 20 common proteinogenic amino acids and these, along with an additional element representing "Unknown" and two elements for tokens denoting both the beginning and the end of a sequence, are encoded into a one-hot vector $\mathbf{a}_c \in \mathbb{R}^{23}$.

The one-hot encodings are supplemented with physicochemical features of each amino acid. These physicochemical properties are to greater or lesser extents involved in the folding process [2, 57], and have been included as features in other work on PSSP. In determining which properties to include, we began with the following: hydrophobicity, polarity, hydropathy index [58], hydrophilicity, isoelectric point[1], van der Waals volume, dissociation constants of the carboxyl and amine groups, graph shape index [2] (steric), and polarizability [59]. (Table 2.1)

We removed any feature sharing an absolute correlation coefficient greater than 0.6 with any other feature to reduce the amount of redundant information. With the exception of hydrophobicity, we normalized these features so that they had zero mean and unit variance, resulting in the final features in Table 2.2. The physicochemical features are thus a vector of 7 real values $\mathbf{a}_f \in \mathbb{R}^7$.

In our experiments, the input vectors consisted of both the one-hot encoded amino acid $\mathbf{a}_c$ and the physicochemical properties of that amino acid $\mathbf{a}_f$ concatenated together. For the supervised dataset CPDB described in the next section, position-specific similarity matrices were also fed to

[1] Isoelectric point is the acidity at which a molecule has a neutral electric charge.

[2] The graph shape index roughly captures the properties of each side chain arising from steric effects, a particular interaction between non-bonded atoms within a molecule. These are calculated from the molecular graph of each residue.

Table 2.1: Raw physicochemical features

	π_1	$pole$	η	π_2	$pH(l)$	υ_v	pK_1	pK_2	Ξ	α
A	0.0	0.0	1.8	3.0	6.01	67.0	2.35	9.87	1.28	0.05
C	1.0	-1.0	2.5	-1.0	5.05	86.0	1.92	10.7	1.77	0.13
D	2.0	1.0	-3.5	3.0	2.85	91.0	1.99	9.9	1.6	0.11
E	2.0	1.0	-3.5	3.0	3.15	109.0	2.1	9.47	0.0	0.15
F	1.0	-1.0	2.8	-2.5	5.49	135.0	2.2	9.31	2.94	0.29
G	0.0	0.0	-0.4	0.0	6.06	48.0	2.35	9.78	0.0	0.0
H	-1.0	1.0	-3.2	-0.5	7.6	118.0	1.8	9.33	2.99	0.23
I	1.0	-1.0	4.5	-1.8	6.05	124.0	2.32	9.76	4.19	0.19
K	2.0	1.0	-3.9	3.0	9.6	135.0	2.16	9.06	1.89	0.22
L	1.0	-1.0	3.8	-1.8	6.01	124.0	2.33	9.74	2.59	0.19
M	1.0	-1.0	1.9	-1.3	5.74	124.0	2.13	9.28	2.35	0.22
N	-2.0	1.0	-3.5	0.2	5.41	96.0	2.14	8.72	1.6	0.13
P	-1.0	0.0	1.6	0.0	6.3	90.0	1.95	10.64	2.67	0.0
Q	-2.0	1.0	-3.5	0.2	5.65	114.0	2.17	9.13	1.56	0.18
R	2.0	1.0	-4.5	3.0	10.76	148.0	1.82	8.99	2.34	0.29
S	-2.0	0.0	-0.8	0.3	5.68	73.0	2.19	9.21	1.31	0.06
T	-2.0	0.0	-0.7	-0.4	5.6	93.0	2.09	9.1	3.03	0.11
V	1.0	-1.0	4.2	-1.5	6.0	105.0	2.39	9.74	3.67	0.14
W	1.0	-1.0	-0.9	-3.4	5.89	163.0	2.46	9.41	3.21	0.41
Y	-2.0	-1.0	-1.3	-2.3	5.64	141.0	2.2	9.21	2.94	0.3
X	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Physicochemical properties of the 20 proteinogenic amino acids: hydrophobicity π_1, polarity $pole$, hydropathy intensity η, hydrophilicity π_2, isoelectric point $pH(l)$, van der Waals volume υ_v, dissociation constants of the $-COOH$ group (pK_1) and $-NH_3$ group (pK_2), graph shape index Ξ, polarizability α

Table 2.2: Normalized physicochemical features.

	π_1	η	$pH(l)$	pK_1	pK_2	Ξ	α
A	0.0	0.71	-0.01	1.06	0.69	-0.84	-1.14
C	1.0	0.94	-0.56	-1.25	2.32	-0.39	-0.38
D	2.0	-1.05	-1.81	-0.88	0.75	-0.55	-0.57
E	2.0	-1.05	-1.64	-0.28	-0.09	-2.01	-0.19
F	1.0	1.04	-0.31	0.25	-0.41	0.68	1.14
G	0.0	-0.02	0.02	1.06	0.51	-2.01	-1.62
H	-1.0	-0.95	0.9	-1.9	-0.37	0.73	0.57
I	1.0	1.6	0.01	0.9	0.48	1.82	0.19
K	2.0	-1.19	2.04	0.04	-0.9	-0.28	0.48
L	1.0	1.37	-0.01	0.95	0.44	0.36	0.19
M	1.0	0.74	-0.16	-0.12	-0.47	0.14	0.48
N	-2.0	-1.05	-0.35	-0.07	-1.56	-0.55	-0.38
P	-1.0	0.64	0.16	-1.09	2.2	0.43	-1.62
Q	-2.0	-1.05	-0.22	0.09	-0.76	-0.58	0.1
R	2.0	-1.39	2.7	-1.79	-1.03	0.13	1.14
S	-2.0	-0.16	-0.2	0.2	-0.6	-0.81	-1.05
T	-2.0	-0.12	-0.24	-0.34	-0.82	0.76	-0.57
V	1.0	1.5	-0.02	1.27	0.44	1.35	-0.29
W	1.0	-0.19	-0.08	1.65	-0.21	0.93	2.29
Y	-2.0	-0.32	-0.22	0.25	-0.6	0.68	1.24
X	0.0	0.0	0.0	0.0	0.0	0.0	0.0

the structure model $\mathcal{K}$. Exactly which features were used and how are detailed for each experiment.

2.4 Datasets

Several datasets of proteins and their structures were used in our experiments. The first section describes a public dataset compiled by the authors of [1] that has been used in a number of past work for secondary structure prediction, CullPDB. To assess the impact that additional unlabelled data has on this problem, we constructed two new datasets. The second section describes a new dataset of proteins with known structures, constructed in the same manner as CullPDB, which we refer to has CullPDB2. The third section then details a large dataset of proteins without known structure gathered by culling UniRef50.

2.4.1 CullPDB (CPDB) and CB513

CullPDB[3] is a publicly available dataset consisting of 6128 non-homologous proteins. As was described in that work, this dataset was constructed by using the PISCES web server [60] to pull all proteins from the Protein Data Bank [25] sharing less than 30% sequence identity and whose 3D structures were resolved to at least 2.5 Angstroms. Additionally, all sequences having fewer than 50 or more than 700 residues were removed. CB513 is an independent test set built in a similar manner.

Each protein was encoded as a sequence of one-hot vectors, indicating amino acids as well as a special "NoSeq" token, concatenated with position-specific scoring matrices. The PSSMs were calculated by running PSIBLAST [61] for 3 iterations with an inclusion threshold of 0.001 against the UniRef90 database. The UniRef clusters [62] group proteins of high sequence similarity using the CD-HIT algorithm [63]. For example, in the case of UniRef50, proteins sharing 50% or higher sequence similarity form a single cluster, represented by a single member, reducing the total number of proteins by roughly 80% - from $\sim$149 million in UniRef100 to $\sim$31 million in UniRef50[4].

UniRef90 was filtered by using the program *pfilt*, available as a part of PSIPRED [64], to remove low information content and coiled-coil like regions. Once calculated, the similarity scores were scaled to the range [0, 1] by passing them through the logistic sigmoid function. Two more features were added to signify both the start and end of each protein.

The DSSP software [15] was used to infer the likely 8-state secondary structure for each amino acid along the protein backbone as well as both the relative and absolute solvent accessibility. [1] discretized these two solvent accessibility measures in a manner similar to was done in [65]. To obtain final model performance, CPDB was further filtered by removing all sequences that shared more than 25% sequence identity with the independent test set CB513.

The original[5] 57 features for CPDB are described in Table 2.3.

[3] Available to download at www.princeton.edu/ jzthree/datasets/ICML2014/

[4] UniRef statistics at www.uniprot.org/statistics/UniRef

[5] Note that we use different features than the ones originally described in [1], with the exception of the sequence profile matrices.

Table 2.3: Protein features used in Zhou and Troyanskaya

Range	**Description**
$[0, 22)$	amino acid label
$[22, 31)$	secondary structure label
$[31, 33)$	N- and C- terminals
$[33, 35)$	solvent accessibility
$[35, 57]$	sequence profile

Table 2.4: Frequencies of DSSP labels in the CPDB dataset

Q8 Label	Description	Frequency in CPDB (%)
H	α-helix	34.54
E	β-strand	21.78
T	Turn	11.28
S	Bend	8.26
G	3_{10} helix	3.9
B	β-bridge	1.03
I	π-helix	.02
L	Unknown or loop / irregular	19.18

2.4.2 CullPDB2 (CPDB2)

Since 2014, over 30,000 new proteins have had their structures resolved and been added to the PDB[6]. To leverage this new data, we followed a similar process as was done for CPDB to build a new dataset, which we refer to as CPDB2.

We pulled proteins from the PDB that shared no more than 30% sequence identity and whose structures were known to a resolution of at least 2.5 Angstroms using the PISCES web server[7]. The secondary structure, solvent accessibility, and torsion angles of each protein were then calculated using the DSSP software[8]. To calculate position-specific scoring matrices, we calculated multiple sequence alignments using PSIBLAST with an e-value of 0.001 for 3 iterations. To reduce computation time, we ran PSIBLAST against the UniRef50 database [62] filtered with *pfilt* to remove coiled-coil and low-complexity regions in sequences [66]. The resulting dataset contains 14726 sequences.

[6]PDB statistics available at rcsb.org
[7]http://dunbrack.fccc.edu/PISCES.php
[8]https://swift.cmbi.umcn.nl/gv/dssp/index.html

2.4.3 CullUniRef50 (CUR50)

Due to relative cheapness of sequencing proteins compared to determining 3D structure, there exist nearly 2 orders of magnitude (131K in the PDB vs. >100mill in UniProtKB) more proteins whose sequences are known but whose structures are not. To build a collection of unlabelled data, we began with the publicly available dataset UniRef50.

We used MMseqs2 [67] to further cluster the representatives in UniRef50 down to 20% sequence identity using a sensitivity parameter of 6[9] and a maximum of 100 aligned sequences per query. By default, MMseqs2 clusters sequences by representing proteins as nodes in a graph. Two protein nodes will share an edge if the user-set alignment criteria are met: % sequence identity, coverage (number of aligned residue pairs divided by the maximum of the lengths of the query or target sequence), and e-value (Gap-corrected Karlin-Altschul statistics). We used the default settings for coverage and e-value. Once edges are calculated, clusters are found by a greedy set cover algorithm, where nodes with the highest degree and their neighbors form a cluster and are removed, iteratively. The filtering → alignment → cluster process is repeated 3 times with increasing sensitivity to find the final clusters, of which there were 14,879,863.

To ensure that no sequences present in the structure dataset are also present in any of the data used for pretraining, we additionally use MMseqs2 to filter out all sequences sharing 20% identity to any sequence in CPDB2. This left 13,341,228 clusters and their representatives. Finally, for practical considerations, we did not include any sequences with lengths outside of 25 - 1040 residues in the training or validation sets.

2.5 Architecture

Our experiments contain two types of bidirectional RNN (BDRNN). The first is a bidirectional language model (BDLM), $\mathcal{L}$, and has the following structure.

The one-hot vectors a_c are passed through a dense projection layer and concatenated with a_f. These are fed to a 1D CNN [68]. We denote the outputs of the CNN as c_{i-k-1}^{i-1}, with the subscript indicating which input steps ($i-1$ to $i-k-1$ in this case) were covered by the kernel.

[9]The sensitivity parameter controls how sensitive sequence queries are. A sensitivity of 6 roughly corresponds to the sensitivity of a search with BLAST.

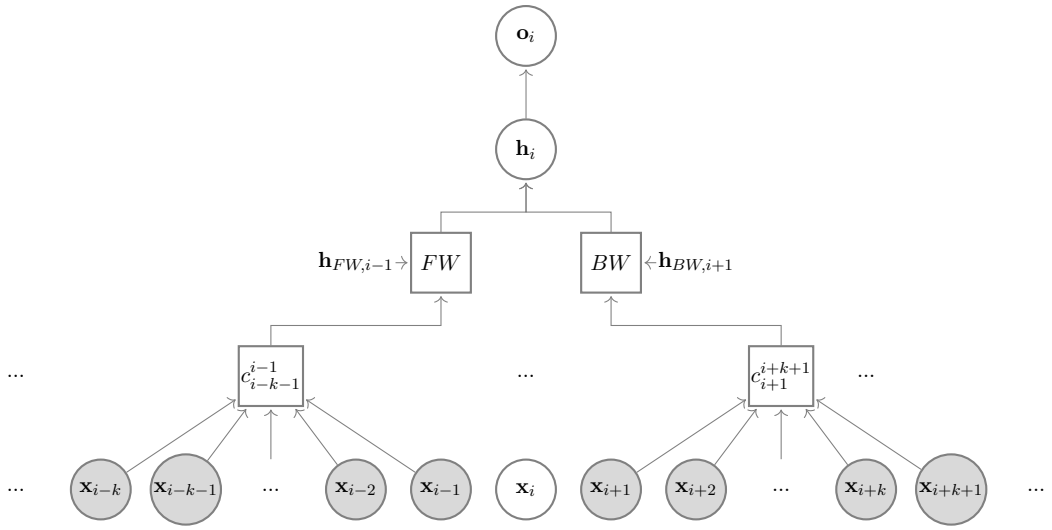


Figure 2.2: Graph of our bidirectional language model architecture. The outputs from the convolutional layer are fed to both the forward and backward RNNs. Circles indicate vectors of features and squares indicate model layers.

A 2-layer bidirectional LSTM calculates outputs for the forward and backward directions, with the outputs linearly projected to have size d. For layer j, the state at step i of the forward (backward) LSTM is written . A residual connection adds the outputs of the CNN to the inputs of the second layer. Both the CNN outputs and the LSTM outputs are linearly projected to have the same size. That is, $c_{i-k-1}^{i-1} \in \mathbb{R}^d$ and $h_{FW,i}, h_{BW,i} \in \mathbb{R}^d$.

Let $FW^{(j)}, BW^{(j)}$ be the jth layers of the forward and backward RNNs, and let $h_{FW,i}^{(j)}$, $h_{BW,T-i}^{(j)}$ denote their hidden states, respectively. Then

$$h_{FW,i}^{(1)} = FW^{(1)}(c_{i-k-1}^{i-1}, h_{FW,i-1}), \text{ and}$$
$$h_{BW,i}^{(1)} = BW^{(1)}(c_{i+1}^{i+k+1}, h_{BW,i+1})$$

The outputs of the second layer are concatenated and fed through a dense softmax layer, which forms the final outputs of the language model, $o_i = softmax(Wh_i)$; $h_i = [h_{FW,i}^{(2)}, h_{BW,i}^{(2)}]$, and W is a weight matrix that projects h_i to the same dimensionality as x. Note that output h_i has no information about the input at step i.

We use a second BDRNN to predict secondary structures: $\mathcal{K}$. Here, a_c is projected as before and concatenated with a_f as well as the position-specific scoring matrix features, a_p.The resulting features are normalized via layer normalization [69] and then passed directly to a bidirectional

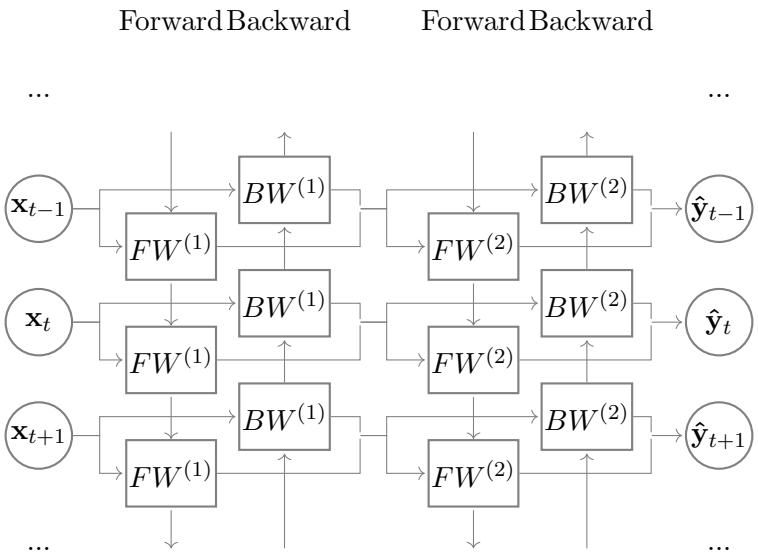


Figure 2.3: Architecture for our structure BDRNN K.

LSTM RNN. The output of the LSTM is then fed through a dense layer with softmax activation to make the final predictions.

The precise hyperparameters of each model - e.g. the number of layers in the RNNs, the size of the dense layers, activation functions, etc. - are specified in detail in the following sections.

2.6 Experiments

2.6.1 Baseline

In order to examine the impact of pretrained language models on secondary structure prediction, we first established performance on a baseline model.

Architecture details

The amino acid labels a_c are transformed to a dense embedding of size 256: $a_d \in \mathbb{R}^{256}$. Once concatenated with the physicochemical properties and PSSMs, the final feature vector is $x_i = [a_d, a_f, a_p] \in \mathbb{R}^{284}$.

The rest of the architecture was determined by a small grid search over the hyperparameters in Table 2.5. The right column indicates the sets of possible values for each hyperparameter, with each experiment consisting of a unique value from each row.

Table 2.5: Hyperparameter grid

Hyperparameter	Values
Layers	{1, 2, 3}
Recurrent State Size	{128, 256}
Dropout	{0.0, 0.3}

The hyperparameters searched over for the baseline model. The row for dropout indicates the dropout probability for non-recurrent layers as well as the variational dropout [70] probability for the recurrent hidden-to-hidden activations.

Architecture and training details are shared between all variations of $\mathcal{K}$. The recurrent layers are standard LSTM cells. All dense layers have ReLU activation functions, with the exception of the input embedding and softmax layers which are linear. All weights are initialized by sampling from the Glorot uniform distribution [71].

Dataset

We used a random train/validation split of the CPDB dataset. The validation set contained 512 samples, leaving 5022 samples for training.

Training regimen

All models were trained via stochastic gradient descent with a batch size of 50. When sampling batches, sequences were grouped by length into buckets [72] to speed up training. The initial learning rate was 0.2, and we divided the learning rate by 2 if there was no improvement in validation loss for 10 consecutive evaluations (roughly 5 epochs). The global norm of the gradient was calculated according to

$$norm_2 = \| \frac{\text{gradient}}{\text{batch size}} \|_2.$$

If the norm exceeded 2.0, the gradient was scaled by $\frac{2}{norm_2}$.

Grid search results

Validation losses and accuracies are reported in Table 2.6, as well as the training step at which those values occurred.

Table 2.6: Hyperparamater search results

Layers	**Units**	**Dropout**	**Accuracy**	**Loss**	**Step**
1	128	0.0	.7002	.577	19200
2	128	0.0	.6914	.5924	11801
3	128	0.0	.6859	.603	11501
1	128	0.3	.6959	.5781	36550
2	128	0.3	.6719	.6266	20550
3	128	0.3	.6828	.6035	17450
1	256	0.0	.696	.5875	15000
2	256	0.0	.6832	.6078	10500
3	256	0.0	.6792	.6175	10400
1	256	0.3	**.7047**	.5632	49800
2	256	0.3	.703	**.561**	40400
3	256	0.3	.7038	.5625	40800

Grid search results on the validation set for the baseline model.

The goal of this grid search was to find a reasonable architecture for our baseline model. While not comprehensive, it did reveal interesting changes in model performance and training behavior at different hyperparameter settings. Some of the general trends we observed were:

- Adding model capacity beyond a single layer BDRNN with 128 units without a concomitant increase in regularization always resulted in poorer performance due to overfitting
- For models with 128 units, performance also suffered after adding layers, even when the model was regularized via dropout
- Adding dropout resulted in up to a 5-fold increase in training time, but improved performance for models with 256 units per layer

There are two important points to make when considering the performance of the models reported in this table: (1) The choices for the baseline and grid search were purposefully kept simple, and (2) The results are noticeably lower than the reported highest accuracies reported in other works.

We justify point (1) by noting that our interest is in evaluating the impact of unsupervised pretraining on this problem, not necessarily in achieving state of the art performance. Point (2) is therefore understandable, as models which perform better on this task are typically much more complicated in design.[10]

We chose an LSTM BDRNN with a single layer of 256 units and dropout values of 0.3 as our baseline. Since there was not a large difference between this model and the 2-layer version, we chose the simpler model.

2.6.2 Protein language model

As touched on in the section on unsupervised pretraining, language models are commonly used in NLP to initialize recurrent neural networks. It is well established that LM performance scales well with larger model sizes and datasets [73] [74], and that such large-scale models can also effectively be integrated into downstream tasks: for semi-supervised sequence tagging [54]; as context-sensitive word embeddings [75]; for inductive transfer learning [76].

These recent successes demonstrate the remarkable power and flexibility that pretrained language models possess across many domains within NLP, and offer a framework for their application to problems involving biological sequence data.

Architecture details

Our language model architecture resembles that of [75], with the size of the LSTM states and output projections reduced to account for available GPU memory. The one-hot amino acids are passed through a dense projection with 25 units; $a_d \in \mathbb{R}^{25}$. The dense embeddings and physicochemical features are concatenated and fed to a 1D CNN with 1024 filters and a kernel size of 7, which is linearly projected down to a size of 256. The recurrent part of the model consists of a 2-layer bidirectional LSTM RNN with 1024 units and 256-dimensional projections. A residual connection adds the inputs of the first layer to its projected outputs before being passed to the second layer. As noted before, the outputs of the forward and backward directions are shifted and concatenated such that the outputs for prediction i only contains information from the preceding ($j < i$) and

[10] Although we would like to note that we were unable to reproduce the results reported in [12] after replicating their proposed model and experimental setup (see Appendix A).

following ($k > i$) steps. The concatenated outputs are then passed through a softmax layer. Our language model contains a total of 22 million trainable parameters.

Dataset

While it would be possible to train on the entire CUR50 dataset, for practical considerations we randomly sampled a training and validation set of 1 million and 10,000 proteins, respectively.

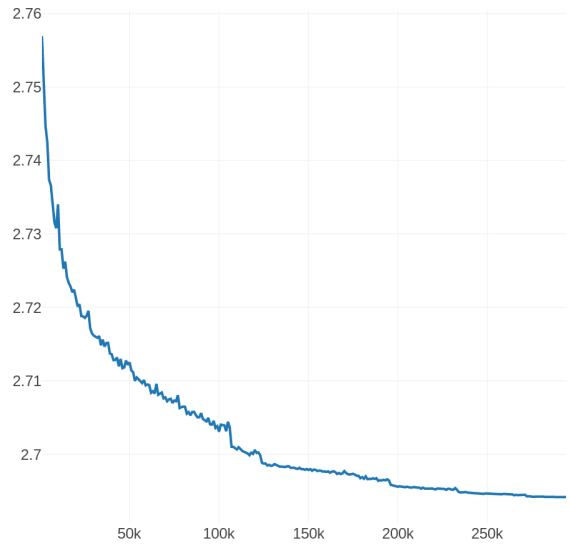
Training regimen

Training was almost identical to the baseline, with the difference that the global norm constraint on the gradients was set to a maximum of 1.0. Additionally, since the forward and backward directions are independent except at the final softmax layer, we parallelize training by assigning each direction to a different GPU. We do not add any regularization to the network.

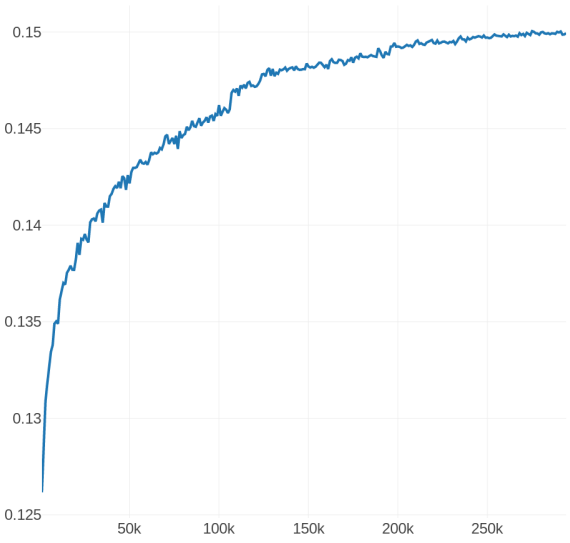
The loss on the validation set converged smoothly over the course of just under 300k training steps, which corresponded to roughly 15 epochs and 30 hours. The occasional steep drops in loss (e.g. around step 110k) occurred when the learning rate was divided in half. Overall, our BDLM achieved a per-residue accuracy of 15% on the validation set. Although we are unaware of recent efforts to apply language models to protein sequences, a unigram Markov model trained on the proteins of a single species reached a perplexity of 16.6 and a cross-species perplexity between 17 and 21 [77]. Perplexity measures how well a model predicts samples from an unknown distribution, and is defined as the exponentiation of the cross entropy, giving our model a perplexity of $e^{2.69} = 14.732$.

We can also compare the performance to similar models in NLP. The recurrent highway network language model of [78] achieved state of the art results for character-level language modeling, with their 21 million parameter model reaching an overall bits-per-character (bpc) of 1.30 on the enwik8[11] test set (although this has since been surpassed by much larger models, e.g. [74]). Since we use the natural logarithm for cross entropy, the loss of our model is in nats-per-character. We can convert to bits by multiplying by $\frac{1}{\ln 2}$, giving a bpc of around 3.89. Note that, since there are 23 characters in our alphabet, it would take $\log_2 23 = 4.52$ bpc to encode an arbitrary sequence, assuming the characters are uniformly distributed.

[11]The enwik8 dataset (http://mattmahoney.net/dc/textdata.html) is a collection of the first 108 bytes of the English Wikipedia, gathered in 2006.



(a) Validation loss



(b) Validation accuracy

Figure 2.4: Results for the bidirectional language model, $\mathcal{L}$.

Comparisons to language models applied to natural language are likely of limited use, however, as biological sequences such as DNA and proteins are known to be intrinsically information dense [79]. One hypothesis for why this is the case is that evolutionary pressure drives proteins to near-optimal compression rates. As such, it is unlikely that language models of proteins can achieve similarly low entropies as in NLP. Speculating what the optimal performance might be, a question closely related to protein compressibility, is an interesting research pursuit but beyond the scope of this thesis.

2.6.3 Pretrained LMs and secondary structure prediction

We experimented with incorporating the pretrained language model $\mathcal{L}$ into the structure BDRNN $\mathcal{K}$ in three different ways, which we refer to as outSEQ, ELMO1, and ELMO2. In each case, we kept the total number of input features the same as the baseline, and adjusted the dimensionality of the amino acid embeddings (a_d) to account for the size of the BDLM features. For each method, we performed 5-fold cross validation on random train/validation splits of the CPDB training data. The results of these trained models are compared to each other, as well as the baseline.

outSEQ

Method 1 uses the softmax outputs of the language model as additional features to the structure BDRNN. Since the language model is trained to predict amino acids, these features comprise a probability distribution over amino acids. The full feature vector is then $[o_i, a_d, a_f, a_p]$, where $o_i \in \mathbb{R}^{23}$ are the softmax outputs of the BDLM, $a_d \in \mathbb{R}^{23312}$ is the dense embedding of the one-hot amino acid tokens, and $a_f \in \mathbb{R}^7$, $a_p \in \mathbb{R}^{21}$ are the physicochemical and PSSM features of each residue, respectively.

ELMO1 and ELMO2

Methods 2 and 3 use ELMo embeddings [75] of the intermediate activations of the language model as extra inputs to the structure BDRNN. An ELMo embedding is a weighted sum of the activations

[12] This was chosen so $[o_i, a_d] \in \mathbb{R}^{256}$, matching the size of a_d from the baseline.

of each layer: if

$$h_i^{(0)} = [c_{i-k-1}^{i-1}, c_{i+1}^{i+k+1}],$$
$$h_i^{(1)} = [h_{FW,i}^{(1)}, h_{BW,i}^{(1)}], \text{ and}$$
$$h_i^{(2)} = [h_{FW,i}^{(2)}, h_{BW,i}^{(2)}],$$

are the intermediate activations of $\mathcal{L}$ for step i, then the ELMo embedding for step i is

$$e_i = \sum_{j=0}^{2} s^{(j)} h_i^{(j)}.$$

[13]

where the $s^{(j)}, \sum_j s^{(j)} = 1$ weight each layer and are learned during training. This results in a single feature vector $e_i \in \mathbb{R}^{2d}$ where d is the size of the intermediate representations of the BDLM - in this case, $d = 256$. ELMos allow the supervised model to weight all of the representations learned by our pretrained model, not just the outputs as in outSEQ.

For ELMO1, we linearly projected e_i to match the dimensionality of o_i. In this case, the input features are $[e_i, a_d, a_f, a_p]$, where $e_i \in \mathbb{R}^{23}$, and a_d, a_f, a_p are identical as outSEQ.

We were also interested in how performance was affected when e_i represented a larger proportion of the input features, and so for ELMO2, the dimensionalities of a_d and e_i were swapped: $a_d \in \mathbb{R}^{23}, e_i \in \mathbb{R}^{233}$.

Architecture details

The architecture for each method is identical with the exception of the inputs to the structure BDRNN. Additionally, since in general the distributions of the intermediate activations of $\mathcal{L}$ (and therefore of the ELMo inputs) may be very different from each other as well as the other inputs (such as the physicochemical features), we normalize the inputs to the structure BDRNN for all methods using layer normalization [69].

[13] While the original formulation included a scaling coefficient γ, all our models normalized their inputs via layer normalization [69] which has its own scale parameter (also called γ). This would make an additional coefficient on e_i redundant.

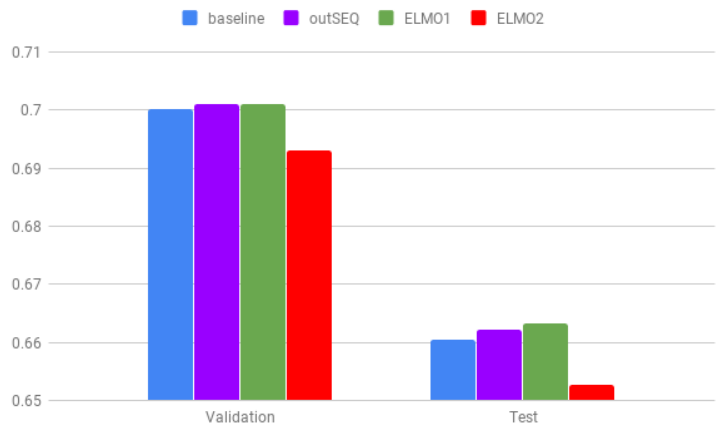


Figure 2.5: Mean validation and test accuracy

Training regimen

We performed the same training procedure as the baseline grid search, with the exception that instead of halving the learning rate every 10 evaluations, the patience was increase exponentially; each time the learning rate was halved, the patience increased by a factor of 1.2 (starting at 10).

Results

All figures are averaged across the 5 folds trained for each model.

The validation and test accuracies in Table 2.5 show that including information from the pretrained BDLM can improve secondary structure prediction, with the notable exception of the ELMO2 model.

The validation loss curves in Table 2.6 show the 95% confidence intervals at each validation step during training. The curve for ELMO2 is interesting, as all copies of that model showed quick improvements to validation performance early on in training, then leveled off. This is consistent with the model overfitting, and could explain its relatively poor performance on the test set. In contrast, the other three models show gradual improvements over a longer span of the training process.

Since the sequences in the pretraining dataset have a slightly higher maximum length (1040 vs. 700), one might expect performance on longer sequences to improve for pretrained models. There appears to be some evidence of this in Table 2.8, with our best model, ELMO1, outperforming

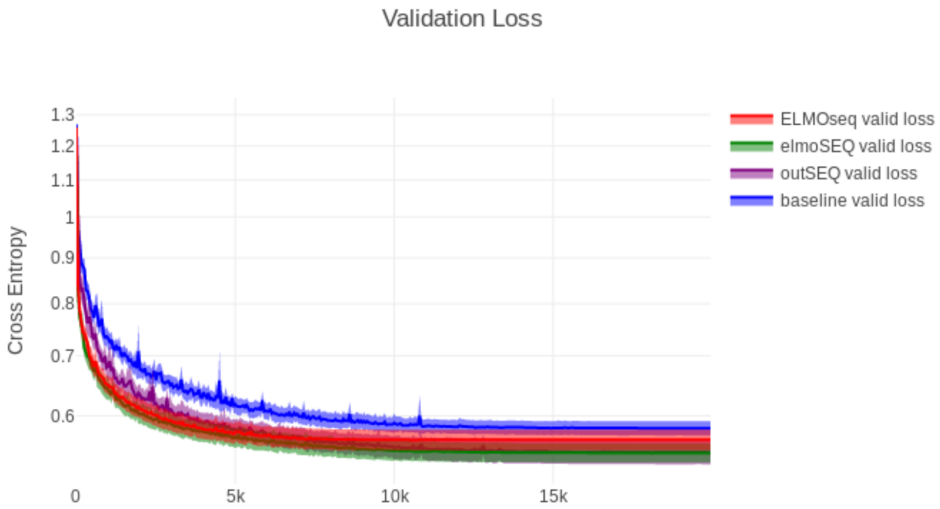
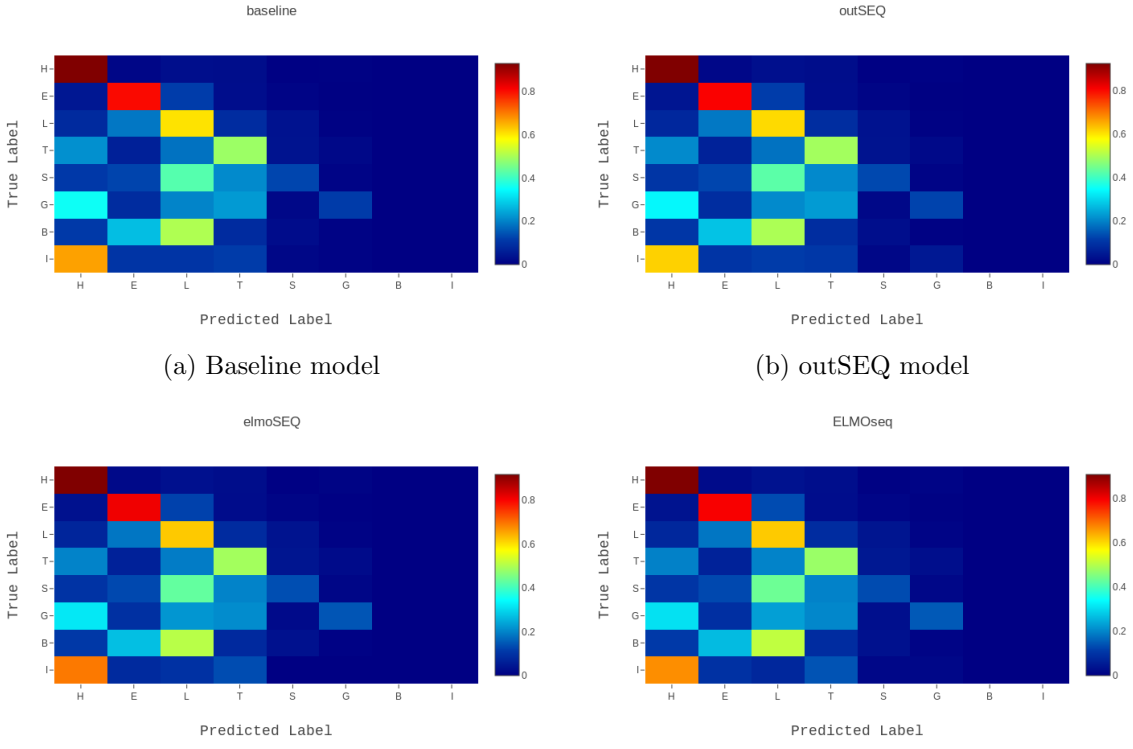


Figure 2.6: Mean validation losses and 95% confidence intervals



(a) Baseline model

(b) outSEQ model

(c) ELMO1 model

(d) ELMO2 model

Figure 2.7: Confusion matrices

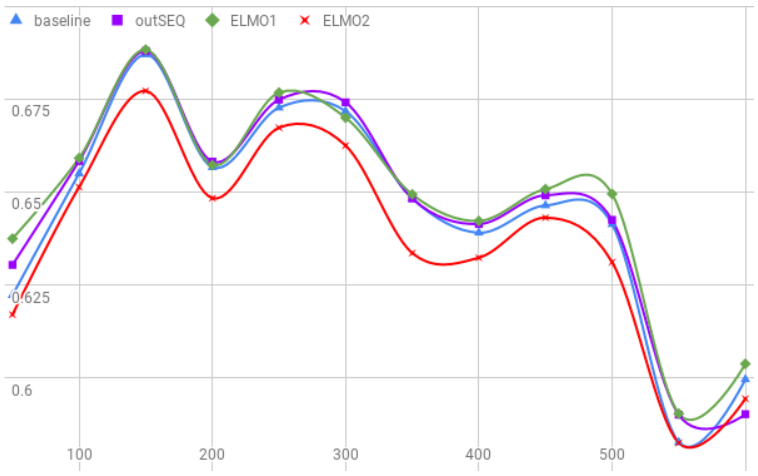


Figure 2.8: Mean test accuracy vs. sequence length

all other models at sequence lengths above 300. Model accuracy for lengths above 550 is somewhat unpredictable, with ELMO2 beating outSEQ despite worse performance across all other lengths.

Sequence/structure mutual information

Information theoretic measures have been used to explore the statistical relationships between protein sequence and structure [80]. In addition to empirically testing the impact of the representations learned by pretrained language models, it is also possible to directly measure the amount of information about the structure labels they contain.

Mutual information captures the amount of information contained in one (potentially vector-valued) random variable about another. It is conceptually similar to the covariance, with the important difference that covariance captures linear relationships whereas mutual information can capture both linear and nonlinear relationship between random variables. For discrete random variables X, Y, the mutual information $I(X;Y)$ is defined as

$$I(X;Y) = \sum_x \sum_y p(x,y) \log \frac{p(x,y)}{p(x)p(y)}.$$

Another way of viewing mutual information is as the difference of the joint distribution $p(X,Y)$ from the product of the marginal distributions $p(X)p(Y)$. Note that $I(X;Y) >= 0$ and is 0 if and only if $p(X,Y) = p(X)p(Y)$; that is, the mutual information is zero only when the two random variables are independent, and positive otherwise.

For discrete distributions, one can calculate the sample mutual information directly. However, this isn't possible in the case of continuous variables, as this typically requires integrating the density function of the joint distribution. There are several methods for estimating MI of continuous random variables that have been shown to do well empirically [81]. Notably, the nearest neighbor-based method introduced in [82], referred to as KSG, estimates entropy by using distances to the k-nearest neighbors of each sample, and has the theoretical guarantee of being an unbiased estimator of MI as the amount of data grows. We use KSG to estimate the mutual information of the PSSMs with the secondary structure labels.

Table 2.7 contains the mutual information for the CPDB 513 test set for several variables of interest and secondary structure, $\mathbf{s}$. Here, $\mathbf{a}$ indicates the true amino acid labels; $\hat{\mathbf{a}}$ are the predictions made by the BDLM, $\mathcal{L}$ ($\hat{\mathbf{a}} = \text{argmax } o_i$); max a_p are the indices of the PSSMs for each position, and a_p are the real-valued PSSM features. We used KSG with $k = 1000$ to estimate $I(\mathbf{s}; a_p)$. The value in the table is equal to the estimate of I minus an estimate of the systematic error, calculated by averaging over random shuffles of the data (randomly shuffling X should result in a mutual information with Y of 0).

Table 2.7: Mutual information results

$I(\mathbf{s}; -)$	
$\mathbf{a}$	0.067055
max a_p	0.091479
$\hat{\mathbf{a}}$	0.594856
$\mathbf{a}_p$	0.616562

Information content of several variables with secondary structure labels for the CB513 test set.

$I(\mathbf{s}; \mathbf{a})$ gives the information shared between amino acids and corresponding secondary structures for each residue in the dataset. The position-wise correlation between amino acid and secondary structure is known to be relatively weak and we use this as a baseline.

$I(\mathbf{s}; \text{max } a_p)$ is an estimate of I for secondary structure and the PSIBLAST-generated PSSMs. These scoring matrices are calculated by an iterative multiple sequence alignment process, and contain much more information about structure than amino acids alone. In this case, we discretize the scores by assigning a label to each sample equal to the amino acid with the largest value in the scoring matrix. This is equivalent to assigning the label of the amino acid which

would maximize the alignment score at that position. Unsurprisingly, this results in a relatively low mutual information with the structure, as in most cases the amino acid that would maximize an alignment score is the precisely the amino acid found at that position.

In comparison, $I(\mathbf{s}; \hat{\mathbf{a}})$ is the mutual information of secondary structure and the predicted amino acids at each position. This is a coarse-grained measurement of the quality of the representations learned by our BDLM, but interestingly it shows a markedly high amount of shared information. It is worth noting that as accuracy of the language model improves, the mutual information of its predictions and the secondary structure will approach $I(\mathbf{s}; \mathbf{a})$; that is, better $\mathcal{L}$ predictions for amino acids would eventually decrease $I(\mathbf{s}; \hat{\mathbf{a}})$.

2.7 Discussion

2.7.1 Model performance

There are subtle differences between the performance of the baseline, outSEQ, and ELMO1. In terms of end performance, the latter two with pretrained feature inputs suggest that these features improve test accuracy, with the richer features provided by ELMo improving performance slightly more than the output probabilities of the language model. All 3 models also have notably different loss curves during training from the baseline model, with validation loss decreasing both more quickly and converging to lower values across the board. In the case of ELMO2, this led to overfitting and the end performance suffered as a result, however for outSEQ and ELMO1, both training and validation loss decreased jointly throughout training.

An examination of the results per secondary structure label reveals that our best 2 models show the same pattern: they are somewhat worse at recognizing α-helices (H), but more accurate for β-strands (E), bends (S), and 3_{10}-helices (G) and roughly on par with the baseline for turns (T).

ELMO1 vs. ELMO2

The disparities between ELMO1 and ELMO2 are worth considering. Both models use a weighted sum of the internal activations of $\mathcal{L}$, however ELMO1 compresses these into a feature vector of size 23 - less than 1/20th the size of the unmodified representations - whereas ELMO2 only reduces

the dimensionality by about half, to 233. Although the overall number of features remains the same, a larger proportion come from e_i for ELMO2, and this results in overfitting. In fact, the training behavior of ELMO2 more closely resembles that of baseline models that were trained with no dropout. We hypothesize that this is due to the model fitting to spurious relationships in the embeddings, and this likely does not occur for ELMO1 due to the size of the projection acting as a bottleneck.

2.7.2 Information content

In section 1.3.4, we mentioned that one way unsupervised pretraining may be helpful for supervised learning is by guiding the model to learn useful features about the *causal factors* of the observed variables, in this case the primary sequence of proteins. It is an interesting exercise to consider what the causal factors of protein sequences might be, and in the case of homologous proteins, we already have some idea: the evolutionary pressures that preserve protein structure in such cases, discussed in section 1.2.2, guide the process of mutation to favor sequences whose associated native structures are similar to one another. This fact is perhaps why PSSMs calculated via multiple sequence alignments (MSA) - which encode the evolutionary neighborhood around related proteins - have such a large positive impact on predicting secondary structure.

Unlike an MSA, however, our language model was trained on proteins filtered to have low sequence identity. With the acknowledgement that our filtering procedure is noisy and therefore likely does not eliminate every homologous protein from the dataset, it seems reasonable to claim that for a given sample $\mathbf{x}_i$, the outputs of $\mathcal{L}$, $\mathbf{o}_i$, encode information about the distribution of sequences $\mathbf{x}_j, j \neq i$ that *aren't* in the evolutionary neighborhood of $\mathbf{x}_i$ - precisely the opposite of the sort of information captured by the PSSM features $\mathbf{a}_p$.

With that being said, Table 2.7 shows that $I(\mathbf{s}; \hat{\mathbf{a}})$ (which is itself only a coarse-grained estimate of $I(\mathbf{s}; \mathbf{o}_i)$) - nearly matches $I(\mathbf{s}; \mathbf{a}_p)$. That is, the predictions of $\mathcal{L}$ contain nearly as much information about secondary structure as the PSSMs. Investigating why this is the case is a compelling question for future research.

Chapter 3

Conclusion

In the final chapter of this thesis, we provide interesting directions for future work and offer some concluding remarks.

3.1 Future Work

Much of our understanding about the utility of language models comes from results reported in natural language processing, and more research needs to be done to build a robust understanding of their representational and predictive power on biological sequence data. There are some clear directions to pursue to further shed light on these models.

Due to resource constraints, both the size of our language model and the amount of unlabeled data on which it was trained were limited. Training on the full set of proteins available (over 13 million) would almost certainly improve the representations learned by this model.

Although we did not emphasize this fact here, one major benefit of pretraining is that the relatively expensive pretraining process can be amortized via transfer to new domains. The pretrained model described in our work can be applied to any sequence prediction task where it might reasonably be expected that protein sequence information be beneficial to task performance. This includes fine-tuning on new protein sequence datasets, or predicting other sequence properties such as solvent accessibility, contact maps, and torsion angles.

In the section on unsupervised pretraining, we noted that sequence autoencoders are also a valid method for pretraining recurrent neural networks. Work published in early 2018 [83] demon-

strated that autoencoders can learn to embed proteins in a latent space wherein euclidean distance corresponds to edit distance. Sequence autoencoders offer a natural way to learn a global representation (in contrast to the local representations learned by language models) that may be amenable to predicting properties of proteins that are also global, e.g. antimicrobial activity, toxicity, edit distance, etc.

3.2 Concluding Remarks

We trained a bidirectional language model on protein sequences and evaluated the effect the representations learned by this model had on secondary structure prediction. Our experiments show that these representations contain a significant amount of information about secondary structure even when trained on primary sequence alone, and suggest that they have a beneficial effect on performance. We believe these results are encouraging, and that further research will elucidate the potential of neural language models for unsupervised learning on biological sequence data.

Appendix A

Reproducing (Sonderby & Winther, 2015)

Validating experiments by reproducing their results is crucially important in science as it strengthens the basis upon which future research is conducted. The goal of this section is to build confidence in recent literature applying deep learning to protein secondary structure prediction. We followed these policies when reproducing results:

- Implement the model as close to its description as possible
- In cases where there is ambiguity, or the authors do not specify, choose a reasonable default, and make that choice clear
- In cases where there are multiple models or training setups, choose the one with the best reported Q3/Q8 accuracy on CB513 (or a similar independent test set)
- Only compare single models, not ensembles

A.1 Protein secondary structure prediction with long short-term memory networks

The model proposed in [12] includes an interesting variation on traditional long short-term memory networks while remaining conceptually straightforward. While the performance reported at the time

was state of the art for Q8 accuracy, we note that the training and evaluation procedures used could have been better detailed and more rigorous. The combination of these facts means that there is value in reproducing these results.

A.1.1 Architecture details

The model consists of a bidirectional long short-term memory recurrent neural network. Both forward and backward networks have 3 layers of 300 or 500 units each. The basic LSTM structure is modified to pass the hidden state at each time step through a feed-forward network with 2 layers of 300 or 500 units with ReLU activations and skip connections. The outputs of the forward and backward networks is concatenated and fed through 2 fully-connected layers of 200 or 400 units each and ReLU activations.

The LSTM used here has the same formulation as described in equations 1.1 through 1.6, with the added feed-forward network between the recurrent hidden-to-hidden activations:

$$h_{t-rec} = h_t + feedforwardnet(h_t)$$

A.1.2 Dataset

We use the same dataset as the original paper, CPDB.

A.1.3 Setup and training

Since we are interested in reproducing a particular paper, we do not attempt to optimize the choice of any hyperparameters. The LSTM weights were initialized by sampling from a uniform random distribution in the range [-0.05, 0.05]. The weights of the fully-connected layers were initialized by the default in the Lasagne library; when we checked on 29/04/2018, this was the Glorot uniform distribution. All biases were initialized to zero. The concatenation network was regularized using dropout with a keep probability of 50%. The model was optimized with the AdaDelta optimizer [84] with default settings (learning rate=1.0, rho=0.95, epsilon=1e-6) using a batch size of 128. After each backward pass, the global norm of the gradient divided by the batch size was calculated,

$$norm_2 = \| \frac{\text{gradient}}{\text{batch size}} \|_2.$$

If the norm exceeded 0.5, the gradient was scaled by $\frac{0.5}{norm_2}$. The initial cell and hidden states for the forward and backward networks were learned.

Our experimental setup replicates the original setup with some small differences:

- No details are provided for how many epochs or steps the models were trained for, nor what stopping conditions (if any) the authors observed. We chose to implement a simple algorithm that stopped training once the validation loss had not improved for 8 validation steps.
- To speed up training, we group samples by sequence length into buckets [72].
- Due to GPU memory constraints, we use a batch size of 50 for all buckets.

A.1.4 Results

The results for both the small and large versions of the model are in the figures below. The training does not run for the same number of steps for both models due to the early stopping procedure mentioned above. In both cases, the total training time was less than 2 hours.

Table A.1: Reproduced Q8 and Q3 accuracies

CB 513 (Test)	**Q8 Accuracy**	**Q3 Accuracy**
LSTM Small (Original)	.671	N/A
LSTM Large (Original)	.674	N/A
LSTM Small (Ours)	.6658	.7990
LSTM Large (Ours)	.6578	.7881

Bibliography

[1] Jian Zhou and Olga Troyanskaya. Deep supervised and convolutional generative stochastic network for protein secondary structure prediction. *CoRR*, abs/1403.1347, 2014.

[2] Ken A. Dill and Justin L. MacCallum. The protein-folding problem, 50 years on. *Science*, 338, 2012. doi:10.1126/science.1219021.

[3] Po-Ssu Huang, Scott E. Boyken, and David Baker. The coming age of de novo protein design. *Nature*, 537:320–327, 2016. doi:10.1038/nature19946.

[4] Thomas C. Terwilliger, David Stuart, and Shigeyuki Yokoyama. Lessons from structural genomics. *Annual Review of Biophysics*, 38:371–383, 2009. doi:10.1146/annurev.biophys.050708.133740.

[5] Elaine R. Mardis. Anticipating the $1,000 genome. *Genome Biology*, 7, 2006.

[6] José Juan Almagro Armenteros, Casper Kaae Sønderby, Søren Kaae Sønderby, Henrik Nielsen, and Ole Winther. Deeploc: prediction of protein subcellular localization using deep learning. *Bioinformatics*, 33(21):3387–3395, 2017. doi:10.1093/bioinformatics/btx431. URL `http://dx.doi.org/10.1093/bioinformatics/btx431`.

[7] Thomas Unterthiner, Andreas Mayr, Günter Klambauer, and Sepp Hochreiter. Toxicity prediction using deep learning. *CoRR*, abs/1503.01445, 2015.

[8] Sheng Wang, Siqi Sun, Zhen Li, Renyu Zhang, and Jinbo Xu. Accurate de novo prediction of protein contact map by ultra-deep learning model. *PLOS Computational Biology*, 13(1):1–34, 01 2017. doi:10.1371/journal.pcbi.1005324. URL `https://doi.org/10.1371/journal.pcbi.1005324`.

[9] Yuedong Yang, Jianzhao Gao, Jihua Wang, Rhys Heffernan, Jack Hanson, Kuldip Paliwal, and Yaoqi Zhou. Sixty-five years of the long march in protein secondary structure prediction: the final stretch? *Briefings in Bioinformatics*, 19(3):482–494, 2018. doi:10.1093/bib/bbw129. URL `http://dx.doi.org/10.1093/bib/bbw129`.

[10] Sergey Ovchinnikov, Hahnbeom Park, David E. Kim, Frank DiMaio, and David Baker. Protein structure prediction using rosetta in casp12. *Proteins: Structure, Function, and Bioinformatics*, 86(S1):113–121, 2018. doi:10.1002/prot.25390. URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/prot.25390`.

[11] Sheng Wang, Jian Peng, Jianzhu Ma, and Jinbo Xu. Protein secondary structure prediction using deep convolutional neural fields. *Scientific Reports*, 6, 2016. doi:10.1038/srep18962.

[12] Soren Sonderby and Ole Winther. Protein secondary structure prediction with long short term memory networks. *CoRR*, abs/1412.7828, 2015.

[13] Rhys Heffernan, Yuedong Yang, Kuldip Paliwal, and Yaoqi Zhou. Capturing non-local interactions by long short-term memory bidirectional recurrent neural networks for improving prediction of protein secondary structure, backbone angles, contact numbers and solvent accessibility. *Bioinformatics*, 33(18):2842–2849, 2017. doi:10.1093/bioinformatics/btx218. URL `+http://dx.doi.org/10.1093/bioinformatics/btx218`.

[14] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. Revisiting unreasonable effectiveness of data in deep learning era. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, pages 843–852. IEEE, 2017.

[15] Wolfgang Kabsch and Chris Sander. Dictionary of protein secondary structure. *Biopolymers*, 22:2577 – 2637, 12 1983.

[16] Cyrus Chothia and Arthur M. Lesk. The relation between the divergence of sequence and structure in proteins. *The EMBO journal*, 5 4:823–6, 1986.

[17] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10:707, February 1966.

[18] Burkhard Rost. Twilight zone of protein sequence alignments. *Protein engineering*, 12 2:85–94, 1999.

[19] Ken A. Dill, S. Banu Ozkan, M. Scott Shell, and Thomas R. Weikl. The protein folding problem. *Annual Review of Biophysics*, 37:289–316, 2008. doi:10.1146/annurev.biophys.37.092707.153558.

[20] Joseph P. Hendrick and F. Ulrich Hartl. The role of molecular chaperones in protein folding. *FASEB journal : official publication of the Federation of American Societies for Experimental Biology*, 9 15:1559–69, 1995.

[21] S. Banu Ozkan, Guohong Albert Wu, John D. Chodera, and Ken A. Dill. Protein folding by zipping and assembly. *Proceedings of the National Academy of Sciences of the United States of America*, 104 29:11987–92, 2007.

[22] Ken A. Dill and H. S. Y. Chan. From levinthal to pathways to funnels. *Nature Structural Biology*, 4:10–19, 1997.

[23] Christian B. Anfinsen. Principles that govern the folding of protein chains. *Science*, 181 4096: 223–30, 1973.

[24] The UniProt Consortium. Uniprot: the universal protein knowledgebase. In *Nucleic Acids Research*, 2017.

[25] Helen M. Berman, John D. Westbrook, Zukang Feng, Gary Gilliland, T. N. Bhat, Helge Weissig, Ilya N. Shindyalov, and Philip E. Bourne. The protein data bank. *Acta crystallographica. Section D, Biological crystallography*, 58 Pt 6 No 1:899–907, 2000.

[26] Peter W. Rose, Bojan Beran, Chunxiao Bi, Wolfgang Bluhm, Dimitris Dimitropoulos, David S. Goodsell, Andreas Prlić, Martha Quesada, Greg B. Quinn, John D. Westbrook, Jasmine Young, Benjamin T. Yukich, Christine Zardecki, Helen M. Berman, and Philip E. Bourne.

The rcsb protein data bank: redesigned web site and web services. In *Nucleic Acids Research*, 2011.

[27] Kim T Simons, Charles Kooperberg, Enoch S. Huang, and David Baker. Assembly of protein tertiary structures from fragments with similar local sequences using simulated annealing and bayesian scoring functions. *Journal of molecular biology*, 268 1:209–25, 1997.

[28] Po-Ssu Huang, Scott E. Boyken, and David Baker. The coming of age of de novo protein design. *Nature*, 537:320–327, 2016.

[29] Ghazaleh Taherzadeh, Yaoqi Zhou, Alan Wee-Chung Liew, and Yuedong Yang. Sequence-based prediction of protein-carbohydrate binding sites using support vector machines. *Journal of chemical information and modeling*, 56 10:2115–2122, 2016.

[30] Rhys Heffernan, Abdollah Dehzangi, James G. Lyons, Kuldip K. Paliwal, Alok Sharma, Jihua Wang, Abdul Sattar, Yaoqi Zhou, and Yuedong Yang. Highly accurate sequence-based prediction of half-sphere exposures of amino acid residues in proteins. *Bioinformatics*, 32 6:843–9, 2016.

[31] Predrag Radivojac, Lilia M. Iakoucheva, Christopher J. Oldfield, Zoran Obradovic, Vladimir N. Uversky, and A. Keith Dunker. Intrinsic disorder and functional proteomics. *Biophysical Journal*, 92 5:1439–56, 2007.

[32] Rhys Heffernan, Kuldip K. Paliwal, James G. Lyons, Abdollah Dehzangi, Alok Sharma, Jihua Wang, Abdul Sattar, Yuedong Yang, and Yaoqi Zhou. Improving prediction of secondary structure, local backbone angles, and solvent accessible surface area of proteins by iterative deep learning. In *Scientific Reports*, 2015.

[33] James G. Lyons, Abdollah Dehzangi, Rhys Heffernan, Alok Sharma, Kuldip K. Paliwal, Abdul Sattar, Yaoqi Zhou, and Yuedong Yang. Predicting backbone cα angles and dihedrals from protein sequences by stacked sparse auto-encoder deep neural network. *Journal of Computational Chemistry*, 35 28:2040–6, 2014.

[34] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *NIPS*, 2006.

[35] Chao Fang, Yi Shang, and Dong Xu. Mufold-ss: New deep inception-inside-inception networks for protein secondary structure prediction. *Proteins*, 86 5:592–598, 2018.

[36] Akosua Busia and Navdeep Jaitly. Next-step conditioned deep convolutional neural networks improve protein secondary structure prediction. *CoRR*, abs/1702.03865, 2017.

[37] Zhen Li and Yizhou Yu. Protein secondary structure prediction using cascaded convolutional and recurrent neural networks. *CoRR*, abs/1604.07176, 2016.

[38] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[39] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, Nov 2012. ISSN 1053-5888. doi:10.1109/MSP.2012.2205597.

[40] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014.

[41] Junshui Ma, Robert P. Sheridan, Andy Liaw, George E. Dahl, and Vladimir Svetnik. Deep neural nets as a method for quantitative structure-activity relationships. *Journal of chemical information and modeling*, 55 2:263–74, 2015.

[42] Alessandro Sperduti. On the computational power of recurrent neural networks for structures. *Neural Networks*, 10:395–400, 1997.

[43] Yoshua Bengio, Patrice Y. Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5 2:157–66, 1994.

[44] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *ICML*, 2013.

[45] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[46] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9: 1735–1780, 1997.

[47] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *EMNLP*, 2014.

[48] Geoffrey E. Hinton and Ruslan Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313 5786:504–7, 2006.

[49] Dumitru Erhan, Yoshua Bengio, Aaron C. Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? In *Journal of Machine Learning Research*, 2010.

[50] Tom Le Paine, Pooya Khorrami, Wei Han, and Thomas S. Huang. An analysis of unsupervised pre-training in light of recent advances. *CoRR*, abs/1412.6597, 2014.

[51] Andrew M. Dai and Quoc V. Le. Semi-supervised sequence learning. In *NIPS*, 2015.

[52] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *NIPS*, 2014.

[53] Ryan Kiros, Yukun Zhu, Ruslan Salakhutdinov, Richard S. Zemel, Antonio Torralba, Raquel Urtasun, and Sanja Fidler. Skip-thought vectors. In *NIPS*, 2015.

[54] Matthew E. Peters, Waleed Ammar, Chandra Bhagavatula, and Russell Power. Semi-supervised sequence tagging with bidirectional language models. In *ACL*, 2017.

[55] Daisuke Kihara. The effect of long-range interactions on the secondary structure formation of proteins. *Protein Science*, 2005.

[56] Jack Hanson, Yuedong Yang, Kuldip K. Paliwal, and Yaoqi Zhou. Improving protein disorder prediction by deep bidirectional long short-term memory recurrent neural networks. *Bioinformatics*, 33 5:685–692, 2017.

[57] Donard S Dwyer. Electronic properties of amino acid side chains: quantum mechanics calculation of substituent effects. *BMC Chemical Biology*, 5:2 – 2, 2005.

[58] Jack Kyte and Russell F. Doolittle. A simple method for displaying the hydropathic character of a protein. *Journal of molecular biology*, 157 1:105–32, 1982.

[59] Jean Luc Fauchère, M N Charton, Lemont B. Kier, A J Verloop, and Vladimir Pliska. Amino acid side chain parameters for correlation studies in biology and pharmacology. *International journal of peptide and protein research*, 32 4:269–78, 1988.

[60] Guoli Wang and Roland L. Dunbrack. Pisces: recent improvements to a pdb sequence culling server. *Nucleic Acids Research*, 33:W94 – W98, 2005.

[61] Stephen Altschul, Thomas Madden, Alejandro Schaffer, Jinghui Zhang, Zheng Zhang, Webb Miller, and David Lipman. Gapped blast and psi-blast: a new generation of protein database search programs. *Nucleic Acids Research*, 25:3389–3402, 1997.

[62] Baris E. Suzek, Yuqi Wang, Hongzhan Huang, Peter B. McGarvey, and Cathy H. Wu. Uniref clusters: a comprehensive and scalable alternative for improving sequence similarity searches. In *Bioinformatics*, 2015.

[63] Weizhong Li and Adam Godzik. Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences. *Bioinformatics*, 22 13:1658–9, 2006.

[64] David T. Jones. Protein secondary structure prediction based on position-specific scoring matrices. *Journal of molecular biology*, 292 2:195–202, 1999.

[65] Yanjun Qi, Merja Oja, Jason Weston, and William S. Noble. A unified multitask architecture for predicting local protein properties. In *PloS one*, 2012.

[66] David T. Jones and Mark B. Swindells. Getting the most from psi-blast. *Trends in biochemical sciences*, 27 3:161–4, 2002.

[67] Martin Steinegger and Johannes Söding. Mmseqs2 enables sensitive protein sequence searching for the analysis of massive data sets. *Nature Biotechnology*, 35:1026–1028, 2017.

[68] Yoon Kim, Yacine Jernite, David A Sontag, and Alexander M. Rush. Character-aware neural language models. In *AAAI*, 2016.

[69] Jimmy Ba, Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *CoRR*, abs/1607.06450, 2016.

[70] Yarin Gal and Zoubin Ghahramani. A theoretically grounded application of dropout in recurrent neural networks. In *NIPS*, 2016.

[71] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, 2010.

[72] Viacheslav Khomenko, Oleg Shyshkov, Olga Radyvonenko, and Kostiantyn Bokhan. Accelerating recurrent neural network training using sequence bucketing and multi-gpu data parallelization. *2016 IEEE First International Conference on Data Stream Mining and Processing (DSMP)*, pages 100–103, 2016.

[73] Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. Exploring the limits of language modeling. *CoRR*, abs/1602.02410, 2016.

[74] Rami Al-Rfou, Dokook Choe, Noah Constant, Mandy Guo, and Llion Jones. Character-level language modeling with deeper self-attention. *CoRR*, abs/1808.04444, 2018.

[75] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke S. Zettlemoyer. Deep contextualized word representations. In *NAACL-HLT*, 2018.

[76] Sebastian Ruder and Jeremy Howard. Universal language model fine-tuning for text classification. In *ACL*, 2018.

[77] Madhavi Ganapathiraju, Deborah K. Weisser, Ronit Rosenfeld, Juan Manuel Carbonell, Raj Reddy, and Judith Klein-Seetharaman. Comparative n-gram analysis of whole-genome protein sequences. 2002.

[78] Julian G. Zilly, Rupesh Kumar Srivastava, Jan Koutník, and Jürgen Schmidhuber. Recurrent highway networks. In *ICML*, 2017.

[79] Craig G. Nevill-Manning and Ian H. Witten. Protein is incompressible. In *Data Compression Conference*, 1999.

[80] Gavin E. Crooks, Jason Wolfe, and Steven E. Brenner. Measurements of protein sequence-structure correlations. *Proteins*, 57 4:804–10, 2004.

[81] Shiraj Khan, Sharba Bandyopadhyay, Auroop R. Ganguly, Sunil Saigal, David J. Erickson, Vladimir Protopopescu, and George Ostrouchov. Relative performance of mutual information estimation methods for quantifying the dependence among short and noisy data. *Physical review. E, Statistical, nonlinear, and soft matter physics*, 76 2 Pt 2:026209, 2007.

[82] Alexander Kraskov, Harald Stögbauer, and Peter Grassberger. Estimating mutual information. *Physical review. E, Statistical, nonlinear, and soft matter physics*, 69 6 Pt 2:066138, 2004.

[83] Sam Sinai, Eric Kelsic, George M. Church, and Martin A. Nowak. Variational auto-encoding of protein sequences. *CoRR*, abs/1712.03346, 2017.

[84] Matthew D. Zeiler. Adadelta: An adaptive learning rate method. *CoRR*, abs/1212.5701, 2012.