DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Variational Inference to Learn Representations for Protein Evolutionary Information

Issar Arab

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Variational Inference to Learn Representations for Protein Evolutionary Information

Variationsinferenz um Repräsentationen für Evolutionäre Information von Proteinen zu lernen

Author:	Issar Arab
Supervisor:	Prof. Dr. Burkhard Rost
Advisor:	Dr. Violetta Cavalli-Sforza
Advisor:	M.Sc. Michael Heinzinger
Submission Date:	15.09.2020

I confirm that this master's thesis in informatics is my own work and I have documented all sources and material used.

Munich, 15.09.2020 Issar Arab

Acknowledgments

I would like to express my sincere and deepest gratitude to my supervisor, Prof. Dr. Burkhard Rost, for his continuous support and guidance throughout this research work. I am also grateful to him for the genuine kindness and the opportunity he has given me to be part of a world-class lab contributing to cutting-edge research and competing with state-of-the-art technologies in the field of protein function and structure predictions. First, in my interdisciplinary project investigating and developing a pipeline to speed up and optimize the current protein sequence alignment process, and now in the domain of embedding language models for protein sequences. Moreover, I would like to extend my gratitude to my guide and advisor, Michael Heinzinger, for his time, constructive criticism, strict feedback, and quick answers to all my annoying questions, even during weekends. My numerous discussions with him have helped me enhance my knowledge and expertise in embedding language models and improve this work's quality. My warmest regards go to Dr. Violetta Cavalli-Sforza, for the valuable time spent reviewing my final report, providing feedback to improve it, and the rigorous correction of its content. I would also like to thank Tobias Olenyi and Michael Bernhofer for their contribution to the abstract's academic German translation. Finally, I would like to thank my parents, Rabia and Mohamed, for their unparalleled support, without which none of this would have been possible.

Danke! Thank you!

Abstract

Many of the machine learning (ML) models used in the field of bioinformatics and computational biology to predict either function or structure of proteins rely on the evolutionary information as summarized in multiple-sequence alignments (MSAs) or the resulting position-specific scoring matrices (PSSMs), as generated by PSI-BLAST. Due to the exhaustive database search to retrieve evolutionary information, the current procedure used in protein structure and function prediction is computationally exhaustive and time-consuming. This issue is becoming more problematic since the protein sequence databases are increasing in size exponentially over time, and hence raising PSI-BLAST runtime as well. According to previous experiments we have conducted, one query protein takes PSI-BLAST on average 14 minutes to build a PSSM profile. Therefore, in order to build a simple ML model, one may need a couple of months waiting for PSI-BLAST to generate PSSMs for a few thousands of query proteins. This runtime bottleneck issue makes it a major problem that requires an efficient alternative solution. A protein sequence is a collection of contiguous tokens or characters called amino acids (AAs). This analogy to natural language allowed us to exploit the recent advancements in the field of Natural Language Processing (NLP) and therefore transferring NLP state-of-the-art algorithms to bioinformatics. A recent prominent alternative to replace the need of PSSMs as input to our prediction methods is the direction of Embedding Language Models (ELMo), converting a protein sequence to a numerical vector representation. ELMo/SeqVec is a state-of-the-art deep learning pre-trained model that embeds a protein sequence into a 3-dimensional tensor of numerical values. This ELMo trained a 2-layer bidirectional Long Short-Term Memory (LSTM) network following a two-path architecture, one for the forward and the second for the backward pass, on an unsupervised task predicting the next AA from the previously seen residues in the sequence. The performance of the embedder was then evaluated on downstream tasks, such as secondary structure and subcellular localization predictions. The results showed that the embeddings succeed to capture the biochemical and the biophysical properties of a protein, but not achieving state-of-the-art performance. By merging the idea of PSSMs with the concept of transfer-learning during pre-training, we are aiming to deploy a new ELMo with a better embedding power than SeqVec by training a novel single branch bidirectional language model (bi-LM), with four times less free parameters. This is the first time that an ELMo is trained not only on predicting the next AA but also on the probability distribution of the next AA derived from similar, yet different sequences as summarized in a PSSM simultaneously (multi-task training), hence learning evolutionary information of protein sequences as well. To train our novel embedder, we compiled the largest currently curated dataset of sequences with their corresponding PSSMs of size 1.83 Million proteins (~0.8 billion amino acids). The dataset of proteins is reduced to 40% sequence identity, with respect to the validation/test sets, and contains sequences ranging between 18 and 9858 residues in length.

Zusammenfassung

Informationen über den evolutionären Hintergrund von Proteinen sind einer der zentralen Bausteine für die maschinengestützte Vorhersage von beispielsweise Struktur oder Funktion von Proteinen in der Bioinformatik. Diese Informationen werden meist in Multiple-Sequence-Alignments (MSAs) oder den daraus abgeleiteten positions-spezifischen Scoring-Matrizen (PSSMs), wie beispielsweise von PSI-Blast generiert, codiert. Aufgrund der großen Menge an bekannten Proteinen ist die Erstellung dieser Informationen jedoch sehr Zeit- und Ressourcen-aufwendig. Zusätzlich dazu steigt die Anzahl an bekannten Proteinen weiter und erhöht damit die Laufzeit und Komplexität der Erstellung um ein Vielfaches. Für das Entwickeln und Testen einfacher Maschine Learning Modellen können monatelange Laufzeiten zur Erzeugung der Trainingsdaten anfallen. Diesen Laufzeitflaschenhals wollen wir mit dieser Arbeit verbessern. Im Allgemeinen ist es möglich Analogien zwischen Protein-Sequenzen und natürlicher Sprache zu finden. Dies erlaubt es uns die aktuellen Entwicklungen in Natural Language Processing (NLP) auszunutzen und diese auf die Bioinformatik zu übertragen. Wir verwenden dabei ein Modell namens ELMo, welches bereits zuvor im Bereich von Sekundärstrukturvorhersage Erfolge erzielt hat. In dieser Arbeit vereinen wir das Konzept von PSSMs und ELMo unter Verwendung relevanter Techniken, wie beispielsweise Transfer Learning, um dadurch den Trainingsaufwand deutlich zu reduzieren. Im Vergleich zu SeqVec, trainieren wir unser Modell nicht nur auf der Vorhersage der nächsten Aminosäuren, sondern auch auf deren Wahrscheinlichkeitsverteilung, welche durch eine passende PSSM gegeben ist. Insgesamt ist unser Modell damit in der Lage die evolutionären Informationen eines Proteins zu erlernen.

Table of Contents

List of Figures

List of tables

List of Abbreviations

AA	Amino Acid	13
AI	Artificial Intelligence	2
bi-LM	Bidirectional Language Model	12
Bi-LSTM	Bidirectional Long Short-Term Memory	3
BLAST	Basic Local Alignment Search Tool	7
BPTT	Backpropagation Through Time	38
CDF	Cumulative Distribution Function	21
CD-HIT	Cluster Database at High Identity with Tolerance	23
CE	Cross-Entropy	25
Char-CNN	Character n-gram filters of a Convolutional Neural Network	2
ELMo	Embedding Language Model	2
KL-**Div**	Kullback-Leibler Divergence	31
LM	Language Model	4
LSTM	Long Short-Term Memory	3
LSTMP	LSTM with recurrent Projection layer	44
ML	Machine Learning	8
MMseqs	Many-against-Many Sequence Searching	2
MSA	Multiple Sequence Alignment	1
MSAM	Multiple Sequence Alignment Method	1, 2
NLP	Natural Language Processing	2
PDF	Probability Density Function	29
PIDE	Pairwise Sequence Identity	23
PPL	Perplexity	41
PSI-BLAST	Position-Specific Iterative Basic Local Alignment Search Tool	2
PSSM	Position-Specific Scoring Matrix	7
Seq_Evo_LM	Sequence Evolutionary Information Language Model Embedder presented in this research work	4
SeqVec	Sequence to Vector Embedder, modeling the un-annotated UniRef50 set of protein sequences with ELMo	3
SIMD	Single Instruction Multiple Data	7
TBPTT	Truncated Backpropagation Through Time	38
UniProt	Universal Protein Resource	2
VI	Variational Inference	29

1- Introduction

Multiple sequence alignment methods, or MSAMs, are a set of algorithmic solutions for the alignment of evolutionarily related sequences. They can be applied to DNA, RNA, or protein sequences. Those algorithms are designed to take into account evolutionary events such as mutations, insertions, deletions, and rearrangements under certain conditions [1]. In their recent Nature publication, Van Noorden R. et al. [2] mentioned that in the field of biology, the most widely adopted modeling approaches use multiple sequence alignment methods. This statement is supported by Altschul et al. [3] in their published manuscript of the tool "*Gapped BLAST and PSI-BLAST*," that is ranked the 11th most cited scientific paper of all times [2].

Indeed, the development of this vital modeling tool, MSA, necessitated the addressing of both complex computational and biological problems. MSA has always been recognized as an NP-complete problem. That is the reason behind the numerous alternative algorithms built, more than 100 variants, aiming at providing an accurate MSA over the last four decades [4]. Given the various alternatives of MSAMs, they all share a core asset: their reliance on estimated and typically greedy heuristics, enforced by the NP-complete nature of the problem. These heuristics are, more or less, dependent on specific data properties, like the length, the relatedness, the type of homology, and others.

The aim of MSA methods is to align any set of biological sequences, either RNA, DNA or proteins, in such a way that will capture their evolutionary, functional, or structural relationships. This is accomplished by adding gaps of different lengths within the sequences, enabling the homologous regions to be aligned with one another, which is analogous to aligning beads of similar colour in an Abacus. This is an interesting analogy simulated by the ancient counting frame, the Abacus, which was used as a calculator in Europe, China and Russia, centuries prior to the adoption of the written Arabic numerical scheme. From an evolutionary point of view, these gaps reflect insertions and deletions within the genome that are hypothesized or believed to have occurred during the evolution of sequences from a common ancestor [1].

Multiple sequence alignment methods can expose evolutionary restricted regions within a sequence. The results of these methods represent an important input to several downstream applications in the field of bioinformatics. Indeed, a large number of in silico studies and analyses depend on MSA methods, and machine learning models are at the hype, nowadays.

In contemporary computational biology and, more specifically, in the area of proteins and the prediction of their properties, sequence alignment forms the de-facto standard input to nearly all machine learning methods [5]. The main functionality of these alignment techniques is to search for homologues of a query sequence in a database of protein sequences, as they tend to share structure and function. For the past two decades, training machine learning models with evolutionary information representations, generated by multiple sequence alignments, has revolutionized the prediction power of AI inducers. Multiple aspects of protein function and structure were studied and investigated following the same approach and achieved significant results in the prediction performance. Such downstream specific tasks include protein secondary structure [6, 7, 8, & 9], transmembrane protein regions [10, 11, & 12], inter-residue contacts [13], and sub-cellular localization predictions [14, 15] as well as protein to protein interactions [16, 17, & 18]. However, this increase in performance is becoming costly in recent years, with the continuous exponential growth of bio-sequence data pools. UniProt is one example of such data stores, in which the entries keep doubling every couple of years [19].

In an earlier project we conducted last year (2019), we suggested the combination of both PSI-BLAST and MMseqs2 in a single pipeline to quickly generate extensive protein sequence alignments (PSI-BLAST vs MMseqs2 runtime analysis, pipeline construction, experiments and results are discussed in supplementary material, Appendix A.1). Currently, MMseqs2 [20, 21] appears to be the only MSA-based alternative solution out there coping with these enormous datasets, but at the expense of substantial hardware requirements. Over time, even MMseqs2 will soon become obsolete. To cope with such tremendous growth, alternative approaches are researched among the community. One prominent solution that can compete with conventional methods is the direction of Embedding Language Models (ELMo) [22], a state-of-the-art technology borrowed from the NLP field.

In the NLP setting, pre-trained word representations are a central component to several natural language comprehension models [23, 24]. However, learning high-quality representations is a difficult task. Ideally, these representations have to model both the dynamic features of word usage, like semantics and morphology, and how they differ across linguistic domains, like polysemy modeling. Pre-trained word vectors [23, 24, & 25] learned from a large corpus of unlabelled content have the ability to model these syntactic and semantic word representations. They represent the core of many state-of-the-art NLP architectures out there, such as semantic role labelling [26], question answering [27], and textual entailment [28]. One major limitation of these conventional word vector learning approaches is their ability to encode a single context-independent representation only for each word. The reason that has pushed other scholars to suggest alternative approaches able to overcome some of these shortcomings. Such solutions include enrichment of word vectors with sub-word extra knowledge [29, 30] or learning different vector representations [31] for word sense. In modern word vector representations, sub-word enrichment is achieved through character n-gram filters of a convolutional neural network (Char-CNN) [32]. Furthermore, other researchers have concentrated their efforts on directly learning context-dependent representations. In their paper Melamud et al. [33] presented a contextualized vector representation of a pivot word in a sentence through the training of bidirectional Long Short-Term Memory (LSTM) [20]. The above concepts were combined, by Peters et al. [22], to present a robust bidirectional embedding language model (ELMo) able to capture both context-independent (via Char-CNN) and context-dependent (via Bi-LSTMs) word representations taking full advantage of a large monolingual corpus of 30 million sentences.

An embedding language model (ELMo) is trained on a large corpus of unlabelled natural text, Wikipedia as an example [22], to predict the next most probable word in a sentence given all the previous seen tokens. However, in a bi-directional language model, during training we learn the probability distribution of the next word in the sentence from both directions, i.e. predicting a pivot word given all the previous tokens from a forward pass and from a backward pass of a sentence. This bi-directional autoregressive [22, 35] paradigm has revolutionized NLP allowing the model to develop a syntactic and a semantic self-learning of the word in a sentence, a.k.a. the context. This means that, for a particular word, the model will provide different contextualized embeddings, depending on the sentence it is used in.

Given the close nature of protein sequences to natural language sentences, the same approach was adopted to train SeqVec by Heinzinger et al. [36] on UniRef50, a corpus of 9.5 billion amino acids, which is around 10 folds larger than Wikipedia in terms of tokens(words). In their research work, Heinzinger et al. [36] proposed a novel embedding tool of protein sequences that replaces the explicit search for evolutionary related proteins in a database. The model was trained on predicting the next amino acid in the sequence. The new approach can be described as an implicit transfer of biophysical and biochemical information learned during the training of a bi-directional language model embedder (ELMo) [22], inspired from NLP, on a large unlabelled set of sequences. The predictive power of the embeddings was then tested on downstream tasks categorized under two levels: per-residue and per-protein predictions. The results showed that the models were able to reach a good performance, but did not outperform the state-of-the-art MSA-based tools. Our idea was then to train a novel bi-language model on PSI-BLAST output with transfer learning, which would eventually encode evolutionary information of the proteins within its embedding representations, with the goal of boosting the final embedding power.

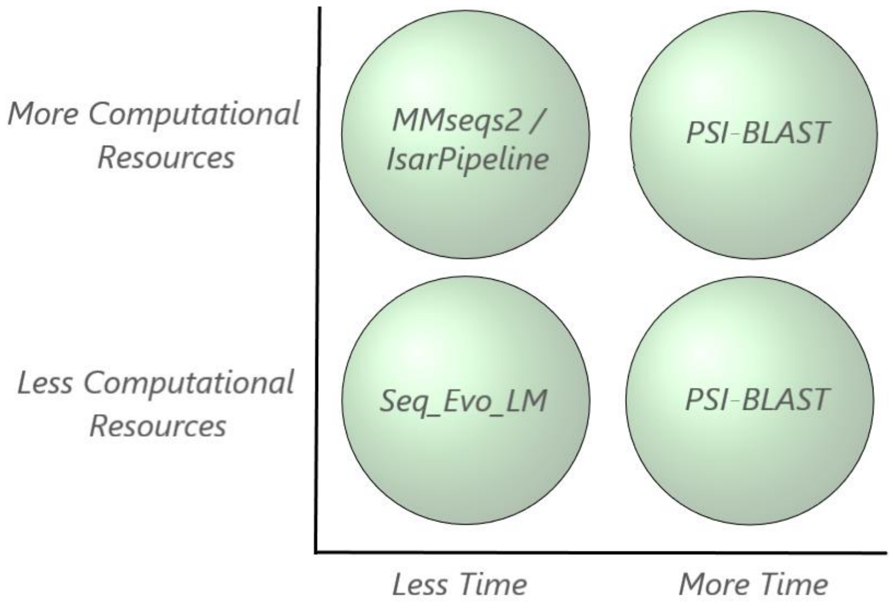


Figure 1. Tools and Software used to translate a protein sequence into numerical representations encoding evolutionary information of the protein. The graph displays the runtime (x-axis) vs the computational resources consumption (y-axis) by each of the 3 tools, PSI-BLAST, MMseqs2, and the Language Model Embedder (model presented in this manuscript and given the name of Seq_Evo_LM) to generate numerical representations encoding evolutionary information.

Figure 1 shows a graphical representation where each of the tools stands when it comes to computing resource consumption and the time needed to generate the output. PSI-BLAST proves to take have a much longer running time, with or without the availability of more computing resources (Figure A.1 in Appendix A), whereas, IsarPipeline, combining both MMseqs2 and PSI-BLAST, proves to be faster given enough computing resources. Our aim in this research is to present a proof-of-concept tool marrying both the evolutionary information and LMs to get the best of both worlds, hence deploying a pre-trained model standing in the bottom left corner of the figure, which achieves an optimal trade-off between time and resource consumption.

Four main aspects must be taken into consideration in the design and the modelling of the new sequence embedder:

1. **Corpus size:** Our training dataset of 1.83 Million proteins (~0.8 billion amino acids) is of a size as similar as the largest existing NLP corpus (1 billion words). Therefore, high capacity models are required to capture the hidden representations of our input sequences.
2. **Vocabulary size:** The English language vocabulary contains 100,000 more tokens(words) compared to the proteins vocabulary, which counts 20 standard amino acid characters. The residues[1] are used to uniquely represent a protein sequence. This

[1] A residue is an amino acid

might be problematic if the complexity encoded in a natural language sentence is comparable to the one encoded in a protein sequence but using a much smaller dictionary.

3. **<u>Sequence length:</u>** On average, the range of an English sentence is between 15 to 30 words [37], with some exceptional extreme cases in the literature, such as James Joyce's Ulysses, reaching up to 4000 words in a single sentence. Proteins, though, go beyond that limit with a range between 30 and 33000 residues in a sequence. Longer proteins might be problematic as they require larger GPU memory to be trained on without mentioning the LSTMs limitations in remembering dependencies within long-range settings.
4. **<u>Architecture:</u>** SeqVec/ELMo model already contains 93.6 Million free parameters, meaning that the network capacity is large enough to capture biophysical and biochemical information within the sequence. The architecture provides a per-residue embedding of size (3 x 1024). The huge number of parameters is due to the two-parallel paths architecture of the network. The first path provides per-residue contextualized embeddings from a forward pass. While the second generates embeddings via a backward traversal of the sequence. Our aim is to deploy an architecture that has a smaller number of free parameters, a lower vector embedding dimensionality, and one path processing both the forward and the backward traversal of a sequence while maintaining a good, or maybe even better, predictive power on downstream tasks.

2- Background

2.1- PSI-BLAST

PSI-BLAST [38] stands for the Position-Specific Iterative Basic Local Alignment Search Tool. It is a software that runs a multiple sequence alignment algorithm powered by the dynamic programming optimization paradigm to search a given database of protein sequences. Given a query sequence, the algorithm retrieves homologous sequences that pass a predefined threshold. This threshold-based computational similarity method, which uses protein-protein BLAST [38, 39] to identify regions of local alignment, is used to construct a Position-Specific Scoring Matrix (PSSM) from the retrieved family of sequences. The matrix is used as input to subsequent iterations of the software to further search the database and retrieve more hits, which are then used to update and improve the PSSM itself. PSI-BLAST was used in this research to generate the PSSMs that serve as our data labels.

2.2- PSSM

A PSSM stands for the Position-Specific Scoring Matrix, which is also referred to as a profile matrix. The PSSM profiles contain statistical representations of residues in a given sequence of a protein with respect to all its relevant aligned protein sequences (i.e. homologs) in the database. A profile captures the conservation pattern in an alignment and stores it as a matrix of scores for each position in the alignment [38], where high scores are assigned to highly conserved positions and lower scores to low conserved ones. Hence, the matrix representation includes what we call the protein evolutionary information. PSSMs were used in this research as targets for training our novel ELMo architecture.

2.3- MMseqs2

MMseqs [20, 21] stands for Many-against-Many Sequence Searching. It is an extremely optimized scalable software used to search and cluster very large protein sequence datasets. It is an open source cross-platform tool developed in C++ using the full power of modern CPU architectures, especially the SIMD optimized instruction set processing unit. The tool implements a SIMD accelerated Smith-Waterman-alignment algorithm [40] to compute the alignment of protein sequences. The owner of the software claims that it is extremely fast that it can run 1000 times faster than BLAST, especially when it comes to batch querying. MMseas2 was used in the research to perform redundancy reduction of protein sequences at a cut-off of 40% sequence identity.

2.4- Artificial Neural Network

Machine learning (ML) is the modern technology enabler behind achieving artificial intelligent capabilities performed by computers. It is simply defined as the process of using algorithms and statistical methods to analyze data and learn patterns within it to start then making decisions or forecasts on future unseen observations. Technically, machine learning is about reducing uncertainty by combining both the sample data and prior assumptions.

One popular machine learning algorithm is the artificial neural network (ANN), also called a multi-layer perceptron (MLP) or a feed-forward network [41, 42]. It is a powerful algorithm that has the ability to extract complex features andto perform hierarchical feature extraction. The ultimate goal of using a Neural Network is to project input data points to a new dimensional space, also called latent space, using non-linear basis functions to allow for better probabilistic decision boundaries. This mechanism is achieved by fixing the number of basis functions in advance but allowing them to be adaptive via the use of parametric forms of functions in which the parameter values are adapted/learned during training time [43]. All those functions are represented as a compute graph[2] in a network, and their parameters; also called weights, are adapted via the use of a loss function that guides the training of all the network parameters during back-propagation. For complex problems, neural networks may contain multiple layers, each with a predefined number of hidden neurons, stacked on top of each other to increase the network's learning capacity. A network of multiple stacked layers is referred to as a deep neural network or deep learning (DL) within the AI community. We built a deep learning model to answer our research question and deploy a novel embedding language model.

2.5- Long Short-Term Memory (LSTM)

Neural networks are powerful ML algorithms that perform exceptionally well in a verity of tasks, such as computer vision and robotics. However, when it comes to the notion of “time” or “sequence,” feed-forward networks perform poorly. To allow for context capturing and more accurate inference, recurrent neural networks (RNN) are widely used for such type of tasks.

Vanilla RNNs are known to suffer from short-term memory. RNNs struggle to carry information all the way from early time steps when it comes to long sequences. Technically, the primary cause is the vanishing/exploding gradient problem. A recurrent neural network cell shares the same weights for all the time steps in a sequence. Since weights in an RNN are multiplied over and over while traversing the sequence, during backward propagation and in case of a long sequence,

[2] A compute graph [44] is a directed graph where on each node we have an operation, and an operation is a function of one or more variables and returns either a number, multiple numbers, or a tensor.

we face vanishing gradient when the weights are small and exploding gradient when weights are large. For the exploding gradients, the issue can simply be resolved by gradient clipping to a norm of 1. Concerning the vanishing gradient problem, a new architecture named “Long short-term memory” has been introduced by Hochreiter and Schmidhuber [34], providing a solution to this raised concern. An LSTM, Figure 2, consists of internal structures called gates used to regulate the flow of information in the network [55], learning pieces of information to be dropped and capturing important ones to be passed along. LSTMs are the main building blocks of our novel bidirectional embedding language model.

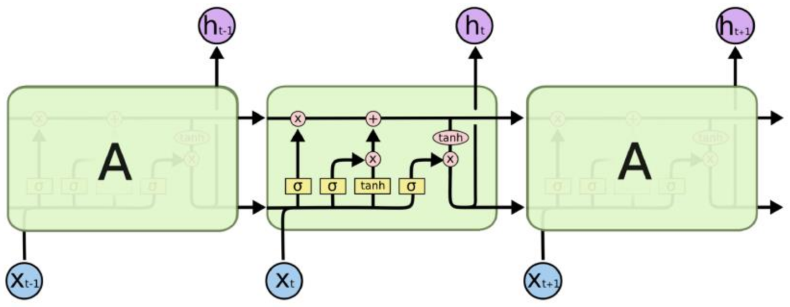


Figure 2. Unrolled Long Short-Term Memory cells

2.6- Dropout and Inverted Dropout

While training a fully connected network, any given layer in the architecture covers all the parameters, which pushes the hidden nodes to build some kind of co-dependency with one another. This behavior negatively affects the individual power of each hidden unit in the layer, and hence, resulting in overfitting to the training dataset. Hinton et al. [45] were the first to present dropout training as a method to combat overfitting in their paper "Improving neural networks by preventing co-adaptation of feature detectors." Dropout is simply defined as a random selection of a given layer neurons on which it is applied, which are then ignored during training iterations of the model. During each iteration, neurons to be dropped are randomly selected with a probability of *1-p*. Similarly, neurons to be kept are selected with a probability of *p*, a hyperparameter referred to as the “keep probability.” The discarded nodes, along with their incoming and outgoing edges, are ignored in the forward and the backward pass during training. Technically speaking, the paths in the network are ignored by setting a value of zero to the corresponding weights. The process might be considered equivalent to model-averaging since the approach simulates the training of several related network architectures on the same data. Figure 3 represents a visualization of the dropout concept applied on a 3-layer fully connected network.

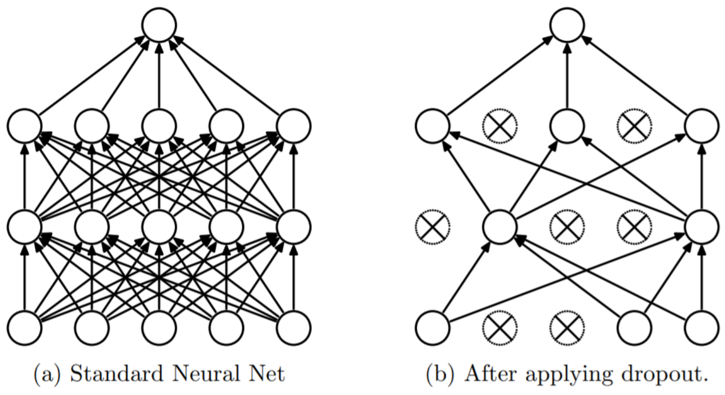


Figure 3. Concept of dropout in a neural network architecture [46]. (a) The figure on the left shows an example of a standard fully connected network of 2 hidden layers and one output layer. (b) The figure on the right shows a reduced network, the result of applying dropout on the original network on the left, where the crossed neurons represent the dropped units.

In fact, training with dropout is a well-known technique falling under the bucket of learning approaches that artificially distort training data to improve predictions, generalize better, and make inference more stable [47, 48, 49, 50]. This artificial feature corruption is a common technique used for regularization during training of AI inducers.

After applying dropout during training, in order to maintain the expected value of the hidden units in our network, we have to scale up the weights during evaluation/testing phase. A variant of the traditional dropout is the “Inverted dropout” technique. It applies the same principle of keeping a subset of neurons according to the “keep probability” *p*. The only difference is that, during training and in order to maintain the same expected value of the non-linear hidden units, we scale the activations by the inverse of the probability *1-p*. Therefore, we maintain the same weight values during both training and evaluation, while preventing the activations of our network from exploding while training. Inverted dropout was used in this research as a regularization technique applied on the linear projection layer module within the customized 2 LSTM layers.

2.7- Residual Block

Neural networks are known as universal approximators, where the learned function accuracy keeps increasing with the increasing network capacity. Simply said, the capacity of a network increases with more hidden layers. In theory, ANNs should be able to learn any simplex or complex function given a large enough network architecture. However, in practice, we notice a degradation problem of the model accuracy with the increasing number of layers. It is definitely not a representational problem, since with deeper architectures we end up with a larger capacity capturing more complex mappings. It is also not an overfitting problem caused by the large capacity of the network since the training error starts also getting higher. It turns out that with larger networks training becomes harder, vanishing and exploding gradients start being noticed, and learning even a simple identity function gets difficult. To solve this problem, He et al. [51] suggested an architectural design solution that will allow us to build larger networks without degradation of our model accuracy. The idea is to add a direct path for information flow by short-circuiting blocks of layers as shown in figure 4. The residual block will then learn a deviation from the identity mapping, where $F(x; w) = H(x) – x$ is the residual mapping. The identity is easier for the residual block to learn and with a skip connection it is guaranteed that performance of the network will not be hurt but only improve. To improve performance of our model, we built a skip connection (residual block) in our final ELMo architecture from the first LSTM layer to the second.

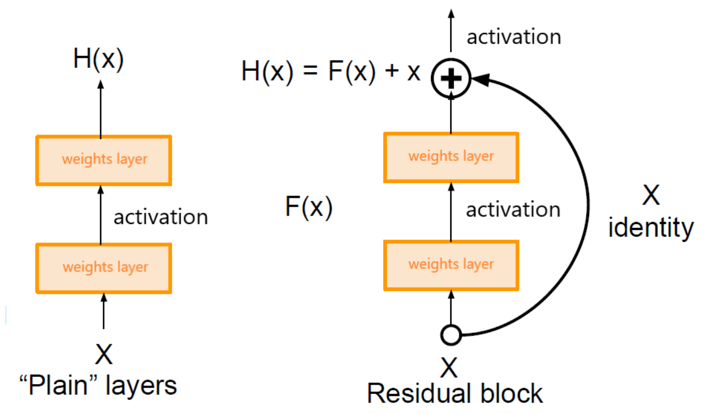


Figure 4. Conceptualized architecture of the skip connection [51]. The sub-figure on the left shows an example of a plain network of a 2-layer architecture, while the sub-figure on the right shows a residual block with two staked layers and one skip connection.

2.8- Transfer learning

Insufficient training data is an unavoidable phenomenon in certain domains, such as bioinformatics, medicine, and robotics. In these contexts, data collection is a complex and costly task, making it incredibly challenging to create massive and high-quality annotated datasets. Since the independent and identically distributed (i.i.d) hypothesis between training and test data is relaxed in transfer learning [52], the later has become an essential strategy in machine learning to apply in order to solve the underlying problem of insufficient training data. The general idea of transfer learning is to transfer the knowledge learned on a huge dataset, called the source domain, to perform better on a new task with few data points, called the target domain. Therefore, the target domain model does not have to be trained from scratch, which will significantly reduce training time and the need for a more extensive training set. Our novel ELMo architecture was trained with transfer learning, where SeqVec output embeddings served as input to our model.

2.9- Data augmentation

When a network architecture behaves in such a way that it perfectly models the training data, we say that it has learned a very high variance function. This behaviour is referred to as overfitting and it is one of the major challenges while training deep learning models. In today's worldwide deployed models, the high empirical performance of deep neural networks in various tasks was extremely reliant on massive datasets to generalize better. Unfortunately, not all application areas have access to a large volume of data.

There exist multiple techniques to fight overfitting while training a model. Data augmentation is one of those strategies that addresses the problem from its root, which is the data itself, under the hypothesis that additional information can be extracted from the original dataset via augmentation [53]. Data augmentation is a generally used routine operating at the data-space level to tackle this issue by enhancing the size and the quality of training datasets. Generally speaking, the process artificially inflates the training dataset size by either data warping or oversampling. In image processing, such applied techniques include rotation, noise injection, flipping, cropping, and colour space transformations. In this research, we built a bi-LM that is trained by traversing the sequence from both a forward and a backward pass. This is achieved by appending a second copy of the protein sequence in the reverse order, which multiplies the training samples by a factor of 2. This plays the role of an implicit data augmentation, in this research project.

3- Research question

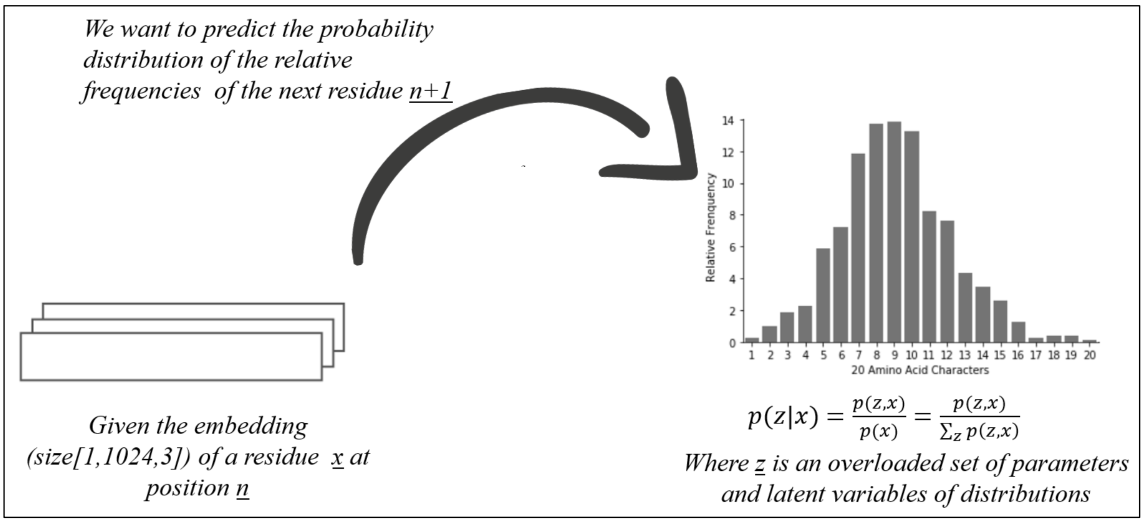
Before jumping forward into our research question formulation, let us have a quick detour and explain a basic probabilistic concept. This concept will allow us to mathematically formulate the core of our research question.

In probabilistic machine learning, a probabilistic model is a joint distribution of hidden variables, referred to as $\underline{z}$, and observed variables, referred to as $\underline{x}$. This statement can be written in probabilistic notations as $p(x, z)$.

In this setting, inference about the unknowns $\underline{z}$ is done through what we call the *posterior* distribution. It is a conditional distribution of the hidden variables given the observations. This statement is translated in probabilistic notations as $p(z \mid x)$.

Our general goal is to go directly from the amino acid sequence of a given protein to a Position-Specific Scoring Matrix, more specifically to the matrix of relative frequencies. In other words, we want to translate a vector of characters of length L, here the protein sequence, to a matrix of dimensions L x 20, which is the Position-Specific Scoring Matrix in our context. In this PSSM map, each row represents the relative frequencies of a residue in the sequence with respect to all the 20 known AAs. These row values represent a discrete probability distribution, where the statistics in each row sum up to 1 or 100 depending on the scale. Our approach aims at learning those representations directly from the sequence following the same paradigm applied in autoregressive models [54]: From the observations of previous time steps given as an input, we want to make predictions of the next time step.

Figure 5 visualizes the main problem statement in this research work, which aims to learn protein evolutionary information. In a protein sequence of length L, given the embedding of a residue $\underline{x}$ at position $\underline{n}$ as well as the previous residues information passed on, we want to predict the probability distribution of the relative frequencies of the next residue $\underline{n+1}$. In mathematical notations, this is equivalent to a conditional probability distribution $p(z \mid x)$, where $\underline{z}$ is an overloaded set of parameters and latent variables of the distribution.



We want to predict the probability distribution of the relative frequencies of the next residue n+1
Relative Frenquency
20 Amino Acid Characters
Given the embedding (size[1,1024,3]) of a residue x at position n
$p(z|x) = \frac{p(z,x)}{p(x)} = \frac{p(z,x)}{\sum_z p(z,x)}$
Where z is an overloaded set of parameters and latent variables of distributions

Figure 5. Problem statement visualization of this research project on learning representations of protein evolutionary information

4- Data and methods

In this major chapter, we present the material used and the methodology followed during this research project. The experiments we performed to answer our research question are described in detail later in the manuscript.

As the project in hand is a Machine Learning task, we abided by the best practices of data science to solve this research problem. Figure 6 [56] illustrates a graphical representation of the model building procedure to develop the final optimal inducer, which will be then used as an Embedding Language Model of protein sequences. Our methodology involves:

- Data gathering, cleaning, and pre-processing
- Splitting data into training, validation, and test sets
- Input features definition
- Model building and hyperparameter tuning to pick the optimal configuration
- Evaluation and benchmarking

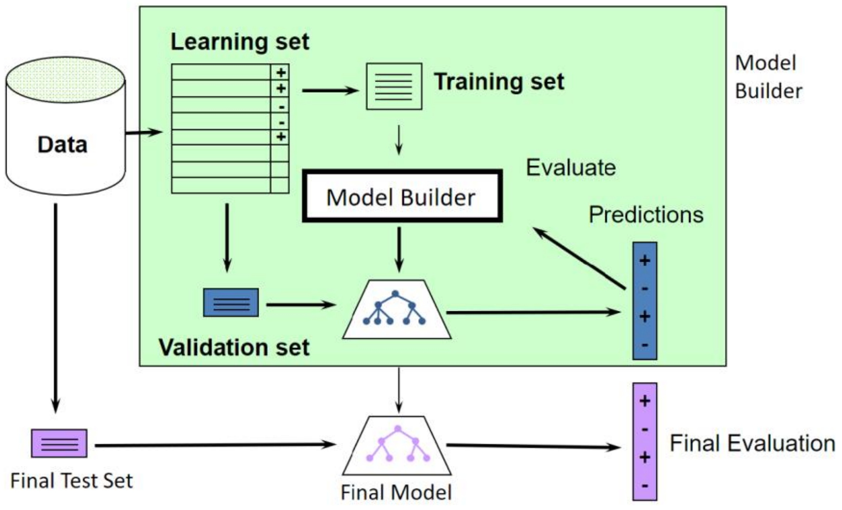


Figure 6. Graphical representation of the Data Science best practices to perform hyperparameter tuning and battle overfitting

4.1- Data gathering, cleaning, and pre-processing

For every machine learning task, careful data preparation is a crucial requirement to build good quality models. Another main requirement is the size of the development set. We wanted to use a larger dataset so that we could deploy a robust model that generalizes better to unseen data. For our training, we needed both a large set of protein sequences and their corresponding profile matrices, i.e. PSSMs. The first approach is to collect a set of unique proteins and run PSI-BLAST on it to generate high quality PSSM matrices. As we planned to work with a large set from UniProt [57] Reference Cluster with 50% sequence identity (uniref50 2019_12), ~38.8 Million proteins, running PSI-BLAST on it will take several months. This is an unrealistic path to take. To overcome this obstacle, we opted for a smart alternative.

Predict-Protein [58] is an Internet service for sequence analysis as well as prediction of protein structure and function. It represents an interface to Rost Clusters where multiple prediction models are deployed and huge amount of data are stored. Those clusters are used by internal Rostlab members and run jobs submitted by external users through the Predict-Protein interface. Since most deployed models and analysis tools make use of PSSM files as an input, running PSI-BLAST each time makes the Web application very slow. To reduce the response time, a clever mechanism was implemented. Whenever a new sequence is submitted by a user, the computed PSSM file as well as the alignment profile are cached within Rost Clusters. Therefore, only the initial submission of the protein sequence has the slow response time. For any upcoming requests, the system just loads the pre-computed results from the cache.

The files in the Predict-Protein cache are stored according to a hash built from the protein sequence. Also, it is always easier to go from a set of sequences to a set of hashes than randomly crawling through the hashes and looking for a suitable set of sequences. The trick was, then, to run a pipeline that would iterate over all the Uniref50 sequences and check whether the cache already contains the PSSM file of a given protein. If the profile was already computed, the pipeline retrieves it. The idea was to gather as many proteins and their PSSM files as possible without the need to run PSI-BLAST on our set. At the same time, a second path in the retrieval pipeline was executed in order to retrieve all stored sequence alignment profiles of proteins in our set. We needed the second file to filter out profile matrices with not enough evolutionary information. This procedure will be explained in more detail bellow.

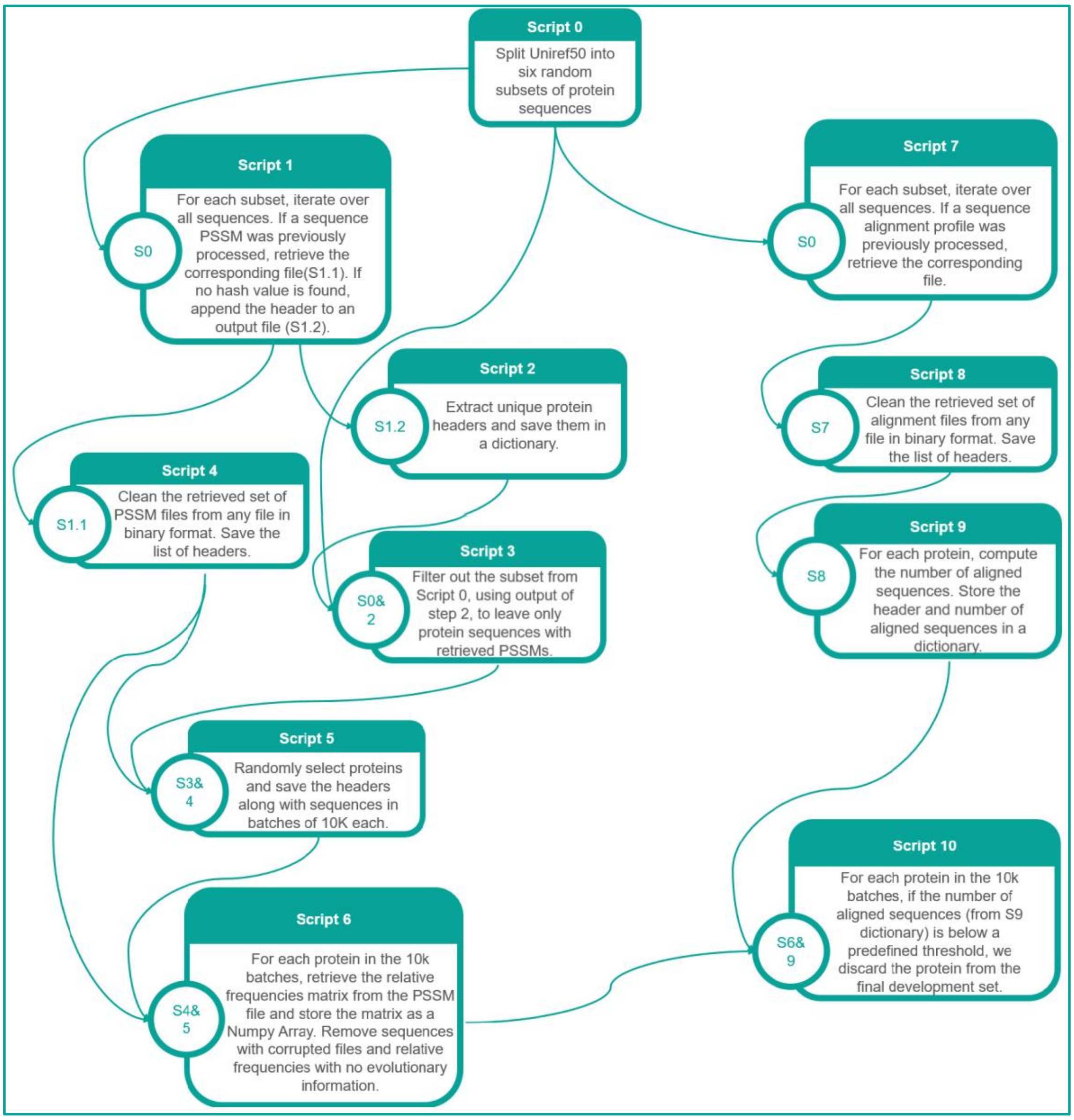

Script 0
Split Uniref50 into six random subsets of protein sequences
Script 1
S0
For each subset, iterate over all sequences. If a sequence PSSM was previously processed, retrieve the corresponding file(S1.1). If no hash value is found, append the header to an output file (S1.2).
Script 7
S0
For each subset, iterate over all sequences. If a sequence alignment profile was previously processed, retrieve the corresponding file.
Script 2
S1.2
Extract unique protein headers and save them in a dictionary.
Script 8
S7
Clean the retrieved set of alignment files from any file in binary format. Save the list of headers.
Script 4
S1.1
Clean the retrieved set of PSSM files from any file in binary format. Save the list of headers.
Script 9
S8
For each protein, compute the number of aligned sequences. Store the header and number of aligned sequences in a dictionary.
Script 3
S0& 2
Filter out the subset from Script 0, using output of step 2, to leave only protein sequences with retrieved PSSMs.
Script 5
S3& 4
Randomly select proteins and save the headers along with sequences in batches of 10K each.
Script 10
S6& 9
For each protein in the 10k batches, if the number of aligned sequences (from S9 dictionary) is below a predefined threshold, we discard the protein from the final development set.
Script 6
S4& 5
For each protein in the 10k batches, retrieve the relative frequencies matrix from the PSSM file and store the matrix as a Numpy Array. Remove sequences with corrupted files and relative frequencies with no evolutionary information.

Figure 7. Pipeline of scripts applied to retrieve, clean, and preprocess the final development set that will be used for building the Sequence Embedding Language Model. The first module (Script 0) provides input to the pipeline, while the last module (Script 10) generated the final output organized in batches of 10k sequences each.

.

Figure 7 represents graphically the flow of the data and provides the description of every script in the protein's evolutionary information retrieval pipeline. The pipeline starts by splitting the initial Uniref50 set of 38.8 Million proteins into 6 subsets, each of around 6.5 Million proteins. The purpose of splitting the initial set is to leverage the availability of multiple nodes in the Clusters in order to run multiple jobs in parallel, hence speeding up the retrieval process. Additionally, we see from the figure that the pipeline branches to two independent paths, the first starts with script 1 and the second starts with script 7. Each path of scripts was run in 6 different copies for parallel processing, each in a separate node in the cluster.

The first path focuses on retrieving all the available PSSM files of corresponding protein sequences from the cache. Script 1 traverses the list of proteins in the subset, and checks, for each sequence, whether the hash code matches one in the cache. If a hit occurs, the corresponding PSSM file is retrieved. Otherwise, the header is appended to an output file. The output file is then parsed to extract all the unique headers, which then are organized in a dictionary data structure. The dictionary helped decreasing the runtime tremendously, as it has a *O(1)*, when it comes to checking if a key/header exists in the set or not. It was used to filter out the original subset of proteins to only keep the sequences of proteins for which a PSSM file was retrieved. Furthermore, we realized that some of the PSSM files were fetched in binary format. Here comes the role of Script 4, which goes over all the files one by one and checks the file content using the "*file*" command from the list of GNU Core Utilities for Unix environments. Via Linux **pipes** capabilities, the filtering task of binary files was performed by the following bash script:

```
$ for i in * ; do file $i \
              | grep -v ASCII    \
              | awk -F: '{print $1}'; done
```

Once all the protein sequences are filtered out along with a clean set of their corresponding PSSM files, an additional pre-processing step was required. The pipeline of scripts was executed in parallel for each of the six subsets. For each batch, a directory of corresponding PSSMs and a second for alignment information files was generated. It was calculated that around 600 GB of diskspace was required to store all of the 12 generated folders. Much of the information included in those files was of no relevance to our problem. Additionally, keeping the evolutionary information for each protein in a single file would heavily affect the performance of the system, as it has to deal with millions of separate files in one folder. Each filesystem has a fixed maximum threshold of what is called *inodes*. It may occur that the hosting OS filesystem will runout of *inodes*, and, hence, ending up with a huge set of overridden corrupted files. The performance hit can also be very noticeable in accessing the files, removing files, or listing the set of files in the folder, and will also impact badly the transfer of PSSM files via the network. To solve this issue, we had to come up with an idea that would decrease both the storage and the huge number of files needed.

As it is mentioned earlier, a PSSM is a matrix based on the nucleic acid (a.k.a. amino acid or residue) frequencies at every position of a multiple sequence alignment. The calculation results of those frequencies are then represented in a matrix that will assign superior scores to amino acids appearing in a certain position more often than by chance. In our PSSM file, there are 2 main matrices both of size Lx20, where L is the length of the protein sequence. The first matrix, the one to the left, is referred to as the substitution score matrix and consists of integer values that might be either positive or negative. Whereas, the second matrix, the one to the right, is referred to as the relative frequencies matrix and consists of values between 0 and 1 (probabilities). In fact, both matrices are similar. It is easy to derive one matrix from the other. Our interest in this research project is on the relative frequencies' matrix. Script 5 parses each PSSM file to extract the relative frequencies matrix for each protein sequence, stores the values in a Numpy Array, and then cast all values from a float to an Integer datatype. During the extraction process, three checks were performed before finally saving the relative frequencies matrix:

- The PSSM file is corrupted
- All values in the relative frequencies' matrix are zeros (i.e. no evolutionary information)
- A value of 100 is assigned to the same amino acid in all the residues of the sequence (i.e. no evolutionary information)

If one of the 3 conditions is met, the corresponding protein is discarded from our final development set. Concerning the second path of the pipeline and following the same procedure, we retrieved all the alignment files corresponding to our Uniref50 set of proteins, cleaned the binary files, and then parsed each file to only keep the family size of the sequences aligned to each protein in our set. The output was then stored in a dictionary, where the key was the protein header and the value was the family size number.

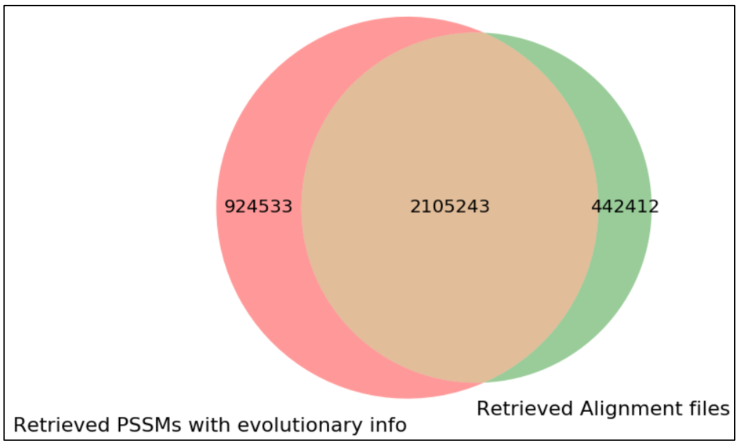


Figure 8. Venn diagram representing the number of retrieved files by each path in the pipeline. The green set is the total number of proteins for which a readable alignment information file was retrieved. The pink set denotes the total number of proteins for which a readable PSSM file with evolutionary in formation was retrieved. Whereas the brown set in the middle is the intersection between them (both aligned family and the relative frequencies matrix are available). The numbers indicate the number of proteins in each set.

Figure 8 shows the number of proteins for which relevant information to our task is retrieved from Rost Clusters. Out of 38.8M proteins, 3M have precomputed relative frequencies matrices and 2.44M have alignment information. Additionally, among the proteins with retrieved relative frequencies, only two-thirds (2/3) have their corresponding aligned family of sequences used to build the PSSM matrices. That is the intersection between the two sets which counts around 2.1M proteins, and that is our final development set.

To analyze our development set in terms of the amount of evolutionary information encoded in the relative frequencies' matrices, we visualized the distribution of protein sequences in terms of the family size of the number of aligned proteins for each sequence in our development set. Figure 9 displays the results of our analysis. It is an extremely right skewed distribution showing that the majority of our protein PSSM files are built on family sizes of more than 1000 aligned sequences while some 200k PSSMs are built on families of size less than 10 aligned sequences.

To maintain a high-quality training set, we set a threshold of the family size for which proteins will be either discarded or used in final training. To pick the threshold, we visualized the percentage of proteins in our development set that will be preserved at each threshold value. We call this percentage the cumulative distribution. In probability theory, the cumulative distribution function (CDF) is simple the percentage/probability of a random variable taking a value less than or equal to certain threshold *x*. In mathematical notations, this can be expressed for a discrete distribution as:

$$F(x) = \Pr[X \leq x] = a$$

$$\Leftrightarrow F(x) = \sum_{i=0}^{x} f(i)$$

Eq. (1)

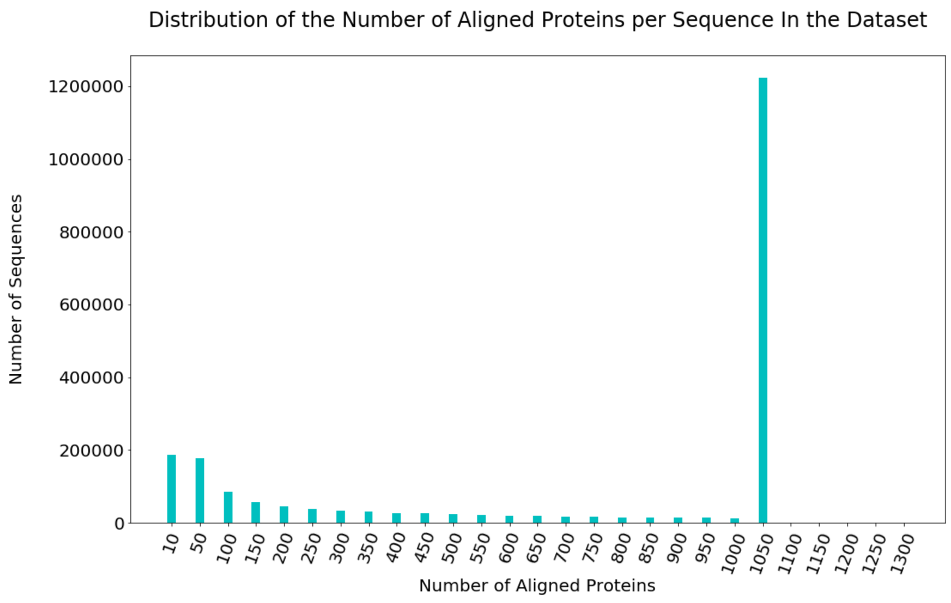


Figure 9. Bar chart representing the distribution of the number of aligned sequences (family size) for each protein in our dataset

In figure 10, we computed and displayed the complement of the cumulative distribution at each threshold value. Mathematically speaking, we visualized the distribution of $1 - F(x)$, where $F(x)$ is the CDF of the number of aligned proteins to each sequence in our development set. Our analysis showed that 9% of the sequences have less than 10 aligned proteins, leaving us with 91% of our total development set with significant encoded evolutionary information. The threshold of 10 led to a number around 1.9 Million cleaned proteins with corresponding position specific scoring matrices. The exact number of sequences output by each script is detailed in table 1.

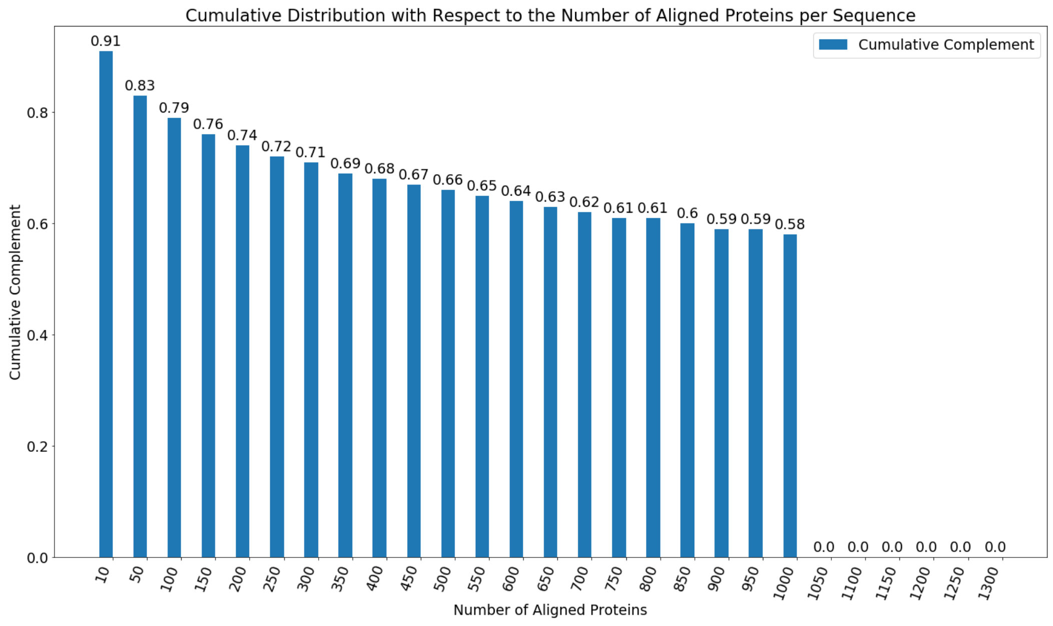


Figure 10. Bar chart representing the complement cumulative distribution with respect to the number of aligned proteins per sequence in our development set (~2.1 M proteins)

Table 1. The numbers in this table represent the output of each retrieval/filtering script in the above pipeline. Since the Uniref50 set was split into 6 batches for parallel processing, each column shows the number of sequences falling in each of the clusters described on the left for each batch. The column in the far right shows the total over the 6 batches.

	Batch #1	**Batch #2**	**Batch #3**	**Batch #4**	**Batch #5**	**Batch #6**	**Total**
Proteins in the batch	5659495	6072076	6878058	6779562	6702723	6764747	**38856661**
All retrieved from Rost Clusters	1111829	883891	558726	594437	475116	146545	**3770544**
Binary format retrieved files	10578	2268	1985	4264	3447	1191	**23733**
Nonbinary format retrieved files	1101251	881623	556741	590173	471669	145354	**3746811**
Corrupted files	90	8477	1813	7	6	13	**10406**
PSSMs with no evolutionary information	177692	163669	115558	123862	102267	24277	**707325**
Clean set with evolutionary information	923559	709477	439970	466304	369402	121064	**3029776**
Proteins with alignment info and PSSMs with significant evolutionary information (alignedProteins >= 10)	663449	437665	263310	267377	201989	84563	**1918353**

4.2- Splitting data into training, validation, and test sets

Having gathered a high-quality development set, we must split it into training, validation and test sets. For that purpose, we randomly selected from the full development set a kingdom of 2k protein sequences, and further split it into two halves: one subset for validation and the second for testing. Since the proteins in our original set are already reduced to 50% pairwise sequence identity (PIDE), we wanted to further reduce our test/validation set to 40% PIDE.

Two options were considered, CD-HIT [59], which is a program for clustering and comparing large sets of protein or nucleotide sequences, and MMseqs2, which is a fast tool for mining massive sequence sets [20]. MMseqs2 utilities for clustering protein sequences that meet a certain PIDE threshold are much faster than CD-HIT. MMseqs2 applies a greedy heuristic incremental algorithm analogous to CD-HIT length-based clustering procedure as described in Figure 11.

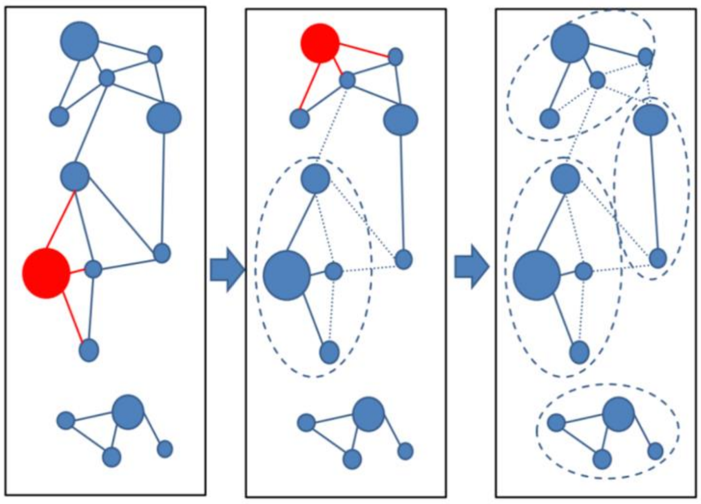


Figure 11. Visual showing how the greedy heuristic incremental clustering algorithm takes the longest sequence, depicted by a larger node within the graph, and places all the sequences that are connected to it in one cluster. Repeatedly, the process is run over and over forming next clusters from the remaining set. This figure was reprinted from [20]

We used MMses2 to perform a target coverage alignment-based clustering on our random subset of 2k proteins. To perform this operation, we run the following two commands:

```
$mmseqs search --cov-mode 1 -c 0.4 queryDB targetDB resultDB

$mmseqs convertalis queryDB targetDB resultDB cluster.tuple --format-output \
                           "qheader,theader,tseq"
```

In our case, both the queryDB and the targetDB represent the same set of 2k proteins. After running the commands, sequences are grouped in the same cluster if they satisfy at least 40% length overlap coverage on the target sequence as shown in the figure 12 below. The procedure results in clusters of similar proteins meeting the predefined threshold, with the longest sequence as the cluster representative. The set is then reduced by keeping only the protein sequence cluster representative and discarding the rest.

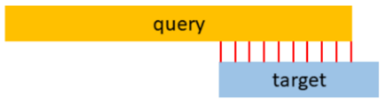


Figure 12. The figure shows that the aligned query sequence (yellow bar) covers at least 80% of the target sequence (blue bar). This figure was thankfully taken from [20]

The redundancy reduction operation resulted in a reduced validation/test set of 1758 protein sequences, which was split into two subsets one for validation and one for testing. The same procedure was applied to reduce the training set to a 40% PIDE cut-off. In the second run, the queryDB was the reduced validation/test set and the targetDB was the training set. The process reduced the training set to a final number of 1.83 Million proteins. Figure 13 visualizes the whole preprocessing pipeline of our dataset starting from the initial Uniref50 to the final training validation and test sets with the resulting filtered number of sequences after each step.

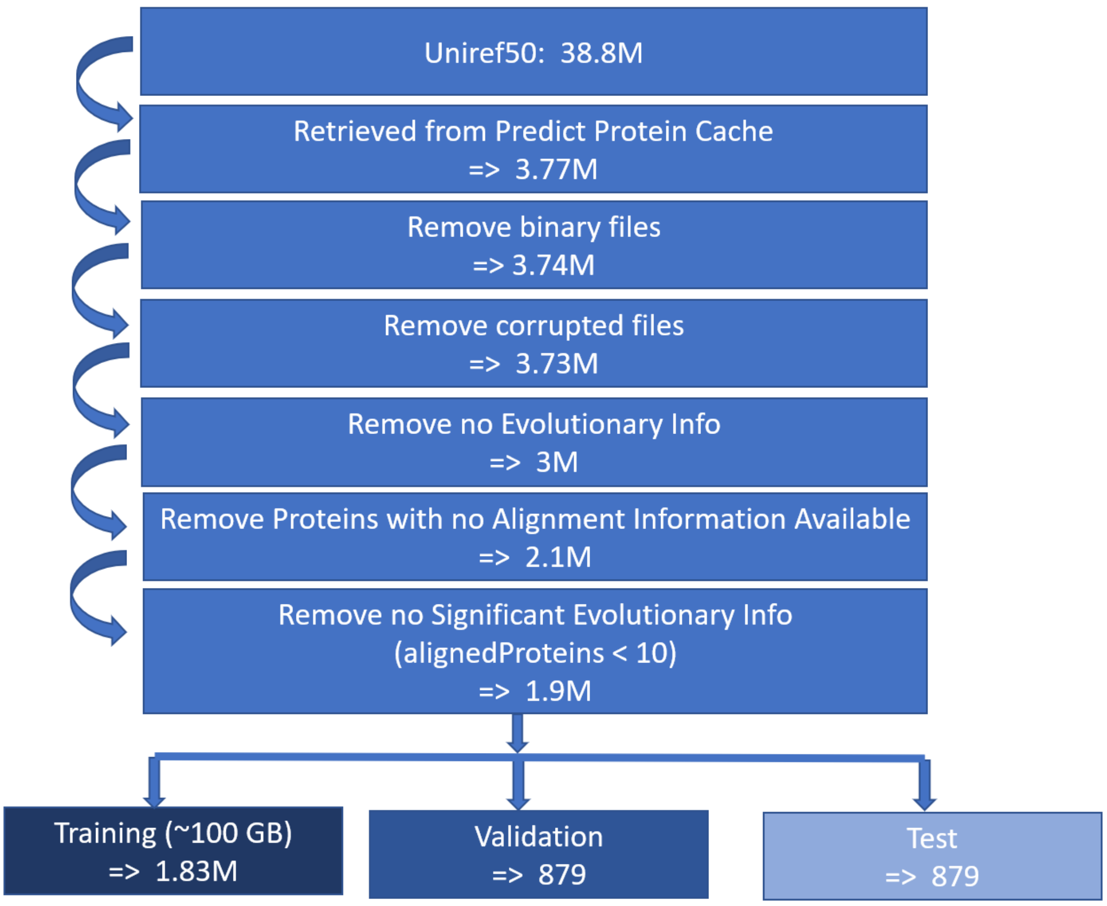

Uniref50: 38.8M
Retrieved from Predict Protein Cache => 3.77M
Remove binary files => 3.74M
Remove corrupted files => 3.73M
Remove no Evolutionary Info => 3M
Remove Proteins with no Alignment Information Available => 2.1M
Remove no Significant Evolutionary Info (alignedProteins < 10) => 1.9M
Training (~100 GB) => 1.83M
Validation => 879
Test => 879

Figure 13. Simplified visualization of the pre-processing pipeline of the whole Uniref50 dataset to obtain the final training validation and test sets

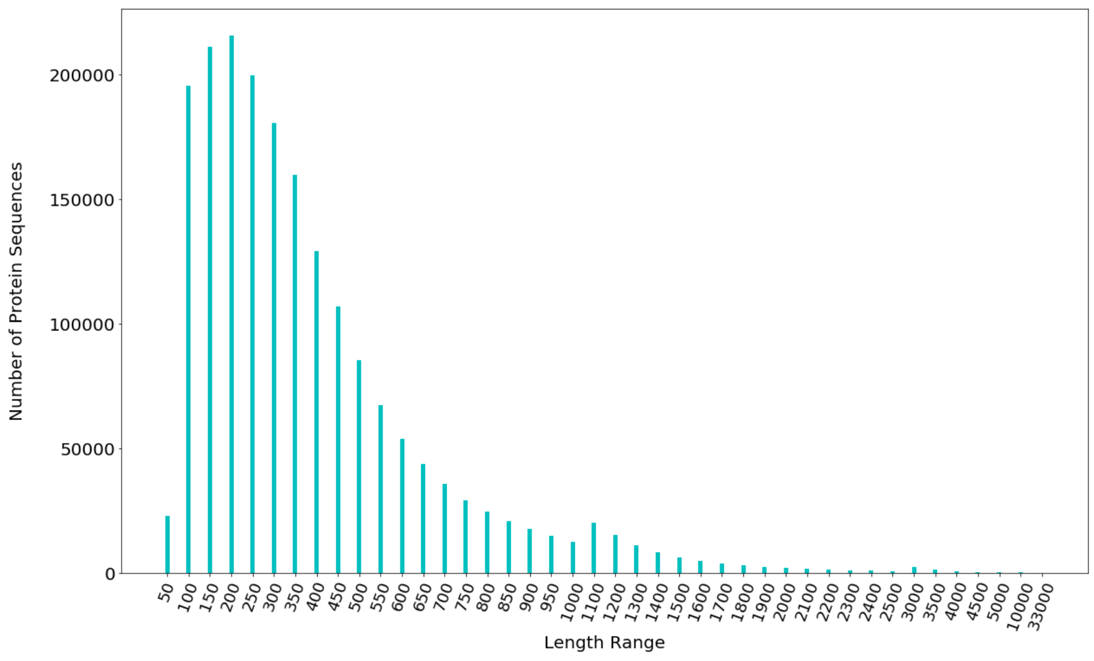


Figure 14. Proteins sequence length distribution in the final training set (~1.83 M protein sequences)

Since we are applying ML-based NLP algorithms to solve our task, it is crucial to know the size of sequences (sentences in NLP terminology) we are dealing with. This knowledge is of great importance as it will guide us in the model architecture design and the hyperparameters selection, especially when it comes to the size of mini-batch training enforced by the GPU memory constraint. For that purpose, we visualized the distribution of proteins in our final training set by their sequence length, as seen in Figure 14. The graph displays the nature of length distribution we are dealing with, and it characterizes a left-skewed distribution with a mean around 300 residues. Another important information is the distribution of vocabulary of our training set. Figure 15 represents the amino acids composition of our dataset, which is a rather left-skewed distribution than a uniform one. These statistics will be used to compute the cross-entropy (CE) random baseline of predicting the next amino acid in the sequence.

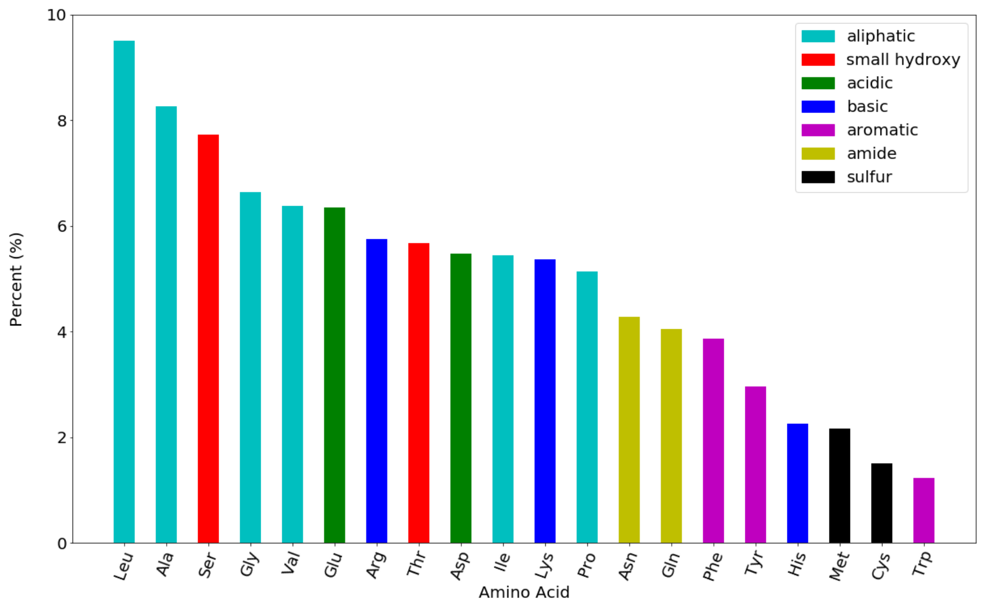


Figure 15. Distribution of the amino acid composition in the final training set (~0.8 billion amino acids)

4.3- Input features definition

The ultimate goal of our project is to go directly from the amino acid sequence to a numerical representation embedding the evolutionary information of the protein. For that purpose, we want to train the model to predict the relative frequencies matrix given the protein sequence. The only input we want to feed our model is the protein sequence. However, in order to reduce the training time for a faster convergence, we opted for transfer learning. Leveraging SeqVec model to generate contextualized and uncontextualized embeddings at the residue level to serve as input to our novel ELMo architecture will definitely boost the performance and lead to faster training. Figure 16 illustrates the processing performed by ELMo/SeqVec. It takes a protein sequence as an input (blue tensor) and generates a per-residue one-hot encoding (purple tensor) of size 20 for each amino acid. The one-hot encoding is fed to a Char-CNN to generate uncontextualized representations (orange tensors), which are then passed to a two-layer bidirectional LSTMs for context learning of a pivot residue (2 stacked yellow tensors for both forward and backward passes). The final embeddings are represented as a concatenation of the 3 layers' outputs, resulting in a tensor of size 3xLx1024 (red tensor). Those embeddings will serve as an input to our model.

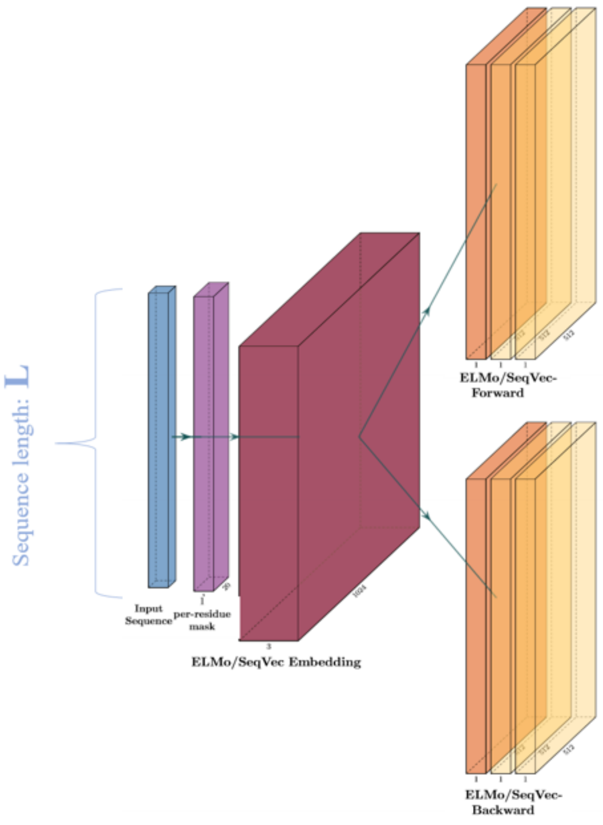


Figure 16. Simplified visualization of ELMo/SeqVec embeddings

An important observation to make here concerns the structure of the embeddings. The tensor of size 3xLx1024 is just a concatenation at the 3rd dimension of 2 tensors of size 3xLx512, where one represents the embeddings of the forward pass and the second the embeddings of the backward pass. Feeding the whole tensor as input to our model will lead to a disastrous data leakage, and hence biasing the training loss. To support this claim, we built a simple model of one linear layer plus a SoftMax layer to predict the next amino acid in the sequence from ELMo/SeqVec last layer embedding. We conducted 3 experiments, each one with a different input vector. The first experiment takes the whole vector *"[:1024]"* of the last layer as an input. The second experiment takes the first half *"[:512]"* of this vector as an input, corresponding to the forward pass embedding. The third experiment takes the second half *"[-512:]"* of this vector as an input, corresponding to the backward pass embedding. Figure B.1, in appendix B, shows the results of our experiments. Unsurprisingly, the backward embedding used as input in two of the experiments provides much better performance (4 curves at the bottom) in predicting the next amino acid as it, somehow, allows the model to cheat and learn from an input representation that has already seen the token we are willing to predict.

Based on the above, the final input to our novel architecture was chosen to be half of the tensor. Since we are building a bi-directional language model, the sequence is processed twice, once from a forward pass using the first half of the tensor (3xLx512) as an input and once from a backward traversal, by flipping the amino acid sequence, and using the second half of the tensor (3xLx512) as an input. This procedure serves as an implicit data augmentation technique, as it increases the number of training samples by a factor of 2, contributing in the model's regularization to build a robust inducer.

4.4- Loss function definition

The last step before proceeding with building our ELMo architecture is to define the loss function that will guide the training of the network's adaptive basis functions during backpropagation. In chapter 3, we formulated mathematically the problem as a posterior probability distribution that we want to learn. The posterior is simply a conditional probability $\boldsymbol{p(z|x)}$ defined as follows:

$$\boldsymbol{p(z|x)} = \frac{\boldsymbol{p(z,x)}}{\boldsymbol{p(x)}} \qquad \textit{Eq. (2)}$$

Here, $\boldsymbol{p(z,x)}$ is the joint probability of the observed variable $\underline{x}$, denoting all previously seen residues information, and the hidden variable $\underline{z}$, which is an overloaded set of parameters and latent variables of the evolutionary information distribution of the next residue. $\boldsymbol{p(x)}$ is the model evidence. Since the evolutionary information embedded within the relative frequencies matrix at the residue level is a discrete probability distribution and the evidence can be written as the marginalization of the joint distribution over $\underline{z}$, our probabilistic model can be reformulated as follows:

$$\boldsymbol{p(z|x)} = \frac{\boldsymbol{p(z,x)}}{\sum_{z} \boldsymbol{p(z,x)}} \qquad \textit{Eq. (3)}$$

For most interesting models, such as ours, the evidence is not tractable. The marginalization over $\underline{z}$, i.e. the sum, makes it hard to maximize. The nature of the distribution can be anything, even a multimodal distribution with multiple maxima. This makes it even harder, even impossible, for the integration to be solved analytically, in the case of a continuous PDF. For that reason, we appeal to approximating our posterior inference. There exist different ways to solve this problem:

a) Metropolis-Hastings [60] is an approximation technique that is more accurate compared to the other two methods and can provide an exact solution given enough iterations. It is a type of method that is based on sampling and it is easy to understand. However, it takes quite a long time to compute the final exact solution.
b) Laplace approximation [61] is the second approach that can be used to solve such problems. It is defined as the simplest deterministic solution, easy to understand and takes less time to compute compared to the other two methods. However, it is restricted to continuous distributions, rather less accurate and gives a poor approximation to the problem.
c) Variational inference (VI) [62] is the third methodology that can be adopted to solve our problem. It is also characterised as a deterministic solution, takes less time to compute compared to Metropolis-Hastings, but is hard to understand. Both Laplace and variational inference provide less accurate approximations compared to Metropolis-Hastings;

however, variational inference provides a much better approximation than its counterpart, Laplace.

For performance objectives and better approximation, we opted for VI to solve our machine learning problem and to approximate our inference. The main idea is to: (1) find a tractable distribution ***q(z)*** that is similar to ***p(z|x)***, and then (2) use ***q(z)*** to answer the questions about ***p(z|x)*** that we care about. Figure 17 shows visually the general approach of variational inference where we have two distributions, $\boldsymbol{q_1(z)}$ and $\boldsymbol{q_2(z)}$, each with known parameters, and an unknown distribution, ***p(z|x)***. In this scenario, $\boldsymbol{q_1}$ is closer to $\boldsymbol{p}$ than $\boldsymbol{q_2}$. We can then use the distribution $\boldsymbol{q_1}$ to answer our questions.

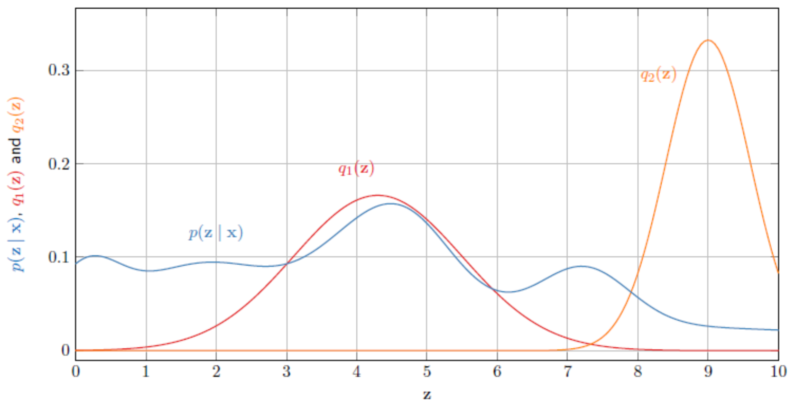


Figure 17. Graphical visualization of variational inference general idea. The graph shows two distributions (orange and red) with known parameters, and a third distribution (blue) with unknown parameters. From the plot, we see that we can use $\boldsymbol{q_1}$ (red) as a distribution to answer related questions of ***p(z|x)*** (blue). This figure was reprinted from [63]

To apply variational inference, two main prerequisites must be satisfied:

1. **A search space must have been defined:** We have to choose a set of tractable candidate distributions Q. In the context of variational inference, we call Q the variational family. This variational family is simply a set of distributions that share some property.
2. **The divergence D must have been chosen:** We have to define a divergence *D(p||q)*, which measures how dissimilar distributions *p* and *q* are.

Once defined, the problem can be then formalized as finding the optimal distribution in the variational family Q which minimizes the divergence *D(p||q)*. In mathematical notations, this is written as:

$$q^* = \underset{q \in Q}{argmin}\, \mathrm{D}(p(z|x)\ ||\ q(z)) \qquad \textit{Eq. (5)}$$

The divergence *D(p||q)* is defined as $D: P \times P \rightarrow \mathbb{R}^+$, and it simply measures the dissimilarity between two probability distributions *p, q* $\in \mathscr{P}$, Where $\mathscr{P}$ represents the set of probability distribution over a given space $\boldsymbol{\Phi}$, such as $\mathbb{R}^d$. In general, the divergence function has to satisfy two conditions:

- It should be non-negative, meaning that $D(p||q) \geq 0$ for all $p, q \in \mathscr{P}$
- And if the $D(p||q) = 0$, we say that $p = q$ almost everywhere

One function of choice meeting the above conditions is the Kullback-Leibler divergence ($\mathbb{KL}$-Div). It is defined as:

$$\mathbb{KL}(p\,||q) = \int p(\boldsymbol{z})\, log\left(\frac{p(\boldsymbol{z})}{q(\boldsymbol{z})}\right) d\boldsymbol{z}$$
$$= \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}[log(p(\boldsymbol{z})) - log\,(q(\boldsymbol{z}))] \qquad \textit{Eq. (6)}$$

Replacing the divergence function in Eq. 5 with the $\mathbb{KL}$ divergence from Eq. 6, defining the loss function translates to finding the **best** distribution q in Q that minimizes the $\mathbb{KL}$ divergence:

$$q^* = \underset{q\in Q}{arg\min}\, \mathbb{KL}(p(z|x)\,||q(z))$$
$$= \underset{q\in Q}{arg\min}\, \mathbb{E}_{p(z|x)}[\log(p(z|x)) - \log\,(q(z))]$$
$$= \underset{q\in Q}{arg\min}[\sum p(z|x)\log(p(z|x)) \;- \sum p(z|x) \log\,(q(z))]$$

Eq. (7)

If we choose Q to be some parametric family Q = *{q(z| v) for all valid parameters v}*, such as Binomial, Normal, Poisson, or Gamma families, we simply have to find the optimal parameters v^* that minimize the $\mathbb{KL}$-divergence (Figure 18 [64]). Following the universal approximation theorem [65], we opted for a deep neural network as a parametric complex function that can learn any distribution given enough neurons in a two-layer network. Therefore, we converted our inference to an optimization problem. The final loss function to minimize is:

$$v^* = \underset{v}{arg\min}[\sum p(z|x)\log(p(z|x)) \;- \sum p(z|x) \log\,(q(z|v))] \qquad \textit{Eq. (8)}$$

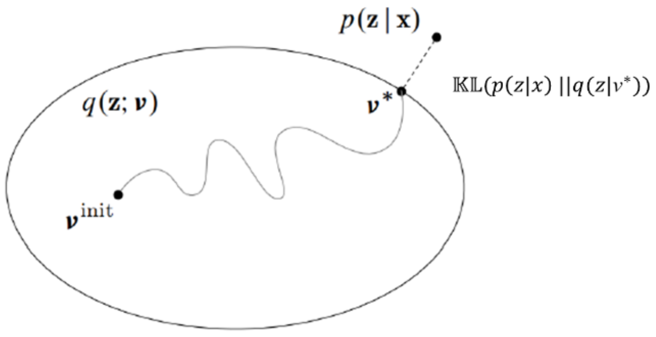

$p(\mathbf{z} \mid \mathbf{x})$
$\mathbb{KL}(p(z|x) \,||q(z|v^*))$
$q(\mathbf{z}; \boldsymbol{\nu})$
$\boldsymbol{\nu}^*$
$\boldsymbol{\nu}^{\text{init}}$

Figure 18. Graphical representation of how variational inference is turned to optimization, given a parametric variational family

After gathering a high-quality training set, analysing it, defining the input, the labels, and the loss function of our main problem we want to solve, we ran three different experiments using different models. The design and the development of the 2nd and 3rd model architectures were motivated by the results obtained in previous experiments and related technologies published in the literature. The next chapter will describe the models and discuss the experiments results in detail followed by a benchmark with respect to downstream tasks.

5- Experiments and results

5.1- Experiment 1 – Initial language model architecture

In this section, we experimented the learning performance of a simple 2-layer plain LSTM embedding language model architecture, with $\mathbb{KL}$-Divergence as a loss function, on a tiny dataset. The purpose of the experiment is to validate that the loss function is able to guide the network to learn protein evolutionary information.

5.1.1. Model description

To build our new embedding language model, we opted for the LSTM technology, the same machinery used in the bi-directional autoregressive approach presented by Peters et al. [22]. Additionally, since the input dimensions are already large enough, and we believe that it carries significant biophysical and biochemical information, we wanted to keep the hidden and the output size of the LSTM layers relatively small. Therefore, we suggested a simple 2-layer LSTM architecture predicting the next PSSM column in a protein sequence, given all the previous residues information (Figure 19). In this initial model design iteration, the network takes a flattened representation, i.e. a concatenation at the 1st dimension, of the SeqVec/ELMo embedding, making the input size=1536 units, for the first LSTM, and the hidden/output size of both LSTMs=256 units. We want to point out here that the same path in the network processes the sequence from both the forward and the backward pass, which corresponds to a bidirectional language model.

As an initial proof of concept that a neural network architecture can learn a function mapping the amino acid sequence to the evolutionary information representations, we randomly chose a set of 40 proteins, ranging between 123 and 389 residues, with corresponding PSSM files for this experiment. The development set was split into a training set, of size 24 proteins, and a validation set, of size 16 proteins. The goal here was to prove that the model can overfit to the training set.

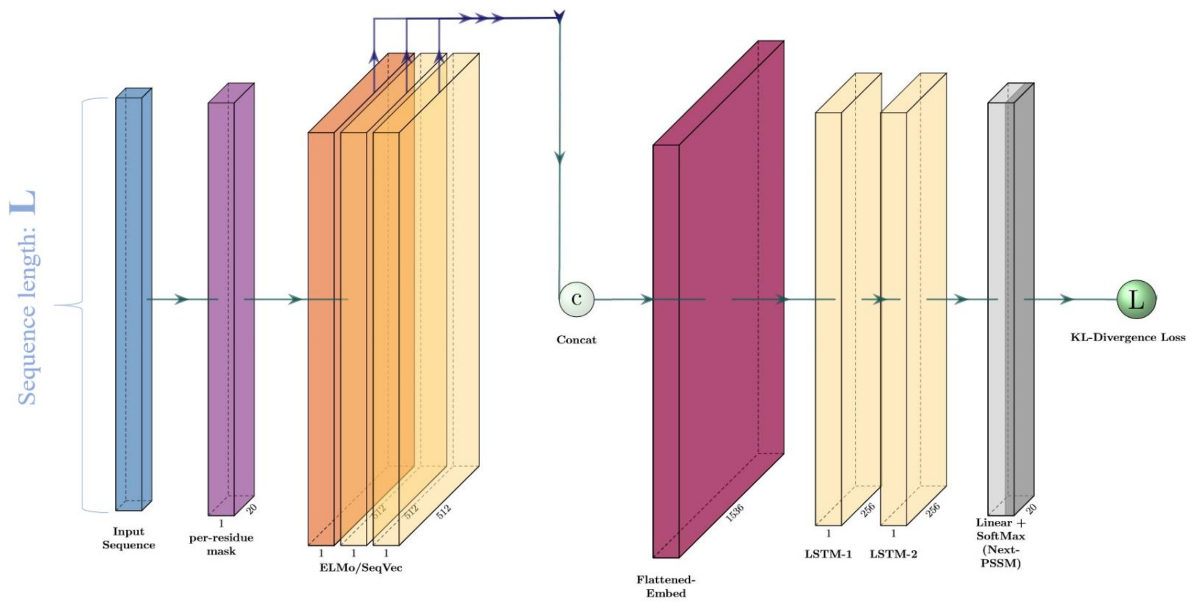


Figure 19. Conceptual visualization of the evolutionary information language model initial architecture consisting of a 2-layer bidirectional LSTM. The network is trained using a $\mathbb{KL}$ -Divergence loss to predict the next PSSM column

5.1.2. Results and discussion

The training results reported in this section were conducted on a personal workstation. The machine had a system memory size of 16GB, an Intel® Core™ i7-8750H 8th generation CPU with 12 logical processors and a maximum speed of 3.5 GHz. The machine contained also an NVIDIA GeForce GTX 1050 Ti Graphical Processing Unit with a dedicated memory of 4GB. The GPU was used for deep learning hyperparameter tuning. The model was trained till convergence with a mini batch size of 8 sequences processed per iteration.

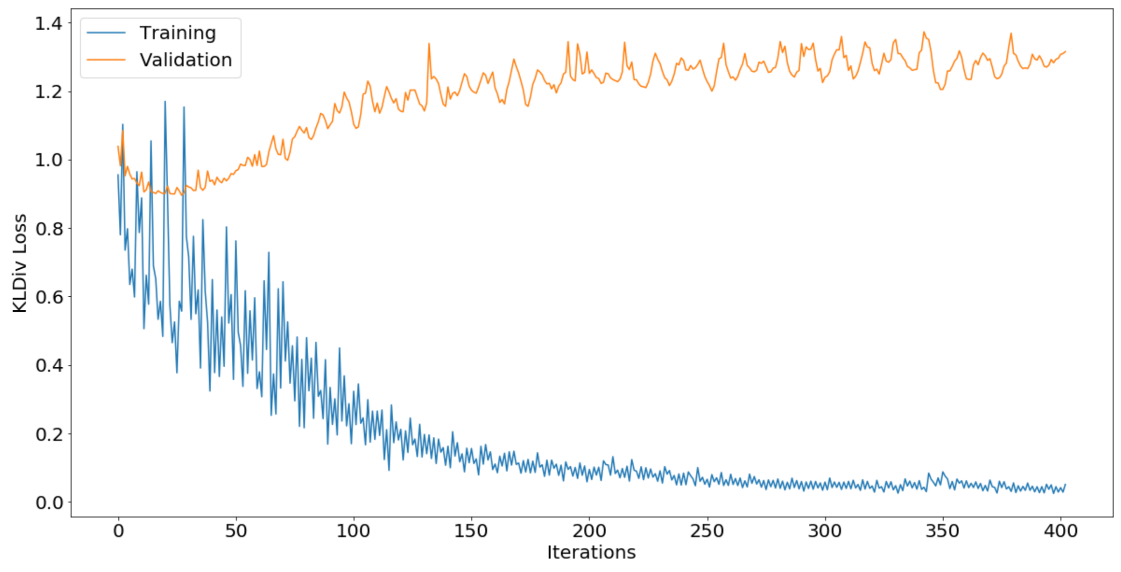


Figure 20. Training vs. validation losses of the initial LM

Figure 20 visuals the training and the validation curves after half an hour of training. Both training and validation 𝕂𝕃-Divergence losses start around 1.1. While the training loss keeps decreasing with the increasing number of epochs, the generalization gap[3] starts getting larger after 13 epochs (~40 iterations). This obviously is a sign of overfitting. The training loss almost approached a value of zero, showing that the network architecture perfectly learned the protein evolutionary information mapping function of the training set.

In order to analyse the prediction performance of the trained model, we randomly picked a protein sequence form the training data and generated the predicted relative frequencies directly from the amino acid sequence. For better examination, we visualized the output distribution and compared it to the ground truth values. Figure 21 compares the discrete probability distributions of both the ground truth and the predicted relative frequencies from a forward pass of a sequence.

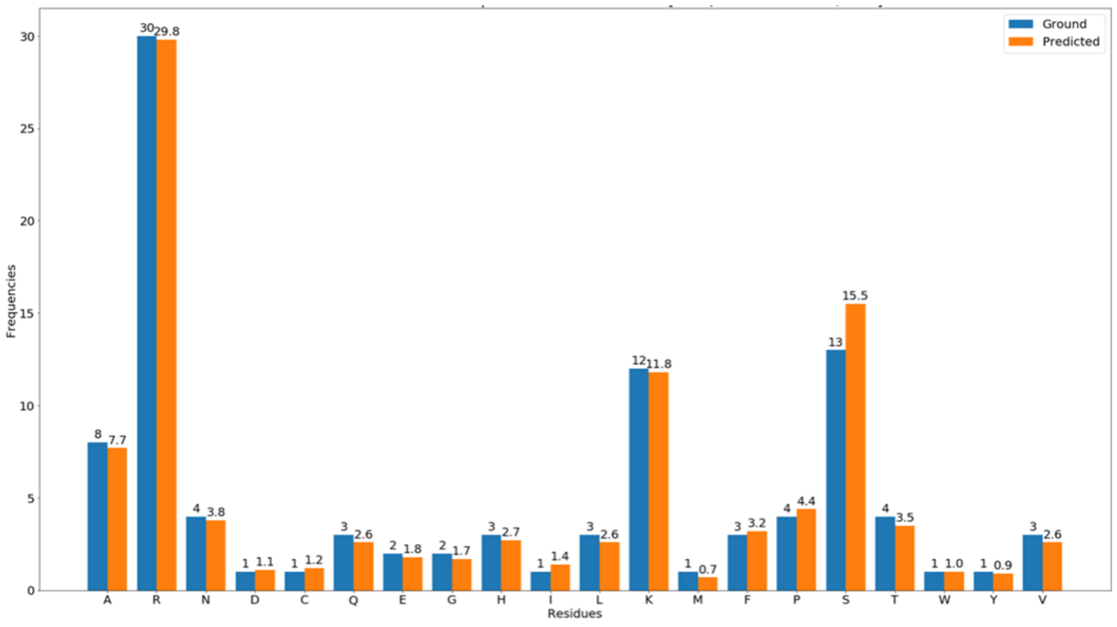


Figure 21. Discrete probability distributions comparison between the ground truth relative frequencies and the predicted ones of an amino acid at position 262 in the protein with PID: Q9H255. The relative frequencies in blue correspond to the results of an MSA-based method aligning the protein against a database, whereas the relative frequencies in orange correspond to the output of the model predicting the distribution of residue 262 from all previously seen residues in a forward pass.

Even though we were able to predict the same distribution, and the model was able to learn the representations, there are 3 major observations to mention:

[3] Generalization gap is defined as the performance difference of a model between the one reported on its training data and the one reported on the validation/unseen data, which is drawn from the same distribution.

- The training set used in the experiment is very tiny
- Though the set was extremely small, the model was struggling at the beginning of the training. It took more than 14 epochs for the optimizer to start pointing towards an optimum
- It took too much time and more than 50 epochs for the model to perfectly overfit to the tiny training data. We needed about half an hour of training to come up with the histogram in Figure 21.

Those observations clearly show the complexity of the task on hand. To draw more realistic conclusions, we needed to scale up and train on a bigger dataset with a larger mini-batch size. For that purpose, we obviously need more computing resources and even tuning the architecture to help the model quickly converge.

5.2- Experiment 2 – Multi-task language model architecture

In this section we investigated the effect of scaling up on the learning process of our ELMo embedder. For that purpose, we selected a random subset, from our large training set (Figure 13), of 200K protein sequences that will be used to train the model, and 879 sequences for validation (Figure 13).

5.2.1. Model description

In order not to forget what was previously learned in SeqVec/ELMo embeddings and with the goal to help the convergence be faster, we added a second path to the head of the network architecture, predicting the next amino acid in the sequence as well. Those two paths are represented in figure 22 as two parallel grey tensors, in the right side of the figure and before the losses. This led our project to shift towards multi-task training.

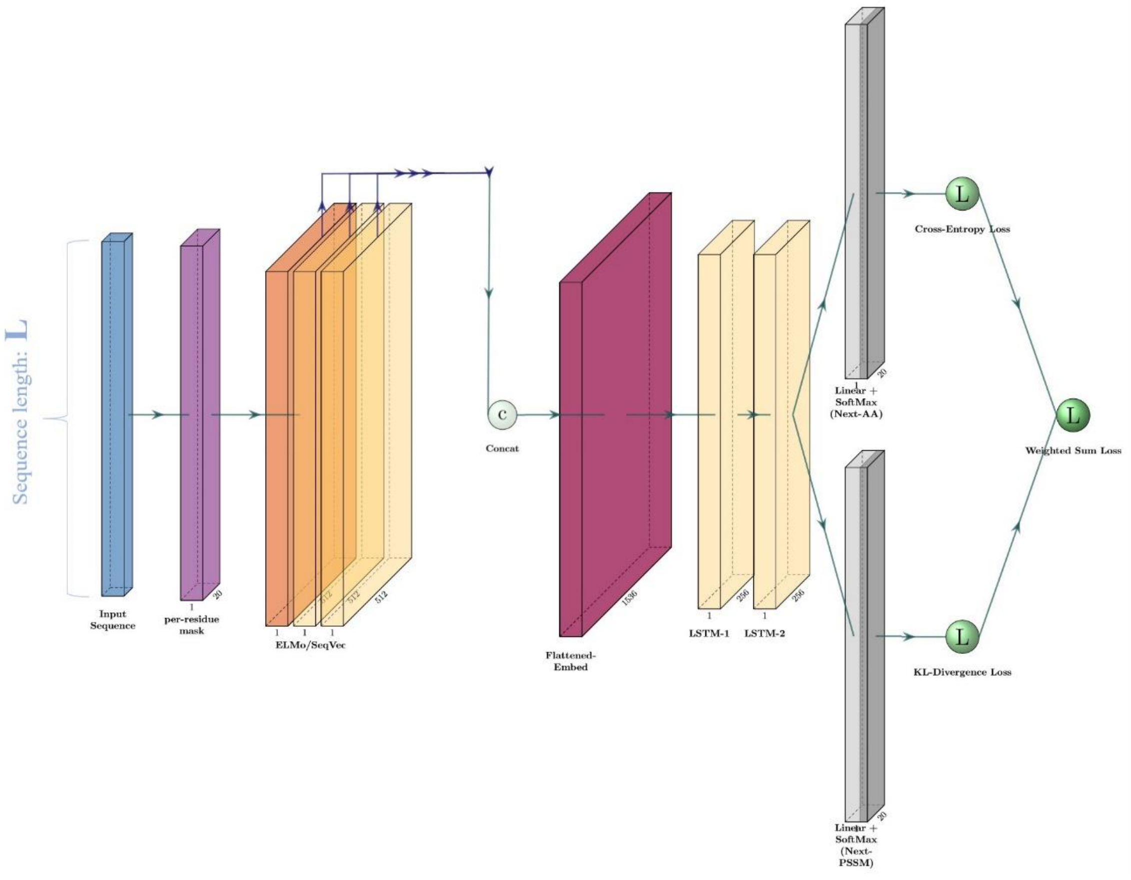


Figure 22. Conceptual visualization of the multi-task network architecture, predicting the next amino acid and the next PSSM column simultaneously in a protein sequence via transfer learning. The model combines both the 𝕂𝕃-Divergence and the cross-entropy losses in a single weighted sum loss.

Using the network architecture visualized in figure 22, we trained the model on the 200K sequences. Since we are training a bi-language model, this translates to an equivalent dataset of 400K sequences counting both the forward and backward traversals of each sequence.

Concerning the loss functions used in the architecture, we have both a cross-entropy loss assessing the predictions of the next amino acid and a $\mathbb{KL}$-Divergence loss measuring the closeness of the predicted discrete probability distribution to the ground truth relative frequencies of the next PSSM column. This corresponds to an optimization problem with respect to multiple objective functions. The commonly adopted approach to solve such multi-task problems computes a weighted linear sum [66] combining all the objective losses:

$$\begin{cases} L_{total} = \sum_i \alpha_i L_i \\ Where \sum_i \alpha_i = 1 \end{cases} \qquad \textit{Eq. (9)}$$

Here, L_i is the objective loss of the task *i* and α_i is the corresponding weight given to that loss. In our experiment, we assigned similar weights (a value of $^1/_2$) to each loss in the final combined objective.

Additionally, the sequences lengths ranged between 21 and 8914 residues. By randomly selecting sequences for batch training, we will run out of memory while trying to allocate space for the variant sizes of proteins. Like most deep neural networks, LSTMs are trained using gradient descent. Such gradients are efficiently computed using a technique called backpropagation through time (BPTT), which applies back propagation to the unrolled network [67]. For long sequential records, such as our case, BPTT is both computational and memory intensive. This is mainly a GPU memory constraint that forces us to look for alternatives for a better control over a batch size that can entirely fit in the GPU dedicated RAM. A dominant approach applied in the field of sequential data processing is to cut the sequence into chunks and process them separately. This is what the approximation technique known as truncated BPTT (TBPTT) [68, 69] does. There are several advantages of using this method:

- It is a lot faster, as we do less work and computations.
- It throws away pieces that have many products of matrices that would eventually either vanish or diverge.
- It has a regularization effect, as it forces the model to focus on the changes that happened recently than changes that would have been observed a very long time in the past.

Therefore, TBPTT trains a model that is more robust and does not suffer much from the butterfly effect, which is also susceptible to overfit. With this technique we are somehow constraining the model space. This adds an additional hyperparameter to our model, which is the step size of the truncation. With insights from figure 14, we set the fixed step size to the mean of the sequence length distribution in our training data, which is ~300 residues.

5.2.2. Results and discussion

With a training set of 200K sequences, if we did train the model using the same workstation as in the previous experiment, it would have taken months and might have ended up damaging the machine. To train this enhanced architecture, we were granted exclusive usage of a remote Linux machine with a system memory size (RAM) of 64GB, Intel® Xeon® 5600 series processors of six-cores with 12MB of shared cache, and a maximum speed of 1.3 GHz. The machine also contained an NVIDIA GeForce GTX TITAN X GPU with a dedicated memory of 12GB. This is 3-fold larger than the previous GPU's dedicated RAM. With a model of 2.8 Million trainable parameters, the training took one week of computation to go through the whole dataset once, i.e. 1 week for one epoch.

As a first step in analyzing the quality of learned representations of the model, we saved the training and validation loss values to track the improvement of the learning over time. With the above fixed maximum sequence length to be processed at a time and given the current hardware, we were able to handle a mini batch of 16 protein sequences in each iteration. To avoid confusion, the sequence processing is detailed as follows:

a. We select a random subset of 16 proteins, called the mini-batch size in this manuscript.
b. For each sequence in the mini-batch, we create a copy of the sequence in the reverse order.
c. We append the new sequences to the mini-batch, resulting in an augmented mini-batch of 32 sequences.
d. SeqVec/ELMo embeddings are generated for the 32 sequences.
e. Short sequences are zero-padded to match the length of the longest one in the batch.
f. All sequences are packed in one big tensor.
g. The tensor is chunked equally to small tensors with a maximum length of 300.
h. Chunks are passed sequentially to the model to be processed.
i. The LSTMs cell and the hidden states are saved at the end of each processed chunk to be passed on to the next chunk in the queue.

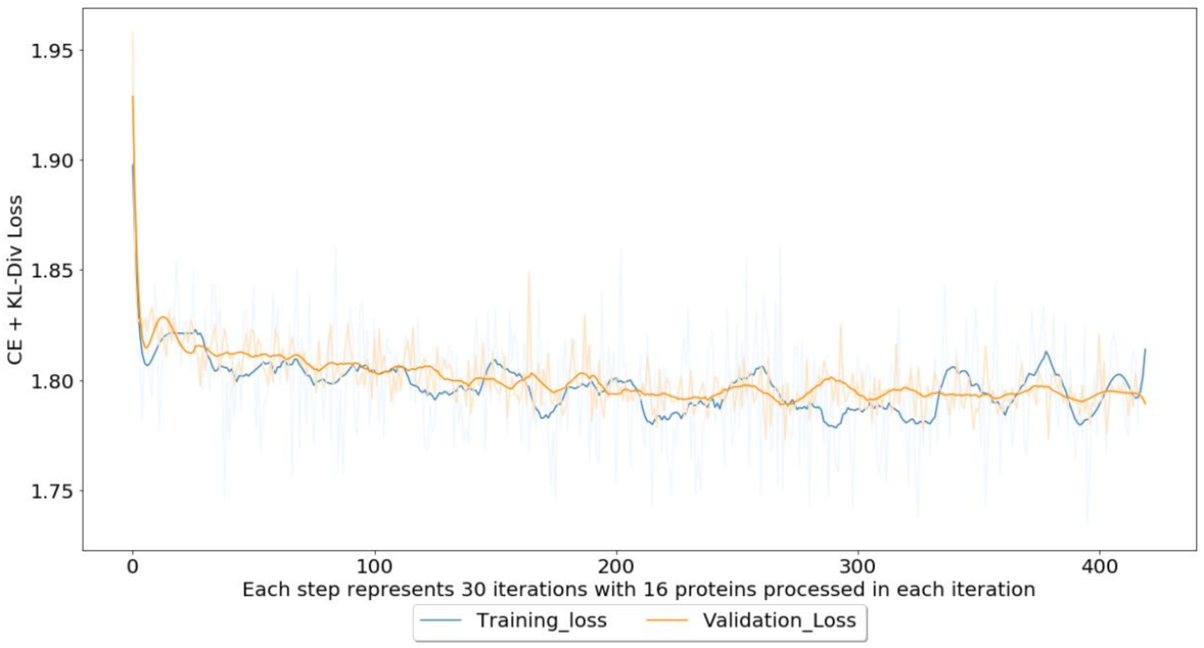


Figure 23. Training and validation curves of the multi-task model with combined objective optimization. The curves correspond to the training data history saved after every 30 iterations

Figure 23 shows how the weighted linear sum of the combined objective loss is improving with increasing number of iterations. To avoid outliers from destroying the visualization and distorting the analysis, the graph displays a smoothing of the original training and validation curves in darker colours. A well-known approach used to tackle such scenarios is the Gaussian process regressor [70]. In our case, we used a second and easy alternative named the Savitzky-Golay smoother [71]. The method applies a filter, using a small window on the data stream and moving one point at a time, to regress the original datapoints to a polynomial using least squares. The polynomial is then used to compute an approximation of the middle point, or the average, in the frame, which will later be used as a point in the smoothed curve.

The results of the above experiment were somehow disappointing as we were expecting the learning curve to decrease more with the increasing number of iterations, especially that we used a larger training set. The loss function started around 1.9, then it quickly dropped in the first iterations to 1.82, to keep oscillating around 1.79 for the rest of the training.

Since our final objective function is combining both a CE and a $\mathbb{KL}$-Divergence loss, decoupling both losses and analysing them separately may give better insights to assess the training quality of the model and tuning our final combined loss function. Regarding the cross-entropy loss that is gauging the predictions of the next amino acid, a better assessment would be to compare the prediction performance with the random baseline and SeqVec reported performance. In the rest of this subsection, we will derive the values of those two baselines.

In general, language models' primary purpose is to learn from the sample corpus a distribution, named Q, that is as close as possible to the actual distribution, named P, of the corpus language.

One widely used measure to compute how close two distributions are to each other is the cross-entropy. In mathematical notations, the CE between the distribution Q and P is defined as follows:

$$H(P,Q) = \mathbb{E}_P[-log\ (Q)] \qquad \textit{Eq. (10)}$$

For discrete probability distributions, the cross entropy is defined as follows:

$$H(P,Q) = -\sum_{C=1}^{M} P(x=c)\ log\ (Q(x=c)) \qquad \textit{Eq. (11)}$$

Where *log* is usually the logarithm base two but in most frameworks, it is replaced by the natural logarithm for fast computation, *c* is a class label from the total number of classes M, P is usually a binary indicator assigning a value/probability of 1 to the correct class and 0 to the wrong one, and Q is the predicted probability of class c. For a uniform probability distribution of the class labels in a dataset, the random CE value is computed as $-\sum_{C=1}^{M} log({}^{1}/_{M})$. In our case, the distribution of amino acids is relatively left skewed, as shown in figure 15. Using the probability frequencies of each of the 20 amino acids in our 0.8 billion residues corpus, the calculated random CE baseline for our task is $H(P,Q) = 3.12$.

Concerning the performance of SeqVec/ELMo embedder, the authors reported the value in terms of perplexity. Perplexity (PPL) is usually known as a measure of uncertainty. From a language model perspective, it is seen as the level of perplexity/uncertainty when predicting the next token. The measure is used more within the natural language processing community, though, it can easily be converted to CE. In mathematical notation, it is defined as follows:

$$PPL(W) = P(w_1 w_2 \dots w_N)^{-\frac{1}{N}} = 2^{cross-entropy} = 2^{H(P,Q)} \qquad \textit{Eq. (12)}$$

Using the inverse function:

$$H(P,Q) = Log(PPL(W)) \qquad \textit{Eq. (13)}$$

Where *W* is the bag of the corpus vocabulary (in our case, it is the set of 20 amino acids), and the *log* is the logarithm base two. For computation performance as explained previously, the log here is the natural logarithm. $PPL(W) = 10.5$ is the reported perplexity by Heinzinger et al. [36]. Using equation 13, the calculated SeqVec/ELMo CE loss baseline is $H(P,Q) = Log(10.5) = 2.35$.

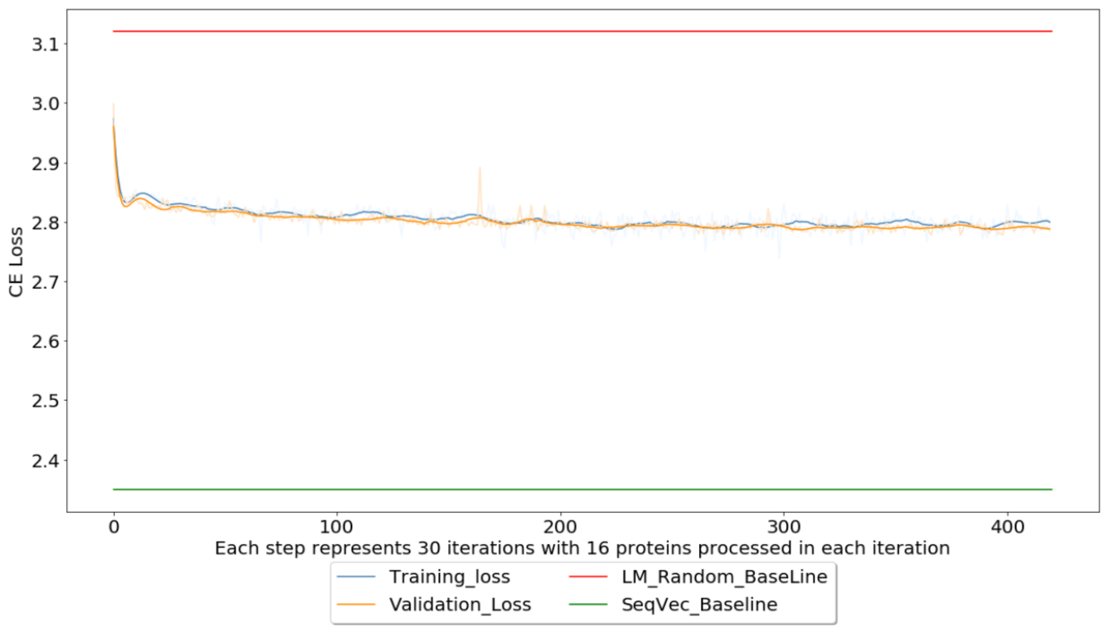


Figure 24. Smoothed training and validation curves of the cross-entropy loss predicting the next residue in the sequence. The graph shows where our learned representations are standing compared to the random language model baseline (Red line) and SeqVec (Green line) performance.

Plotting the two baselines, the training, and validation curves of our ELMo will give us a better assessment of the network architecture and the hyperparameters used during training to predict a distribution of the next amino acid in a sequence. From figure 24 and with a value of ~2.8, we notice that our model is doing better than the random baseline. However, it is performing poorly compared SeqVec/ELMo learned representations. Concerning the $\mathbb{KL}$-divergence loss predicting the next PSSM column in the sequence, Figure 25 shows that the model is not learning much from our dataset. The loss keeps oscillating around 0.8 with no improvement noticed over time.

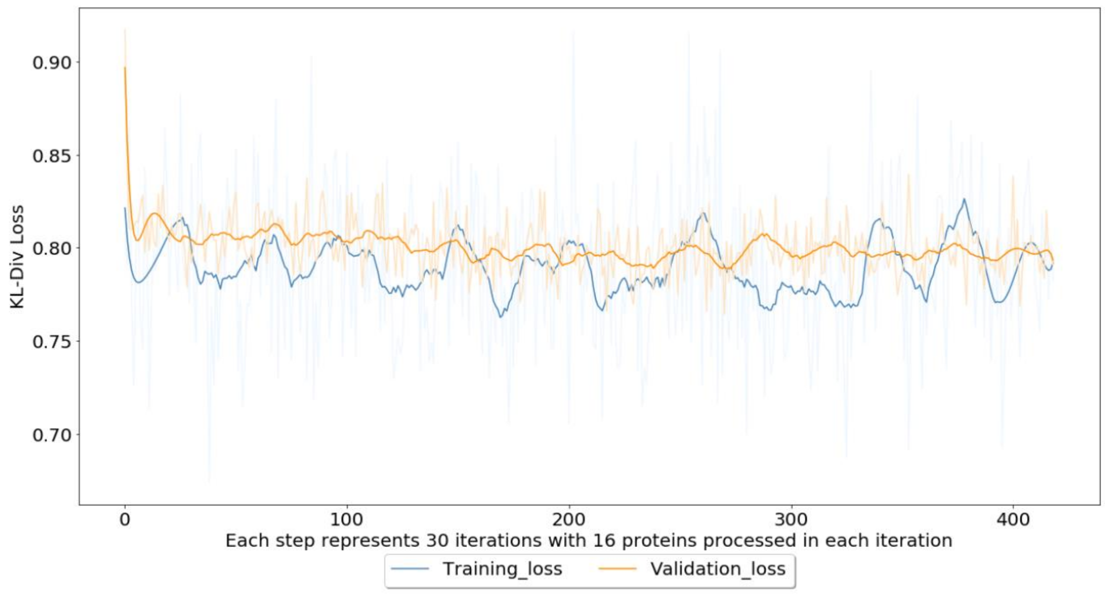


Figure 25. Smoothed training and validation curves of the $\mathbb{KL}$-Divergence loss predicting the next PSSM column in the sequence.

From the above results, we plainly see that the model performance is poor. Nevertheless, we should not neglect that we are putting too much regularization on our training both architecture-wise, learning forward and backward representations of a sequence using the same architecture, and sequence lengthwise, by restricting the maximum sequence size to be processed to a maximum of 300 residues. Additionally, the visuals give us a sign that the current network architecture and its capacity are not enough to learn complex representations. Moreover, the batch size used for training might also contribute in this observed performance, as it is a training hyperparameter and it has been shown in many research projects [72-75] that batch size plays a role in providing good/bad expectation estimations, hence affecting the training performance. Table 2 displays all the losses reported in this section and offers an easy comparison of the statistics. We notice from the table that the CE loss is almost 4 folds larger than the $\mathbb{KL}$-Divergence loss. Hence, making the CE as the dominant loss in our final objective function.

Table 2. The table compares the performance of our suggested ELMo training a 2 bidirectional LSTM (2 bi-LSTM) architecture via transfer learning (TL). The table shows the separate values of the CE and $\mathbb{KL}$-Divergence losses as well as the joint final loss. Also, the table displays the values of the random and SeqVec reported baselines.

	Cross-Entropy loss (AA)	**$\mathbb{KL}$-Divergence Loss (PSSM)**	**Joint loss (AA + PSSM)**
Random baseline	3.12	-	-
SeqVec reported baseline	**2.35**	-	-
TL+2 bi-LSTM	2.8	**0.8**	**1.8**

5.3- Experiment 3 – Final language model architecture

In the experiment described in this section, we are increasing the capacity of our plain LSTM layers by adding recurrent projection layers to each module, adding a skip connection to the architecture, reducing the truncation step size to 100 residues, and training on the whole set of 1.83 Million proteins (~0.8 billion amino acids).

5.3.1. Model description

From the training/validation curve analysis, the network capacity appears to be limited as both the training and validation losses in all 3 of the figures (figures 25, 26, and 27) look as if they are struggling to improve with the increasing number of iterations. To boost the capacity of a network, one can either go deeper by increasing the number of layers, go wider by increasing the number of neurons, or both. A joint research work between a faculty member at the University of Toronto and a scientist at Microsoft Research [76] has shown empirically that shallow feed-forward networks can learn the complex functions formerly learned by deep neural networks and obtain performances that were previously only possible with deep models. This conclusion pushed us to drop the idea of going deeper, in terms of layers, but rather look towards the direction of increasing the number of hidden units.

An LSTM architecture contains several components named *memory blocks*. Such blocks are called gates controlling the information flow in a network, including the input, output and forget gates. The LSTM structure is uniquely defined by the number of its input and output units. To raise the capacity of an LSTM, one can just increase the hidden size units, hence increasing the capacity of the cell state to carry more information along to the next time steps for complex tasks. However, those numbers determine the computational complexity for training an LSTM network. For a moderate number of hidden input units, dimensionality, i.e. the complexity of the network, is largely dominated by the size of the output hidden units. Sak et al. [77] from Google labs suggested an alternative approach that addresses the complexity of large capacity LSTMs. They proposed the addition of a linear projection layer applied right after the output and before the recurrent connection to the cell. Therefore, the recurrence is applied from a smaller projected hidden state, which ultimately reduces the number of computations within the whole LSTM cell, while maintaining the large capacity of the cell state. The later described module is referred to as LSTM with recurrent projection layer (LSTMP) [77]. This technology has been adopted by Jozefowicz et al. [78] in Google Brain labs to train different variations of language models on a very large corpus of 0.8 billion words and a vocabulary size of 793471 words [79]. Jozefowicz et al. [78] empirically proved that when trying to fit an LSTM network architecture on very large and complex datasets, the size of the LSTM heavily matters.

For the sake of improving our LM architecture, we customized the CUDA optimized Long-Short Term Memory cell implemented with TorchScript[4] to include a recurrent linear projection layer. For the hidden and projection sizes, we decided to go with half the dimensions used in [22 & 36], i.e. 2048 units and 256-dimension projections.

Inspired by the original ELMo paper by Peters et al. [22], we have also added a residual skip connection from the first to the second LSTM layer, with the goal of boosting the training performance. To further increase the capacity of the network and have more control over the memory size allocated by the LSTM layers for variable input sizes, we also added a fixed size (1024 hidden unit) non-linear input layer, to the LSTM cell, with LeakyReLU as an activation function. Concerning the weights initialization of the LSTM matrices, we initialized the input projection layer with a Kaiming [80] uniform initialization, the biases of the LSTM gates to a value of 1.0, as it was shown to perform well for long size dependencies [81], and the rest of the matrices were initialized with a uniform distribution of a standard deviation $\sigma = \frac{6.0}{hidden\_size + projection\_size}$.

Combining all the LM state-of-the-art technologies discussed in this subsection, we constructed the complex network architecture visualized in figure 26. The model comprises 2 layers of LSTMs, with projections, stacked one after the other. The first layer takes as input SeqVec uncontextualized embeddings concatenated with a one-hot encoding vector of size 20, making it a total input size of 512+20 = 532 units. The second LSTM layer takes as input contextual representations concatenating the output of the first layer with the 2-layer context aware embeddings from SeqVec, resulting in an input size of 512*2+256 = 1280 units. As for the residual block, the two representations, of size 256, from both layers are then summed element-wise to serve as input to the two parallel linear layers, one for predicting the next amino acid and the second for inferring the next PSSM column distribution.

As mentioned earlier, the two losses are computed and summed according to eq. 9 to form the final objective function to be minimized. As our primary goal is to learn evolutionary information representations, we want to assign higher weight to the $\mathbb{KL}$-Divergence loss. In addition, we have seen from table 2 that the CE loss is more dominant in the final objective function, being nearly 4 times larger than the $\mathbb{KL}$-divergence loss. Thus, we decided to bring both losses to the same scale and give more weight to the $\mathbb{KL}$-divergence loss. Tuning the loss coefficients, our final objective function is now defined as:

$$L_{final} = 0.25 * CE_{loss} + 0.75 * KL_{loss}$$

[4] https://pytorch.org/blog/optimizing-cuda-rnn-with-torchscript/

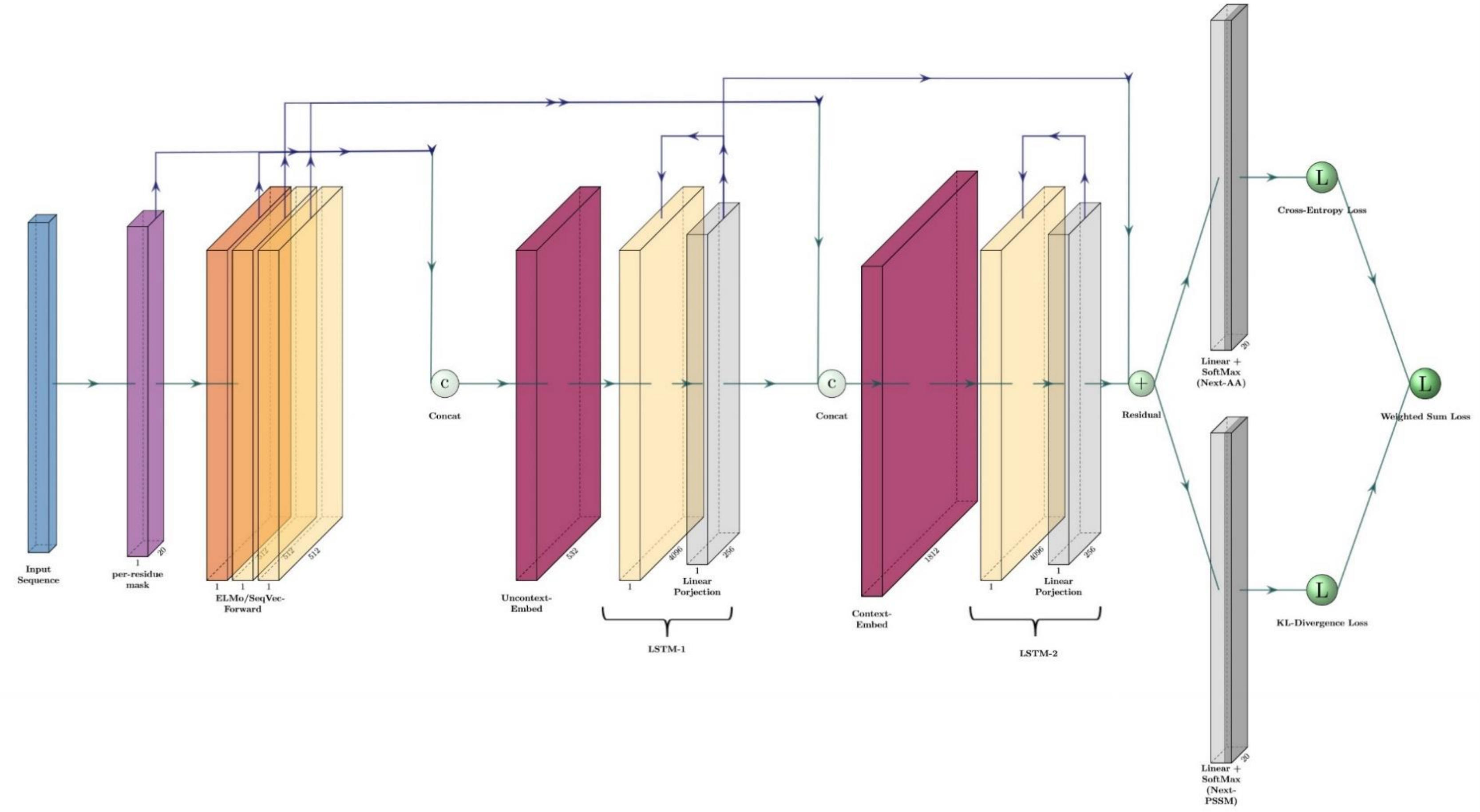


Figure 26. Conceptual visualization of our novel single path 2-layer bidirectional embedding language model architecture. The model is trained on predicting both a distribution of the next amino acid and its corresponding evolutionary information relative frequencies. The architecture makes use of LSTMs with projection and a residual connection

In addition, multiple previous research works have raised the concern that a too large sequence length might disturb the learning process and negatively affect the performance. We suspected that even a maximum sequence length of 300 residues might still be too large and therefore decided to investigate how the maximum time step could be affecting our ELMo performance by training two separate models, one with a maximum sequence length of 100 and the second of 300 residues. The ELMo performance was then evaluated on the secondary structure prediction downstream task.

5.3.2. Results and discussion

After implementing the novel biLM architecture along with its data loader and the model solver[5], we trained it on the whole 1.83 Million proteins. In the second experiment and with a GPU of 12Gib dedicated RAM, the training took 1 week for just an epoch on 200k sequences. Using the same machine in this experiment would have taken around 2 months to process the whole dataset once. Therefore, large computing resources were heavily needed.

Luckily enough, during the same period, my direct thesis guide, Michael Heinzinger, received the first phase of a $4.5 million grant from Google through the Covid-19 High-Performance Computing (HPC) Consortium for a volunteer research project. I was granted part of this considerable computing resource to train my final 24 Million parameters model, referred to as Seq_Evo_LM in the rest of this manuscript. I got approximately four weeks of exclusive usage of a remote Linux VM with a system memory size (RAM) of 120GB and 32 Intel® Xeon® CPUs with a maximum speed of 2.3 GHz. The machine also contained a cluster of 8 Tesla V100 SXM2 GPUs with a dedicated memory of 16GB each. For my training, I used 2 GPUs for each experiment. With these resources, we were able to increase the training batch size and fit 64 and 128 sequences per iteration for a maximum of 300 and 100 time steps, respectively. For a maximum time step of 100 residues, the training took one week, whereas it took a week and a half for a maximum length of 300 residues.

For a given protein sequence of length L, Seq_Evo_LM generates a numerical representation (embedding) of size 3xLx512. The first layer is just the SeqVec uncontextualized embedding (512); while, the second and third layers are the output of the two LSTMPs of our final trained model. Since the projections are of size 256 each, the output of the forward pass is concatenated with the backward pass representations, making it a bidirectional context aware embedding from

[5] https://github.com/issararab/Evol_Info_Prediction

each of the two layers. After one epoch of training on the 1.83M sequences, we used the learned representations as input to train two models one predicting the 3 and second the 8 states of proteins secondary structure. Figures B.2 and B.3, in appendix B, show the results comparing input embeddings generated from a model trained with a maximum sequence length of 300 residues and a second of 100 residues. The performance was tested on three sets, where the $CASP_{12}$ [82] dataset being the most difficult. We see that *Seq_Evo_LM/SeqLen=100* outperforms its rival by almost an increase of ~10% and ~6.5% for the 3-state and 8-state secondary structure predictions respectively, on all of the three test sets. We can conclude that a larger sequence length negatively affects the learned representations. Therefore, we retained 100 time-steps as our best maximum sequence length hyperparameter.

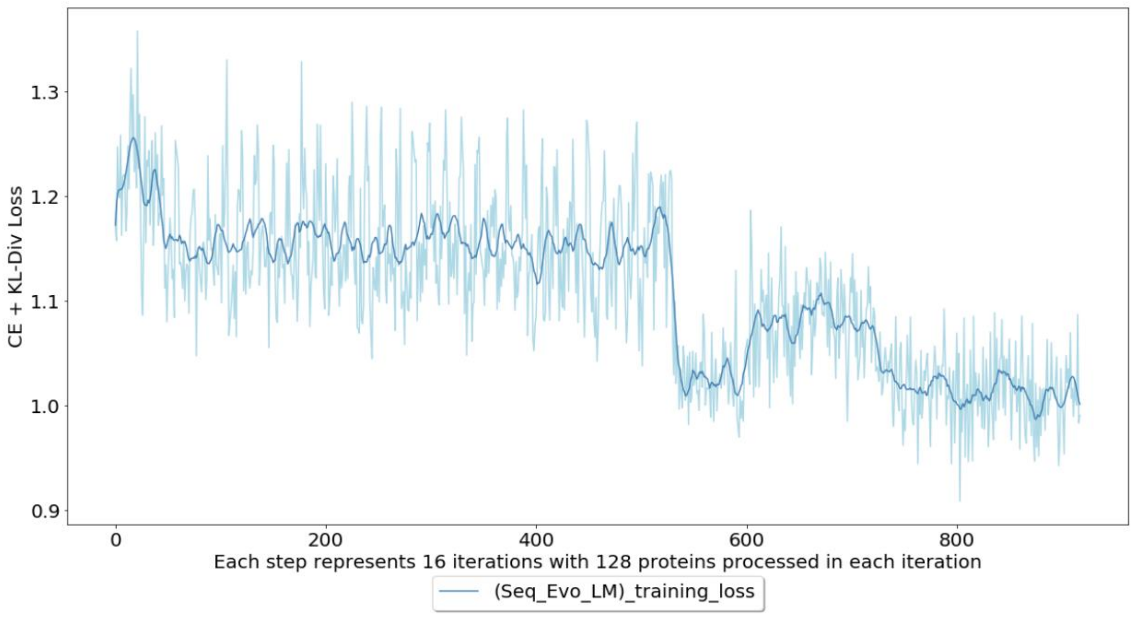


Figure 27. Smoothed training curve of Seq_Evo_LM multi-task training on a dataset of 1.83 Million proteins using a maximum time step of 100 residues.

For our best ELMo model, trained on a maximum sequence of 100 residues, we have displayed the training curve of the joint final loss in figure 27. We see how the high capacity architecture as well as the large batch size have allowed the model to learn representations from our 1.83 Million training sequences. The visual shows that the loss function has made 3 improvement drops: The first around 800 iterations after processing 100K sequences, the second around 8300 iterations after processing 1 million sequences, and the third 12K iterations after processing 1.5 million proteins. To assess how well each of the separate tasks, predicting either the next residue or the next PSSM column, did contribute to the final loss, we are displaying the training curves of each task separately and comparing it to the previous results and baselines.

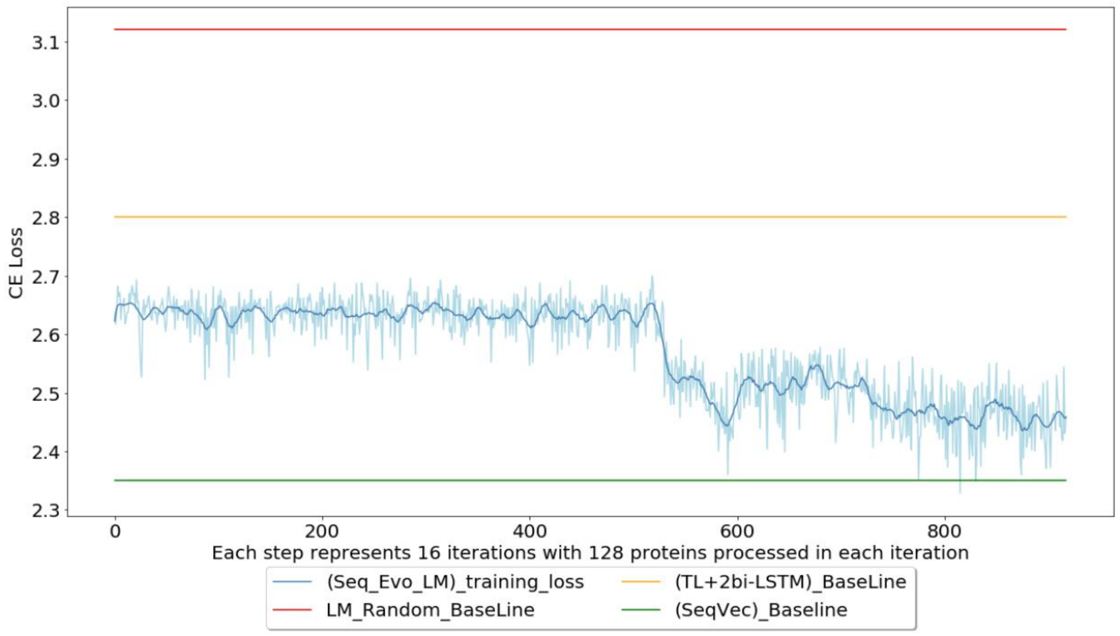


Figure 28. Seq_Evo_LM smoothed training curve predicting the next amino acid in the sequence. The model was trained on a dataset of 1.83 Million proteins using a maximum time step of 100 residues. The training curve is compared with three baselines.

Figure 28 shows the training curve (darker blue) of the final self-learning task predicting the next amino acid in a sequence from all previously seen residues. Comparing the learning trend in both figure 28 and 26, we see how the new architecture did improve the training significantly when compared to a plain LSTM architecture. While the initial architecture's training (figure 24) maintained a constant rate (~2.8, orange for baseline) over time, the training loss of our last implementation starts at a much lower value (~2.65), with a low range oscillation throughout the whole training, and converged at a loss (~2.4) close to SeqVec reported performance (~2.35, green for baseline).

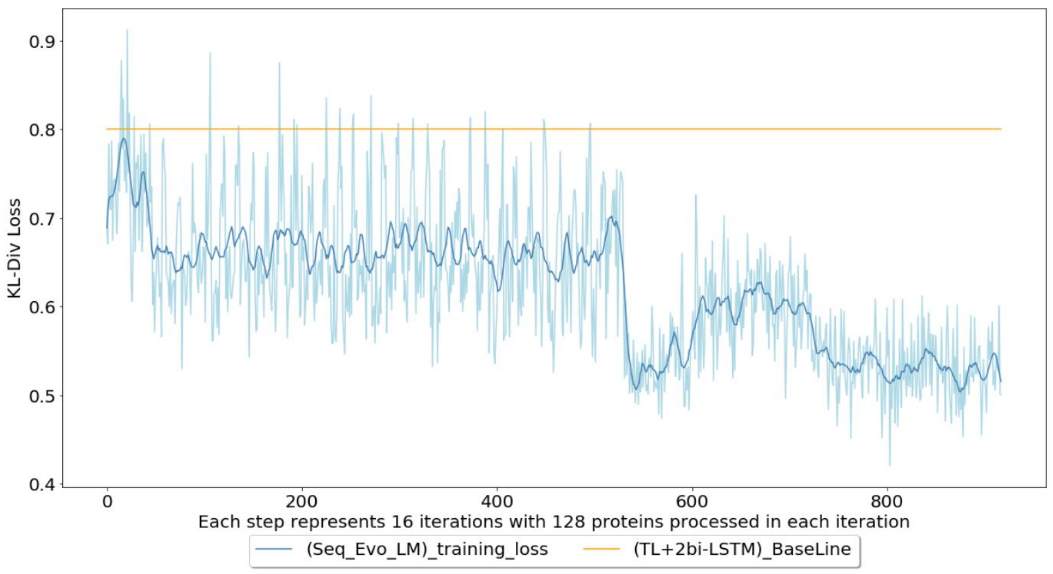


Figure 29. Seq_Evo_LM smoothed training curve predicting the next PSSM column in the sequence. The model was trained on a dataset of 1.83 Million proteins with corresponding relative frequencies using a maximum time step of 100 residues. The training curve is compared with the vanilla LSTM architecture baseline.

Similarly to the cross-entropy loss, the 𝕂𝕃-divergence showed a comparable behaviour for the novel ELMo architecture. Figure 29 displays the next PSSM column smoothed training curve where the model converged around 0.5, more than $1/3$ lower than the plain LSTM architecture (orange). Table 3 summarizes the performance improvement throughout this research project, achieving a joint best loss of 1.0 with a cross-entropy converging around CE = 2.4 and a 𝕂𝕃-Divergence loss value at 𝕂𝕃-Div.= 0.5. The best model architecture is defined with a hidden size of 2048 and an output projection size of 256 units. The model, named Seq_Evo_LM, is now used as a protein sequence embedder for downstream tasks.

Table 3. The table compares the performance of the final ELMo architecture training 2 stacked bidirectional LSTMs with projection via transfer learning (TL). The architecture connects also the first LSTM with second via a skip connection. The table shows the separate values of the CE and 𝕂𝕃-Divergence losses as well as the joint final loss. Besides, the table displays the values of the random, initial architecture, and SeqVec reported baselines.

	Cross-Entropy loss (AA)	**𝕂𝕃-Divergence Loss (PSSM)**	**Joint loss (AA + PSSM)**	**Hidden size / Output size**
Random baseline	3.12	-	-	-
SeqVec reported baseline	**2.35**	-	-	4096/512
TL+2 bi-LSTM	2.8	0.8	1.8	256/256
Seq_Evo_LM	2.4	**0.5**	**1.0**	2048/256

5.4- Evaluation of the best model on downstream tasks

After convergence, we evaluated our novel ELMo embedder, Seq_Evo_LM, in downstream tasks. As a reminder, our ultimate goal was to deploy a tool that converts sequences into numerical representations encoding the protein evolutionary information, hoping to improve the current prediction power of SeqVec embeddings. The embedding predictive power was evaluated on two categories of downstream tasks: the first task involves secondary structure, which is a per-residue type of predictions; the second comprises subcellular localization and soluble vs. membrane proteins as a per-protein level of predictions.

Concerning the first evaluation, we trained a model using the new embeddings as input to predict the three states of a protein secondary structure: helix (H), strand (E), and coil (C). We trained the model until convergence and tested it on three test sets:

- TS_{115} [83]: a set of 115 sequences derived from high-quality protein structures (i.e. < 3 Å) with no more than 30% PIDE to any protein of known structure in the PDB [84] in 2015.
- CB_{513} [85]: a set of 513 non-redundant sequences compiled after a Structure Integration with Function, Taxonomy and Sequence (SIFTS) [86] mapping.
- $CASP_{12}$ [82]: a set of 21 protein sequences retrieved in 2018 from the $CASP_{12}$ free-modelling targets after a SIFTS mapping.

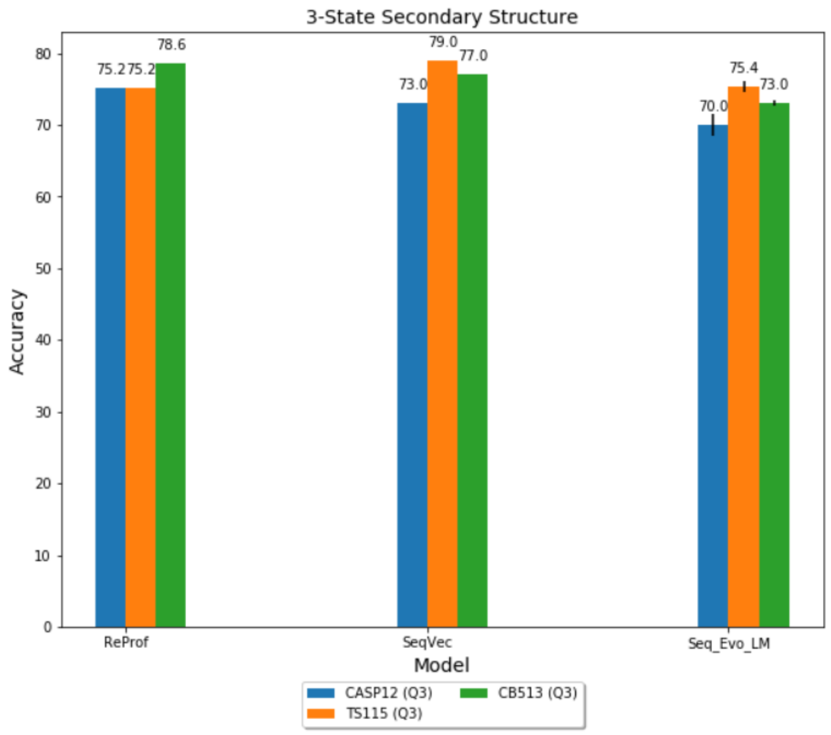


Figure 30. 3-state secondary structure prediction comparison between MSA-based and ELMo-based inducers

Figure 30 displays the performance results of Seq_Evo_LM embedder compared to SeqVec, another ELMo-based model, and ReProf [87], an MSA-based model. ReProf is built on PSSM matrices input generated by MSA methods. The latter is still considered as one of the state-of-the-art methods on this task. From the histogram plot, we see that SeqVec is performing quite well without the need of evolutionary information to make predictions. we also observe that SeqVec and Seq_Evo_Lm are performing in a quite comparable way where SeqVec is still doing better. However, we should mention that SeqVec is relying on embeddings of size 3x1024 for each residue in the sequence to reach this performance, whereas Seq_Evo_LM requires embeddings with only half the size of its rival (3x512). The same observation was drawn from figure 31, visualizing two per-protein tasks.

The second evaluation was conducted on predicting the membrane-bound proteins from the water-soluble ones. The two-state predictions were tested on a set of 846 proteins retrieved from DeepLoc [88] published supplementary data set. DeepLoc is a state-of-the-art tool relying on MSA profiles output to build its models. It also includes a model predicting the 10 states of subcellular localization, corresponding to our third benchmarking evaluation. The same annotated test set was used to evaluate and benchmark Seq_Evo_LM and SeqVec embeddings with DeepLoc. The orange bars correspond to the performance results of the Q_2 membrane predictions. We see that Seq_Evo_LM is lower by 10% and 4% compared to DeepLoc and SeqVec, respectively.

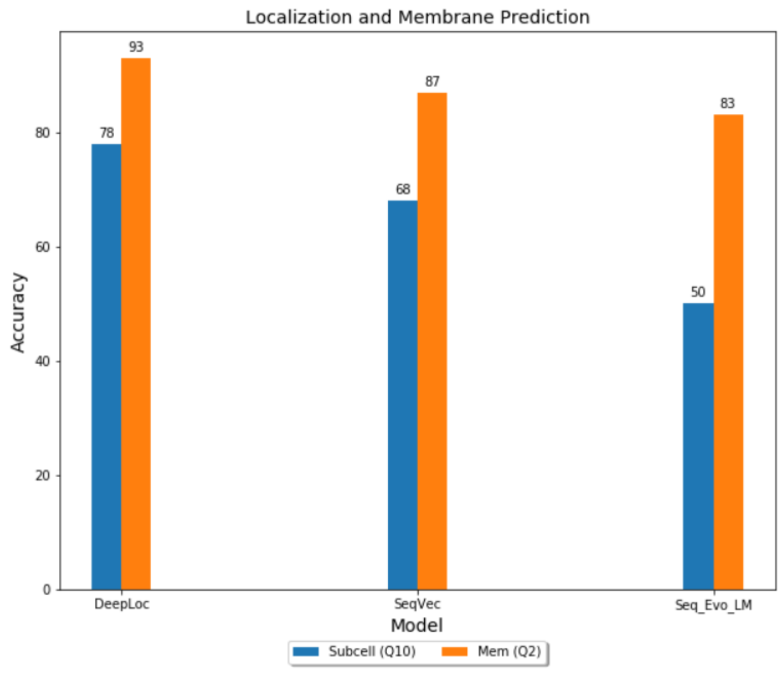


Figure 31. Subcellular localization and membrane vs soluble protein prediction comparison between MSA-based and ELMo-based inducers

While DeepLoc is still outperforming both of the ELMo-based methods ($Q_2 = 93\%$), our suggested embedding performance is still significantly high hitting an 83% with an embedding size of only 3x512 per residue. However, the Q_{10} localization predictions seem to have suffered badly, with an overall accuracy of 50%, which is 28% lower than DeepLoc and 18% lower than SeqVec.

For multi-task problems, it is always better to analyse the prediction results by displaying the confusion matrix of all the classes. Figure 32 compares the precision performance of Seq_Evo_LM to SeqVec in terms of subcellular localization predictions. While SeqVec tends towards a diagonal matrix (i.e. perfect classification in the best case scenario), Seq_Evo_LM is not. Both representations succeed to classify *NC*, *EXT*, and *ER* with quite a good performance, where Seq_Evo_LM *NC* score is surpassing the one of SeqVec by 14%. Also, with the new embeddings, the model seems confusing most of the CYTO as NC, and with a very high confidence of 87%. In fact, Seq_Evo_LM looks in general biased towards the *NC* class. An observation that is worth investigating more in future work.

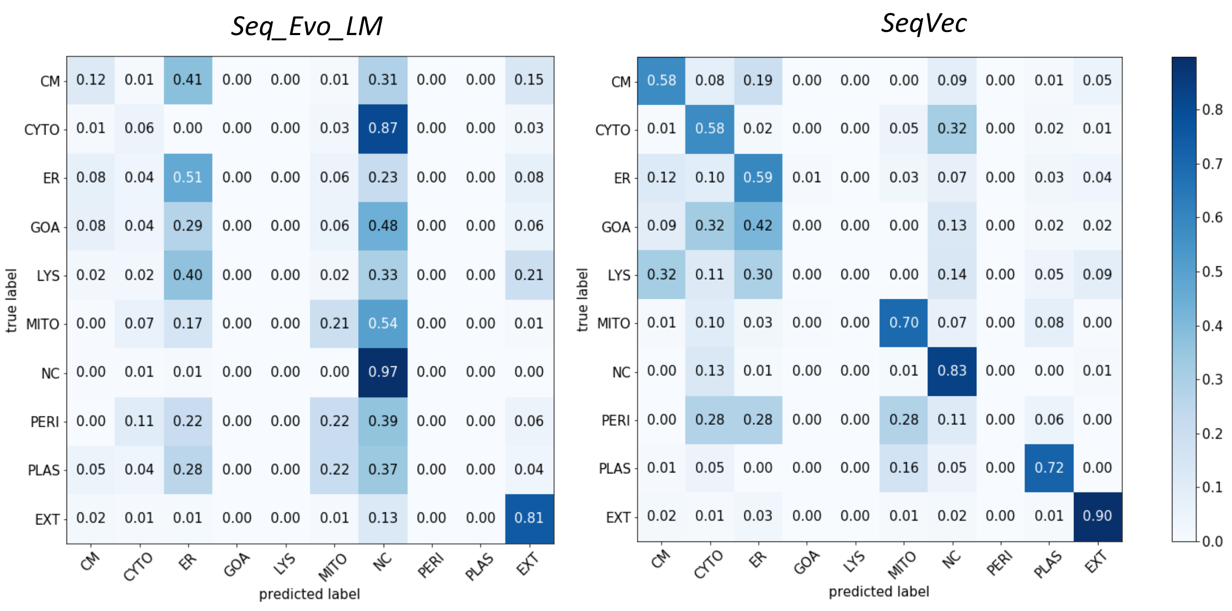


Figure 32. Precision confusion matrix for Subcellular localization predictions comparing SeqVec predictions to Seq_Evo_LM at the class level

6- Conclusion

The current algorithms for generating evolutionarily related information of protein sequences is largely dominated by multiple sequence alignment methods. Those algorithms that can be applied to DNA, RNA, and protein sequences are designed to take into account evolutionary events such as mutations, insertions, or deletions. We have seen that this technique is one of the most widely used modeling approaches in biology. MSAMs are used to expose those restricted evolutionary regions within a sequence. The results of these methods represent an essential input to several downstream applications in the field of bioinformatics. We have also seen that many of the machine learning models used in the field of bioinformatics and computational biology to predict either function or structure of proteins rely on the output of PSI-BLAST, an MSA-based tool. The process is simply described as searching for homologs of a query sequence in a database of protein sequences, capturing the conservation patterns in the alignment, and storing this information as a matrix of numerical scores for each position in the alignment.

Indeed, the evolutionary information representations generated by MSA methods have revolutionized the prediction power of AI methods for the past two decades. However, this increase in performance has become costly in recent years, with the continuous exponential growth of bio-sequence data pools. Thanks to the similarity of protein sequences to natural language, researchers are now trying to exploit the recent advancements in the field of natural language processing and therefore transferring NLP state-of-the-art algorithms to bioinformatics. One prominent solution that can cope with such tremendous growth and compete with conventional methods is the use of embedding language models (ELMo). This was the core of this research project, where we aimed to leverage AI to learn representations of evolutionary information of a protein.

Ultimately, at Rostlab, we wanted to deploy an ELMo tool incorporating evolutionary information in its representations and offering a better trade-off between computing resources and runtime. While pre-training is costly, inference is cheaper. So we trained a novel bidirectional language model following the autoregressive approach. The model was trained on a set of 1.83 million proteins (~0.8 billion residues), predicting both the next amino acid in a sequence and the probability distribution of the next residue derived from similar, yet different, sequences, as summarized in a PSSM. With 975MB required GPU memory to load the final pre-trained model, the average inference time takes around 1.03 second to embed a human protein sequence segment of size 512 residues. The embeddings' prediction power was evaluated on three downstream tasks: secondary structure, subcellular localization, and membrane vs. soluble protein predictions.

According to the experiments results, the embedder performance was comparable to SeqVec, for the 3-state secondary structure predictions, but did not outperform it. The same trend was observed for the 2-state membrane prediction, achieving a performance of 83%, and satisfactory but relatively low in terms of the 10-state subcellular localization compared to DeepLoc. Even though we did not succeed in reaching our initial goal of outperforming SeqVec, we did achieve a performance close to SeqVec with half the embedding dimensions (i.e. 3 x 512). Additionally, the results of the third experiment confirmed a finding by Jozefowicz et al. [78], who did empirically prove that, when trying to fit an LSTM network architecture on very large and complex datasets, the size of the LSTM matters.

In general and from downstream tasks perspective, we did not find any evidence that adding PSSM input improved the knowledge that LMs could learn from a sequence. Different hypotheses can be drawn here: SeqVec was able to learn certain patterns already without PSSMs, our dataset was too small for training a LM, we should have trained the whole architecture from scratch, we should have used larger capacity LSTMPs, or we should have simply trained longer.

References

[1] Maria Chatzou, Cedrik Magis, Jia-Ming Chang, Carsten Kemena, Giovanni Bussotti, Ionas Erb, Cedric Notredame, Multiple sequence alignment modeling: methods and applications, *Briefings in Bioinformatics*, Volume 17, Issue 6, November 2016, Pages 1009–1023, doi: https://doi.org/10.1093/bib/bbv099

[2] Van Noorden RMaher BNuzzo R. The top 100 papers. Nature 2014;514:550–3.

[3] Altschul, S. F., Madden, T. L., Schäffer, A. A., Zhang, J., Zhang, Z., Miller, W., & Lipman, D. J. (1997). Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. Nucleic acids research, 25(17), 3389–3402. doi: https://doi.org/10.1093/nar/25.17.3389

[4] Kemena CNotredame C. Upcoming challenges for multiple sequence alignment methods in the high-throughput era. Bioinformatics 2009;25:2455–65.

[5] Barton J. Geoffrey, Protein Sequence Alignment Techniques, European Molecular Biology Laboratory Outstation, The European Bioinformatics Institute, Wellcome Trust Genome Campus, Hinxton, Cambridge CB10 LSD, England. Acta Cryst. (1998). D54,1139-1146

[6] Rost B., Sander C. (1993) Prediction of protein secondary structure at better than 70% accuracy. J Mol Biol. 232:584–99.

[7] Rost B, Sander C. Improved prediction of protein secondary structure by use of sequence profiles and neural networks. Proc Natl Acad Sci. 1993;90:7558–62.

[8] Barton GJ. Protein secondary structure prediction. Curr Opin Struct Biol. 1995;5:372–6.

[9] Chandonia J-M, Karplus M. Neural networks for secondary structure and structural class predictions. Protein Sci. 1995;4:275–85.

[10] Bigelow H, Petrey D, Liu J, Przybylski D, Rost B. Predicting transmembrane beta-barrels in proteomes. Nucleic Acids Res. 2004;32:2566–77.

[11] Rost B, Casadio R, Fariselli P. Topology prediction for helical transmembrane proteins at 86% accuracy. Protein Sci. 1996;5:1704–18.

[12] Rost B, Casadio R, Fariselli P, Sander C. Transmembrane helix prediction at 95% accuracy. Protein Sci. 1995;4:521–33.

[13] Punta M, Rost B. PROFcon: novel prediction of long-range contacts.Bioinform. 2005;21(13):2960–8.

[14] Nair R, Rost B. Better prediction of sub-cellular localization by combining evolutionary and structural information. Proteins. 2003;53(4):917–30.

[15] Nair R, Rost B. Mimicking cellular sorting improves prediction of subcellular localization. J Mol Biol. 2005;348(1):85–100.

[16] Marino Buslje C, Teppa E, Di Domenico T, Delfino JM, Nielsen M. Networks of high mutual information define the structural proximity of catalytic sites:implications for catalytic residue identification. PLoS Comput Biol. 2010;6(11):e1000978.

[17] Ofran Y, Rost B. ISIS: interaction sites identified from sequence. Bioinform. 2007;23(2):e13–6.

[18] Ofran Y, Rost B. Protein-protein interaction hot spots carved into sequences.PLoS Comput Biol. 2007;3(7):e119.

[19] Suzek BE, Wang Y, Huang H, McGarvey PB, Wu CH, UniProt C. UniRef clusters: a comprehensive and scalable alternative for improving sequence similarity searches. Bioinform. 2015;31(6):926–32.

[20] Steinegger M and Soeding J. (2017) MMseqs2 enables sensitive protein sequence searching for the analysis of massive data sets. Nature Biotechnology, doi: 10.1038/nbt.3988.

[21] Steinegger M and Soeding J. (2018) Clustering huge protein sequence sets in linear time. Nature Communications, doi: 10.1038/s41467-018-04964.

[22] Peters M., Neumann M., Iyyer M., Gardner M., Clark C., Lee K., Zettlemoyer L. (2018). Deep contextualized word representations. arXiv:1802.05365

[23] Mikolov T., Sutskever I., Chen K., Corrado G. S., and Dean J. (2013). Distributed representations of words and phrases and their compositionality. In NIPS.

[24] Pennington J., Socher R., and Manning C. D. (2014) . Glove: Global vectors for word representation. In EMNLP

[25] Turian J. P., Ratinov L., and Bengio Y. (2010). Word representations: A simple and general method for semi-supervised learning. In ACL

[26] He L., Lee K., Lewis M., and Zettlemoyer L. S. (2017). Deep semantic role labeling: What works and what’s next. In ACL

[27] Liu X., Shen Y., Duh K., and Gao. J. (2017). Stochastic answer networks for machine reading comprehension. arXiv preprint arXiv:1712.03556 .

[28] Chen Q., Zhu X., Ling Z., Wei S., Jiang H., and Inkpen D. (2017). Enhanced lstm for natural language inference. In ACL.

[29] Wieting J., Bansal M., Gimpel K., and Livescu K. (2016). Charagram: Embedding words and sentences via character n-grams. In EMNLP

[30] Bojanowski P., Grave E., Joulin A., and Mikolov T. (2017). Enriching word vectors with subword information. TACL 5:135–146

[31] Neelakantan A., Shankar J., Passos A., and McCallum A. (2014). Efficient nonparametric estimation of multiple embeddings per word in vector space. In EMNLP.

[32] Srivastava R. K., Greff K., and Schmidhuber J. (2015). Training very deep networks. In NIPS.

[33] Melamud O., Goldberger J., and Dagan I. (2016). context2vec: Learning generic context embedding with bidirectional lstm. In CoNLL.

[34] Hochreiter S. and Schmidhuber J. 1997. Long short-term memory. Neural Computation 9.

[35] Akaike H. (1998) Autoregressive Model Fitting for Control. In: Parzen E., Tanabe K., Kitagawa G. (eds) Selected Papers of Hirotugu Akaike. Springer Series in Statistics (Perspectives in Statistics). Springer, New York, NY

[36] Heinzinger, M., Elnaggar, A., Wang, Y. et al. Modeling aspects of the language of life through transfer-learning protein sequences. BMC Bioinformatics 20, 723 (2019). https://doi.org/10.1186/s12859-019-3220-8

[37] Schils E., Haan P., Characteristics of Sentence Length in Running Text, Literary and Linguistic Computing, Volume 8, Issue 1, 1993, Pages 20–26, https://doi.org/10.1093/llc/8.1.20

[38] Bhagwat M, Aravind L. PSI-BLAST Tutorial. In: Bergman NH, editor. Comparative Genomics: Volumes 1 and 2. Totowa (NJ): Humana Press; 2007. Chapter 10. Available from: https://www.ncbi.nlm.nih.gov/books/NBK2590/

[39] Altschul S. F., Gish W., Miller W., Myers E. W., Lipman D. J. (1990) Basic local alignment search tool. J. Mol. Biol ., 215, 403–410.

[40] Farrar M. Striped Smith-Waterman speeds database searches six times over other SIMD implementations. Bioinformatics, 23(2):156–161, Jan. 2007.

[41] Fine T.L., Feed forward Neural Network Methodology, third ed., Springer-Verlag, New York, 1999.

[42] Bishop C.M., Neural Networks for Pattern Recognition, third ed., Oxford University Press, Oxford, 1995.

[43] Bishop C.M., Pattern Recognition and Machine Learning: Continuous Latent Variables, third ed., Oxford University Press, Cambridge CB3 0FB, U.K, 2006.

[44] Teuwen J., Moriakov N. (2019) Handbook of Medical Image Computing and Computer Assisted Intervention, Chapter 20 - Convolutional neural networks, Academic Press, 481 - 501

[45] Hinton E.G., Srivastava N., Krizhevsky A., Sutskever I., and Salakhutdi-nov R.R. (2012) Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprintarXiv:1207.0580

[46] Srivastava N., Hinton G., Krizhevsky A., Sutskever I., Salakhutdinov R. (2014), Dropout: A Simple Way to Prevent Neural Networks from Overfitting Journal of Machine Learning Research 15 (2014) 1929-1958

[47] Abu-Mostafa S. Y. (1980). Learning from hints in neural networks. Journal of Complexity, 6(2):192–198,1990.

[48] Burges J.C.C. and Schlkopf B. (1997). Improving the accuracy and speed of support vector machines.In Advances in Neural Information Processing Systems, pages 375–381.

[49] Simard P.Y., Le Cun Y.A., Denker J.S., and Victorri B. (2000). Transformation invariance inpattern recognition: Tangent distance and propagation. International Journal of Imaging Systems andTechnology, 11(3):181–197

[50] Rifai S., Dauphin Y., Vincent P., Bengio Y., and Muller X. (2011). The manifold tangentclassifier. Advances in Neural Information Processing Systems, 24:2294–2302.

[51] He k., Zhang X., Ren S., Sun J. (2015) Deep Residual Learning for Image Recognition. 770-778. 10.1109/CVPR.2016.90.

[52] Tan C., Sun F., Kong T., Zhang W., Yang C., and Liu C. (2018). A Survey on Deep Transfer Learning. The 27th International Conference on Artificial Neural Networks. doi: arXiv:1808.01974v1

[53] Shorten C. & Khoshgoftaar T. (2019). A survey on Image Data Augmentation for Deep Learning. Journal of Big Data. 6. 10.1186/s40537-019-0197-0.

[54] Akaike H. (1998) Autoregressive Model Fitting for Control. In: Parzen E., Tanabe K., Kitagawa G. (eds) Selected Papers of Hirotugu Akaike. Springer Series in Statistics (Perspectives in Statistics). Springer, New York, NY

[55] Leal-Taixé L. and Niessner M., Introduction to Deep Learning: Recurrent Neural Networks, Computer Vision Group, Department of Computer Science, Technical University of Munich, 22.01.2019

[56] Günnemann S., Machine Learning: Lecture1 - Decision Trees, Data Mining and Analytics, Technical University of Munich, 22.10.2018

[57] The UniProt Consortium, UniProt: a worldwide hub of protein knowledge, *Nucleic Acids Research*, Volume 47, Issue D1, 08 January 2019, Pages D506–D515, https://doi.org/10.1093/nar/gky1049

[58] Rost, B., Yachdav, G., & Liu, J. (2004). The PredictProtein server. Nucleic acids research, 32(Web Server issue), W321–W326. doi:10.1093/nar/gkh377

[59] Li W, Godzik A. Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences. Bioinformatics. 2006;22(13):1658-1659. doi:10.1093/bioinformatics/btl158

[60] Robert C. (2015). The Metropolis—Hastings Algorithm. 10.1007/978-1-4757-4145-2_7.

[61] Rubio V. G., Bivand R. & Rue H.. (2014). Spatial Models Using Laplace Approximation Methods. 10.1007/978-3-642-23430-9_104.

[62] C. Zhang, J. Bütepage, H. Kjellström and S. Mandt, "Advances in Variational Inference," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 41, no. 8, pp. 2008-2026, 1 Aug. 2019, doi: 10.1109/TPAMI.2018.2889774.

[63] S. Günnemann, Machine Learning: Lecture12 - Variational Inference, Data Mining and Analytics, Technical University of Munich, 22.10.2018

[64] Blei D., Ranganath R. & Mohamed S. "Variational Inference: Foundations and Modern Methods", NIPS, Columbia University, 5. 12.2016

[65] Lu Y. & Lu J. (2020). A Universal Approximation Theorem of Deep Neural Networks for Expressing Distributions. arxiv:2004.08867v2

[66] Cipolla R., Gal Y., Kendall A. (2018). Multi-task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics. 7482-7491. 10.1109/CVPR.2018.00781.

[67] Werbos J. P. (1990) Backpropagation through time: What it does and how to do it. Proceedings of the IEEE, 78(10):1550–1560

[68] Williams J. R., Zipser D. (1995). Gradient-based learning algorithms for recurrent networks and their computational complexity. Backpropagation: Theory, Architectures, and Applications, 433.

[69] Sutskever I. (2013) Training Recurrent Neural Networks. University of Toronto Toronto, Ontario, Canada

[70] Schulz E., Speekenbrink M., Krause A. , (2017) A tutorial on Gaussian process regression: Modelling, exploring, and exploiting functions. bioRxiv doi: https://doi.org/10.1101/095190

[71] Ostertagova E. & Ostertag O. (2016). Methodology and Application of Savitzky-Golay Moving Average Polynomial Smoother. Global Journal of Pure and Applied Mathematics. 12. 3201-3210.

[72] Radiuk P.M. (2017) Impact of training set batch size on the performance of convolutional neural networks fordiverse datasets.Inf. Technol. Manag. Sci.2017,20. doi: 10.1515/itms-2017-0003

[73] Kandel I., Castelli M. (2020) The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset. doi: doi.org/10.1016/j.icte.2020.04.010

[74] Hoffer E., Ben-Nun T., Hubara I., Gilad N., Hoefler T., Soudry D. (2019) Augment your batch: better training with larger batches. doi: arXiv:1901.09335v1

[75] Hoffer E., Hubara I., Soudry D. (2018) Train longer, generalize better: closing the generalization gap in large batch training of neural networks. doi: arXiv:1705.08741v2

[76] Ba J., Caruana R. (2014) Do Deep Nets Really Need to be Deep? doi: arXiv:1312.6184v7

[77] Sak H., Senior A., Beaufays F. (2014) Long Short-Term Memory Based Recurrent Neural Network Architectures for Large Vocabulary Speech Recognition. doi: arXiv:1402.1128v1

[78] Jozefowicz R., Vinyals O., Schuster M., Shazeer N, and Wu Y. (2016) Exploring the limits of language modeling. CoRR abs/1602.02410.

[79] Chelba C., Mikolov T., Schuster M., Ge Q., Brants T., Koehn P., and Robinson T. (2013) One billion word benchmark for measuring progress in statistical language modeling. arXiv preprint arXiv:1312.3005

[80] He K., Xiangyu Z., Shaoqing R., and Jian S. (2015) Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In International Conference on Computer Vision (ICCV), 2015. URL http://arxiv.org/abs/1502.01852.

[81] Jozefowicz R., Zaremba W., and Sutskever I. (2015) An empirical exploration of recurrent network architectures. In Proceedings of the 32nd International Conference on Machine Learning (ICML-15), pp. 2342–2350, 2015.

[82] Abriata LA, Tamò GE, Monastyrskyy B, Kryshtafovych A, Dal Peraro M. Assessment of hard target modeling in CASP12 reveals an emerging role of alignment-based contact prediction methods. Proteins. 2018;86:97–112

[83] Yang Y, Gao J, Wang J, Heffernan R, Hanson J, Paliwal K, Zhou Y. Sixty-five years of the long march in protein secondary structure prediction: the final stretch? Brief Bioinform. 2016;19(3):482–94.

[84] Berman HM, Westbrook J, Feng Z, Gilliland G, Bhat TN, Weissig H, Shindyalov IN, Bourne PE. The protein data bank. Nucleic Acids Res. 2000;28(1):235–42.

[85] Cuff JA, Barton GJ. Evaluation and improvement of multiple sequence methods for protein secondary structure prediction. Proteins Struct Funct Genet. 1999;34(4):508–19.

[86] Velankar S, Dana JM, Jacobsen J, Van Ginkel G, Gane PJ, Luo J, Oldfield TJ, O'donovan C, Martin M-J, Kleywegt GJ. SIFTS: structure integration with function, taxonomy and sequences resource. Nucleic Acids Res. 2012;41(D1):D483–9.

[87] Rost B., Sander C. (1993). Improved prediction of protein secondary structure by use of sequence profiles and neural networks. Proceedings of the National Academy of Sciences, 90, 7558-7562.

[88] Almagro Armenteros JJ, Sonderby CK, Sonderby SK, Nielsen H, Winther O. DeepLoc: prediction of protein subcellular localization using deep learning. Bioinform. 2017;33(24):4049.

Supplementary Material

Appendix A

A.1 - Isar-Pipeline

In the Machine Learning arena of the bioinformatics field, PSI-BLAST output represents the main input to many models out there. Therefore, ways to speed up this process are heavily needed. During a previous study, we conducted an analysis of this tool in a research project that aimed at combining state-of-the-art methods to quickly generate extensive protein sequence alignments. In this research, we compared the computational resources and time needed to generate alignment profiles of a set of query proteins against a similar database. The benchmarking was done with another state-of-the-art algorithm, MMseqs2, that claims to speed up this process significantly.

A.1.1 - Experiment setting

To compare the runtime of each algorithm in retrieving relevant aligned sequences, we run multiple experiments on the same dataset and using the same machine. Experiments were conducted using variant numbers of proteins in each batch.

All the results presented in this section were conducted on a server granted by the University of Luxembourg. The server system memory size had about 196Gib with an Intel Xeon E312xx CPU of 4 cores supporting SSE4.1 instruction set.

Concerning the database of proteins used, we sampled a random set from UniProt Reference Cluster with 90% sequence identity (uniref90 2019_02). The reason behind using only a sample from uniref90 was the RAM size constraint. For full performance analysis, we needed the index table of the database to fit entirely in the RAM. Since the whole uniref90 will generate an index table of roughly 250Gib, we run our tests on a random sample of sequences (~ 68 million proteins). This sample generated an index table of size 181Gib.

A.1.2 - Experiment results and discussion

The experiments were conducted on query sets of 10 different batch sizes, between 1 to 1000 sequences per batch. Each experiment with the same batch size was run 3 times using the same algorithm, either MMseqs2 or PSI-BLAST. The final result is reported as the average sum of values.

Table A.1. The table shows the effect of random bulk sequences (number of seqs) as a single query on the runtime of MMseqs2 and PSI-BLAST

Number of Sequences in Batch	Runtime	
	MMseqs2	PSI-BLAST
1	24m 48s	14m 05s
5	28m 52s	69m 39s
10	22m 11s	109m 22s
20	23m 02s	218m 56s
30	26m 08s	340m 32s
50	28m 03s	605m 12s
100	29m 21s	1211m 15s
200	28m 42s	-
500	25m 43s	-
1000	32m 06s	-

In table A.1, we clearly see that the increasing number of sequences in a single bulk query heavily affects the runtime of PSI-BLAST. On the other hand, MMseqs2 is barely affected by the batch size (Figure 1). MMseqs2 runtime ranges between 22min and 32min with an average runtime of ~30min per batch. Whereas, PSI-BLAST runtime follows a linear trend (Figure 1). PSI-BLAST processes the sequences in the batch in a sequential fashion. This runtime becomes even higher when the algorithm is run against larger databases.

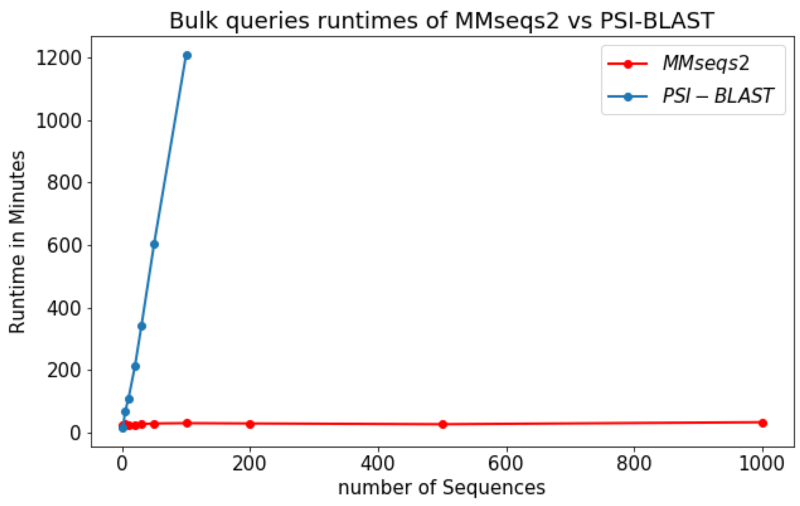


Figure A.1. Runtime comparison between MMseqs2 and PSI-BLAST and how they scale up with the increasing number of proteins in a single query batch size.

MMseqs2 scales much better in terms of batch query sequences compared to PSI-BLAST (Figure 1). This property was exploited to build a pipeline, called IsarPipeline, that combined both MMseqs2 and PSI-BLAST. The need for combining both tools was due to the absence of a PSSM output file directly generated from the result of MMseqs2. Therefore, MMseqs2 was used to run batch queries in order to quickly generate the aligned family of sequences to each protein in the query batch, and then PSI-BLAST is run afterwards to generate PSSM profiles. The general idea of the pipeline was that MMseqs2 will take care of the massive search on the large database, then it will provide PSI-BLAST with a list consisting only of the relevant sequences, a maximum of 1k, for each query protein.

After a thorough analysis of each module runtime in MMseqs2 search, we discovered that loading the index table to perform the prefiltering procedure takes more than 90% of the whole runtime reported. Therefore, a way to load the whole DB and the index table into RAM, then using it directly whenever the search command is called will decrease the runtime tremendously. The hack that we came up with was to load the whole database into RAM and lock the pages there, using "vmtouch[6] ". Later in the search, we map the sequences from the locked DB instead of loading it again from the disk. This trick allowed us to reach almost an instant sequence alignment search result, of about 53 secs via IsarPipeline, for an average length protein sequence (450 residues).

The instant sequence alignment is very impressive, but this huge gain in runtime is achieved at the expense of enormous hardware resources. In the above-presented experiment results, we had to allocate 190Gib of RAM for a database size of 68 million proteins. With a rough extrapolation estimation of the results on hand, a set of 200 million proteins will require about 1/2TB of RAM to allocate. This solution is not affordable to the general bioinformatics community. From here came the idea of researching an optimal solution incorporating evolutionary information of a protein sequence in a numerical representation with an optimal trade-off between computing resources and runtime of the software. With the current revolution of AI and the advancements achieved in the NLP field, we are aiming in this research project to transfer this knowledge in order to learn representations of protein evolutionary information.

[6] Hoyte D. (2020, August 31), vmtouch - the Virtual Memory Toucher, source: https://hoytech.com/vmtouch/

Appendix B

B.1 – SeqVec/ELMo embeddings data leakage experiment results

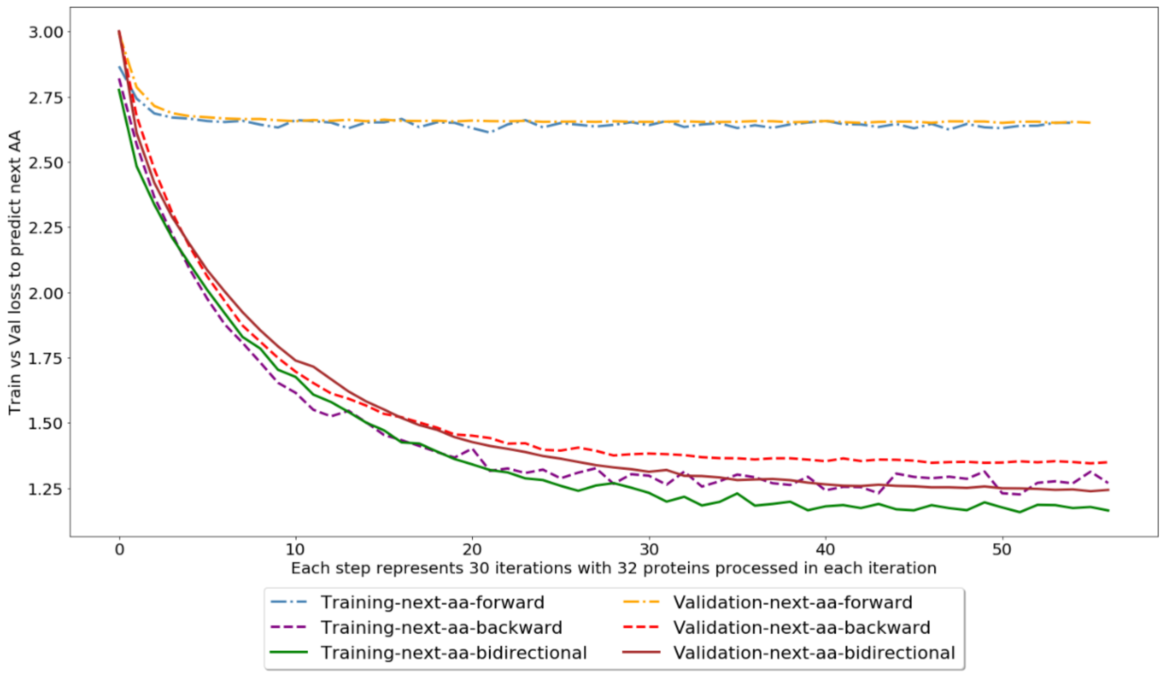


Figure B.1: Training and validation cross-entropy losses of a linear layer predicting the next amino acid in a sequence from ELMo/SeqVec last layer. The visual shows a comparison of three experiments each using a different input of SeqVec embeddings. The graph clearly presents the data leakage problem caused by including the backward embedding as an input to the model.

B.2 – Training step size length effect on the 3-state secondary structure performance

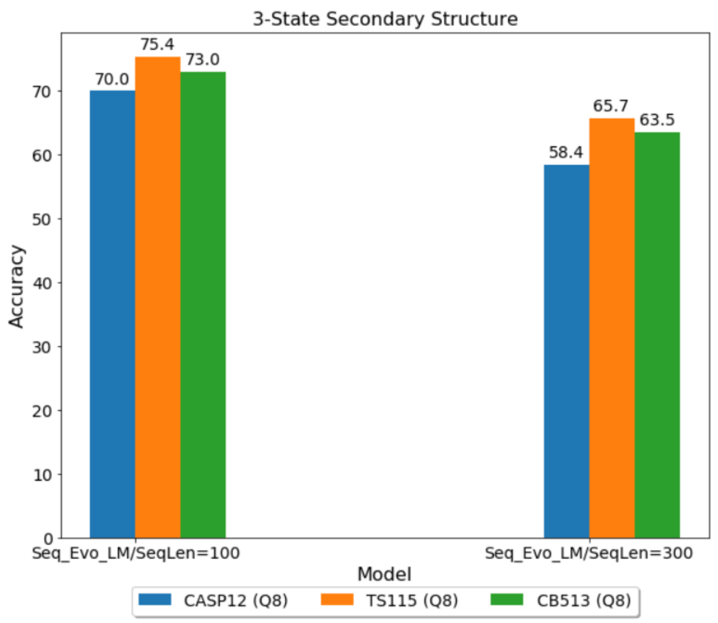


Figure B.2: Comparing the performance of Seq_Evo_LM trained with a maximum sequence length of 100 (left) to 300 (right) on the 3 State Secondary Structure.

B.3 – Training step size length effect on the 8-state secondary structure performance

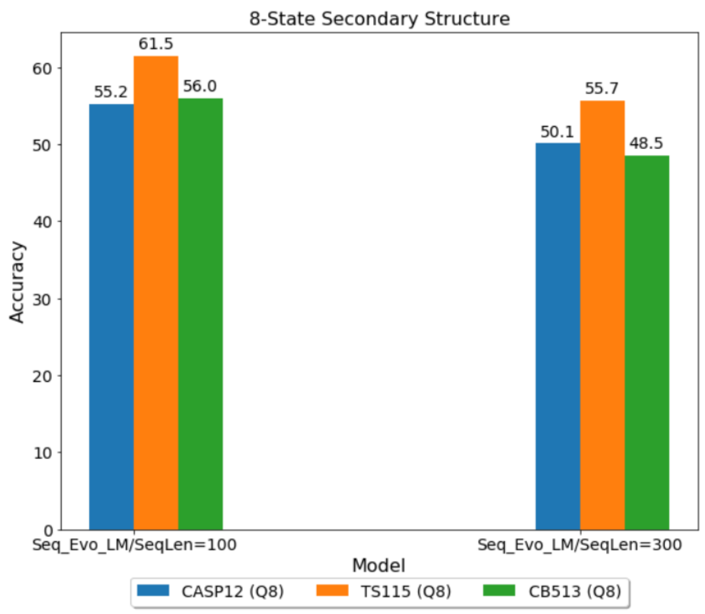


Figure B.3: Comparing the performance of Seq_Evo_LM trained with a maximum sequence length of 100 (left) to 300 (right) on the 8 State Secondary Structure.