Methodology

Dereck Piché Guillermo Martinez Jonas Gabirot

March 17, 2023

1 Task

Let us briefly resume our task once again in order to make this document self-contained. Our task is to create models capable of reproducing the classical algorithms used by the NUPACK foundation to predict the free energy ($\in R$) of a DNA strand (a sequence of the elements of $\{A, C, G, T\}$).

2 Planned Methodology

2.1 Training dataset

First, we will have to generate our training data. We shall implement a simple python script which uses the NUPACK python library in order to generate a .json file containing our training data. We randomely generated a regressively labeled datasets of a milion DNA strands of length 30.

2.2 Baseline Algorithms

2.2.1 Multilayered Perceptron

2.2.2 Decision trees with AdaBoost

We might want to use this as a baseline.

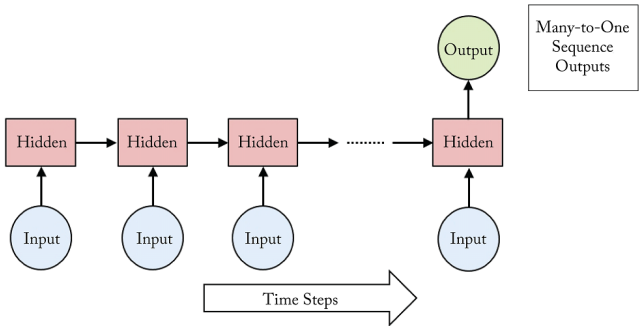
2.3 Advanced Algorithms

2.3.1 Recurrent Neural Network

Recurrent neural network (RNN) is a deep lRecurrent neural network (RNN) is a deep learning architecture used for sequential data prediction using both current and past inputs. earning architecture used for sequential data prediction using both current and past inputs.

it in a simple way, RRN architectures are composed of an encoder and a decoder. The initial input is vectorized by the encoder and processed as a function of the initial state, which is random at first. As a result, the encoder's weights and biases are adjusted in the form a second state to incorporate both current

Figure 1: Illustration of RNN architecures



and past input information. The encoder recursively processes the following vectorized inputs as functions of current states, while updating the weights and biases of current states to produce new states at each iteration. The encoder terminates this recursion when it has iterated over an entire sequence of features and concludes by transmitting its final state, which incorporates all previous states, to the decoder. In the case of a many-to-one RNN underlying architecture, this paper's architecture of interest, the decoder produces one output prediction as a function of the final state received from the encoder and as a function of its own current state, random at first. By contrasting the predicted output value to the actual value of the sequence, the decoder performs gradient descent to minimize the loss function, updates its current state and backpropagates it to the encoder's states. Once the weights and biases of the encoder states are adjusted, it iterates over the following sequence of features following the same recursion procedure. This recursive process is repeated for the entire length of sequences within the training set, and the regression model is cross-validated on its ability to minimize the squared mean error between target and predicted output values in the validation and test set.

RNNs are not without challenges. In order to update parameters, the back-propagation algorithm needs to calculate gradients at each different step. This usually results in unstable neural networks due to vanishing and exploding gradients which are unable to learn long-term dependencies. Long Short Term Memory networks (LSTMs) have been proposed to avoid these problems and designed to handle long-term dependencies. Initially proposed by Hochreiter and Schmidhuber (1997), LSTMs use cells with input, output and forget-gate to control the flow of information.

Given this paper's task to predict the level of free energy given a sequence of 30 features, we will train a many-to-one LSTMs for a regression task with multiple input time series. We will divide our entire dataset in train 80%, validation 20%. For each training instance, we will give the model a sequence of observations and a corresponding target value. The goal will be to forecast time series' free energy within the validation set. Time given, hyperparameter tuning

Figure 2: Long Short Term Memory cell

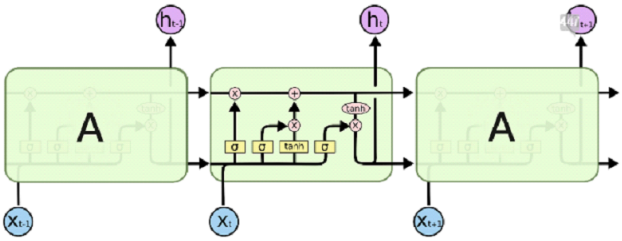
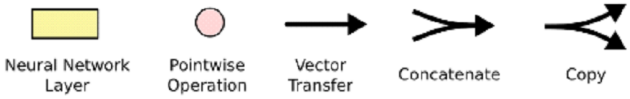


Figure 3: Information flow



will be performed on the validation set to optimize the choice of the learning rate, the number of units or layers, and the weight regularization techniques used as penalties on the loss function. Finally, the tunned model's prediction accuracy will be calculated on the test set using Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE) metrics.

2.3.2 Tranformer

Architecture The transformer architecture was initially made to translate text. However, it's subsequent use was mostly tied to token generation. This use only required the decoder to be part of the architecture, and recursively fed the predicted token in the current input while truncating if the input size was over the limit. Our task is vastly different. We are dealing with a regressive task, since we are trying to learn a function of the form $R^n \mapsto R$. Thus, we shall only keep the decoder part of the transformer architecture. We only need to set the feedworward neural network in the decoder such that it only has one output neuron.

Cost Function Since this is a regressive task, we shall use various instances of the l_p-norm as our cost function.

Tokenisation, encoding, positionnal encoding Since the tranformer learns the embedding in the attention heads, [1] we shall simply use an integer mapping for the set of tokens $\{A, C, G, T\}$ as opposed to one-hot encoding. This is done partly due to the way the Pytorch library works.

We will train and compare our the accuracy of our decoder-only tranformer with respect to the absence and inclusion of positionnal embedding to the input tokens.

Advantages What makes the decoder-only transformer different from other models? What are we taking advantage of by it's use? As opposed to recurrent neural networks, this model is less sequential in nature. We are predicting by taking the sequence of tokens all at once. We have high hopes for the distributivity of attention made possible by the head multiplicity. We can imagine that there is high importance between the ends of the DNA sequence, and at the center. A transformer, given enough data, would be able to take advantage of this structure in order to simplify the task.

2.4 Comparisons and analysis

It would be more logical to compare learned models with a similar number of parameters. This would give us more information as to if the comparative advantages were caused by the particularities of the architecture or simply because of the increase of expressivity tied to bigger models.

3 Preliminary Results and Challenges Ahead

4 Changes from the initial plan

We have not strayed afar from the original plan. We are only more sure that time will lack to implement generative models like we intended after the attempt at reproducing the aforementionned NUPACK classical algorithm.

References

[1] Ashish Vaswani et al. Attention is all you need. *CoRR*, abs/1706.03762, 2017.