**Predicting gene expression using millions of random promoter sequences by mt**

***Abstract***

I developed an end-to-end procedure to predict the expression of genes using random promoters. Using information encoded in the forward and reverse DNA strands, my approach uses Recurrent (GRU) and Convolution Neural Networks to regress the strength of the targeted promoters. The starting point of my model, two bi-bidirectional GRUs, have been recently used in a machine learning competition at Kaggle to predict the stability of RNA vaccines (https://arxiv.org/abs/2110.07531). In this work, I expanded on this architecture and found that the addition of convolutions and fully connected layers can efficiently extract features from DNA sequences and predict gene expression.

**1. Description of data usage**

The DNA forward and reverse strands were one-hot encoded using the canonical four bases (A, T, C, G) plus two special tokens, one for the bases that were not possible to sequence (N) and one for the padding (P) the sequences shorter than the longest sequence in the dataset. The padding was applied at the 5’ of the forward strand. The one-hot encoded sequences were stacked together to create an input of 142 (the longest sequence in the dataset) x 12 (6 for the forward and 6 for the reverse encoding of the DNA strands).

A custom data generator was developed to train the sequences. The data generator takes care of grabbing the promoter sequences from a pandas data frame. The padding, reverse-complement, one-hot encoding and stacking of the forward and reverse strands happen at the batch level.

The model was trained with the whole dataset minus ten thousand promoters (randomly chosen) used for validation.

**2. Description of the model**

The model I developed uses one branch starting from two bi-bidirectional GRUs followed by three convolutions and max pooling operations. At the end of the convolutions, the data is flattened and fed to two fully connected layers. The addition of dropouts at any level in this architecture makes the model train slower with less competitive results. For this reason, the model was trained without dropouts.

**3. Training procedure**

The training of this model uses the Binary Crossentropy loss coupled with the sigmoid activation of the output layer. Any other losses coupled with a liner activation of the output layer make the model unstable, with nans appearing in the metric values. The model scores are recorded after each epoch and available in the jupyter notebook. The epoch chosen for submission is 11. I’m choosing to stop training the first time that the val Pearson stops improving +1 epoch. This strategy, supported by leaderboard probing, seems successful for this architecture. I believe the likely explanation for this finding is the different nature of the train set (more noise) with respect to the test set.

**4. Other important features**

I was not able to find a combination of train \ test split, dropout or more complex architecture that would make this model overfit. On the contrary, the model slightly underfits, with testing metrics always better than training metrics.

**5. Contributions and Acknowledgement**

**5.1 Contributions**

|  |  |  |
| --- | --- | --- |
| **Name** | **Affiliation** | **Email** |
| Michele Tinti | The Wellcome Centre for Anti-Infectives Research, Dundee University | m.tinti@dundee.ac.uk |

**5.2 Acknowledgement**

I would like to acknowledge Dr Susan Wyllie, Prof. David Horn and Prof. Mike Ferguson for their continuous support and contributions to my salary.