**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. My approach uses Recurrent (GRU) and Convolution Neural Networks to regress the strength of the targeted promoters using information encoded in the forward and reverse DNA strands. 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 token for the bases that were miss-sequenced sequence (N) and one token for the padding (P). The sequences that are shorter than the longest sequence in the dataset (n=142) were padded. 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 columns (the longest sequence in the dataset) x 12 rows (6 for the forward plus 6 for the reverse 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 computation, one-hot encoding and stacking of the forward and reverse strands happen at the batch level in the data loader. The model was trained with the whole dataset minus ten thousand promoters (randomly chosen) used for validation.

**2. Description of the model**

I used TensorFlow and Keras to develop a model with one branch. The model passes the inputs to 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 model was trained without dropouts.

**3. Training procedure**

The training of this model uses the BinaryCrossentropy loss coupled with the sigmoid activation of the output layer. The model scores are recorded after each epoch and reported in the jupyter notebook of the GitHub repository. I decided to use epoch 11 for submission by leaderboard probing.

**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 slightly better than the training metrics.
* The addition of dropouts at any level of the chosen architecture makes the model train slower with less competitive results. For this reason, the model was trained without dropouts.
* Until two weeks before the competition deadline, the model was very unstable, with nans appearing randomly in the evaluation metric during training. I figured out that the problem was the loss. The Keras MeanSquaredError or MeanAbsoluteError losses coupled with a liner activation of the output layer make the model unstable. The only way to train the model end-to-end is to use the Binary Crossentropy loss coupled with the sigmoid activation. As the sigmoid activation is bound from 0 to 1, it seemed logical to me to scale the target values between 0 and 1. The model instability made me train one epoch at a time, so I could re-start training from the last successful epoch in the case of nans. This is not necessary anymore, but I didn't change the training procedures. In fact, I found it useful to have the weights for all the epochs to see how they behave against the test set.
* Training the model after epoch 11 increases the validation scores but does not improve the leaderboard score. Increasing the number of promoters in the test set or adding dropouts to the model does not change this training behaviour. At the moment, I do not see any other method to decide when to stop training this architecture other than using the leaderboard feedback.
* I have been using the parameters epsilon=1e-6 and global\_clipnorm=1.0 in the Adam optimizer to prevent the nan problem during training. They didn’t prevent the nan issue, but the model performs worse without these parameters.

**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.

**7. Feedback (optional)**

I want to thank the organizer for setting up this interesting competition. It has been a very pleasant journey, with a lot of experience gained. Thanks especially to Abdul, that has been extremely helpful in the discussion forum. Possibly, requesting to make the models to run in colab, would make it easier to share and experiment with the code.