Learning the Language of the Genome using RNNs

Govinda M. Kamath
Department of Electrical Engineering
Stanford University
Stanford, CA 94305
gkamath@stanford.edu

Jesse M. Zhang
Department of Electrical Engineering
Stanford University
Stanford, CA 94305
jessez@stanford.edu

Abstract

We explore how deep recurrent neural network architectures can be used to capture the structure within a genetic sequence. We first confirm that a character-level RNN can capture the non-random parts of DNA by comparing the perplexity obtained after training on a real genome to that obtained after training on a random sequence of nucleotides. We then train a bidirectional character-level RNN to predict whether a given genomic sequence will interact with a variety of transcription factors, DNase I hypersensitive sites, and histone marks. Because multiple biological objects can interact with a given sequence, we cast this latter problem as a multitask learning problem.

1 Introduction

In recent years, deep recurrent neural networks (RNNs) have allowed researchers to tackle a variety of machine learning problems in the domain of natural language processing. Most of these applications investigate problems such as translation, named entity recognition, and sentiment analysis. Less work has been done with RNNs on what is perhaps the most natural language: the genome, a sequence of four letters (A, C, G, T). The purpose of this project is to explore how an RNN architecture can be used to learn sequential patterns in genomic sequences.

To confirm that an RNN can model the structure within a genomic sequence, we first train a simple character-level RNN to predict one of the four possible characters given the previous string of characters. If the ability of an RNN to predict the next character for a real genome is the same as for a random genome, then we will have to more carefully tweak our model until we can identify some signal. We will also use this simpler task to help choose a proper model architecture.

Once we have empirical evidence that an RNN can capture the non-random structure within a genome, we explore a sequence classification problem. Epigenetics is the study of how the genome is regulated by external factors. Biological experiments have shown that subsequences of the human genome are often regulated by both nearby and very distant sequences Specifically, biologists have been able to identify whether a particular genomic feature will be observed for a particular sequence. Examples of these features include

1. Transcription factors: proteins that bind to a particular sequence
2. DNase I hypersensitive sites: a sequence that is sensitive to cleavage by the DNase I enzyme
3. Histone marks: chemical modifications to histone proteins, biomolecules which regulate certain sequences.

These results are available via the ENCODE [2] and Roadmap Epigenomics [4] data releases. For each of many features, we will assess an RNN's ability to predict whether a given feature will be present based solely on the sequence.

2 Related work

Several deep genomic models have emerged in the past year, and the ones discussed below attempt to predict whether a certain genomic feature will be observed at a particular sequence.

Two tools were published in August 2015. DeepSEA [6] performs prediction based solely on a fixed-length genomic sequence. The authors use a deep convolutional neural network (CNN) over an input sequence, alternating between convolutional and pooling layers to extract sequence features. They use a sigmoid output layer to compute the probability of seeing a particular genomic feature. DeepBind [1] also uses a deep CNN with 4 stages of convolution, rectification, pooling, and some nonlinearity. Unlike for DeepSEA, DeepBind accommodates varying-length inputs and also incorporates extra information in addition to just the input sequence. This extra information comes from known facts about a particular sequence gathered from other experiments. Basset [3] was published a few months later in October 2015. This tool likewise uses a deep CNN architecture to learn the functional activity of genomics sequences.

DanQ [5], published in December 2015, incorporates a bidirectional long-short-term-memory (LSTM) RNN on top of the outputs of a max pooling layer after convolution on the input sequence.

Each of these tools feeds a one-hot encoding of an input sequence into a convolution layer. In other words, the networks operate on a $4 \times L$ matrix obtained from a length-L sequence.

3 Approach

All RNN models in this project were built using Google's `TensorFlow` Python package. For the character-level genome modeling task, we tested a variety of RNN-based neural network architectures including gated recurrent units (GRUs) and long-short-term-memory RNNs (LSTMs). We experimented with a variety of hyperparameters on multiple genomes in order to assess the robustness of a given model. We will measure the ability of a model to fit a given dataset using the average perplexity across all character prediction tasks:

$$PP(\mathbf{y}, \hat{\mathbf{y}}) = \exp\left\{-\frac{1}{n-1}\sum_{t=1}^{n-1}\sum_{i=1}^{|V|} y_i^{(t)} \log \hat{y}_i^{(t)}\right\}$$

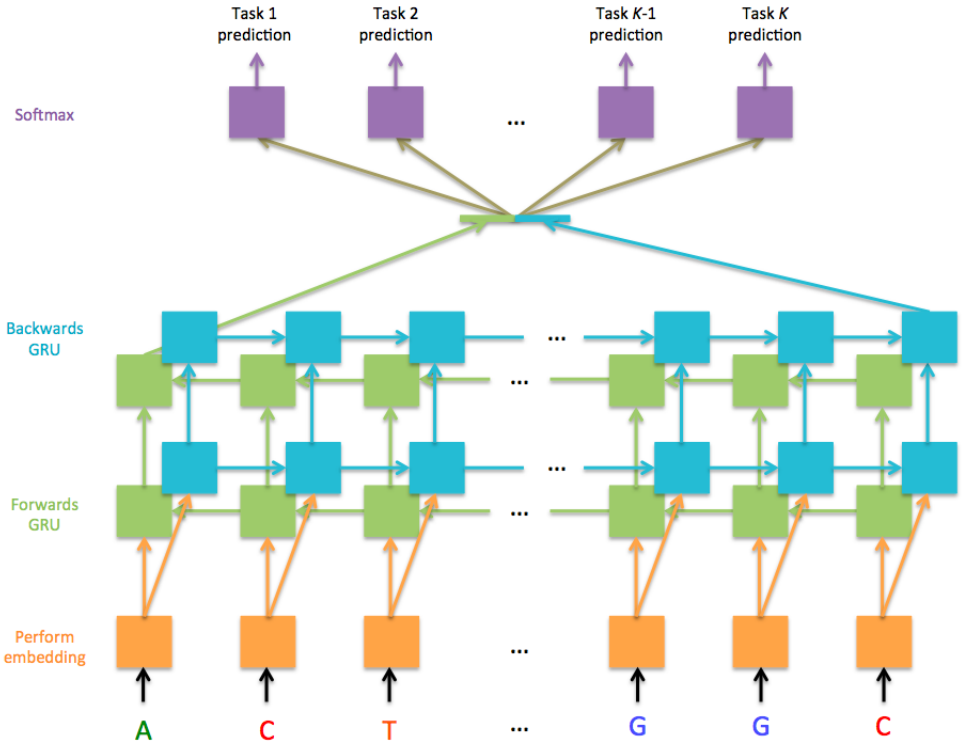
where n is the number of training samples (roughly the length of our genome), $|V|$ is the size of our vocabulary, $\hat{y}_i^{(t)}$ is the predicted probability of the predicted word at time t being word i, and $\mathbf{y}^{(t)}$ is a one-hot vector describing the actual word at time t.

From the above experiment, we selected a cell type to use for the multitask learning model. We used a bidirectional GRU for character-level processing. Each character in the input sequence is mapped to some low-dimensional embedding. Intuitively, a genetic sequences can be regulated by both flanking sequences, and therefore bidirectionality is a crucial part of the architecture. The outputs of the forward and backward RNNs are fed through a softmax layer. For each of K tasks, a prediction of 0 or 1 is made, indicating whether or not a particular feature should be observed for the input sequence. For all tasks, all weights are shared before the final softmax. Each task has its own weight and bias matrices at the softmax layer. The overall architecture is summarized in Figure 1. The overall loss is the geometric average perplexity across all K tasks.

4 Experiment

Experimentation was performed by first gathering datasets of synthetic genomes, real genomes, and sequence-label training examples. Due to the size of the training set for the multitask prediction problem, we used a simpler problem (genome character-level prediction) to choose certain hyperparameters. Our evaluation metric was based on the perplexity cost function above for non-classification tasks and an F1 score for classification tasks. For the multitask prediction problem, we implemented a baseline that involved both feature extraction and logistic regression.

Figure 1: The network architecture used to perform multitask prediction for a given input genomic sequence. Each character in the input sequence is mapped to a low-dimensional embedding, which is fed into a bi-directional recurrent neural network consisting of gated recurrent units. The outputs of the last GRU in the forwards and backwards RNNs are concatenated before being fed to one of K prediction tasks. A final prediction of 1 or 0 is made for each task.



4.1 Dataset

For the character-level genome prediction task, four genomes were prepared:

1. A length-30,000 repeating genome where the repeated unit is `AGCTTGAGGC`
2. A length-30,000 random genome
3. The length-4,639,675 genome for *E. coli*
4. The length-23,264,338 genome for malaria

For the multitask prediction task, we used a random subset of the dataset used in the DanQ [5] and DeepSEA [6] papers. The dataset was collected from experiments reported in the ENCODE [2] and Roadmap Epigenomics [4] data releases. We chose 80,000 sequence-label pairs for the training dataset, 8000 pairs for the testing dataset, and 2000 for the validation dataset. Each sequence has a length of exactly 1000. All sequences are length-200 subsequences of the human genome with 400 bp flanking regions. The length-200 portion is nonoverlapping across different sequences. The set of labels describes 919 binary prediction tasks, and each indicates whether part of a particular sequence was observed as accessible for a particular ChIP-seq or DNase-seq experiment (i.e. if a ChIP-seq or DNase-seq peak for a particular experiment overlaps with the 200-bp region by at least half of the length of the peak). Both of these types of experiments evaluate the locations where one of the three features (see introduction) can bind. The dataset is noisy, and about 10% of the tasks' labels were all 0. In general, tasks are strongly skewed towards negative examples. For 421 of the 919 tasks, less than 1% of the training examples were labeled 1.

Due to computational limitations, training an RNN on all 1000 characters in each example was difficult. The RNNs discussed below were instead trained on the middle 100 characters of each sequence. For a fair comparison, the baseline model was trained on the same set of truncated sequences.

4.2 Character-Level Genome Prediction Results

For this task, our baselines were the artificial genomes (first two genomes in the list). Any model that does not perform as expected on these two genomes will not be used on other genomes. We wanted to perform better than the random genome, and the repeating genome serves as an empirical lower bound. For these genomes, we used an LSTM architecture, a dropout keep rate of 1, a batch size of 50, a sequence length of 50, a learning rate of 0.002, and 10 epochs. The achieved perplexity of the random genome was 4.003, just as expected. This is the case when the model always assigns equal probability to each of the 4 possible character A, C, G, T. The achieved perplexity of repeating genome is 1.002, which was also just as expected. Since the sequence repeats without noise, we expect the model to be able to put all the probability mass on a single character.

For the *E. coli* and malaria genomes, we used a dropout keep rate of 1, a batch size of 50, and 1 epoch. We tested the GRU, LSTM, and RNN architectures, learning rates of 0.0001 and 0.0008, sequence lengths of 50, 100, 500, and 1000, and 2 and 3 layers. The number of epochs was limited to 1 in order to quickly test every combination of the above hyperparameters. For both genomes, all combinations of hyperparameters performed about the same, reaching a perplexity of 3.679 for *E. coli* and a perplexity of 2.938 for malaria. All training errors were continuing to decrease after 1 epoch, however, indicating that increasing the epochs will allow for even smaller perplexities (Figures 1 and 2).

The learning rate of 0.0001 performed worse than the learning rate of 0.0008, but this is likely because the former is slower at updating. Both 2 and 3 layers worked about the same. Surprisingly, 50 or 100-length sequences perform better than 500 or 1000-length sequences despite the fact that genomes often involve long-range interactions. Finally, both RNNs and GRUs appear to outperform LSTMs.

While the perplexity being less than random is reassuring, the results obtained here are not useful by themselves. Genomes are known to have repeat structures in certain regions. This and the fact that that there existed an unequal distribution of the four nucleotides can produce a lower perplexity. Nevertheless, these experiments showed that a 2 layer network using GRUs may be appropriate for modeling a genomic sequence. These will be used for training a model to handle the primary problem in this project: multitask prediction.

4.3 Multitask Prediction Baseline and Evaluation Metric

Just as in [5], a baseline was drawn by first mapping each sequence in the training set to a feature vector and then running the vector-label pairs through a simple logistic regression model. The first feature mapping used involving mapping each of the 4 nucleotides to an integer 0, 1, 2, or 3. As expected, this feature mapping performed quite poorly as the mapping fails to capture how adjacent characters interact.

Instead, we used a feature mapping inspired by the authors of [5]. We used a k-mer bag of words method where each length k subsequence of an input sequence was counted. We used $k = 1, 2, 3, 4, 5$, resulting in $4^1 + 4^2 + 4^3 + 4^4 + 4^5 = 1364$ features. For example, the sequence `ACTGG` would produce a length-1364 feature vector where the entries corresponding to the k-mers `A`, `C`, `T`, `AC`, `CT`, `TG`, `GG`, `ACT`, `CTG`, `TGG`, `ACTG`, `CTGG`, and `ACTGG` would each equal 1, and the entry corresponding to `G` would equal 2. All other entries equal 0.

Logistic regression was performed using the Python module `scikit-learn`. Default parameters were used except for the regularization constant of the ℓ_2 regularization term, which was set to 10^{-6}. A separate logistic regression model was trained for each of the 919 tasks on all 80,000 training examples (with length-100 sequences). The performance of a model was evaluated based on the its prediction accuracy on the 8000 training examples. Note that a validation set was not used here; each logistic regression model was simply trained until the default tolerance of 0.0001 was reached. Because all examples are severely skewed towards the 0 class (i.e. for all tasks, there are significantly more examples without the feature than with the feature), the raw error rate is not a

good indicator of prediction accuracy. We instead looked at the F1 score, defined as:

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

where precision is the proportion of correct positive predictions, and recall is the proportion of positive examples that received a positive prediction.

To account for the fact that less than 1% of the training examples were labeled 1, we looked only at the 100 tasks where logistic regression performed the best and where at least 1% of the training examples were labeled 1.

4.4 Multitask Prediction Results

The final model used a learning rate of 0.001 for an Adam Optimizer, 2 layers, an embedding size of 2 (as there were only 4 unique characters), hidden layer sizes of 128, a dropout keep probability of 0.95, and batch size of 20. The RNN was bidirectional and used GRU units with tanh activation function. The RNN processed all 100 characters of each input sequence. The model was trained for 61 epochs on the 80,000 training examples, running for approximately 3 days using an NVIDIA GRID K520 GPU on an Amazon Web Services Elastic Compute Cloud instance.

5 Conclusion

References

[1] Babak Alipanahi, Andrew Delong, Matthew T Weirauch, and Brendan J Frey. Predicting the sequence specificities of dna-and rna-binding proteins by deep learning. *Nature biotechnology*, 2015.

[2] ENCODE Project Consortium et al. An integrated encyclopedia of dna elements in the human genome. *Nature*, 489(7414):57–74, 2012.

[3] David R Kelley, Jasper Snoek, and John Rinn. Basset: Learning the regulatory code of the accessible genome with deep convolutional neural networks. *bioRxiv*, page 028399, 2015.

[4] Anshul Kundaje, Wouter Meuleman, Jason Ernst, Misha Bilenky, Angela Yen, Alireza Heravi-Moussavi, Pouya Kheradpour, Zhizhuo Zhang, Jianrong Wang, Michael J Ziller, et al. Integrative analysis of 111 reference human epigenomes. *Nature*, 518(7539):317–330, 2015.

[5] Daniel Quang and Xiaohui Xie. Danq: a hybrid convolutional and recurrent deep neural network for quantifying the function of dna sequences. *bioRxiv*, page 032821, 2015.

[6] Jian Zhou and Olga G Troyanskaya. Predicting effects of noncoding variants with deep learning-based sequence model. *Nature methods*, 12(10):931–934, 2015.