BioQ: a reimplementation of DanQ Hybrid Deep Neural Network for DNA sequence function prediction

Carter Conboy and Yeshwanth Konka

Department of Electrical and Computer Engineering, University of Victoria, P.O. Box 1700 STN CSC Victoria, B.C., V8W 2Y2, Canada

ABSTRACT

DanQ is a hybrid convolutional and recurrent deep neural network which predicts non-coding function of DNA sequences. The model was developed in 2016 and achieved considerable performance improvements to comparable solutions. In this paper we report on a re-implementation of the DanQ model: BioQ, which applies the original paper's methodology (model architecture and training procedure) with up-to-date machine learning frameworks. BioQ achieved similar results to the original paper and introduced nice-to-have usability features. Implementation details are available on our Github.

INTRODUCTION

As one of the first methods to apply Recurrent Neural Networks to DNA sequence processing, DanQ made important progress in the field of bio-informatics. Currently the model's code base is open source and available on the DanQ Github, however, its reliance on legacy packages and their cascading dependencies severely compromise its usability. This project implements DanQ using modern supported frameworks, making it easily usable. We also make changes to memory access allowing for more usability with low memory systems, as well as updating training hyper-parameters to boost performance. Finally, we test BioQ against the same metrics as DanQ and compare the results.

The ability to predict the function of non-coding DNA has significant scientific benefits, as it makes up over 98% of the human genome and contains 93% of disease associated variants (1). Deep Neural Networks (DNN) are effective at learning high levels of abstraction from large feature rich datasets, making them an appropriate solution for direct sequence processing. The ability to directly process sequences is important for revealing novel insights about non-coding DNA whose function is not yet defined. Convolutional Neural Networks (CNN), use convolutional layers to capture local patterns in data. The application of CNNs to DNA allows the convolutional filters to capture "sequence motifs", which are short recurring patterns linked to biological function. Recurrent Neural Networks (RNN) are deep neural networks with cyclic components designed to process spatial structures of sequential data. A bi-directional long short-term memory network (BLSTM) is a variant of RNN used to learn long term dependencies. The hybrid framework combines CNN and BLSTM models, first using the CNN to extract motifs, which are used by the BLSTM to learn the structural patterns these motifs follow (presumed to be a regulatory grammar governed by physical constraints).

MATERIALS AND METHODS

Model Architecture

The model begins with a convolutional layer that applies 320 filters, each of size 26, to scan the input DNA sequences encoded as one-hot vectors of size 4 (A, C, G, T). The output of the convolution is passed through a ReLU activation function and down-sampled using a MaxPooling operation with a stride and kernel size of 13 to reduce spatial dimensions. To prevent over-fitting, a Dropout layer (p=0.2) follows. The output is then fed into a Bidirectional LSTM (Long Short-Term Memory) network with 2 layers, each having 320 units in both forward and backward directions (640 units in total for bi-directionality). This recurrent architecture captures long-range dependencies in the sequence data. Following the LSTM, another Dropout layer (p=0.5) is added for regularization. The LSTM outputs are flattened and passed through a series of fully connected layers, starting with a linear layer of 925 neurons (ReLU-activated) and another with 919 neurons, corresponding to the number of functional annotations being predicted. The model concludes with a Sigmoid activation function, converting outputs to probabilities.

The model contains approximately 49.4 million trainable parameters, distributed among the convolutional, recurrent, and dense layers as shown in Figure 1.

Dataset

The dataset is sourced from ENCODE (Encyclopedia of DNA Elements). It contains labeled DNA sequences annotated for functional properties such as transcription factor binding and histone modifications.

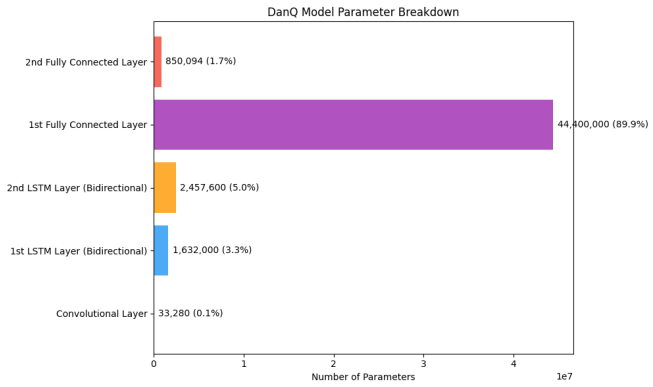

Figure 1. Model weights/parameters over different layer

The DNA sequences are of length 1,000, with four channels for each one-hot encoded base (shape: 4 × 1,000). A binary label vector of length 919, indicates the presence or absence of specific functional annotations in a given sequence. In total, the training dataset is 3.3GB and contains 4,400,000 samples.

Implementation and Training

The model was implemented in Python using Pytorch (a popular machine learning library), and training was handled on the CUDA platform with an NVIDIA RTX 3070 GPU.

The network was trained with the RMSprop algorithm with a mini batch of size 100 and max norm gradient clipping was used to prevent exploding gradients. Further training hyper-parameters are shown in Table 1. Training time was approximately 1 hour per epoch.

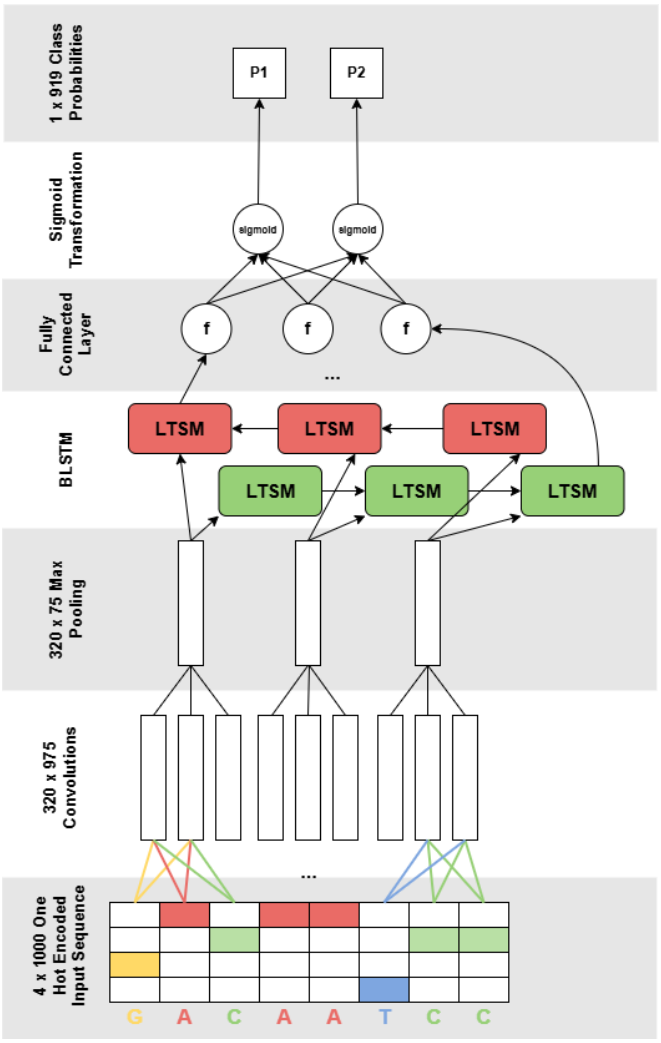

Figure 2. High Level Diagram of Hybrid Model Architecture

RESULTS

Figure 3 displays loss per epoch, demonstrating expected learning behavior of steep initial loss reduction flattening out as the optimization converges after 55 epochs.

Model performance is measured with the area under receiver operating characteristics curve (ROC AUC) and area under precision recall curve (PR AUC) averaged across all 919 targets. The PR AUC curve is in this case a better metric, as it accounts for true negatives, making it less sensitive to the class imbalance present in the data set (high sparsity of positive targets).

Average model accuracy is displayed in Table 2. An initial prediction threshold of 0.5 was used for comparison with the original paper, and an optimal threshold of 0.215 was derived from the PR curve. The optimal prediction threshold resulted in a marginal accuracy decrease with a substantial increase in F1 score.

Table 1. Training Hyper-Parameters

Parameter	Value
Epochs	55
Learning Rate	1e-5
Weight Initialization	PyTorch Default
Loss Function	Binary Cross Entropy

Table 2. Accuracy and F1 Score Results

Threshold	Mean Accuracy	F1 Score
0.5	98.2%	0.281
0.215	97.6%	0.404

Optimized Threshold Greatly Increases F1 Score

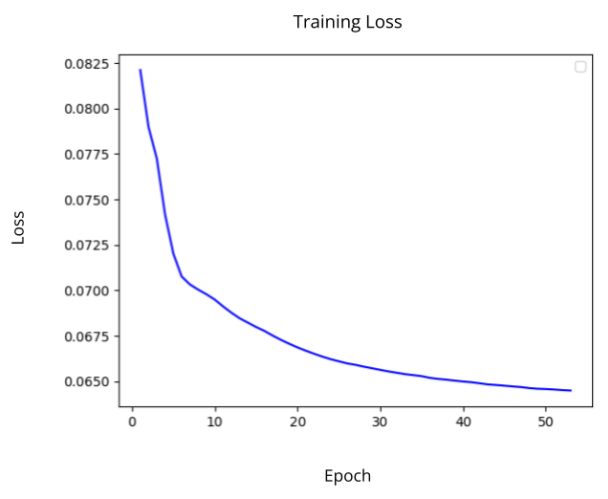

Figure 3. BCE Loss by Training Epoch

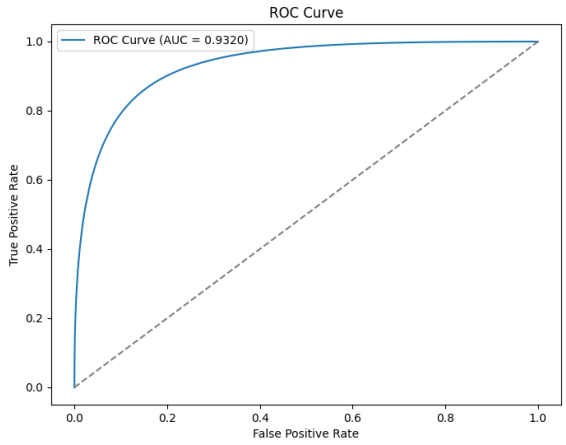

Figure 4. Area under receiver operating characteristics curve: 0.9320

DISCUSSION

BioQ was able to closely replicate DanQ's results with mean accuracy of 98.2% compared to DanQ's 98.24%. The ROC AUC and PR AUC scores are both within the range of results demonstrated in the DanQ paper (0.927 to 0.972 for ROC AUC and 0.291 to 0.469 for PR AUC). The primary diversions from the original paper are training hyper-parameters and the decision not to implement the DanQ-JASPAR model. The DanQ-JASPAR model uses larger convolutional layers and known motifs for weight initialization. The expanded model size would not have fit into our available GPU memory making training infeasible. Learning rate, gradient clipping and training epochs were selected heuristically due to lack of information in the original paper and unknown default settings of deprecated packages. The main contributions BioQ adds are code usability and batch memory access. Our implementation

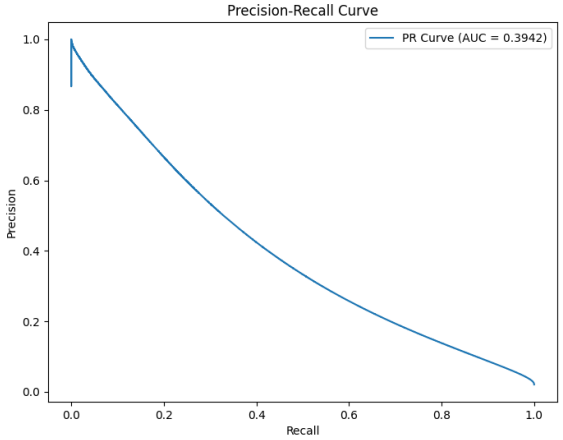

Figure 5. Area under precision recall curve: 0.3942

is built on recent and supported releases of Python (3.12.7), PyTorch (2.5.1), and CUDA (12.6.3) making it trivial to download and train the model. We also expect that these modern packages greatly reduce training time, though it is difficult to compare across hardware. Additionally, the use of batch memory access facilitates training on lower memory systems. Next steps include implementing motif based weight initialization, tracking validation scores during training, and exploring how the last 8 years of advances in machine learning model architecture, such as transformers, could be applied to this problem.

ACKNOWLEDGEMENTS

We gratefully acknowledge the authors of the original DanQ paper Daniel Quang and Xiaohui Xie, as well as Proffesor Ibrahim Numanagić for his teaching and mentorship.

REFERENCES

1. Author,Daniel Quang and Author,Xiaohui Xie. (2015) Article title. *DanQ: a hybrid convolutional and recurrent deep neural network for quantifying the function of DNA sequences*. https://www.biorxiv.org/content/10.1101/032821v1.full
2. Author, Jason Brownlee (2023) *How to Use ROC Curves and Precision-Recall Curves for Classification in Python*. In Machine Learning Mastery.
3. *Pytorch:* https://pytorch.org/docs/stable/index.html
4. *CUDA for GPU usage:* https://docs.nvidia.com/cuda
5. *Mathplotlib Plotting results:* https://matplotlib.org/stable/users/explain/quick_start.html