Capstone Project

Rubén Chaves Moreno

Machine Learning Engineer Nanodegree

January 31th, 2020

Classifying non-coding DNA sequences

Definition

Project Overview:

Genomics, as other life sciences like biology and medicine, is generating big amounts of data. As prices of whole genome sequencing keeps to going down (about $700 for commercial users right now, $100 predicted for 2020), genomics becomes more and more data-driven and it aims to be one of the big contenders of big data[1]. However, to extract the real value, acquiring the data is only the first step; we still have to storage it, distribute it and analyze it.

For this last step, there have been numerous studies since the first full sequencing of the human genome in the Human Genome Project (HGP) where it was estimated that only 1.5% of it, around 20,000 genes, are coding genes. The 98.5% remaining, firstly known as 'junk DNA', as been subject of study in projects like the continuation of the HGP, the Encyclopedia of DNA Elements (ENCODE), where it was found that at least 80% of the DNA is functional through mechanisms as promoters, enhancers, regulatory RNA or chromatin formation regions.

Over 150 genome-wide association studies with 500 trait/disease-associated single-nucleotide polymorphisms (SNPs) located 88% of them in non-coding regions [6], revealing a greater need for predictive algorithms focused on these parts of the genome.

For this case, the use of data driven algorithms, a field known as machine learning, is a perfect fit. While many traditional approaches have been extensively used in bioinformatics, deep learning, a new family of algorithms that have given great results in other fields like computer vision, robotics and many others, is an ideal tool for genomics due to lots of data available, a big and complex problem and no hand crafted features.

We will leverage the advances of a special subset of deep learning algorithms specialized on sequences of words, field known as natural language processing (NLP), with powerful tools that easily adapt to our problem: a dataset from the DeepSEA framework (download link), a dataset of the GRCh37 reference genome with targets extracted from ChIp-seq and Dnase-seq peak sets from the ENCODE and Roadmap Epigenomics data.

Problem Statement:

The challenge consists in building a multi-task classifier (many independent targets) for non-coding sequences. Since the input is only formed by the four nucleotide bases of the DNA (A, T, C, G) and the targets are on the hundreds, we'll need a powerful algorithm able to create rich internal representations. Currently, the best NLP deep models are based on the original transformer [8] extending it with improvement like the Transformer XL [9], adapted for longer sequences and which we'll use for our

genomics data. Additionally, we'll augment the input with convolution layers to enrich and reduce the length of the input without losing information.

The main library used will be PyTorch, a deep learning framework that support most of the latest advances giving the user flexibility to make its custom model without sacrificing code readability nor speed and supporting hardware accelerators such as GPUs and half-precision floating-point (FP16). The steps taken will be:

1. Data exploration: looking for the distribution of the target to see how much our data is imbalanced, the total number of positive vs negatives and the distribution of each label, looking if there is any category without positive labels and the percentage of each base.
2. Data preprocessing: the data is already processed and ready to train, the only things to do would be preparing the batching policy and transforming one hot encoding to class labels if we are using embeddings.
3. Model design: code our different models implementations that we'll briefly compare doing short training loops. The main ideas to try are:
    - Hot encoded vs embedding.
    - Convolution 1D, ResNet before the transformer.
    - Implementation of Transformer-XL.
    - Attention head with or w/o intra convolution.
4. Training: using the training and validation test.
    - Compare the different model by data efficiency (loss/epoch) and speed (loss/training time).
    - Hyperparameters tuning: after the try-outs of the models, we'll dive into one model and try to search the hyperparameters to speed up the training and maximize our metrics and improve the generalization capabilities of the network.
5. Evaluation on the test set and comparing to the benchmark models.
6. Model interpretation: deep learning models are usually seen as black box algorithms, but we'll look for patterns on the activations of layers and kernels looking for common motifs to get insights of how non-coding sequences work.

Metrics:

The dataset has a low number of positive targets, making metrics like accuracy bad for our purpose. We'll focus on:

- ROC AUC: area under the true positive rate (TPR) vs false positive rate (FPR). Bad for our imbalanced data because positive targets are sparse, an algorithm only predicting negative would perform well. We use it to compare with others models.
- PR AUC: precision (PPV) vs recall (TPR). There is no focus on the negative class making it better for imbalanced cases.

$$TPR = \frac{TP}{TP + FN} \qquad FPR = \frac{FP}{FP + TN} \qquad PPV = \frac{TP}{TP + FP}$$

The metrics are calculated for each label and then we aggregate them using a weighted average where weights are the percentages of existence of each corresponding target in the training set.

Analysis

Data Exploration

The data consist of 1,000-bp fragments, with 919 binary targets for TF binding (104), DNase I sensitivity (125) and histone-mark profiles (690). Each sequence is centered around a 200-bp bin that overlaps at least one target but as reported in [3] 10% of the target vectors are all negative.

Each of the four bases (A, G, C and T) is represented in the input with a dimension of a one-hot encoded vector and as expected each one is present with a frequency between 20 and 25%.

The data is divided in training set (4,400,000 samples), validation set (8,000 samples), and testing set (455,024 samples). Due to the small size of the validation size, this dataset may not be representative (34 targets are not even present) and we have to be careful not to overfit to this while choosing our hyperparameters. We could take more data from the training set but we don't do it for the sake of fair comparison with other algorithms.

As illustrated in [4], it is a highly unbalanced dataset with the majority of targets observed in less than 5% of the training samples. There is not only class imbalance, but the number of negative classes outnumber the positive labels by a factor of 45, making the data 98% negative.

Exploratory Visualization

Next we show some of the problems discussed before. In Figure 1 we observe the counting of each class. We can differentiate three groups by its count that correspond with the three types of label of the datasets: first DNAases, second transcription factors and third the histones marks. After sorting this data, we notice the imbalance between classes. For the validation set, we have the same distribution but lacking some of the less common targets.

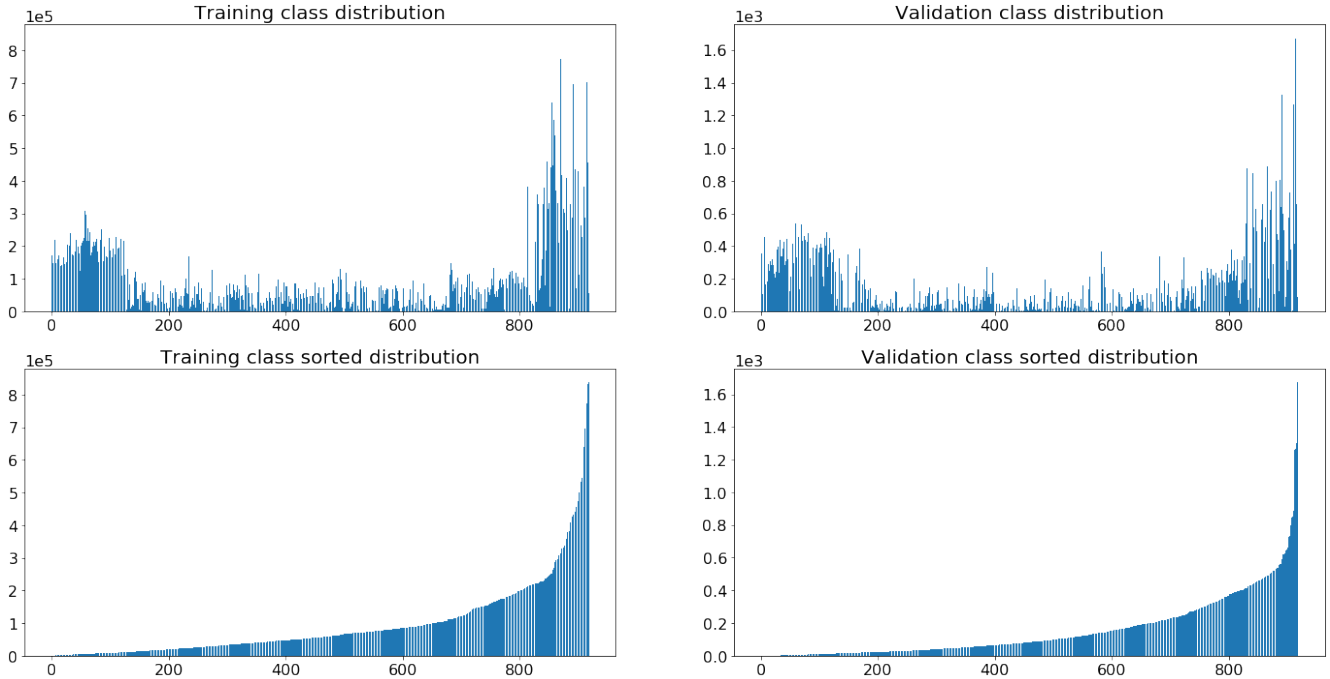

Fig 1. Class distribution on training and validation set by class index and sorted. Axes: count vs index

Calculating the frequency in which the targets are present on every sample of the training set on a histogram, we discover another type of imbalance between positives and negatives. As we observe in Fig. 2a most targets are in the less frequent bins, with the first bin having the highest count. In fact, 90% of the targets are present in less than 5% of the samples.

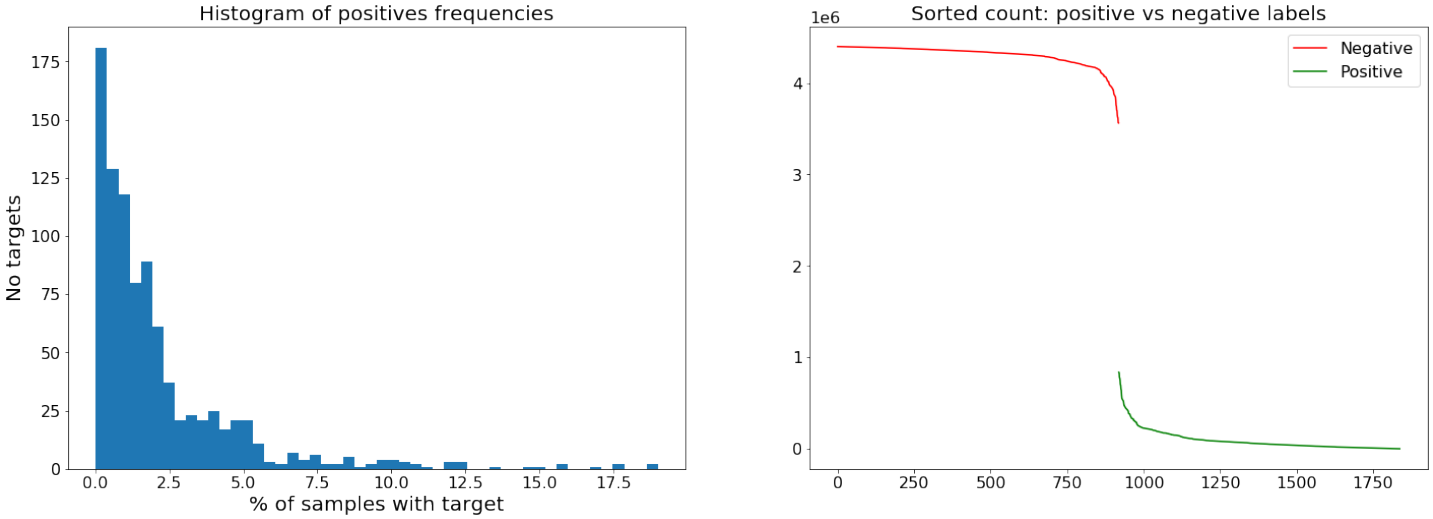

Fig 2. a) Histogram. b) Count.

Finally, we visualize the size of the issue in Fig. 2B, the most frequent positive class is less common than the less common negative class.

Algorithms and Techniques

We'll combine different techniques and types of deep learning architectures to try to reach our objective:

1. Embeddings[11]: a NLP technique to represent words on a numerical and high dimensional space that can be learned. Each base is transformed to a n dimensional vector increasing the number of features of the input.
2. WideResNet 1D: ResNet [12] is a type of convolutional architecture from computer vision which introduced skip-connections between layers wit the purpose of letting the gradients of backpropagation flow more easily. Convolution can be used for sequences with a one dimension convolution (in our case, each of the embedding dimensions), leveraging its parameter efficiency and good local connectivity, reasons why it has been used in NLP and genomics. Furthermore, we can reduce the length of the sequence by aggregating contiguous elements without losing information thanks to an increasing of features dimensions. Shortening the length will save us computation and lower the memory requirements on the next steps. We also increase the number features on the 3x3 convolution by a expansion factor as in Wide ResNets [18].
3. Transformer-XL[13]: transformer [14] based models are responsible for the latest improvement on NLP datasets. They are composed by attention layers, which main components are the three matrices called key, query and value, representing the words of the input, which are multiplied between them creating rich relational information. To add to these, Transformer-XL enables learning dependency beyond a fixed, making it better for long sequences as in our case.

4. Fully connected: the last part of the network will be a couple of linear layers that will take the output of the transformer directly and the output of the ResNet through a skip-connection and convert it to the multiple labels of the target.

It is important to take in consideration the number of parameters of the model for fair compassion with other models. For reference, the only model available, DanQ, has 47m parameters. The final hyperparameters and total number of weights are represented in Table 1.

Part	Hyperparameters	No parameters
Embedding	Embedding dimensions = 64	256
ResNet	No blocks = 3 Expansion factor = 16	4710k
Transformer XL	No layers = 6 No heads = 4 Head dim. =16 Inner dim. = 256 Hidden dim. = 256	1290k
Fully connected	Hidden dimension = 512	33240k
		39242k

Table 1. Model summary.

The model is trained with Adam with a learning rate of 1e-4, with batches of size 25, accumulating the gradient for four batches, achieving an effective batch size of 100. For a max of 60 epochs or until overfit occurs (the validation start to increase, while the training loss keeps decreasing).

Benchmark:

The dataset was originally used in DeepSEA[2] and the used in the DanQ[3] and NCNet[4] papers. Metrics are summarized next:

	Accuracy	ROC AUC	PR AUC
DeepSEA	98.21%	0.9046	0.4463
DanQ	98.24%	0.9109	0.4698
NCNet-bRR*	98.35%	0.9441	0.5358
NCNet-RbR*	98.36%	0.9507	0.5519

*Ncnet data is reported with relative values compared to an reimplementation of DanQ(r-DanQ), but they don't give any absolute value from either NCNets nor r-DanQ.

Methodology

Data Preprocessing

The dataset is already nicely processed, the only change we make is converting the one encoding on a vector of integers that represent each class to feed it to our embedding matrix.

Implementation

The main training loop is coded using PyTorch Lightning, a wrap on top of PyTorch designed to help you write less boilerplate code and improve the reproducibility of deep learning experiments by setting a template of common functions of the training loop like data loading, training steps, optimizer steps, etc., and letting you easily change what you need, making easy to understand other peoples code or making variations of your experiments without changing everything. Furthermore, it makes easier little things like checkpointing, logging, graddient accumulation and to train in FP16, parallel or in GPU. We'll make use of this last on to train on GPU to speed the training.

We batch the data with PyTorch dataloaders putting the fetched data Tensors in pinned memory, and thus enabling faster data transfer to CUDA-enabled GPUs. Using a number of workers different than 0 (only the main process) caused slower data loading, but this problem didn't mean much because the process was already fast since there was no data processing involved.

The activation function used in the model is ReLU, followed by batch norm and on some layers of dropout on the transformer and linear parts. Since we are using batch norm, we remove the bias from convolution and linear layers to avoid redundant parameters.

The ResNet part use convolutions with stride of two instead of pooling layers as mean of dimensional reduction as in the original implementation. We implement the bottleneck version of the residual blocks, with the pre-activation variation where the identity is added before the activation.

For the Transformer-XL, we use the implementation from the Hugginface's transformer library. We utilize the form without the language model head, with the only complication of FP16 not working. This is the part of the network that takes more time, so we keep it small.

The final classifier head consists on two linear layers, where the first is connected to the output of the transformer and also to the output of the ResNet with a skipped connection. These outputs are concatenated instead of added like in the original residual blocks, which gives better results but consume more memory. With the imbalance of the dataset in mind, we add Weight Vector Normalization (WVN) [16] to the last layer and initialize its bias according to [17] to take in consideration the fact that most labels are negative.

Our loss function (Binary Cross Entropy) takes as input the logits of the last layer for numerical stability reasons, meaning the last layer have no activation function. Nevertheless, metrics are calculated with the output of a sigmoid, better coupled for multi-task problems than softmax.

To calculate the AUC ROC we use an implementation based on the one from fastai, instead of sklearn, as the former one is faster and can be accelerated by GPU. We modify it to also calculate the PR ROC. We also take care for the targets missing on validation and test sets producing NaNs.

Refinement
The initial solution consisted of the ResNet based part with bottleneck blocks, without pre-activation, skip-connections to the linear layers nor expansion factor. The hyperparameters for the transformer don't change much since the trials where we vary them didn't show much improvement.

Due to problems with the implementation, increasing the batch size slowed the time for epoch a lot. For most of the model and hyperparameters explorations, a batch size of 32 is used, until the need for larger batches to counter the fact that there is little positive labels made us change to gradient accumulation of 4 epoch of size 25 to match the size of 100 used in other models for this dataset.

Apart from Adam, other optimization algorithms tried are AdamW, RMSprop and SGD, with different learning rates from 1e-3 to 1e-5 and cyclical learning rates. As methods for regularization, weight decay (1e-2 to 1e-6) and momentum (0.5 to 0.9) are tested, without improvement to the evaluation metrics, in contrast with low-mid values of dropout (0.2-0.5).

Independently of the hyperparameters, loss at the start of every training experiment started really high in comparison to the rest of the training. These high losses can make the parameters take bigger update steps, entering a bad space where takes long to get out. To solve this issue, we add a learning rate warm-up period where it linearly increase for 500 batches until our assigned initial value. On the same line, we initialize the bias of the last linear layer to match the distribution of positive vs negative labels as in [16]. WNM is tested with no luck.

Improvements started to appear when introducing some changes to the structure of the model: implementing the pre-activation version of ResNet and skip-conection between convolution and linear layers, specially when followed with batch norm, otherwise the output of the convolutions and the transformer would be on different scales (transformer-XL was using batch norm already).

Realizing the importance of the residual blocks, and how little parameters they took (94k of tens of millions from the total model), the next effective improvement was to make it bigger by adding an expansion factor as in Wide ResNets [18].

WIP: After all, roc saturate on 0.89. transformer is not magical, alone did not much, lots of little changes. last lstm, with fp16 bidirectional, better results.

Results

Model Evaluation and Validation
WIP: Training vs validation loss.
WIP: Test ROC AUC, PR ROC, acc.
WIP: https://github.com/HaebinShin/grad-cam-text

Justification
WIP: Comparison with benchmark models
WIP: NCNet do not show absolute values so we don't know if we are really doing better or worse than them.

References

[1] Big Data: Astronomical or Genomical. Link
[2] Predicting effects of noncoding variants with deep learning–based sequence model. Link Code
[3] DanQ: a hybrid convolutional and recurrent deep neural network for quantifying the function of DNA sequences Link Code
[4] NCNet: Deep Learning Network Models for Predicting Function of Non-coding DNA. Link
[5] Opportunities and obstacles for deep learning in biology and medicine. Link
[6] Potential etiologic and functional implications of genome-wide association loci for human diseases and traits. Link
[7] Novel transformer networks for improved sequence labeling in genomics. Link
[8] Attention Is All You Need. Link
[9] Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context. Link
[10] PyTorch: An Imperative Style, High-Performance Deep Learning Library. Link Web
[11] Distributed Representations of Words and Phrases and their Compositionality Link
[12] Deep Residual Learning for Image Recognition Link
[13] Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context. Link
[14] Attention Is All You Need Link
[15] Identity Mappings in Deep Residual Networks Link
[16] Adjusting Decision Boundary for Class Imbalanced Learning Link
[17] Class-Balanced Loss Based on Effective Number of Samples. Link
[18] Wide Residual Networks. Link