DRASTIC: A FULL DEEP LEARNING BACTERIAL GENOME ANNOTATOR

Jorge Carrasco Muriel, Arianna Taormina, Matias Tofteby Bach Andersen

S192528, S163671, S145152

ABSTRACT

Since the success of the Human Genome Project, sequencing services have become cheap enough to turn the process of extracting the DNA of an organism into a routine task. Thus, the number of sequenced genomes has grown exponentially since the beginning of the century. This knowledge has improved our understanding of biology, leading to important theoretical insights and biotechnological applications in fields such as drug discovery, industrial biological production and genetics.

Today's bottleneck is to define the function of every piece of information in the genomes. This task has been historically tackled with costly experimental assays and, later, with the emergence of Bioinformatics, with homology-based heuristics algorithms. However, these heuristic methods are limited and error prone.

For this project, given the current availability of public data and the similarity with Natural Language Processing tasks, we propose a Deep Learning LSTM genome annotator to takle this challenge.

Index Terms— Genome annotation, genome sequencing.

1. INTRODUCTION

1.1. The problem of genome annotation

Genomes are the hereditary material of organisms. Encoded by four molecules of DNA - nucleotides - as building blocks [1], they contain the information to perform biological functions, whose corresponding particular sequence is called a "gene".

The cellular machinery is capable of recognising different patterns in the genome to produce the gene products that ultimately performs the different biochemical function. This cellular machinery is composed of cellular agents - mainly proteins called Transcription Factors (TFs) - that diferentially binds to motifs across the genome. Once they are attached to a domain, by using a diverse range of biochemical mechanisms, the TFs can attract or hamper the tool to transcribe the genes so they can develop their function: the RNA polymerase.

Since the first characterization of a sorted sequence of DNA in 1975, progressively more efficient and accurate technologies have been developed [2]. This fact has allowed an exponential growth in the number of sequenced genomes since the start of the millennium. Rather than data gathering, the bottleneck in genomics is annotation: the process of identifying genes and its function inside a genome.

1.2. Classical approaches and machine learning

The aforementioned exponential explosion on genome sequencing was driven by the Human Genome Project [3]. Initially based on an international experimental laboratory workhorse, the final success of the Human Genome Project was the introduction of Bioinformatics, that would later speed up the sequencing of a plethora of genomes of different species to the point of doing genetic mappings of entire human populations, such as the UK Biobank in 2019 [4]. This discipline was firmly built upon homology methods, heuristics that exploit similarity of previous annotations with new sequences.

Bioinformatics has now matured into a discipline with both accessible homology annotation workflows [5] and the use of different machine learning tools, with especial attention to the early adoption of Hidden Markov Models [6] [7].

1.3. Previous work

With the recent advances of Deep Learning in Natural Language Processing, it was only natural to try to develop methods to this annotation process. Propelled by the availability of data and the nature of the problem, the use of Deep Learning in Genomics and in Biology in general was beautifully reviewed in 2018 by a community effort [8].

On this work, we have chiefly focused on [9] to extend and develop further models that tackle the problems originated from the difference between Natural Language Processing and the use in genomes, starting from the simplest task possible in this subject: supervised classficiation of genes inside a prokaryotic genomic sequence.

Here, we propose that the "language" of the genes can be modeled since they have been selected by evolution to have conserved sequences. Resulting from this hypothesis, two different Deep Learning topologies have been tested and compared with different hyperparameters and regularization techniques.

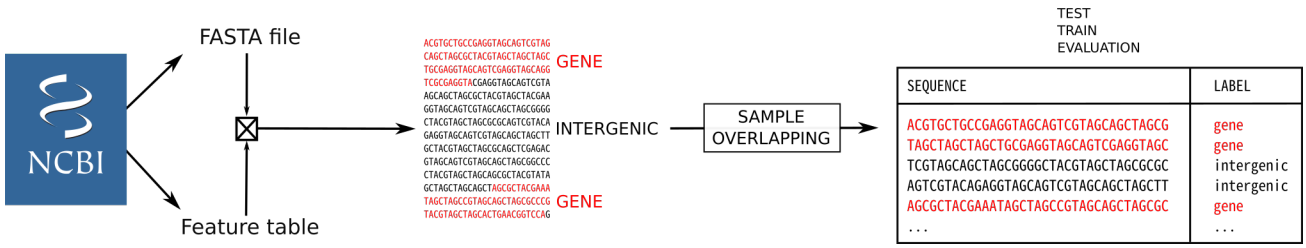


Fig. 1. Preprocessing workflow. Two files are retrieved from the NCBI Assembly database. The feature table with information about the position of the genes is mapped into the raw sequence of the genome of an organism. Then, this map is processed with an overlapping window of fixed length. Finally, the resulting dataset is divided into train, test and evaluation.

2. METHODS

The source code for this project was developed using the Python 3 programming language, with pytorch as the Deep Learning framework. All data and code is open sourced and it is available at `https://github.com/carrascomj/drastic`.

2.1. Data preparation

The assembly information of 10 prokayoriotic genomes (see 2) was downloaded from the database NCBI Assembly `https://www.ncbi.nlm.nih.gov/assembly`. By assembly information, here we refer to two files: the raw genomic sequence in a FNA file and a feature table TSV, which contains the start and end position of each gene in the sequence with its corresponding identifiers. Extra-genomic DNA - e.g., plasmids - were ignored.

Figure 1 shows the whole pre-processing workflow. First, the positions of the genes contained in the feature table were mapped to the raw genomic sequences. This map defines genes and intergenic regions in the DNA sequence. Second, overlapping fixed-length sequences were extracted and annotated with labels corresponding to the mapped regions and information about the genes. The overlapping space was chosen to be 10. Figure 2 illustrates the sampling process within a region. Finally, the built dataset was divided in test, test and evaluation data.

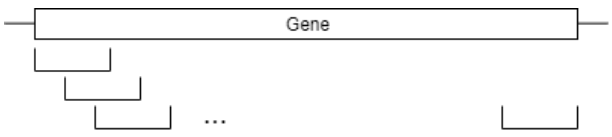


Fig. 2. Sampling method illustrated in genetic part. The samples overlaps each other in the same way that the samples of the genome will be input to the system.

2.2. Embedding approaches

In this project two different model approaches have been tested. The two models are very similar in their structure except in the manner they embed the input sequence of nucleotides. The first model is shown in Figure 3(a). This model uses the Continuous Bag of Words (CBOW) structure to train embedding vectors.

The second model that has been explored in this project is shown in Figure 3(b). This model uses a 1D-convolutional layer instead of the CBOW structure to create an embedding for the input sequence.

2.2.1. CBOW

In Natural Language Processing one of the main objectives is to create a meaningful numerical representation of a word– referred to as an embedding - capable of capturing its meaning. In well trained models, embeddings of related words should group together in their corresponding vector space.

Introduced first time by Mikolov et al. [10], word2vec (W2V) algorithms produce a distributed numerical representation of words based on the intuitions that the context is related to the meaning of a word and that words that closely related frequently occur together. The W2V algorithms include skip-gram and CBOW models:

- Skipgram is trained to predict context words from one input.
- Continuous Bag-of-Words (CBOW) works in the opposite direction, taking a window of n words as input surrounding the target word.

Numerous other models (e.g. fastText, GloVe) have been developed to obtain a meaningful word representation.

W2Vs can be applied not only to words interpretation and sentences sentiment analysis; their applications extend to codes in which unknown patterns might be encrypted, such symbolic series or genes.

In the field of genes and DNA our words are referred to as "kmers". That is, subsequences of nucleotides of

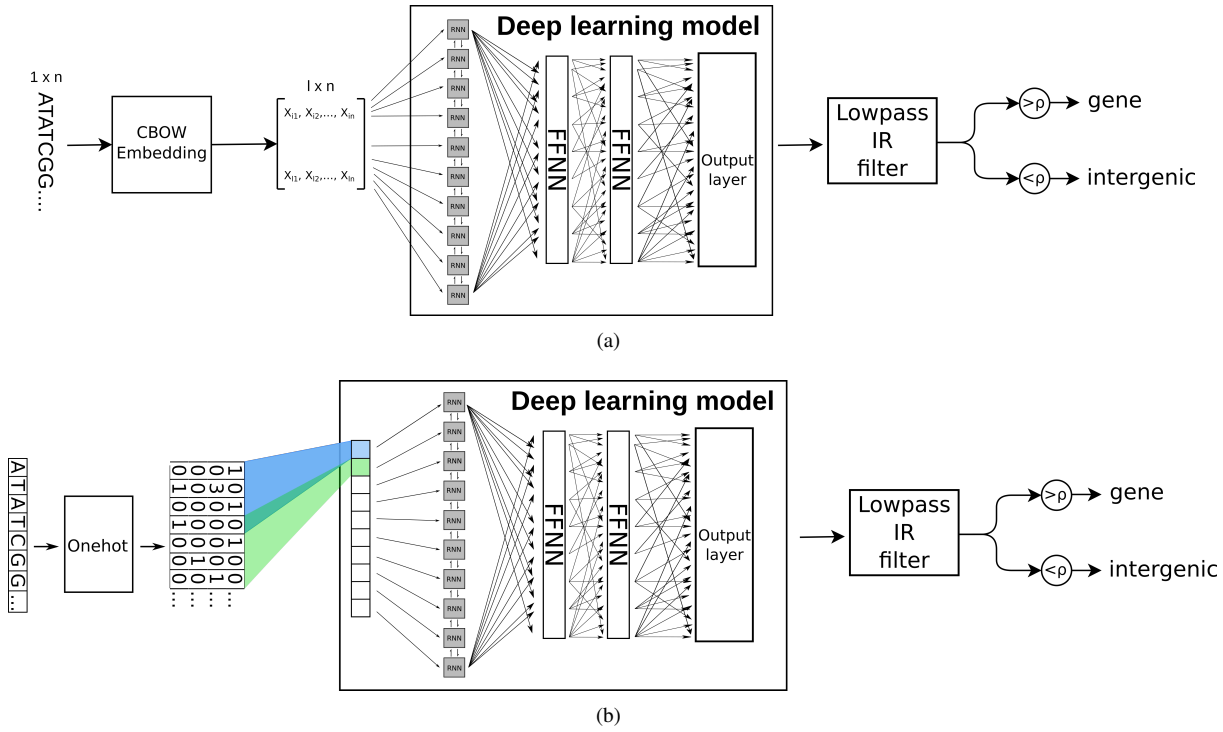


Fig. 3. (a) Deep learning model using CBOW embedding strategy. (b) Deep learning model using a convolutional layer as embedding strategy.

length k. Experimentally good results in gene annotation are usually achieved when $k = 3$ or $k = 7$ [9]. The number of possible unique kmers is equal to 4^k (eg for $k = 3$: AAA,AAC,AAG,AAT...TTT). For our Bacterial DNA analysis, we adopted the CBOW model to get a meaningful representation of each kmer in our sequence, predicting it from the kmers surrounding it.

The number of unique kmers that we encounter in our string is the size of our vocabulary, V. The size of the context window can be chosen arbitrarily; we set it to ± 2. This means each target kmer will be predicted using the two kmers before and two kmers after it. We train the model sliding over a long DNA sequence to get meaningful representations of the kmers in the form of embeddings.

As said, CBOW is trained to predict a target word w(t) from its context words. The CBOW model is shallow. It consists of an input layer of one-hot vectors, a "hidden layer" also called "embedding layer" or "projection layer" and one output layer.

Each input word in the context is first represented as a one-hot vector size V and fed to the model. These one-hot vectors are then projected in a N-dimensional vector at the hidden layer via an input Matrix size VxN. The N-dimensional vector is in fact the average of the context vectors. From the embedding layer, via an output Matrix size NxV we get an output vector of size V of distributed probabilities: The highest probability identifies the prediction. The prediction is compared to the target and weights of the input matrix and adjusted to minimize the error between the two. We train the model over the full genome, sliding each time and thus feeding a new context-target pair to CBOW.

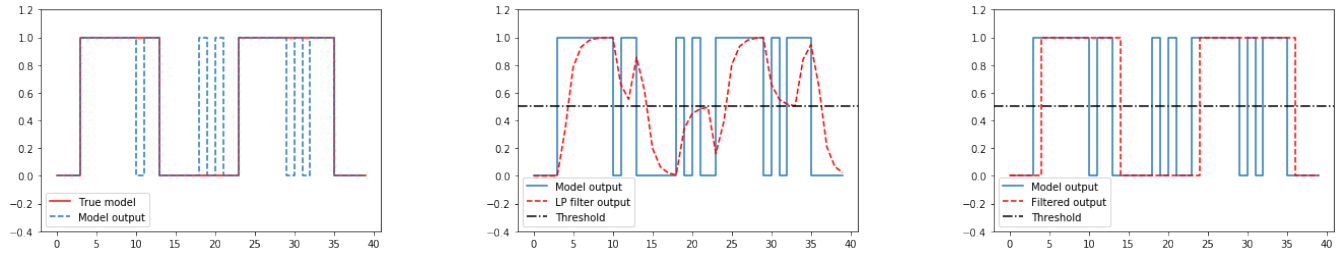
The CBOW model is a strong embedding technique and widely used, but it assumes that the actual split of words is known. This is of course the case with classic NLP tasks, but in the case of genomes no natural split of words exists. Essentially, that means that for a choice of k-length words the model has to be trained k times to ensure all word splits are taken into account. Given a DNA Sequence of length L with $k = 3$ we see how different word splits may be obtained:

TAGCCCATTTGCATTTTTCTGT...
TAG CCC ATT TGC ATT TTT CTG T... k =3, s = 0
T AGC CCA TTT GCA TTT TTC TGT... k =3, s = 1
TA GCC CAT TTG CAT TTT TCT GT... k =3, s = 2

It is possible to train models for all the combinations of these, but this is computationally expensive and for larger systems simply not viable. Thus, another approach that does not rely on hard choices of word splits has been investigated and is presented in subsequent section.

2.2.2. *Convolutional embeddings*

The main difficulty of the first approach is that we are looking for meaningful sequences of amino acids - the words in the "language" of proteins - without really knowing where these words start or end. Therefore, another model structure have been investigated.



(a) (b) (c)

Fig. 4. (a) Illustrates misclassifications in the output function. (b) Smoothing of output function. (c) Corrected function that needs time-shifting.

The second model that has been explored in this project is shown in Figure 3(b). This model uses a 1D-convolutional layer instead of the CBOW structure to create an embedding for the input sequence. Using a convolutional layer resolves the issue of having to choose where the "words" in the genome sequence splits. This structure was first proposed in [11].

The four nucleotides, A, C, G, T are one-hot encoded like with the CBOW model. A sub-sequence of the input sequence is then fed into the convolutional layer giving one output number. The window of convolution is slid with a fixed distance after every convolution. The idea is that performing convolution in a sub-sequence condenses the information found in that sub-sequence and will pass on the important piece of information to the subsequent layers in the model. In this way the input sequence is embedded via the convolution operation and using overlap between the convolution ensures that no hard encoding of word start and stops has to be chosen in advance. The window overlap used in this project was 62.5% and the sub-sequence length fed to the convolutional layer found via cross-validation was 8.

2.3. Deep learning model

The deep learning model itself is fairly simple. After the input sequence has been embedded via the CBOW-structure or the convolutional layer, the embedded input sequences are input to a bidirectional RNN layer. This layer is the key part of the model given the input's sequential nature.

The final output of the RNN layer is used as input to a feed forward layer. From there the output is fed through another feed forward layer and finally from their fed into the output layer.

Different optimizers (Adam, ADADelta, ASGD) have been tested to improve the performance of the models. The optimizer used for the CBOW model and convolutional model was ASGD and Adam, respectively. These were the ones giving the best results in the two models.

2.4. Regularization

Even with the fairly shallow network structures presented in Figure 3(a) and 3(b) the models were still overfitting when training. This is a good indication that the model is learning the parameters, but the need of regularization is present to be able to fit the model well to these genome types in general.

CBOW model was implemented without regularization, while the convolutional model uses dropout heavily as well as batch normalization in its feed forward and convolutional layers.

2.5. Smoothing the output-function

The task of the deep learning model is to determine whether a sub-sequence stems from a gene a not. The output should be 1 if the sub-sequence stems from a region within a gene and 0 otherwise. The average size of a gene in the bacteria used in this project is 900 characters. The input length of sequence put into the deep learning model is 50 characters. This means that whenever the model encounters a gene in the genome it is expected to output multiple 1's in a row. Similarly, the intergenetic regions may be much longer than the input size to the system. Therefore, the expected output of the model is a step function where the 1's and 0's come in clusters.

The system will be prone to errors and misclassifications will occur. However, the majority of the time the system classifies the input sequences correctly. This is illustrated in Figure 4(a). Practically, this means that when the system encounters a gene the majority of the outputs will be 1's with some misclassifications in between. Knowing this output function it is possible to obtain a higher accuracy simply by smoothing the output using a 1st order lowpass IIR filter. The filter has been made in such a way that single misclassifications in between correct classifications will be counted as the correct classifications. However, when the majority of the recent classifications shift from one class to the other the filter will stabilize in the new region. Figure 4(b) illustrates the filtering of the output function. Finally, Figure 4(c) illustrates the corrected output function. It is clear that the corrected output function becomes time-shifted. This happens because the filter requires a couple of subsequent outputs in the same class to overpass the threshold. Knowing the delay of the filter it is possible to simply shift the output function correspondingly in time.

	Training Accuracy (%)	Test Accuracy (%)	Smoothed Accuracy (%)	F1 Score (%)
CBOW Model	72.31	57.5	67.4	**79.8**
Convolutional Model	66.75	**64.5**	67.5	78.7
Old Sampling CBOW	87.25	64	**68.8**	$\sim$
Old Sampling Convolutional	71.36	59.4	63.9	$\sim$

Table 1. Table showing the results of the different models investigated in the project.

2.6. Analysis of convolutional activations

The convolutional layer was investigated to try disentangle the pattern that the model is looking for. This is not only covered in previous work for image recognition [12], but also in Deep Learning used in genomics, as in [13]. Briefly, a sequence of the appropriate length used during training was optimized to minimize the loss of a convolution operation for a specific layer and filter. The output was visualized with Logomaker [14] to generate a classical logo of the probability distribution of the sequences in the genome.

3. RESULTS

3.1. Model performance

Table 2.5 shows the results of the best performing models within the two different models investigated. The two upper rows of the table shows the results from the model structure using the sampling method described in Figure 2. The two bottom rows shows the results obtained in the two model structures using the previous sampling method i.e. only sampling start, middle and end of a gene.

The convolutional model using the sampling method illustrated in Figure 2 scores the highest accuracy when presented to new parts of the studied genome. However, the CBOW using the old sampling method is practically performing equally. The CBOW model using the old sampling method has the highest accuracy after applying the smoothing filter. The smoothing filter generally has a good effect on the overall performances of the models again, however, most of them are fairly equal in performance. F1 scores has only been obtained on the models using the new sampling method. Here, the CBOW model score the highest with 79.8% over the convolutional model's 78.7%.

3.2. Logos of convolutional activation

On one hand, the optimized sequence generated for the first filter of the Convolutional model is shown in 5(a). This sequence shows enrichment in citosine (C) and guanine (G). On the other hand, the sequence optimized to activate the second filter (5(a)) shows some enrichment and patterns involving tymine (T) and adenine (A).

4. DISCUSSION

All of the models show potential, but in general they still have too low accuracy to be used for completely automated annotation processes. Both model architectures look promising and none of them outperform the others in a significant way.

One possible cause of poor performance in our models is the sparsity of the information in our data: despite having a large corpora of bacterial DNA sequences, extracting information-dense chunks to train our models properly is not trivial.

NLP models such CBOW have the big drawback of requiring very long time to train on very large amount of data. Therefore, while convolutional model was trained with 9000,12000 and 17000 samples from each genome, CBOW was only trained using 2000 samples from each genome.

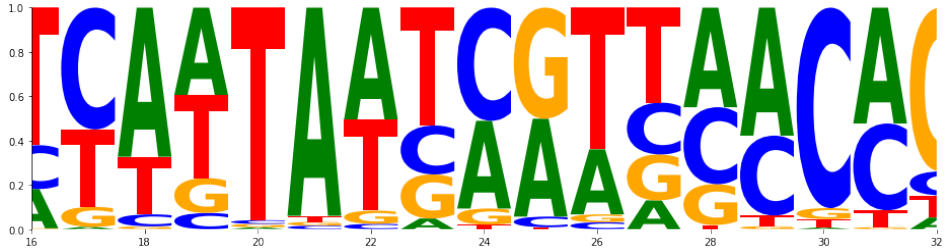
The lack the computational resources limited possible further improvement for the CBOW in terms of exploring higher embedding dimensionalities and different kmers lengths, which would have led to a vocabulary exploding in size. As mentioned this is due to the fact that the size of the vocabulary grows with 4^k.

The CBOW model gave the best result without any regularization or dropout. This is probably due to the lower amount of data used for the CBOW and thus it performs better when having all the data available.

On the opposite, the convolutional model tendency to overfit was mitigated by using heavy regularization. In fact, the convolutional model was overfitting after 15 epochs of training even using 0.5 dropout in both feed forward layers and in the convolutional layer. The RNN layer chosen in the convolutional model was a GRU which has an built-in dropout option in PyTorch. Furthermore, batch normalization was used. The convolutional model still overfitted which was an issue all throughout the development of the model. In the following section future work to over come this challenge is proposed.

About the analysis of convolution activations, the filters showed attention of different patterns. Interestingly enough, the second filter is optimized by using patterns that resembles as Pribnow box, which like the eukaryotic TATA box, is bound preserved on the core promoter of prokaryotic genes [15], demonstrating biological sense.

(a)



(b)

Fig. 5. Optimized sequences zoomed for twice the filter size for the Convolutional Model, but these patters are consistent among the whole optimized sequences. (a) First filter. (b) Second filter.

5. FURTHER WORK

It has shown to be difficult to automatically annotate gene locations in genome because of the very sparse nature of the data. In future work one thing that may help give better performance is to change the task of the network slightly. Instead of having the deep learning model categorizing whether a sequence stems from a gene a not, it would probably be easier to simply be able to recognize the beginnings and endings of gene sequences.

Genomes as well as genes are very sparse. However, the beginning and ending of genes follow certain patterns that may be more easily recognizable. At the same time the information carried in the beginning and ending of genes may be less sparse in nature than in an arbitrary point within a gene.

Further regularization of the models is something that has to be looked more into as well. A lot of regularization has been used already, but is unfortunately not working as intended as the models are still overfitting the data. This is probably because the RNN layer does not respond well to dropout given that it loses important information that is carried in the hidden state. More advanced regularization techniques such as the AWD-LSTM proposed in Stephen Merity et. al. [16] will be interesting to investigate. This would also allow for more deep networks that may more accurately annotate the genome sequences.

6. REFERENCES

[1] James D. Watson and Francis H. C. Crick, "Molecular structure of nucleic acids: A structure for deoxyribose nucleic acid," *Nature News*, 1953.

[2] Alice Maria Giani, Guido Roberto Gallo, Luca Gianfranceschi, and Giulio Formenti, "Long walk to genomics: History and current approaches to genome sequencing and assembly," *Computational and Structural Biotechnology Journal*, vol. 18, pp. 9–19, 2020.

[3] "A physical map of the human genome," *Nature*, vol. 409, no. 6822, pp. 934–941, 2001.

[4] N Graham and et al, "Impact of major depression on cardiovascular outcomes for individuals with hypertension: prospective survival analysis in uk biobank," *BMJ Open*, 2019.

[5] Ross Overbeek, Robert Olson, Gordon D. Pusch, Gary J. Olsen, James J. Davis, Terry Disz, Robert A. Edwards, Svetlana Gerdes, Bruce Parrello, and Maulik Shukla, "The seed and the rapid annotation of microbial genomes using subsystems technology (rast)," *Nucl. Acids Res.*, vol. 42, pp. D206–D214, 01 2014.

[6] Richard Durbin, Sean R. Eddy, Anders Krogh, and Graeme Mitchison, *Markov chains and hidden Markov models*, p. 47–80, Cambridge University Press, 1998.

[7] Byung-Jun Yoon, "Hidden markov models and their applications in biological sequence analysis," *Current Genomics*, vol. 10, no. 6, pp. 402–415, Jan 2009.

[8] Travers Ching, Daniel S. Himmelstein, Brett K. Beaulieu-Jones, Alexandr A. Kalinin, Brian T. Do, Gregory P. Way, Enrico Ferrero, Paul-Michael Agapow, Michael Zietz, and Michael M. Hoffman, "Opportunities and obstacles for deep learning in biology and medicine," *J. R. Soc. Interface*, vol. 15, pp. 20170387, 01 2018.

[9] Mohammad Ruhul Amin, Alisa Yurovsky, Yingtao Tian, and Steven Skiena, "Deepannotator," 01 2018.

[10] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean, "Efficient estimation of word representations in vector space," 2013.

[11] Daniel Quang and Xiaohui Xie, "Danq: a hybrid convolutional and recurrent deep neural network for quantifying the function of dna sequences," *Nucleic Acids Res*, vol. 44, pp. e107–e107, 01 2016.

[12] Utku Ozbulak, "Pytorch cnn visualizations," `https://github.com/utkuozbulak/pytorch-cnn-visualizations`, 2019.

[13] David R. Kelley, Jasper Snoek, and John L. Rinn, "Basset: learning the regulatory code of the accessible genome with deep convolutional neural networks," *Genome Research*, vol. 26, no. 7, pp. 990–999, Mar 2016.

[14] Ammar Tareen and Justin B. Kinney, "Logomaker: Beautiful sequence logos in python," *bioRxiv*, 2019.

[15] D. Pribnow, "Nucleotide sequence of an rna polymerase binding site at an early t7 promoter.," *Proceedings of the National Academy of Sciences*, vol. 72, no. 3, pp. 784–788, Jan 1975.

[16] Stephen Merity, Nitish Shirish Keskar, and Richard Socher, "Regularizing and optimizing LSTM language models," *CoRR*, vol. abs/1708.02182, 2017.

A. SUPPLEMENTARY MATERIAL

A.1. Prokaryotic assembly genomes

Genome Assembly	Organism
ASM886v2	*Escherichia coli O157:H7 str. Sakai*
ASM904v1	*Bacillus subtilis subsp. subtilis str. 168*
ASM2730v1	*Haemophilus influenzae Rd KW20*
ASM852v1	*Helicobacter pylori 26695*
ASM2732v1	*Mycoplasma genitalium G37*
ASM2734v1	*Mycoplasma pneumoniae M129*
ASM21472v1	*Methanobacterium paludis*
ASM301522v1	*Synechocystis sp. IPPAS B-1465*
ASM866v1	*Archaeoglobus fulgidus DSM 4304*

Table 2. Assembly number of each genome with its corresponding organism.

A.2. CBOW Training procedure

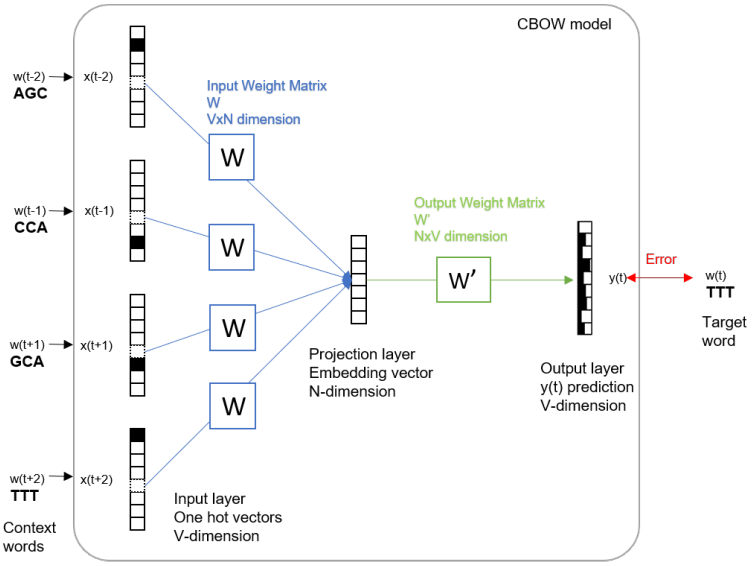


Fig. 6. Illustration of training procedure of CBOW method.