Spring 2018

A Neural Network Classifier for the COI Barcode Gene

Saurabh Marathe
San Jose State University

Follow this and additional works at: https://scholarworks.sjsu.edu/etd_projects

Part of the [Computer Sciences Commons](#)

Recommended Citation

Marathe, Saurabh, "A Neural Network Classifier for the COI Barcode Gene" (2018). *Master's Projects*. 604.
DOI: <https://doi.org/10.31979/etd.2tmv-y2rq>
https://scholarworks.sjsu.edu/etd_projects/604

This Master's Project is brought to you for free and open access by the Master's Theses and Graduate Research at SJSU ScholarWorks. It has been accepted for inclusion in Master's Projects by an authorized administrator of SJSU ScholarWorks. For more information, please contact scholarworks@sjsu.edu.

A Neural Network Classifier for the COI Barcode Gene

A Writing Project

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Computer Science

By

Saurabh Marathe

Spring 2018

ABSTRACT

Mitochondrial Cytochrome C Oxidase subunit I (CO I – to be read as “see – oh one”) is a 658 base pair region in the gene encoding that is proposed as standard barcode for animals. Meaning, the CO I is a special region found in animal DNA that is studied to identify the species of the animal. Currently, there is an implementation of an algorithm called ARBitrator which identifies and extracts these CO I sequences from enormous genes database called GenBank. The ARBitrator is good at extracting the CO I sequences that have better specificity and accuracy as compared to other existing algorithms for CO I sequence identification^{[1][2]}. Now, this project aims at training a neural network to learn the features of the CO I sequences extracted by ARBitrator, so that this neural network can be used in future to further recognize CO I sequences. Effectively, we are aiming to successfully design, train, and use a deep learning neural network to learn to recognize CO I sequences in a supervised way. This is the first time that a neural network is explored and used for this purpose.

ACKNOWLEDGMENT

I would like to express my honest gratitude to my project advisor Dr. Philip Heller for his continuous guidance, time, and assistance during this project. I am grateful, and I thank my committee members Dr. Robert Chun and Dr. Steven Silver for their thoughts, time, and suggestions.

Special thanks to Dr. Robert Chun for teaching me advanced parallel algorithms and techniques which were very useful in methods devised in this project.

My sincere thanks and regards to continuous support and valuable suggestions from Dr. Steven Silver, throughout my master's program, and also for this project which is very relevant to my work with him before.

We would also like to thank Department of Computer Science for help in setting up this project defense. We would also like to express our thanks to the library for giving required books and materials to learn different aspects and concepts of our project and extended study hours for students.

Last but not the least, I would also like to heart fully thank my friends and family for their moral support, encouragement, and blessings without which I could not able to finish my project and all the required study successfully.

TABLE OF CONTENTS

1. Introduction.....	6
2. Related Work.....	8
3. Methods.....	11
4. Results and Discussion.....	35
5. Conclusion.....	40
REFERENCES.....	41

TABLE OF FIGURES

Figure 1: Model 1 Architecture.....	13
Figure 2: Snapshot of final training accuracy of Model 1 for 2000 quanta of data....	15
Figure 3: Model 1 accuracy graph after running on 2000 quanta of data.....	16
Figure 4: Snapshot when Model 1 is run on only positive sequences of data.....	19
Figure 5: Snapshot when Model 1 is run on only negative sequences of data.....	21
Figure 6: Model 2 Architecture.....	25
Figure 7: Snapshot showing Model 2 accuracy when run on 39,900 quanta of data...	26
Figure 8: Model 2 accuracy graph after running on 39,900 quanta of data.....	27
Figure 9: Snapshot showing Model 2 accuracy when run on 8000 quanta of data....	28
Figure 10: Model 2 accuracy graph after running on 8000 quanta of data.....	28
Figure 11: Snapshot of Model 2 accuracy when run on 2000 quanta of data.....	30
Figure 12: Model 2 accuracy graph after running on 2000 quanta of data.....	30
Figure 13: Snapshot of Model 2 failed training because of only positive data.....	32
Figure 14: Snapshot of Model 2 failed training because of only negative data.....	34

1. Introduction

This project is the first of its kind of attempt to explore, study, design, implement, train and evaluate a Artificial Neural Network (ANN) to recognize the CO I sequences from DNA sequences found in the GenBank database. This report is final culmination of a number of attempts along with successful attempts towards this goal.

The end deliverable of this project will be a Neural Network model (A deep Learning Neural Network model) that is trainable and will be useful to extract CO I sequences from GenBank and its performance is comparable to that of the existing algorithms.

This report is divided into 4 sections consequently. First section talks about previous attempts related to this project. It discusses papers which have detailed explanation of Deep Learning Neural Networks implemented for recognizing DNA sequences different from the CO I sequences.

Second section talks about the experiments with respect to different models and designs of the neural network that were carried out as part of this project. It describes in detail all the experiments along with topological details of the models that were used for study and application to the domain. It further elucidates the accuracy of these models as obtained when they were used for classification purpose on different quantum of CO I sequences data. It provides actual implementation results.

Third section is objective comparison of all the models on basis of their performance parameters, especially with respect to the accuracy criteria. Most importantly, it will

compare the ARBitrator and BLAST algorithms with the Artificial Neural Network at hand.

Last section is brief conclusion of the culmination of all the experiments followed by detailed references of the sources used to form this project.

2. Related Work

There is no previous work of this kind that is done before. This is the first time an artificial neural network is being built to extract CO I sequences from the GenBank database. First time the neural network will be implemented to perform as a classifier for a highly specific and accurate CO I sequences obtained from ARBitrator algorithm.

Neural network are part of machine learning techniques that aim to help computers classify CO I nucleotide sequences without having to explicitly program them. This power of automation, without explicit programming, means that the problem is beyond regular software development. Main challenge is to try to demonstrate that computers can learn to classify with help of neural networks, without having the need to program them, as programming is not possible for such classification problems. Also, automation using machine learning in form of neural networks, will lead to a new way of validating CO I sequences from gene sequence database like GenBank.

Automation and supervised neural network based learning means that there will be false positive and false negative in classification. Key challenge would be to explore and implement different machine learning techniques like bootstrapping, basic model building, ensemble learning, hyper-parameter tuning etc. to try and bring up the accuracy level for neural network. So, making use of right techniques for right problems, will be key experimental challenge in this project. Previously, different flavors of neural network have been used to train on recognizing species of animals using CO I sequences. Meaning, the question addressed in the previous attempts was “Given a CO I sequence, what is the species?”. Radial Basis Function Neural Network^[5], Back Propagation Neural Network^[6], Convolutional

Neural Network^[4], General Regression Neural Network^[10] are different types of neural network that have been used previously for this purpose.

In this project, we will be studying and implementing a neural network to train on the CO I database we have obtained from GenBank using ARBitrator. Meaning, the question being addressed here is, “Given a sequence, is it a CO I sequence?” Hence, it will be first time to use the deep learning neural network to form a classifier for CO I sequences we have as obtained from ARBitrator algorithm. These sequences have more specificity and accuracy as compared to the other existing gene databases^[1]. Trying to build a neural network classifier for such a database of highly specific sequences is the challenge, and will require understanding of ARBitrator algorithm-derived CO I sequences, their specialty. Additionally, the project will require understanding of neural networks and their design types and mathematical working, study of similar previous art, understanding and implementation of various machine learning techniques like hyperparameter tuning, ensemble learning, model evaluation and cross validation, so that a strong neural network classifier is implemented. After the different stages of implementation and study are completed ^{[4][14]}, we aim to have a neural network that will act as classifier for the CO I sequences derived from ARBitrator algorithm. Performance analysis and result quantification in terms of accuracy of the neural network will be the culminating step of the project, that will dictate future usage of the neural networks as a technology candidate for the classifier and next steps for future work.

We intend to publish the findings with respect to our research and the performance of our Neural Network in Bioinformatics and Computer Science journals. The

publication will focus on the novelty of usage of Neural Networks for first time in this domain. A detailed study of previously existing neural networks, their modifications and applications for making them suitable candidates for the highly specific database at hand, and implementing them at parallel scales to yield accuracy results, will all be novelty, as it all is the first attempt for the specific data of CO I sequences. The idea here is to publish the results for providing a solid foundation for future research to build on it, so that a highly autonomous and self-learning technology like Artificial Neural Network could lead the classification task for important bio-markers like CO I sequences in the future. That is the precise goal of this thesis, which is novel in its purpose, and approach and results. Hence, the content will be seminal and so we wish to publish the results in the near future.

3. Methods

3.1. Deep Learning Model 1 (based on reference [4])

This is a paper that used a Convolutional Neural Network for DNA sequence classification. Its model implementation was made available. It was required to try this model out on the CO I positive sequences and negative sequences of the data as we had to see the feasibility of the project. Neural Network parameter changing was required and was implemented to run this model from scratch on our data. Total Data consisted of 1,70,000 positive sequences of CO I, and only 160 negative sequences that symbolized confirmed non-CO I sequences. This huge dis-proportion of data is due to the fact that non-CO I sequences are actually paralogs of CO I sequences, and the paralogs of CO I sequences are non-existent in large numbers in nature ^[2]. Hence, we have very few non CO I sequences. The performance of the neural network will be evaluated keeping this dis-proportion in mind. In this model, the input data is vectorized by treating all the 1,70,000 sequences as one document and each 220-amino acid long sequence is then produced as a numerical vector using TensorFlow VocabularyProcessor fit_transform function. Basic use of this fit_transform technique is to use a limited size array (in our case 220 length array) to represent similar sequences by similar numerical representations and different sequences by different representations.

A. Model 1 run snapshots for CO I sequences of total size of 2000 sequences:

Model 1 here is elucidated in graphical topological form using TensorBoard.

The Model 1 ANN for data size of 2000 sequences is as follows in

architecture: (Legend to read the model is given here)

Figure 1: Model 1 Architecture

Following is snapshot of various optimization parameters used in the Model1 training that are shown as important nodes in TensorBoard graph. The model architecture and these nodes are connected and to be seen together:

The Model 1 actual implementation snapshot showing final training accuracy after cross-validation approach:

```

Writing to /home/saurabh/Downloads/New folder/DNA-Sequence-Classification-using-QNN-master/runs/1525791187

Initializing Saver:
WARNING:tensorflow:From train2_with_train_summaries.py:138: all_variables (from tensorflow.python.ops.variables) is deprecated and will be removed after 2017-03-02.
Instructions for updating:
Please use tf.global_variables instead.
Done Initializing Saver
WARNING:tensorflow:From /home/saurabh/.local/lib/python2.7/site-packages/tensorflow/python/util/tf_should_use.py:118: initialize_all_variables (from tensorflow.python.ops.variables) is deprecated and will
be removed after 2017-03-02.
Instructions for updating:
Use 'tf.global_variables_initializer' instead.
('Total Number of Iterations for RT Data: ', 320)
('Number of Iterations per Epoch: ', 160)

Evaluation:
2018-05-08T07:56:25.378646: step 160, loss 0.312931, acc 0.9255

Evaluation:
2018-05-08T08:00:59.718134: step 320, loss 0.410866, acc 0.9255

('ACCURACY: ', 0.9255)

```

Figure 2: Snapshot of final training accuracy of Model 1 for 2000 quanta of data

The following is TensorBoard snapshot of the accuracy graph across the training phase of the Model 1. It is important to know the trend of training of the model and how it was tuned to recognize our data:

Figure 3: Model 1 accuracy graph after running on 2000 quanta of data

Each dip and rise above is the symbolization of one training epoch of the neural network over the data.

Important highlights of Model 1:

- The model 1 has more weights and biases to calculate in terms of architecture. It has very high execution time per epoch. Also, in this model we are using an entire dataset for training in one execution. The training samples are shuffled, and batches-based training does happen, but the batches cover the whole training dataset in first feed itself.
- The execution time is so high that the neural network would take weeks if trained on higher number of data sequences.
- Not a very feasible model
- Achieved relatively low accuracy of 92.6%

B. Model 1 run snapshots for CO I sequences of total size of 8000 sequences

Following is the snapshot showing the accuracy levels achieved by the Model 1 when a smaller dataset of 8000 sequences out of 39,900 unique sequences is passed

```
Writing to /home/saurabh/Downloads/New folder/DNA-Sequence-Classification-using-CNN-master/runs/1525799070

Initialising Saver:
WARNING:tensorflow:From train2_with_train_summaries.py:138: all_variables (from tensorflow.python.ops.variables) is deprecated and will be removed after 2017-03-02.
Instructions for updating:
Please use tf.global_variables instead.
Done Initializing Saver
WARNING:tensorflow:From /home/saurabh/.local/lib/python2.7/site-packages/tensorflow/python/util/tf_should_use.py:118: initialize_all_variables (from tensorflow.python.ops.variables) is deprecated and will be removed after 2017-03-02.
Instructions for updating:
Use 'tf.global_variables_initializer' instead.
('Total Number of Iterations for RT Data: ', 12288)
('Number of Iterations per Epoch: ', 6148)
^CTraceback (most recent call last):
  File "train2_with_train_summaries.py", line 217, in <module>
    train_step(x_batch, y_batch)
  File "train2_with_train_summaries.py", line 166, in train_step
    feed_dict)
  File "/home/saurabh/.local/lib/python2.7/site-packages/tensorflow/python/client/session.py", line 905, in run
    run_metadata_ptr)
  File "/home/saurabh/.local/lib/python2.7/site-packages/tensorflow/python/client/session.py", line 1140, in _run
    feed_dict_tensor, options, run_metadata)
  File "/home/saurabh/.local/lib/python2.7/site-packages/tensorflow/python/client/session.py", line 1321, in _do_run
    run_metadata)
  File "/home/saurabh/.local/lib/python2.7/site-packages/tensorflow/python/client/session.py", line 1327, in _do_call
    return fn(*args)
  File "/home/saurabh/.local/lib/python2.7/site-packages/tensorflow/python/client/session.py", line 1312, in _run_fn
    options, feed_dict, fetch_list, target_list, run_metadata)
  File "/home/saurabh/.local/lib/python2.7/site-packages/tensorflow/python/client/session.py", line 1420, in _call_tf_sessionrun
    status, run_metadata)
KeyboardInterrupt
saurabh@saurabh-VirtualBox:~/Downloads/New folder/DNA-Sequence-Classification-using-CNN-master$
```

As you can see, the first epoch itself took so long that after 30 minutes of waiting for the first batch of training to even start, we had aborted the program by KeyBoard Interrupt. Similar, very high execution times for each epoch were encountered even when the Model 1 was run on high compute power resource like Amazon AMI instance. Hence, Model 1 was found to be very time consuming for practical training of very large dataset.

Because of this high execution time even for 8000 sequences, we did not run the neural network for the whole dataset of 39,900 unique sequences. Reasons

for this exceptionally high time consumption are further discussed in Results and Discussion part of the report.

C. Model 1 run snapshots when only positive CO I sequences are passed as data

Now, this may seem like an odd case. Why should we pass all positive sequence data to a neural network? As mentioned in previous section of the report, there is very high dis-proportion between quantum of positive data and negative sequence data. So, by giving the neural network a fully positive dataset and fully negative dataset, we train them on these highly homogenous data. So, if neural network architecture is sound, then, it will detect that very similar data is fed to it as both positive samples and negative samples. Meaning, it will, through its training and weight update changes, will show that it is more confused about the input data. So, good neural network architectures will have low accuracy scores whenever we give them same or very similar data as both positive and negative dataset. On the other hand, weak neural network architecture will fail, and may not detect similarity between the negative and positive samples leading to it learning positive samples as negative samples – a phenomenon undesirable for our neural network considering dis-proportion of data we have.

Another important phenomenon that dis-proportionate data across two classes may cause is that a neural network may become good at recognizing only one class of the two classes. This is also otherwise called as over fitting. In our case, there is high chance for the neural network to become over fitted to the

positive sequences of the data, which are 1000 times more in quantum than negative sequences.

So, to see if our neural network architecture is good and not susceptible to over fitting, and to make sure that it learns about both the classes equally well in spite of data dis-proportion, we supply same or similar data as positive and negative samples and monitor the accuracy levels of recognition.

```
Writing to /home/saurabh/Downloads/New folder/DNA-Sequence-Classification-using-CNN-naster/runs/1525796063

Initializing Saver:
WARNING:tensorflow:From train2_with_train_summaries.py:138: all_variables (from tensorflow.python.ops.variables) is deprecated and will be removed after 2017-03-02.
Instructions for updating:
Please use tf.global variables instead.
Done Initializing Saver
WARNING:tensorflow:From /home/saurabh/.local/lib/python2.7/site-packages/tensorflow/python/util/tf_should_use.py:118: initialize_all_variables (from tensorflow.python.ops.variables) is deprecated and will be removed after 2017-03-02.
Instructions for updating:
Use 'tf.global_variables_initializer' instead.
('Total Number of Iterations for RT Data: ', 4000)
('Number of Iterations per Epoch: ', 2000)

Evaluation:
2018-05-08T09:26:36.724468: step 2000, loss 0.695104, acc 0.486

Evaluation:
2018-05-08T09:39:06.659865: step 4000, loss 0.693071, acc 0.514

('ACCURACY: ', 0.514)
```

Figure 4: Snapshot when Model 1 is run on only positive sequences of data

As you can see in above snapshot, the Model 1 scored a low accuracy of 51%. Meaning, when only positive samples are given, the neural network detected the similarity between the sequences that were passed as both positive and negative samples. This led to its confusion rather than learning. Hence now,

the new accuracy after training on this homogenous positive sequences being supplied as both negative and positive dataset is as low as 51%.

This confirms that Model 1 is actually a good architecture to detect similarities between similar input patterns even if we give them with opposite labels for training.

Although, the most ideal case is where a neural network training itself crashes, because of the continuous changes of weights not leading to convergence. So, Model 1 has clearly not crashed. Meaning, it has performed poor but still it was fooled in believing that input sequences were still different in pattern, leading to 51% accuracy of recognition, when it should have been 0%.

So, Model 1 is a good model but not the best in this regard.

D. Model 1 run snapshots for when only negative CO I sequences are passed as data

Just like running Model 1 on all positive data sequences, it is important to run it on all negative sequences as well. This will especially address problem of over fitting. If the accuracy of the neural network for the negative sequences is very low as compared to that of the positive sequences, then it means that neural network has actually learnt more about the negative sequences more, and was able to detect similarity between the negative sequences being passed as both positive and negative datasets. This means there is more information gain from negative sequences and the neural network is over fitted to negative sequences. If the accuracy of the neural network for the negative sequences alone is very high as compared to that of the positive sequences, then consequently we can say, that the neural network is over fitted on the positive

sequences and there is more information gain for the neural network from the positive sequences than the negative sequences. An accuracy score similar to that of positive sequences means that the model is balanced and hence there is not over fitting problem. In that case, the model has equal knowledge of both the sequences, and that both the data sequences are highly unique in nature and quality. These will qualm any problems with over fitting and data disproportion.

```
Writing to /home/saurabh/Downloads/New folder/DNA-Sequence-Classification-using-CNN-master/runs/1525790016

Initializing Saver:
WARNING:tensorflow:From train2_with_train_summaries.py:138: all_variables (from tensorflow.python.ops.variables) is deprecated and will be removed after 2017-03-02.
Instructions for updating:
Please use tf.global_variables instead.
Done Initializing Saver
WARNING:tensorflow:From /home/saurabh/.local/lib/python2.7/site-packages/tensorflow/python/util/tf_should_use.py:118: initialize_all_variables (from tensorflow.python.ops.variables) is deprecated and will be removed after 2017-03-02.
Instructions for updating:
Use 'tf.global_variables_initializer' instead.
('Total Number of Iterations for RT Data: ', 600)
('Number of Iterations per Epoch: ', 300)

Evaluation:
2018-05-08T09:47:04.700140: step 300, loss 0.705677, acc 0.45

Evaluation:
2018-05-08T09:47:12.068367: step 600, loss 0.699081, acc 0.45

('ACCURACY: ', 0.45)
```

Figure 5: Snapshot when Model 1 is run on only negative sequences of data

As seen in the snapshot above, when the Model 1 is trained only on the negative sequences, we see the average training accuracy is 45%. That means, it is 5-6% lesser than that for only positive sequences.

For this deviation in accuracy, with the data dis-proportion we have, we can safely say that there is no problem of over fitting and that each sequences in the positive set is very different from each negative set – a quality highly desirable. As for the architecture of Model 1, the above snapshot makes it clear that it has learnt about both the positive as well negative sequences equally.

In machine learning, to have a model this balanced is rare. And the observation is that, the reason for this is the good quality of the sequences, the sequences are very unique and are high information gain sequences in terms of their pattern. The primary reason for this could be the fact that the sequences were extracted by ARBitrator algorithm, which has only 0.03% error rate in recognizing CO I sequences.

More detailed implications and overall performance review of the Model 1 is discussed in detail in Results and Discussion section.

Performance comparison with ARBitrator algorithm is also available in subsequent independent sections.

3.2.Deep learning model 2 (Based on reference [12])

The model used in [4] is originally based on this model. This model is an early application using TensorFlow to build a text categorization application. So this model 2 code was available for public use on GitHub, but for a completely different purpose of text categorization. Now, CO I sequences are themselves expressed as sequences of amino acids. So, they are also a 220-character long string of amino acids (building blocks of DNA). So, key idea used by [4] was to modify this model to suit the DNA

sequence nature. We used this model directly on the CO I sequences where the CO I sequences are treated as text by the model. This is the culmination of previous knowledge of neural networks, and intuition of its applicability, that it was decided to go forward with this. The parameters of the neural network had to be changed to suit the data at hand. Accordingly the parameters like the `maximum_document_length` along with data pruning of extracting 39,900 unique sequences of CO I from 1,70,00 data sequences was performed. This had to be done, because training the neural network on all the 1,70,000 sequences data would take weeks of continuous run on a cloud-based cluster. Even with 39,900 sequences this model had to be run on specialized Amazon Machine Instances (AMI) for Deep Learning. In this model, the input 220-long amino acid sequences are represented numerically as vector by using the TensorFlow VocabularyProcessor `fit_transform()` function. Thus, vectorization of input character-based CO I sequence in form of numerical arrays is similar to both Model 1 and Model 2. Except in Model 2 there is an extra Embedding layer. This extra layer takes care of problems like making all string equal if they are not by padding zeroes, as also it again enhances the `fit_transform()` functionality of representing similar strings or CO I sequences as really similar. Thus, Embedding layers embeds the input into a representation more easy to compute and consistent with the neural network architecture.

A. Model 2 run snapshots for CO I sequences of total size of 39,900 unique sequences

Model 2 here is elucidated in graphical topological form using TensorBoard. The Model 2 ANN for data size of 39,900 sequences is as follows in architecture: (Legend to read the model is given here)

▼ Close legend.

Graph (* = expandable)

Namespace* [?](#)

OpNode [?](#)

Unconnected series* [?](#)

Connected series* [?](#)

Constant [?](#)

Summary [?](#)

Dataflow edge [?](#)

Control dependency edge [?](#)

Reference edge [?](#)

Figure 6: Model 2 Architecture

The Model 2 actual implementation snapshot showing final training accuracy after cross-validation approach:

```

Evaluation:
2018-05-01T08:20:55.617892: step 7200, loss 0.338681, acc 0.928857

Saved model checkpoint to /home/ubuntu/second experiment/runs/1525160965/checkpo
ints/model-7200

2018-05-01T08:20:56.011374: step 7201, loss 0.0021389, acc 1
2018-05-01T08:20:56.240882: step 7202, loss 0.00732566, acc 0.99
2018-05-01T08:20:56.460466: step 7203, loss 1.37604e-05, acc 1
2018-05-01T08:20:56.684545: step 7204, loss 1.39838e-05, acc 1
2018-05-01T08:20:56.915301: step 7205, loss 1.02039e-06, acc 1
2018-05-01T08:20:57.135621: step 7206, loss 6.4968e-07, acc 1
2018-05-01T08:20:57.361998: step 7207, loss 2.08128e-06, acc 1
2018-05-01T08:20:57.588375: step 7208, loss 1.58547e-07, acc 1
2018-05-01T08:20:57.806542: step 7209, loss 6.72211e-06, acc 1
2018-05-01T08:20:58.031135: step 7210, loss 3.63168e-06, acc 1
2018-05-01T08:20:58.257840: step 7211, loss 3.70737e-07, acc 1
2018-05-01T08:20:58.478865: step 7212, loss 7.58142e-07, acc 1
2018-05-01T08:20:58.705837: step 7213, loss 7.99867e-07, acc 1
2018-05-01T08:20:58.930658: step 7214, loss 1.19209e-09, acc 1
2018-05-01T08:20:59.148621: step 7215, loss 1.25354e-05, acc 1
2018-05-01T08:20:59.368367: step 7216, loss 2.38419e-09, acc 1
2018-05-01T08:20:59.587584: step 7217, loss 0.00035908, acc 1
2018-05-01T08:20:59.815904: step 7218, loss 3.57628e-09, acc 1
2018-05-01T08:21:00.041561: step 7219, loss 1.00848e-06, acc 1
2018-05-01T08:21:00.238748: step 7220, loss 0, acc 1
Avg training loss 0.083711
Avg training accuracy 0.980862
ubuntu@ip-172-31-18-228:~/second experiments$

```

Figure 7: Snapshot of Model 2 accuracy when run on 39,900 quanta of data

Please notice the high 98% average accuracy as compared to 92% by Model 1 when it was run on 2000 sequences of data.

The following are TensorBoard snapshot of training phase accuracy graph. Each dip to rise is training for each batch of data. There is flat line at end of training symbolizing the convergence reached by the ANN in the very middle of training. But one dip-to-rise again at the end symbolizes the encounter of new training data by the neural network. Thus, we see that the ANN in Model 2 reaches convergence early even though total unique sequences are 39,900, meaning that most of the sequences are **highly similar**, and new sequences or very different sequences are there at the end of the whole dataset.

accuracy_1

Figure 8: Model 2 accuracy graph after running on 39,900 quanta of data

There is convergence reached at about first 3000 sequences itself. That is why we see accuracy being 1.0 (100%) for a long time till Neural Network reaches new data at the end of the 39,900 total sequences.

B. Model 2 run snapshots for CO I sequences of total size of 8000 data sequences

The same Model 2 was tried on 8000 sequences of data. We will observe two snapshots, one of the actual accuracy recorded. The other will be the learning curve of the accuracy. We will be comparing these results with Model 2 performance when it was given 39,900 data sequences. The snapshots are on the subsequent pages.

```

2018-05-08T17:34:26.524714: step 1460, loss 0.000305487, acc 1
2018-05-08T17:34:26.622434: step 1461, loss 0.000362446, acc 1
2018-05-08T17:34:26.718328: step 1462, loss 0.000203079, acc 1
2018-05-08T17:34:26.811869: step 1463, loss 0.000306559, acc 1
2018-05-08T17:34:26.908715: step 1464, loss 0.000181223, acc 1
2018-05-08T17:34:27.007263: step 1465, loss 0.000241828, acc 1
2018-05-08T17:34:27.104620: step 1466, loss 0.00120163, acc 1
2018-05-08T17:34:27.199404: step 1467, loss 9.83386e-05, acc 1
2018-05-08T17:34:27.296042: step 1468, loss 0.00101699, acc 1
2018-05-08T17:34:27.393950: step 1469, loss 0.00068415, acc 1
2018-05-08T17:34:27.489509: step 1470, loss 0.000889419, acc 1
2018-05-08T17:34:27.583494: step 1471, loss 0.000724424, acc 1
2018-05-08T17:34:27.680234: step 1472, loss 0.000811943, acc 1
2018-05-08T17:34:27.775397: step 1473, loss 0.0006738, acc 1
2018-05-08T17:34:27.871596: step 1474, loss 0.000362353, acc 1
2018-05-08T17:34:27.969195: step 1475, loss 7.17998e-05, acc 1
2018-05-08T17:34:28.062594: step 1476, loss 0.000153269, acc 1
2018-05-08T17:34:28.157793: step 1477, loss 0.000615644, acc 1
2018-05-08T17:34:28.252119: step 1478, loss 0.000163657, acc 1
2018-05-08T17:34:28.349125: step 1479, loss 0.000275892, acc 1
2018-05-08T17:34:28.408568: step 1480, loss 0.00101581, acc 1
Avg training loss 0.159036
Avg training accuracy 0.966128
ubuntu@ip-172-31-18-228:~/second_experiment$

```

Figure 9: Snapshot of Model 2 accuracy when run on 8000 quanta of data

As seen in the above snapshot, we observe that for 8000 sequences of data as the input, Model 2 has an accuracy of 96%.

Figure 10: Model 2 accuracy graph after running on 8000 quanta of data

With the learning graph of the accuracy for the 8000 sequences data, we see that initially there is lot of drop-and-rises, but later on the training curve evens out consistently maintaining an accuracy of 1.0. So, Model 2 is seen to learn and detect new patterns in the 8000 sequences data but only in the initial phases, later on it has learnt all the patterns available, and next incoming data is something similar to what it has seen before or learnt before, leading to a evening out or convergence of the neural network in later epochs of training for later parts of the data. That is very similar to Model 2 behavior when it was given 39,000 sequences of data, except that in that bigger dataset the neural network had encountered some new data gain at the end of the training data. Here, we don't see that happening, so there is no new data that neural network discovers at the end of the overall data sequences. This is an important observation and what it signifies is the information loss that Model 2 has as compared to when Model 2 was run on 39,000 data sequences. Consequently, we see that the accuracy of Model 2 has also reduced to 96% from 98%. So, this proves that more the data, the better the accuracy.

C. Model 2 run snapshots for CO I sequences of total size of 2000 unique sequences

Now, we will see how the Model 2 behaves when we run it on a total data set of 2000 sequences. Following are the snapshots of Model 2 average training accuracy and the learning curve of the neural network.

```

2018-05-08T10:50:23.316702: step 388, loss 0.0266183, acc 1
2018-05-08T10:50:24.566566: step 389, loss 0.0395093, acc 0.99
2018-05-08T10:50:25.652323: step 390, loss 0.0143927, acc 1
2018-05-08T10:50:26.758118: step 391, loss 0.0198533, acc 1
2018-05-08T10:50:27.918093: step 392, loss 0.0261862, acc 0.99
2018-05-08T10:50:29.111303: step 393, loss 0.0259588, acc 1
2018-05-08T10:50:30.204528: step 394, loss 0.0201309, acc 1
2018-05-08T10:50:31.289452: step 395, loss 0.0109055, acc 1
2018-05-08T10:50:32.433806: step 396, loss 0.0214182, acc 1
2018-05-08T10:50:33.636930: step 397, loss 0.0337226, acc 0.99
2018-05-08T10:50:34.803833: step 398, loss 0.0112576, acc 1
2018-05-08T10:50:35.895392: step 399, loss 0.0218639, acc 1
2018-05-08T10:50:36.401293: step 400, loss 0.0163657, acc 1

Evaluation:
2018-05-08T10:50:37.128286: step 400, loss 0.398542, acc 0.875

Saved model checkpoint to /home/saurabh/Downloads/cs298_my_code/cnn-text-classification-tf-master/cs298_code/runs/1525801383/checkpoints/model-400

Avg training loss 0.30432728
Avg training accuracy 0.9166667
saurabh@saurabh-VirtualBox:~/Downloads/cs298_my_code/cnn-text-classification-tf-master/cs298_code$

```

Figure 11: Snapshot of Model 2 accuracy when run on 2000 quanta of data

The Model 2 accuracy is very low when it is run on 2000 sequences of the data. It is only 91.6%

accuracy_1

Figure 12: Model 2 accuracy graph after running n 2000 quanta of data

The training curve graph is very peculiar and interesting for Model 2 when it was trained on only 2000 sequences. As seen here, the neural network only

learned increasing its accuracy after every epoch. This tells us that the initial samples were all very unique and information-rich. Hence, we see that dropout happened only once, as there was positive information gain or knowledge gain thereafter. So, there was no need of any dropout as the weights were changing rapidly, indicating no sign of local minima or maxima phenomena. From this example, we learn that training a neural network with smaller training dataset as batches is more effective as the neural network is less likely to get caught up in local minima or maxima as mostly for the datasets like CO I the initial sequences are going to be very information rich.

Now, Model 1 ad training accuracy of 91% with 200 data sequences, so is true for Model 2 with same amount of data. But, Model 2 accuracy jumped to 96% when trained on 8000 data sequences. And the Model 2 accuracy jumped to 98% magic number when it was trained on all the 39,900 unique sequences. This symbolizes that our machine learning algorithm here i.e. the ANN are really learning patterns with each increase in quantum of data which are impossible to be discovered and remembered by the traditional algorithms like ARBitrator or BLAST.

In conclusion, Model 2 has performed very well.

D. Model 2 run snapshots for when only positive CO I sequences are passed as data


```

Writing to /home/ubuntu/second experiment/runs/1525802749

2018-05-08T18:07:19.248297: step 1, loss 1.45043, acc 0.45
2018-05-08T18:07:19.319892: step 2, loss 3.21256, acc 0.56
2018-05-08T18:07:19.386136: step 3, loss 2.40092, acc 0.48
2018-05-08T18:07:19.454299: step 4, loss 1.11593, acc 0.5
2018-05-08T18:07:19.521726: step 5, loss 1.917, acc 0.52
2018-05-08T18:07:19.588000: step 6, loss 2.33735, acc 0.5
2018-05-08T18:07:19.655906: step 7, loss 1.70437, acc 0.47
2018-05-08T18:07:19.722168: step 8, loss 1.1911, acc 0.41
2018-05-08T18:07:19.788624: step 9, loss 1.35289, acc 0.48
2018-05-08T18:07:19.853368: step 10, loss 1.6135, acc 0.41
2018-05-08T18:07:19.919152: step 11, loss 1.40737, acc 0.49
2018-05-08T18:07:19.985484: step 12, loss 1.05742, acc 0.52
2018-05-08T18:07:20.049769: step 13, loss 1.08733, acc 0.5
2018-05-08T18:07:20.117146: step 14, loss 1.13568, acc 0.52
2018-05-08T18:07:20.185171: step 15, loss 1.05601, acc 0.58
2018-05-08T18:07:20.249554: step 16, loss 0.903079, acc 0.61
2018-05-08T18:07:20.317454: step 17, loss 1.03695, acc 0.41
2018-05-08T18:07:20.384245: step 18, loss 1.11787, acc 0.46
2018-05-08T18:07:20.449902: step 19, loss 0.859048, acc 0.52
2018-05-08T18:07:20.515437: step 20, loss 1.16731, acc 0.47
2018-05-08T18:07:20.581955: step 21, loss 1.21925, acc 0.47
2018-05-08T18:07:20.649120: step 22, loss 0.972309, acc 0.47
2018-05-08T18:07:20.713367: step 23, loss 0.975033, acc 0.49
2018-05-08T18:07:20.780467: step 24, loss 1.15782, acc 0.48
2018-05-08T18:07:20.845893: step 25, loss 1.20332, acc 0.43
2018-05-08T18:07:20.912704: step 26, loss 1.18134, acc 0.44
2018-05-08T18:07:20.978416: step 27, loss 0.959231, acc 0.49
2018-05-08T18:07:21.043758: step 28, loss 0.994432, acc 0.43
2018-05-08T18:07:21.110206: step 29, loss 0.880997, acc 0.55
2018-05-08T18:07:21.174397: step 30, loss 0.994544, acc 0.46
2018-05-08T18:07:21.240736: step 31, loss 1.04972, acc 0.5
2018-05-08T18:07:21.306900: step 32, loss 0.916426, acc 0.42
2018-05-08T18:07:21.374057: step 33, loss 0.854663, acc 0.54
2018-05-08T18:07:21.438883: step 34, loss 0.847965, acc 0.48
2018-05-08T18:07:21.506041: step 35, loss 0.862857, acc 0.48
2018-05-08T18:07:21.571215: step 36, loss 0.841565, acc 0.55
Traceback (most recent call last):
  File "train.py", line 187, in <module>
    x_batch, y_batch = zip(*batch)
ValueError: not enough values to unpack (expected 2, got 0)
ubuntu@ip-172-31-18-228:~/second experiment$

```

Figure 13: Snapshot of Model 2 failed training because of only positive data

This snapshot is central to the project and important. This is the snapshot when the Model 2 is trained on dataset having all the positive sequences of CO I. Please observe the training logs. They show us that after every epoch the neural network is realizing that we are giving it same data with conflicting

labels. So, we see the accuracy of neural network keeps on decreasing and decreasing. Ultimately, the weights are so non-convergent that no matter what the weights wouldn't change in any meaningful direction. Ultimately the program ends with an error that symbolizes the non-convergence of the neural network and hence abort of the training. In effect, Model 2 recognized the fact that we were giving very similar data sequences with conflicting labels. Recall that Model 1 had failed to do so completely. It had still completed its training with accuracy of 50%. But, Model 2 could not even complete the training with conflicting labels. The reason for this strength of the Model 2 lies in its architecture. The Model 1 is stacked layers of convolution filters. Model 2 is parallel executing layers of convolution filters. How this helps Model 2 avoid fooling itself is explained more in subsequent Result and Discussion section. But, as we can clearly observe, Model 2 is better than Model 1, as we could not fool it by giving same data with conflicting labels for training.

E. Model 2 run snapshots for when only negative CO I sequences are passed as data

Following page has a detailed snapshot along with training logs for the Model 2 when it was trained only on negative sequences.

```

2018-05-08T18:11:28.268805: step 27, loss 0.934857, acc 0.477273
2018-05-08T18:11:28.330404: step 28, loss 0.819525, acc 0.6
2018-05-08T18:11:28.392704: step 29, loss 0.899591, acc 0.52
2018-05-08T18:11:28.449082: step 30, loss 1.03878, acc 0.431818
2018-05-08T18:11:28.512871: step 31, loss 0.88438, acc 0.47
2018-05-08T18:11:28.574730: step 32, loss 0.893038, acc 0.49
2018-05-08T18:11:28.633414: step 33, loss 0.911019, acc 0.5
2018-05-08T18:11:28.697276: step 34, loss 0.688368, acc 0.61
2018-05-08T18:11:28.759059: step 35, loss 0.886376, acc 0.46
2018-05-08T18:11:28.815987: step 36, loss 0.770698, acc 0.579545
2018-05-08T18:11:28.876499: step 37, loss 0.871697, acc 0.48
2018-05-08T18:11:28.939633: step 38, loss 0.754307, acc 0.49
2018-05-08T18:11:28.995301: step 39, loss 0.823856, acc 0.477273
2018-05-08T18:11:29.056570: step 40, loss 0.793733, acc 0.56
2018-05-08T18:11:29.117310: step 41, loss 0.864147, acc 0.48
2018-05-08T18:11:29.176384: step 42, loss 0.834908, acc 0.420455
2018-05-08T18:11:29.237444: step 43, loss 0.755082, acc 0.48
2018-05-08T18:11:29.298785: step 44, loss 0.801029, acc 0.56
2018-05-08T18:11:29.354657: step 45, loss 0.786405, acc 0.488636
2018-05-08T18:11:29.415789: step 46, loss 0.727592, acc 0.53
2018-05-08T18:11:29.477252: step 47, loss 0.752023, acc 0.51
2018-05-08T18:11:29.532958: step 48, loss 0.782495, acc 0.488636
2018-05-08T18:11:29.595907: step 49, loss 0.73688, acc 0.48
2018-05-08T18:11:29.657085: step 50, loss 0.759816, acc 0.58
2018-05-08T18:11:29.714015: step 51, loss 0.767977, acc 0.454545
2018-05-08T18:11:29.776281: step 52, loss 0.746998, acc 0.5
2018-05-08T18:11:29.836705: step 53, loss 0.789006, acc 0.5
2018-05-08T18:11:29.891768: step 54, loss 0.663656, acc 0.579545
2018-05-08T18:11:29.951996: step 55, loss 0.734938, acc 0.52
2018-05-08T18:11:30.012056: step 56, loss 0.727893, acc 0.49
2018-05-08T18:11:30.068164: step 57, loss 0.747949, acc 0.477273
2018-05-08T18:11:30.131637: step 58, loss 0.727963, acc 0.58
2018-05-08T18:11:30.193060: step 59, loss 0.749831, acc 0.53
2018-05-08T18:11:30.248883: step 60, loss 0.700491, acc 0.590909
/home/ubuntu/anaconda3/lib/python3.6/site-packages/numpy/core/fromnumeric.py:2889: RuntimeWarning: Mean of empty slice.
  out=out, **kwargs)
/home/ubuntu/anaconda3/lib/python3.6/site-packages/numpy/core/_methods.py:80: RuntimeWarning: invalid value encountered in double_scalars
  ret = ret.dtype.type(ret / rcount)
Avg training loss nan
Avg training accuracy nan
ubuntu@ip-172-31-18-228:~/second experiment$

```

Figure 14: Snapshot of Model 2 failed training because of only negative data

As you can clearly see, the training accuracy keeps on decreasing, till the training weights and weight changes go into negative numbers leading to inconsistency and so the training abruptly ends with NaN (Not a Number) error. This really shows that as compared to Model 1, Model 2 is really very solid in its foundational architecture, with key being that is it wider and not deeper.

This one aspect of its architecture has made it better than Model 1 in terms of instruction execution speed, accuracy, data pattern recognition ability,

avoidance of over fitting and also it is harder and even impossible to fool Model 2. In conclusion, Model 2 has performed very well.

4. Results and Discussions

Key Results:

1. Model 1 had more weights and biases (i.e. fully connected layers after each convolution with RELU activation function). Hence it is 10 times more time consuming than Model 2.
2. Model 2 **is different** to model 1 in architecture.
3. Model 1 has Convolutional filters, followed by RELU activation layer, followed by max pooling layer architecture. So CONV-RELU-POOL is the general neural network architecture. There are 3 such CONV-RELU_POOL architectures that are stacked up one after the other, followed by a fully connected layer
4. Thus, Model 1 is highly sequential in its architecture and very heavy, because it has more weights and biases at each of the convolution filter layer, pooling layer.
5. This is precisely why the execution time for Model 1 is very high, because the design being sequential leaves very little scope for parallelization of the training phase of the neural networks.
6. Contrarily, the Model 2 is wider network, **not as deep as Model 1.**
7. Model 2 has similar CONV_RELU_POOL layers. But, there are 2 key differences.

8. First difference is that Model 2 has the 3 CONV-RELU-POOL layers not stacked up one after the other. Instead, it has them in **parallel**.
9. So, the embedded layer (absent in model 1), is the first layer that stores a normalized embedded nuclear form of input sequences, and these sequences are supplied to all the 3 layers or architectures in parallel.
10. This is precisely why the Model 2 is faster than Model 1.
11. Also, when it comes to adjusting of weights using back propagation (**actual learning process of neural networks**), it happens in parallel for all the CONV-RELU-POOL layers in parallel.
12. But for Model 1, this again as to happen in sequence.
13. This is exactly why execution time per epoch is exponentially high for Model 1 as compared to Model 2
14. Another key difference is that Model 1 has same number of convolution filters in each of the three CONV-RELU_POOL layers.
15. There are 16 convolution filters (a very high number actually) for each CONV-RELU_POOL layer
16. Model 2 on the other hand, has variable number of filters for each CONV-RELU-POOL layers
17. Models 2 has 3 filter for first Convolution layer and 4 filters for next Convolution layer, and 5 filters for the final Convolution layer
18. Model 2 therefore develops different features because of different depth or filter sizes even though it is not as deep as Model 1

19. In observation, it is noteworthy that we have encountered a rare technique where a neural network is more wider than deep, but with number of filters along each convolution, and all of them running in parallel.
20. This type of architecture is new, and is smart, because it yield more feature extraction (pattern extraction) at relatively lesser amount of time pertaining to the parallel aspect of the architecture
21. Both have a fully connected layer at the end of the model
22. Model 1 has [680,2] shaped fully connected layer
23. Model 2 has [384,2] shaped fully connected layer
24. Both the models have same auxiliary nodes and parameters and functions for optimization process
25. Model 2 and Model 1 performed similarly for 2000-pair long sequence dataset
26. Model 2 performed outstandingly with an agreeable performance of accuracy score of 98% when given all the unique sequences in form of 39,900 data. sequences
27. Model 1 could not be run on the bigger dataset as it had higher execution time per training cycle leading to it being requiring at least a week to complete training. Thus Model 1 is example of hyper-engineering in terms of Deep Learning.
28. Model 2 is far better in all aspects than Model 1.

Discussion:

ARBitrator has 99.97% accuracy. The Model 2 has 98% accuracy. The ARBitrator algorithm was developed by Professor Dr. Heller requiring rigorous work for 5 years. Contrary to this, Model 2 was studied, implemented for the CO I data and trained with results in only several months effectively. This is only an initial exploratory work, and the initial 98% accuracy can probably be improved by further work until it is comparable to ARBitrary algorithm's accuracy.

Also, ARBitrator is a traditional model. Model 2 is ANN meaning it requires no explicit instruction to tell it what pattern to recognize and remember to decipher CO I sequences. ANN does it on its own. It is purest form of machine intelligence.

TensorFlow is highly modularized library where we always have to code in form of template usage. So anyone wanting to recreate this model can do it easily. Moreover, most important factor is that all these models including Model 2 have been saved in form a graph which can be reloaded with all the data values for further prediction. (links for pre-trained model are available in the references section). So no retraining is required. Model can be used directly for prediction with error rate of 2%. ANN codebase is smaller than ARBitrator and also TensorBoard is very useful utility that provides visual understanding.

In brief, ARBitrator and Model 2 are fundamentally different. But, the emergence of high computing power with machine learning techniques and algorithms has led to emergence of a new computer science software that is capable of updating itself with

new and new information without hassles of repetitive coding. On the other hand, for BLAST or ARBitrator algorithms, cannot update unless someone programs and changes these algorithms. Also code base of ARBitrator is very heavy. For these reasons, this Model 2 and related study presented through all the methods and results so far will be instrumental to make CO I sequence extraction from GenBank more accurate, hassle-free, autonomous, and machine intelligent.

Hence, this study and its outcomes are comparable to the performance of ARBitrator or BLAST algorithm, especially for CO I sequence detection and extraction.

5. Conclusion

We have implemented an ANN for the first time in the world to recognize the CO I sequences. This was achieved in relatively shorter time of 3 months. Modularity, reusability and provision of retraining is what made the ANN to be developed and retrained for this purpose in such a short period of time.

Further the neural network can be trained using 3 different methods to make it more accurate and useable as an alternative to ARBitrator algorithm

3 ways are:

- Reload the saved Model 2 and train on all the 170,000 sequences increasing the accuracy
- Reload the saved Model2 and add/subtract layers or modify its architecture to attempt to find new architecture guaranteeing higher accuracy
- Reload the weights (W_CN, B_CN) values of final layer alone and changing the dimension of convolutional and pooling layer to form new ANN with better accuracy. This is very famous and prevalent method and is known as Transfer Learning.
- Pretrained Model 2 on 39,900 sequences is stored on GitHub and its link is made available in the References section
- Pretrained Model 1 on 2000 sequences is also made available on the GitHub. Link in the reference section
- Python TenorFlow code to reload these models is also made available on GitHub. Link is in the references section.

Thus the objective setup at start of the project is satisfied and delivered in full.

REFERENCES

- [1] “ARBitrator: A software pipeline for on-demand retrieval of autocuratednifH sequences from GenBank”, Philip Heller, H. James Tripp, Kendra Turk-Kubo, Jonathan P. Zehr, *Bioinformatics Oxford Journal*, November 12, 2016
- [2] “Sequence-based classification significantly improves mining of GenBank for CO I barcode sequences”, Philip Heller, James Casaletto, Gregory M. Ruiz, Jonathan B. Geller (Submitted to *Nature Scientific Data*, in peer review)
- [3] “DNA barcoding for ecologists”, Alice Valentini, Francois Pompanon, Pierre Tabarlet, *Cell Press*, 25 December 2008
- [4] “DNA sequence classification by Convolutional Neural Network”, Ngoc Giang Nguyen, Vu Anh Tran, Duc Loo Ngo, Dau Phan, FavorisenRosykingLumbenraja, Mohammad Reza Faisal, BahridinAbapihi, Mamaru Kubo, Kenji Satou, *Journal of Biomedical Science and Engineering*, Vol. 9 No. 5, April 2016
- [5] “A New method for Species Identification via Protein-Coding and Non-Coding DNA Barcodes by Combining Machine Learning with Bioinformatic Methods”, Ai-bing Zhang, Jie Feng, Robert D. Ward, Ping Wan, Qiang Gao, Jun Wu, Weizhong Zhao, *PLOS Journal*, February 20, 2012
- [6] “Inferring species membership using DNA sequences with back-propagation neural networks”, Zhang AB, Sikes DS, Muster C, Li SQ, *Systematic Biology*, Volume 57, Issue 2, 1 April 2008

- [7] “A C/C++ interface program for species identification via DNA barcoding with a BP-neural Network by calling the Matlab engine”, Zhang AB, Savolinen P, Molecular Recology Resources Journal, Volume 9. Issue 1 , 7 October, 2008
- [8] “A k-mer-based barcode DNA classification methodology based on spectral representation and a neural gas network”, Fiannaca A, La Rosa M, Rizzo R, Urso A, NCBI Literature PubMed, Publication Type: Comparative Study, July 2015
- [9] “A Deep Learning Approach to DNA sequence Classification”, Riccardo Rizzo, Antonino Fiannaca, Massimo La Rosa, Alfonso Urso, International Meeting on Computational Intelligence Methods for Bioinformatics and Biostatistics, July 2016
- [10] “The General Regression Neural Network to classify Barcode and mini-barcode DNA”, Riccardo Rizzo, Antonino Fiannaca, Massimo La Rosa, Alfonso Urso, International Meeting on Computational Intelligence Methods for Bioinformatics and Biostatistics, 18 November 2015
- [11] “Taxonomic Classification for Living Organisms Using Convolutional Neural Networks”, Saed Khawaldeh, Usama Pervaiz, Mohammed Elsharnoby, AlaaEdinAlchalabiaandNayel Al-Zubi, Genes Journal, Volume 8, Issue 11, 17 November 2017
- [12] "Convolutional Neural Networks for Sentence Classification", Yoon Kim, Computer Science Department, Cornell University Library, 3 September 2014
- [13] Model 2 based on [12] : <http://www.wildml.com/2015/12/implementing-a-cnn-for-text-classification-in-tensorflow/>
- [14] GitHub link for base architecture of Model 2 under Apache License for free

private use : <https://github.com/dennybritz/cnn-text-classification-tf>

[15] Model 1 is based on [4]. Model 1 is modified architecture of Model 2 and

reference [13]. Model 1 GitHub link for base architecture is :

<https://github.com/tariqul-islam/DNA-Sequence-Classification-using-CNN>

[16] Reference GitHub link for code on how to reload a pretrained TensorFlow

model :

https://github.com/SAURABHMARATHE/cs298_code_for_reloading_tf_model