Implementation of a TensorFlow Convolutional Neural Network to Discriminate Similar Gene Functions

Rory Butler, Ralph Butler[1], and Chrisila Pettey
Department of Computer Science, Middle Tennessee State University
Murfreesboro, TN, USA
roryb@me.com, ralph.butler@mtsu.edu[1], and chrisila.pettey@mtsu.edu
[1]Contact Author

Abstract *- Convolutional neural networks are used widely in natural language processing [2, 3], image recognition [4], and recommender systems [9]. And, while deep NNs have been used for protein function prediction [1, 5], we present a novel method of using a convolutional neural network to discriminate among a set of similar protein-encoding genes from amino acid sequences. Our CNN uses a method often seen in those used for image recognition, i.e. it applies a sliding window-like technique to obtain convolutions that can be pooled before fully-connected layers. However, there are no images here, only one-hot representations of amino acids that form genes with interesting functions. This method had a 97% accuracy rate on our primary validation set and compares favorably to recurrent NNs and ordinary (i.e., not convolutional or recurrent) deep NNs which we developed for comparison.*

Keywords: Neural Networks, Convolutional Neural Networks, Recurrent Neural Networks, TensorFlow, Gene Function

Submission Type: Regular Research Paper

1 Introduction

One aspect of previous research we have done involved determining gene function as part of genome annotation [6]. An approach that we sometimes used was the k-mer (specifically 6-mer) approach. One problem that evolved from that research is that of discriminating among smaller sets of similar protein-encoding genes. Ordinarily we would simply develop some new algorithm using the old 6-mer ideas. However, we have some experience with machine learning (among other things, with scikit-learn [7]), and we were also interested in investigating TensorFlow [8]. That fact coupled with inspiration derived from the use of convolutional neural networks (CNNs) in the area of image recognition led to the results reported in this paper.

In the remainder of this paper we discuss a novel way of applying a CNN to the problem of discriminating among a set of similar protein-encoding genes from amino acid sequences. Section 2 contains a description of the CNN method, the experiments, and the results. A comparison of the CNN results to both a recurrent neural network (RNN) and an ordinary deep neural network (ODNN) - i.e., one that is neither convolutional nor recurrent - is given in section 3.

All three neural networks presented in this paper were TensorFlow-based NNs. The set of similar protein-encoding gene functions of interest to us is given in Figure 1. All tests involved 433 genes with the given functions. 333 of the genes were used for training, and 100 genes were used for validation.

Sodium-transporting ATPase subunit A
Sodium-transporting ATPase subunit B
Sodium-transporting ATPase subunit C
Sodium-transporting ATPase subunit D
Sodium-transporting ATPase subunit E
Sodium-transporting ATPase subunit F
Sodium-transporting ATPase subunit G
Sodium-transporting ATPase subunit Q
Sodium-transporting ATPase subunit R

Figure 1. The set of similar protein-encoding gene functions used in all discrimination experiments.

2 The CNN

In our investigation of CNNs we realized that the CNN notions that are applicable to image recognition, might have a useful analog to our problem, even to the extent of somewhat utilizing the old idea of the 6-mers. In particular we refer to the idea of the *local receptive field (patch)* for the hidden neurons. We like to think of the combination of the hidden neurons and their local receptive fields as a sliding window with a fixed height and width that moves successively through the data with a particular stride. Figure 2 shows a typical setup where there is a 5x5 sliding window over a 32x32 image of a robot. The figure also depicts a common scenario of multiple convolutions, max-pooling, and fully-connected layers.

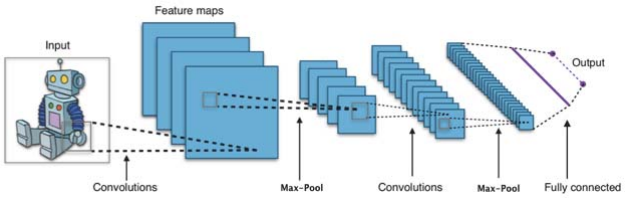

Figure 2. Typical CNN [10]

The sliding window concept is particularly appealing in the context of searching through strings of amino acids for substrings of length 6 (i.e., 6-mers). To further adapt the data to the sliding window concept, the amino acids were encoded using one-hot notation. One-hot is a method for encoding data such that each item is represented as a string of 0's with a single 1. Table 1 shows a comparison of a one-hot encoding vs. a binary encoding of the decimal digits 0 through 7. For our data, each amino acid can be represented as a one-hot encoding using 20 digits. Since a gene is represented as a string of amino acids, the input to the CNN is an array of 20 digits x N digits where N is the length of the gene (i.e., the number of amino acids). Specifically, each column represents a single amino acid in one-hot encoding (see Figure 3).

Table 1. Possible one-hot encoding vs. binary encoding for the digits 0 - 7.

digit	binary encoding	one-hot encoding
0	000	10000000
1	001	01000000
2	010	00100000
3	011	00010000
4	100	00001000
5	101	00000100
6	110	00000010
7	111	00000001

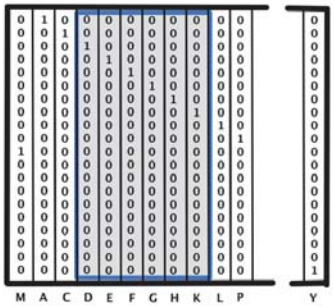

Figure 3. 20x6 sliding window over one-hot amino acids

Since we typically search for substrings of length 6, the local receptive field for the hidden neurons was 20x6 with a stride of one. Figure 3 shows the 6-wide sliding window over the 20-high one-hot values. The CNN had two convolutional layers, each with 2x2 max-pooling. This was followed by two fully-connected layers which used the relu (Rectified Linear Unit) activation function. It used the mean of *softmax_cross_entropy* as the loss function and a learning rate of 0.0001 in an *Adam* optimizer. After 300 epochs of training (approximately three minutes), the CNN achieved a 97% accuracy rate on the validation set.

To construct a CNN using TensorFlow requires several TensorFlow procedures. We used the following:

- To construct new weights we used *tf.Variable*(*tf.truncated_normal(shape, stddev=0.05))*
- To construct new biases we used *tf.Variable(tf.constant(0.05, shape=[length]))*
- To build an enhanced convolutional layer we needed the following three:
  tf.nn.conv2d()
  tf.nn.max_pool()
  tf.nn.relu()
- To reshape convolutional results for fully connected layers we used *tf.reshape()*
- To build each fully connected layer we used *tf.matmul()* and *tf.nn.relu()*
- To turn on various options we used some subset of the following:
  tf.nn.softmax()
  tf.argmax()
  tf.nn.softmax_cross_entropy_with_logits()
  tf.train.AdamOptimizer().minimize()
  tf.equal()
  tf.reduce_mean()

Figure 4 contains the pseudo-code for training a CNN in TensorFlow. Note that TensorFlow does not actually run a neural network until you create a *Session* and *run* it. You can also access intermediate values being computed by the graph through the Session. TensorFlow provides a facility called TensorBoard that will provide visualizations of the flowgraph, histograms, etc., if you correctly instrument the code. For those who are interested, Figure 5 is the TensorBoard flowgraph representation of our CNN.
The success of the CNN led us to consider whether or not an ODNN or a RNN would perform better. A discussion of the results of those two implementations are presented in Section 3.

```
# basic TensorFlow algorithm
# set up graph (see Table 2)
# create a session and initialize - you must have a session
   session = tf.Session()
   session.run(tf.global_variables_initializer())
# run the epochs
   for epoch in range(num_epochs):
      session.run(optimizer, ...)
      #every 10th epoch validate and checkpoint
         if (epoch % 10) == 0:
               session.run(accuracy, ...)
               saver.save(session, ...)
```

Figure 4. Pseudo-code for training a CNN in TensorFlow

3 ODNN and RNN Results

As mentioned previously, all tests involved 433 genes with the functions given in Table 1. 333 of the genes were used for training, and 100 genes were used for validation. Additionally, we considered amino acid strings of length 6 (6-mers).

3.1 ODNN

For this set of genes, the number of actual 6-mers is 75,000. Thus, there were 75,000 inputs to the ODNN. While this would normally be considered a very large number of inputs for a neural network, our ODNN was able to handle it.

For this experiment, the ODNN was built of five layers, each using the *tanh* evaluation function, and each employing a 0.5 drop-out strategy during training. It used the mean of *softmax_cross_entropy* as the loss function, and a learning rate of 0.001 in an *Adam* optimizer (see Figure 6 for the TensorBoard flowgraph of the ODNN). After 300 epochs of training (about 20 minutes), it was able to achieve 99% accuracy on the validation set.

It should be noted that this particular approach will, in general, be intractable due to the 64,000,000 possible 6-mers that could be required for input for larger problems.

3.2 RNN

Admittedly an RNN is probably not the right tool for this problem, but for completeness sake, we investigated it as well. Our RNN was built with a layer of 128 *BasicLSTMCell's* (lstm - long short-term memory). It used the mean of *softmax_cross_entropy* as the loss function, and a learning rate of 0.001 in an *Adam* optimizer. After 2400 epochs of training (about three hours), it was only able to achieve 89% accuracy on the validation set.

In an attempt to improve the accuracy rate, we tried other options. Switching the *BasicLSTMCell* to a wrapping of *LSTMCell*, *DropoutWrapper*, and *MultiRNNCell*, did not help, nor did the use of *xavier* weights. We also tried adjusting various options to the *Adam* optimizer (e.g., learning rate, beta1, beta2, epsilon) and eventually achieved a 90% accuracy rate on the validation set. However, the 90% rate took slightly more than 4900 epochs (14 hours).

Given the lower accuracy, and the much longer training time, we do not believe a RNN is appropriate for this particular problem.

4 Conclusions

In this paper we presented a novel way of applying a CNN to the problem of discriminating among a set of similar protein-encoding genes from amino acid sequences. The technique had a 97% success rate on the validation set after approximately three minutes of training. By comparison, the ODNN achieved a success rate of 99% after a similar amount of training. While the success rate of the CNN was not as good as the ODNN success rate, the ODNN's high number of inputs (75,000 on a small problem to as many as 64,000,000 on a bigger problem) make them too complex of a model to be considered for future work. We also investigated a RNN, but multiple adjustments could produce no more than 90% accuracy with 14 hours of training.

5 References

[1] Cerri, R., and deCarvalho, A., Hierarchical Multilabel Protein Function Prediction Using Local Neural Networks," *Proceedings of the Brazilian Symposium on Bioinformatics*, 2011, pp. 10 - 17.

[2] Kalchbrenner, N., Grefenstette, E., and Blunsom, P., "A Convolutional Neural Network for Modelling Sentences," *Proceedings of ACL 2014: The 52nd Annual Meeting of the Association for Computational Linguistics*, 2014, pp. 655 - 665.

[3] Kim, Y., "Convolutional Neural Networks for Sentence Classification," *Proceedings of EMNLP 2014: Conference on Empirical Methods in Natural Language Processing*, 2014, pp. 1746 - 1751.

[4] Krizhevsky, A., Sutskever, I., and Hinton, G., "ImageNet Classification with Deep Convolutional Neural Networks," *Proceedings of Advances in Neural Information Processing Systems 25 (NIPS 2012)*, 2012.

[5] Murvai, J. Vlahoviček, K., Szepesvári, C., Pongor, S., "Prediction of Protein Functional Domains from Sequences Using Artificial neural networks," *Genome Research* 11:1410-1417, 2001.

[6] Overbeek, R., et. al., "The Subsystems Approach to Genome Annotation and its Use in the Project to Annotate 1000 Genomes," *Nucleic Acids Research*, (2005) 33 (17): 5691-5702.

[7] scikit-learn: Machine Learning in Python http://scikit-learn.org/stable/ last accessed 3/16/17

[8] TensorFlow https://www.tensorflow.org/ last accessed 3/16/17

[9] Van den Oord, A., Dieleman, S., Schrauwen, B., "Deep Content-based Music Recommendation," in *Proceedings of the Neural Information Processing Systems Conference*, 2013.

[10] Wikimedia Commons *File:Typical cnn.png* https://commons.wikimedia.org/wiki/File:Typical_cnn.png

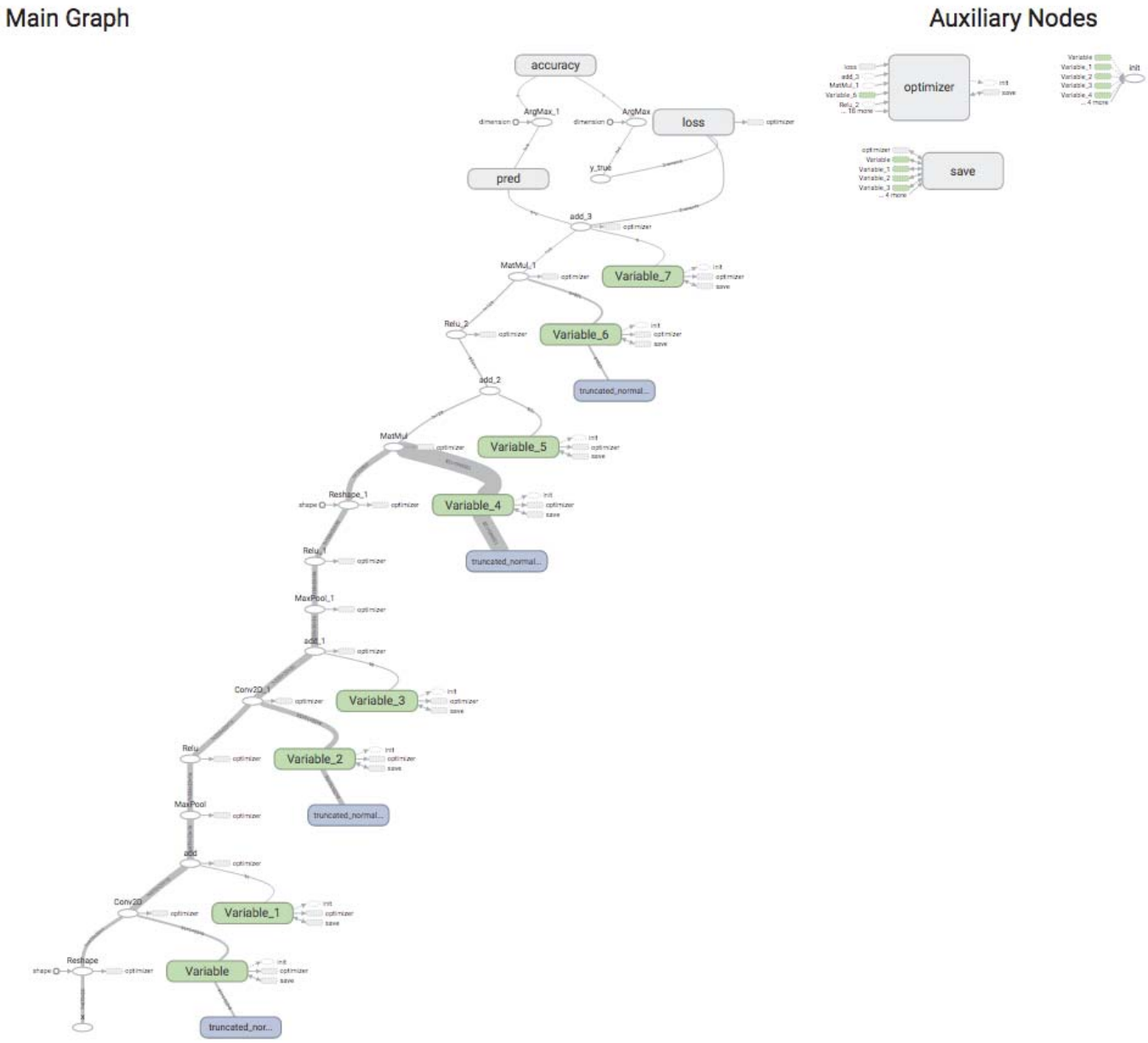

Figure 5. TensorBoard visualization of the flow graph for our CNN

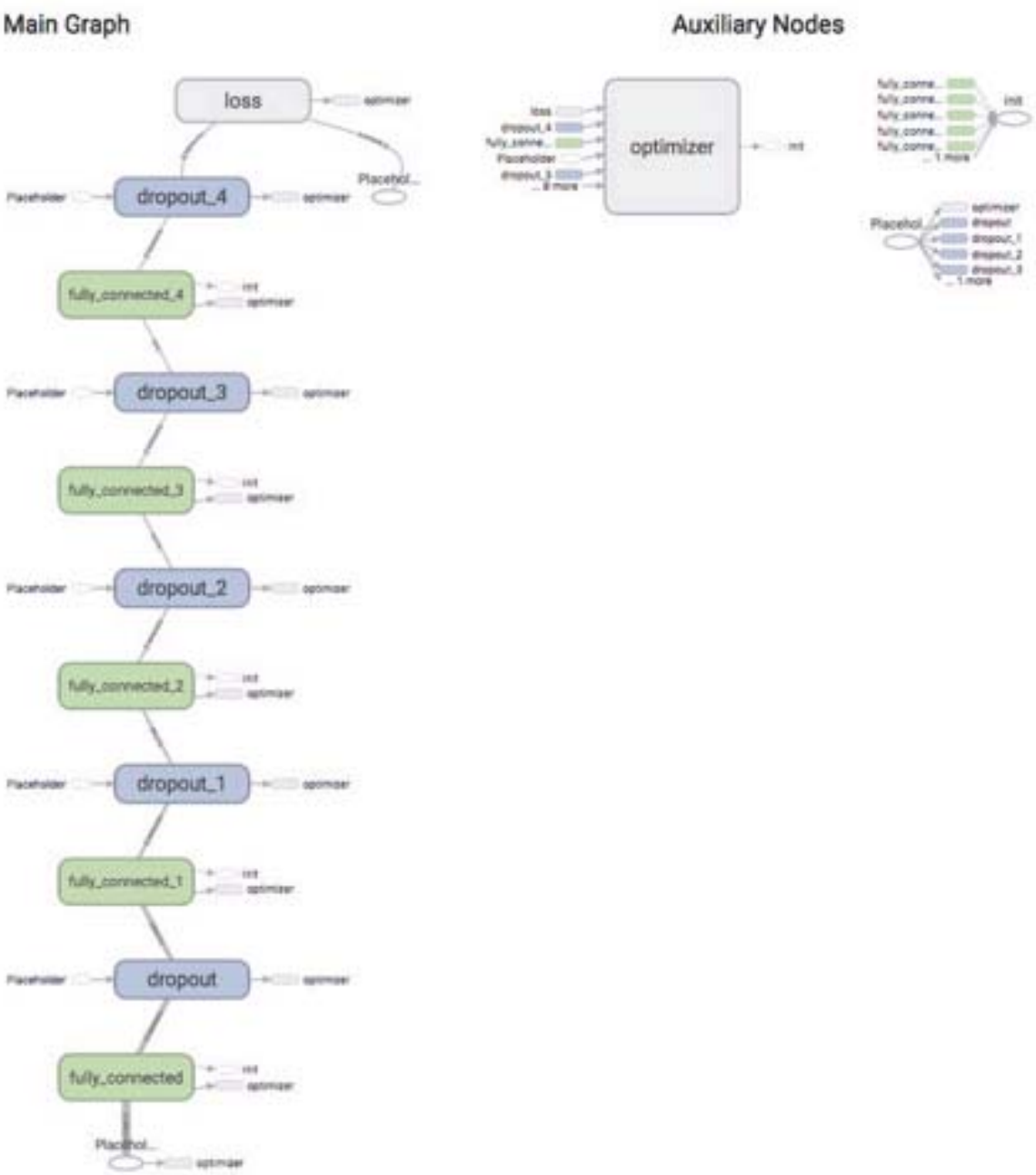

Figure 6. TensorBoard visualization of the flowgraph for our ODNN