Gene Ontology Classification of Protein Sequence Using Fully Connected Neural Networks

Safia Fatima

December 10, 2017

1 Introduction

There are many techniques available in Bioinformatics which are used to annotate the non-classified Protein Sequences in a given specie. One way to do this task in an efficient way is to utilize the power of Neural Networks on such complex data so that it can generalize any type of sequences in the future for prediction of their Gene Ontology (GO) classes .

2 Problem in Simple Neural Network

The main issue in simple Neural Networks is they need hand-made features against each example of any dataset. Also the network doesn't have the ability to learn the model well on high complex data.

That is why these days a variant of Neural Network i-e Deep Learning is used which can take the feature extraction part from human as well as provides more options to add in the architecture for bigger datasets.

3 Dataset

The dataset contain two files: one for the Protein IDs against their GO classes named as "AllProteinswithFunctions-Bakers Yeast.txt" and another file for providing the sequences against each protein ID named as "YEAST.fasta".

The first file contains 9368 Proteins along with their GO functions, and after the parsing of both files based on requirements mentioned below they will be equal to 3351 Proteins with their sequences and GO classes.

4 Parsing

The Parsing is done in 'Parsing.py' script where the both files are joined together in a single DataFrame using pandas library after multiple steps.

```
def ExtractingSequences(m):
    count=0
    pp=0
    ind=0
    f = open('sequences.txt','w')
    infile = open(m, 'r')
    text = infile.readlines()
    for j in (text):

        pp=pp+1
        if any(word in j[4:10] for word in lines):
            f.write(j[4:10])
            f.write(',')
            count=count+1
            ind=pp
            for i in range(100):

                if (text[ind][0]==">"):
                    break
                else:
                    f.write(text[ind].rstrip('\n'))
                ind=ind+1
            f.write('\n')
    f.close()
    print (count)
ExtractingSequences("YEAST.fasta")
```

Figure 1: Extract Sequences from YEAST.fasta

```
#Reading Bakers.csv into pandas Dataframe
with open('Bakers.csv') as f:
    df = pd.DataFrame(
        [[x, y] for x, *ys in map(m, f.readlines()) for y in ys if y],
        columns=['ProteinID', 'Class']
    )
```

Figure 2: Read Classes per Instance

```
#Extracting top 20 frequenct classes (in our case its 21)
countt=df_new['Class'].value_counts()
#Choose all Classes above frequency of 54
df_neww=df_new.groupby('Class')['ProteinID','Sequence','Class'].filter(lambda x: len(x) > 54)
df_neww = df_neww.reset_index(drop=True)
```

Figure 3: Picking top 20 Frequent Classes

The Script is documented in detail if someone wants to learn the steps of parsing both files successfully.

After parsing the file is saved in the Class and Sequence format and the classes are saved in separate file named classes.txt from where they are referred as integer index values. Finally the dataset is divided into train.csv and test.csv for Deep Learning.

5 Word Embeddings

The embedding I have used for now is one hot vector encoding. In this case the encoding will be done on character level for each sequence and vector length will be equal to the number of unique amino-acids+indel which is 23. This process is called Quantization which is used in [1] and shown in Figure 4 as well.

```
with tf.name_scope("Embedding-Layer"), tf.device('/gpu:0'):
    #Quantization layer

    Q = tf.concat(
                  [
                      tf.zeros([1, alphabet_size]), # Zero padding vector for out of alphabet characters
                      tf.one_hot(list(range(alphabet_size)), alphabet_size, 1.0, 0.0) # one-hot vector representation for alphabets
                  ],
                  0,
                  name='Q')

    x = tf.nn.embedding_lookup(Q, self.input_x)
    x = tf.expand_dims(x, -1) # Add the channel dim, thus the shape of x is [batch_size, l0, alphabet_size, 1]
```

Figure 4: Convolution filters and Hidden Layers

Using such embedding scheme one can treat a sequnce as a combination of encoded vectors so that they can be passed to Convolutional Neural Network.

6 Architecture

The architecture I tried to explore for this scenario was Convolutional Neural Networks (CNN) which is widely used these days for text classification of sentences. The hurdle was interpreting the sequences instead of sentences. Sequence is a long string having no multiple words while in sentence multiple words can form vector embeddings to a matrix level easily. To achieve the same in here there is been work going on for sequence or characters classification. The idea is recently implemented in [1] where the whole CNN architecture as well as desired filters are used for character level text classification. I have modified the model according to out dataset and transferred the architecture to Tensorflow for project requirements as well as learning aspect.

7 Hyperparameters

There were several hyperparameters tried for the Neural Network before it starts learning.

- p = 0.5 (Dropout Probabiility)

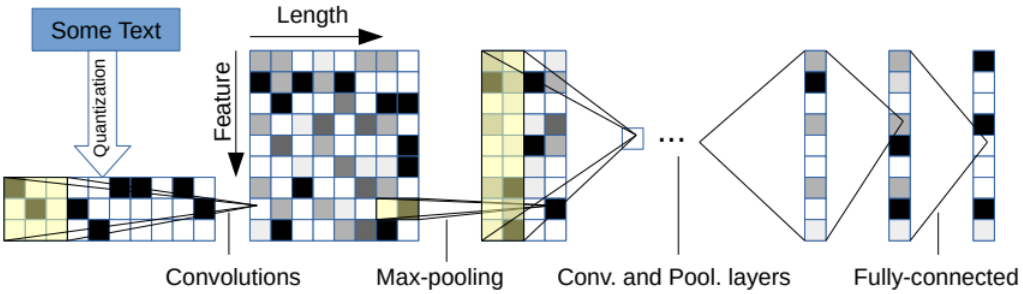

Figure 5: Architecture

- base_rate = 1e-2 (for Learning Rate)
- momentum = 0.9
- epoches = 1000
- evaluate_every = 100 (for evaluating test.csv data according to batch size)
- checkpoint_every = 100 (for saving the model file)

The filters also called Convolutions were chose provided in [1] and show in Figure 6. There are six filters used in Convolutions and after the max pooling of these filter maps the final vector is passed to the two hidden layered fully connected architecture.

```
class ModelConfig(object):
    conv_layers = [[256, 7, 3],
                   [256, 7, 3],
                   [256, 3, None],
                   [256, 3, None],
                   [256, 3, None],
                   [256, 3, 3]]

    fully_connected_layers = [250, 200]
    th = 1e-6
```

Figure 6: Convolution filters and Hidden Layers

Output layer will be equal to the number of classes which is 20 in our model.

The other configurations of the model are shown in Figure 7. The alphabets are amino-acids present in the sequences. The length of sequence is taken 200 although the length of sequences in dataset is not equal. Batch size is taken either 64, 128 or 512 interchangeably for testing purposes.

```
alphabet = ['_', 'A', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'K', 'L', 'M',
            'N', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y']
alphabet_size = len(alphabet)
l0 = 200
batch_size = 128
no_of_classes = 21

train_data_source = 'data/ag_news_csv/train.csv'
dev_data_source = 'data/ag_news_csv/test.csv'

training = TrainingConfig()

model = ModelConfig()
```

Figure 7: Basic Configurations

8 Results

Multiple tries have been made for the improvement of model evaluation metrics but the results were not good. The results are been displayed in Figure 8. The overall final

	precision	recall	f1-score	support
class 1	0.19	1.00	0.32	98
class 2	0.00	0.00	0.00	36
class 3	0.00	0.00	0.00	60
class 4	0.00	0.00	0.00	9
class 5	0.00	0.00	0.00	54
class 6	0.00	0.00	0.00	2
class 7	0.00	0.00	0.00	85
class 8	0.00	0.00	0.00	3
class 9	0.00	0.00	0.00	1
class 10	0.00	0.00	0.00	14
class 11	0.00	0.00	0.00	5
class 12	0.00	0.00	0.00	4
class 13	0.00	0.00	0.00	3
class 14	0.00	0.00	0.00	7
class 15	0.00	0.00	0.00	2
class 16	0.00	0.00	0.00	4
class 17	0.00	0.00	0.00	38
class 18	0.00	0.00	0.00	37
class 19	0.00	0.00	0.00	10
class 20	0.00	0.00	0.00	40
avg / total	0.04	0.19	0.06	512

Figure 8: Final Model Evaluation

accuracy of the test data was 26%. The influence of top frequent GO term was on the model. Also the padding was not implemented between this attempt to see the difference in results. The hidden layers have been changed between range of 100 to 500 nodes per layer and also the learning rate, dropout, Optimizer changed to Momentum and Adam, and length of sequence have been altered but the results were not affected.

9 Improvement

Due to shortage of time and lack of practice for problem I wanted to try various things regarding improvement of results. There are few areas which can be looked into to explore more about improving the final results.

1. Change of architecture to Keras and Theano along with the usage of GPU instance.
2. Using Word2vec and other variants of encoding a sequence.
3. Change of Convolution filters.
4. Learning more about the use of hyperparameters.
5. Implementing LSTM architecture.
6. Usage of padding in the sequences.

References

[1] Xiang Zhang, Junbo Jake Zhao, and Yann LeCun. Character-level convolutional networks for text classification. *CoRR*, abs/1509.01626, 2015.