Anirudh Krishna L.
2015A7PS0081P
Venkateshwaran.M
2015A7PS0122P

Adhithya wagh 2015A3PS0382P
Pranay K. Ganguly 2015A7PS0124P

Variable Computation in Recurrent Neural Networks

Link to the Paper:https://arxiv.org/pdf/1611.06188.pdf

WORK DONE

Variable Computation in Recurrent Neural Networks

Implemented a Elman Recurrent Neural Network.
Implemented a Variable Computation variant of Elman RNN using softmask.
Results indicating the improvement in character prediction.

Methodology and pipeline

- A pre-processed dataset was acquired from pen-treebank.
- Converting all characters to lowercase.
- Made a dictionary of characters in the pen-treebank and Id’s are assigned to each character.
- Made the elman rnn class with its weights, biases and a function for each epoch.
- The function for running the elman rnn
- Modification to the elman to make it use lesser no. of hidden nodes.

Methodology and pipeline

- Evaluating the results of elman rnn and its variant.
- It is being evaluated for different dimensions of the hidden layer.
-

Source of data

The Pen-treebank
The testdatabase contains 6M characters which are being trained in batches.

Software/Framework

python
Numpy

Implementation of Paper

The part of paper implemeted was the elman rnn, doing character level prediction, using the Pen-treebank and the Variable computation variant.

Elman RNN Back propogation

```
    def train_inst (self, x, y):
        x_comp = np.dot(self.wx, x)
        h_comp = np.transpose(self.u) * self.ht_new
        self.ht_old = self.ht_new
        comp = x_comp + h_comp + self.b
        self.ht_new = np.tanh(comp)
        y_pred = sigmoid(np.dot(self.wy, self.ht_new))

        d_out = (y - y_pred) * (1 - y_pred) * (y_pred)
        del_wy = np.dot(d_out, np.transpose(self.ht_new)) * self.lc
        self.wy += (self.gamma * self.del_wy) + del_wy
        self.del_wy = del_wy

        d_h = np.dot(np.transpose(self.wy), d_out) * (1 - self.ht_new ** 2)
        del_wx = np.dot(d_h, np.transpose(x)) * self.lc
        del_b = d_h * self.lc
        del_u = (d_h * self.ht_old)
        del_u = del_u * self.lc
        self.u += ((self.del_u.T) * self.gamma) + del_u.T
        self.wx += (self.del_wx * self.gamma) + del_wx
        self.b += (self.del_b * self.gamma) + del_b
        self.del_wx = del_wx
        self.del_b = del_b
        self.del_u = del_u

        #print(np.sum(np.absolute(del_u)))
        #print(np.sum(np.absolute(del_wy)))
        #print(np.sum(np.absolute(del_b)))
```

VCRNN Back propogation (Using Softmask)

```
    def train_inst (self, x, y):
        mt = 1.0/(1.0+np.exp(-(np.dot(self.param_u, self.ht_new) + np.dot(self.param_v, x)+ self.param_b)))
        etx = sigmoid(np.array([self.sh * (mt*len(x) - ind) for ind in range(0, len(x))]))
        htx = sigmoid(np.array([self.sh * (mt*len(self.ht_new) - ind) for ind in range(0, len(self.ht_new))]))
        etx = np.array([[thres(a, self.ep) for a in etx]]).T
        htx = np.array([[thres(a, self.ep) for a in htx]]).T
        x = etx * x

        x_comp = np.dot(self.wx, x)
        self.ht_old = self.ht_new
        self.ht_new = htx * self.ht_new
        h_comp = np.transpose(self.u) * self.ht_new
        self.ht_new = (np.tanh(x_comp + h_comp + self.b)) * htx + (1 - htx) * self.ht_old
        y_pred = sigmoid(np.dot(self.wy, self.ht_new))

        d_out = (y - y_pred) * (1 - y_pred) * (y_pred)
        del_wy = np.dot(d_out, np.transpose(self.ht_new)) * self.lc
        self.wy += (self.gamma * self.del_wy) + del_wy
        self.del_wy = del_wy

        d_h = np.dot(np.transpose(self.wy), d_out) * (1 - self.ht_new ** 2)
        del_wx = np.dot(d_h, np.transpose(x)) * self.lc
        del_b = d_h * self.lc
        del_u = (d_h * self.ht_old)
        del_u = del_u * self.lc
        self.u += ((self.del_u.T) * self.gamma) + del_u.T
        self.wx += (self.del_wx * self.gamma) + del_wx
        self.b += (self.del_b * self.gamma) + del_b
        self.del_wx = del_wx
        self.del_b = del_b
        self.del_u = del_u
```

Result

Elman RNN (At the end of 100 batches)

D = 32
Bpc = 5.04

```
Select C:\Windows\System32\cmd.exe
D:\STUDIES\Code\python\NNFL\NNFL>python Assignment.py
ElmanRNN
No of batches = 100
Batch - 0
Checking
305606 out of 425975 are incorrect!
Batch - 10
Checking
284099 out of 425975 are incorrect!
Batch - 20
Checking
278286 out of 425975 are incorrect!
Batch - 30
Checking
276212 out of 425975 are incorrect!
Batch - 40
Checking
283548 out of 425975 are incorrect!
Batch - 50
Checking
274673 out of 425975 are incorrect!
Batch - 60
Checking
271810 out of 425975 are incorrect!
Batch - 70
Checking
269940 out of 425975 are incorrect!
Batch - 80
Checking
271622 out of 425975 are incorrect!
Batch - 90
Checking
272128 out of 425975 are incorrect!
268815 out of 425975 are incorrect!

D:\STUDIES\Code\python\NNFL\NNFL>
```

D = 96
Bpc = 5.03

```
C:\Users\venky\Anaconda3\python.exe D:/STUDIES/Code/python/NNFL/NNF
ElmanRNN
No of batches = 100
Batch - 0
Checking
313282 out of 425975 are incorrect!
Batch - 10
Checking
287942 out of 425975 are incorrect!
Batch - 20
Checking
276736 out of 425975 are incorrect!
Batch - 30
Checking
274614 out of 425975 are incorrect!
Batch - 40
Checking
274990 out of 425975 are incorrect!
Batch - 50
Checking
272925 out of 425975 are incorrect!
Batch - 60
Checking
271644 out of 425975 are incorrect!
Batch - 70
Checking
268379 out of 425975 are incorrect!
Batch - 80
Checking
266486 out of 425975 are incorrect!
Batch - 90
Checking
272007 out of 425975 are incorrect!
268015 out of 425975 are incorrect!

Process finished with exit code 0
```

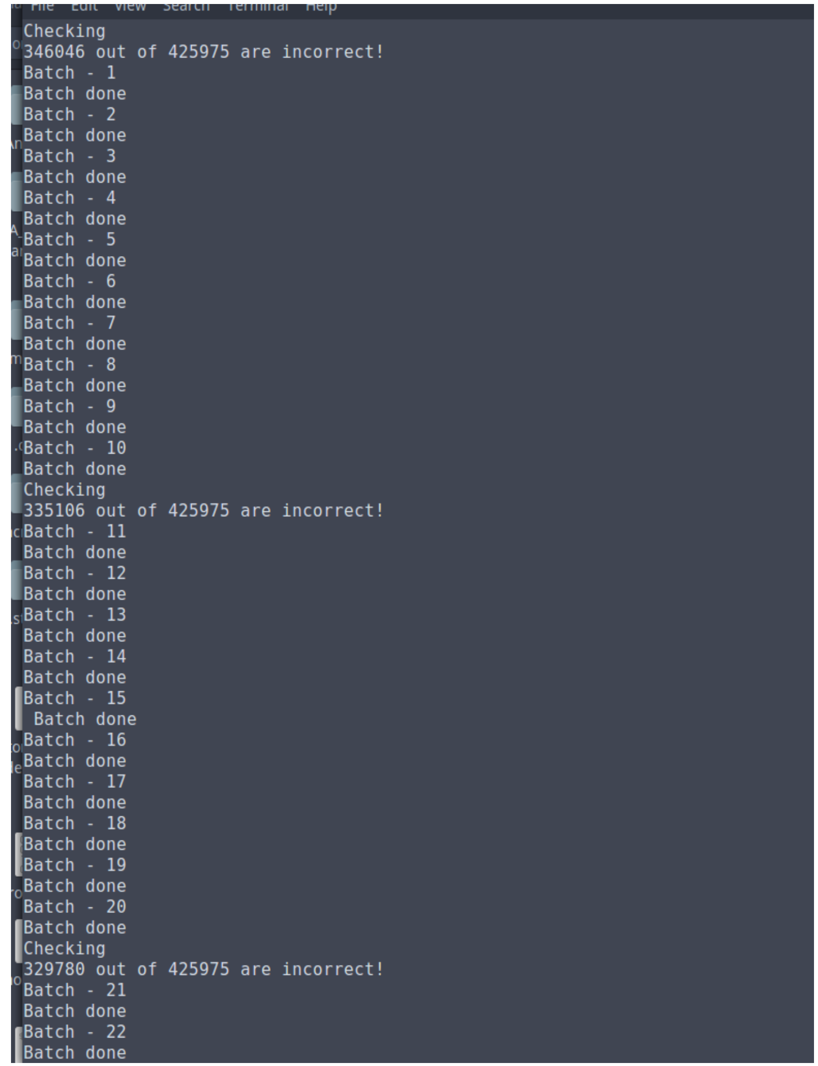
D = 256
Bpc = 5.07

```
ElmanRNN
No of batches = 100
Batch - 0
Checking
337959 out of 425975 are incorrect!
Batch - 10
Checking
307927 out of 425975 are incorrect!
Batch - 20
Checking
300124 out of 425975 are incorrect!
Batch - 30
Checking
291795 out of 425975 are incorrect!
Batch - 40
Checking
289002 out of 425975 are incorrect!
Batch - 50
Checking
290143 out of 425975 are incorrect!
Batch - 60
Checking
282501 out of 425975 are incorrect!
Batch - 70
Checking
278274 out of 425975 are incorrect!
Batch - 80
Checking
274875 out of 425975 are incorrect!
Batch - 90
Checking
271430 out of 425975 are incorrect!
271676 out of 425975 are incorrect!

Process finished with exit code 0
```

Result

Variable Computation Rnn
(Faster)
D = 32
Bpc = 5.19
(At the end of 100 batches)



Limitations, improvements and futurework

The softmask parameters are randomized rather than being learnt via back-propagation
Momentum has been added and learning curve I being reduced overtime to allow to settle at maxima.
The code can be improved for efficiency.
Performance can be further improved by limiting back-propagation based on softmask.

References

PTB dataset - https://github.com/tomsercu/lstm/tree/master/data
Paper - https://arxiv.org/pdf/1611.06188.pdf
Numpy docs
Python docs
Elman RNN - http://mnemstudio.org/neural-networks-elman.htm