Soynet Week 4

2022312686

The Rise of Deep Learning

Deep-Learning & Backpropagation

MLP & Backpropagation

- Perceptron(Weight, Bias) ➔ AND, OR (Solved) XOR (?)
- Marvin Minksy ➔ Not by Single Perceptron, Maybe Multilayer?
  - But how to train MLP? (1st crisis)
- Solution ➔ Backpropagation (역전사) & Convolutinal Neural Networks (분할하여 인식)
  - Scalability Issue (A Big Problem) → Backpropagation didn’t work well with lower layers (2nd crisis)
- Weight initialization in a clever way

Deep Learning

- Neural Network ➔ Deep Learning
    - ImageNet Classification
    - Explain Photo
    - Deep API Learning
    - Speech recognition
    - Etc

Lab 09-00

Tensor manipulation

Neural Network

Introduction

Back Propagation

- $f = wx + b$ ➔ $\frac{\delta f}{\delta w} = x, \frac{\delta f}{\delta x} = w, \frac{\delta f}{\delta b} = 1$
- Chain Rule
    - $f = wg + b,\ \frac{\delta f}{\delta w} = \frac{\delta f}{\delta g} * \frac{\delta g}{\delta w}$ Utilize previous 'local' value.
    - Tensor Flow use Chain Rule for Back Propagation.
        - Follow the Multiple Tensors …
        - $\frac{\delta Cost}{\delta w_0} = \frac{\delta Cost}{\delta x_n} \times \frac{\delta x_{n-1}}{\delta x_{n-2}} \times \cdots \times \frac{\delta x_1}{\delta x_0}$

Lab 09-01

Neural Net for XOR

Single Layered NN

Rate ➔ 0.5 ...

```
[11] X = tf.placeholder(tf.float32)
     Y = tf.placeholder(tf.float32)
     W = tf.Variable(tf.random_normal([2,1]),name='weight')
     b = tf.Variable(tf.random_normal([1]),name='bias')

[13] hypothesis = tf.sigmoid(tf.matmul(X,W)+b)

[14] cost = -tf.reduce_mean(Y*tf.log(hypothesis)+(1-Y)*tf.log(1-hypothesis))
     train = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)

[16] predicted = tf.cast(hypothesis>0.5, dtype=tf.float32)
     accuracy = tf.reduce_mean(tf.cast(tf.equal(predicted, Y), dtype=tf.float32))

[18] with tf.Session() as sess:
       sess.run(tf.global_variables_initializer())

       for step in range(10001):
         sess.run(train, feed_dict={X: x_data, Y: y_data})
         if step %100 == 0:
           print(step, sess.run(cost, feed_dict={X: x_data, Y: y_data}))

       h, c, a = sess.run([hypothesis, predicted, accuracy],feed_dict={X: x_data, Y: y_data})
       print("\nHypothesis: ",h,"\nCorrect:",c, "\nAccuracy", a)
```

Multi Layered NN

Rate ➔ 1.0 !
Careful with Input Size

```
[25] W1 = tf.Variable(tf.random_normal([2,2]), name='weight1')
     b1 = tf.Variable(tf.random_normal([2]), name='bias1')
     layer1 = tf.sigmoid(tf.matmul(X,W1)+b1)

     W2 = tf.Variable(tf.random_normal([2,1]), name='weight2')
     b2 = tf.Variable(tf.random_normal([1]), name='bias2')
     hypothesis = tf.sigmoid(tf.matmul(layer1,W2)+b2)
```

```
[26] cost = -tf.reduce_mean(Y*tf.log(hypothesis)+(1-Y)*tf.log(1-hypothesis))
     train = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)
```

```
[27] predicted = tf.cast(hypothesis>0.5, dtype=tf.float32)
     accuracy = tf.reduce_mean(tf.cast(tf.equal(predicted, Y), dtype=tf.float32))
```

+ 코드 + 텍스트

```
with tf.Session() as sess:
  sess.run(tf.global_variables_initializer())

  for step in range(10001):
    sess.run(train, feed_dict={X: x_data, Y: y_data})
    if step %100 == 0:
      print(step, sess.run(cost, feed_dict={X: x_data, Y: y_data}))

  h, c, a = sess.run([hypothesis, predicted, accuracy],feed_dict={X: x_data, Y: y_data})
  print("₩nHypothesis: ",h,"₩nCorrect:",c, "₩nAccuracy", a)
```

Wide Multi Layered NN

```
W1 = tf.Variable(tf.random_normal([2,2]), name='weight1')
b1 = tf.Variable(tf.random_normal([2]), name='bias1')
layer1 = tf.sigmoid(tf.matmul(X,W1)+b1)

W2 = tf.Variable(tf.random_normal([2,1]), name='weight2')
b2 = tf.Variable(tf.random_normal([1]), name='bias2')
hypothesis = tf.sigmoid(tf.matmul(layer1,W2)+b2)
```

```
[29] W1 = tf.Variable(tf.random_normal([2,10]), name='weight1')
b1 = tf.Variable(tf.random_normal([10]), name='bias1')
layer1 = tf.sigmoid(tf.matmul(X,W1)+b1)

W2 = tf.Variable(tf.random_normal([10,1]), name='weight2')
b2 = tf.Variable(tf.random_normal([1]), name='bias2')
hypothesis = tf.sigmoid(tf.matmul(layer1,W2)+b2)
```

```
[30] cost = -tf.reduce_mean(Y*tf.log(hypothesis)+(1-Y)*tf.log(1-hypothesis))
train = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)
```

```
[31] predicted = tf.cast(hypothesis>0.5, dtype=tf.float32)
accuracy = tf.reduce_mean(tf.cast(tf.equal(predicted, Y), dtype=tf.float32))
```

```
with tf.Session() as sess:
  sess.run(tf.global_variables_initializer())

  for step in range(10001):
    sess.run(train, feed_dict={X: x_data, Y: y_data})
    if step %100 == 0:
      print(step, sess.run(cost, feed_dict={X: x_data, Y: y_data}))

  h, c, a = sess.run([hypothesis, predicted, accuracy],feed_dict={X: x_data, Y: y_data})
  print("₩nHypothesis: ",h,"₩nCorrect:",c, "₩nAccuracy", a)
```

Wide Multi Layered NN

```
W1 = tf.Variable(tf.random_normal([2,10]), name='weight1')
b1 = tf.Variable(tf.random_normal([10]), name='bias1')
layer1 = tf.sigmoid(tf.matmul(X,W1)+b1)

W2 = tf.Variable(tf.random_normal([10,10]), name='weight2')
b2 = tf.Variable(tf.random_normal([10]), name='bias2')
layer2 = tf.sigmoid(tf.matmul(layer1,W2)+b2)

W3 = tf.Variable(tf.random_normal([10,10]), name='weight1')
b3 = tf.Variable(tf.random_normal([10]), name='bias1')
layer3 = tf.sigmoid(tf.matmul(X,W3)+b3)

W4 = tf.Variable(tf.random_normal([10,1]), name='weight2')
b4 = tf.Variable(tf.random_normal([1]), name='bias2')
hypothesis = tf.sigmoid(tf.matmul(layer1,W4)+b4)
```

```
[34] cost = -tf.reduce_mean(Y*tf.log(hypothesis)+(1-Y)*tf.log(1-hypothesis))
     train = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)
```

```
[35] predicted = tf.cast(hypothesis>0.5, dtype=tf.float32)
     accuracy = tf.reduce_mean(tf.cast(tf.equal(predicted, Y), dtype=tf.float32))
```

```
[36] with tf.Session() as sess:
       sess.run(tf.global_variables_initializer())

       for step in range(10001):
         sess.run(train, feed_dict={X: x_data, Y: y_data})
         if step %100 == 0:
           print(step, sess.run(cost, feed_dict={X: x_data, Y: y_data}))

       h, c, a = sess.run([hypothesis, predicted, accuracy],feed_dict={X: x_data, Y: y_data})
       print("₩nHypothesis: ",h,"₩nCorrect:",c, "₩nAccuracy", a)
```

Lab 09-02

TensorBoard

How to Log

1. From the TF graph, decide which tensors you want to log
2. Merge all summaries
3. Create writer and add graph
4. Run summary merge and add_summary
5. Launch TensorBoard

Exercise

MNST

Neural Network

ReLU

Activation Function; ReLU Function

- Drawback of Deep Learning
  - so deep to be poor
    - Sigmoid value is less than 1 ➔ with chain rule, final feedback would be lesser (Vanishing gradient)
  - New function?
- ReLU (Rectified Linear Unit) Function
  - $ReLU(x) = \max(0, x)$
  - W/O final layer ➔ apply sigmoid

Activation Function; Variations

- Sigmoid Family
  - tanh
- ReLU Family
  - Leaky ReLU
  - Maxout
  - ELU

Neural Network

Weight Initialization

Weight Initialization

- Why Important
  - used for not only forward but also backward propagation
  - Initialize to zero result in fail ➔ no feed back
- Many Ways of Pre-Training
  - RBM (Restricted Boatman Machine)
    - Initialize weight for encoding and decoding value to be same
    - Pretrain weights sequentially (Fine Tuning)
  - Xavier Initialization & He’s Initialization
    - Make sure the weight are ‘just right’ not too big and small

Neural Network

Overfitting

Solutions; Regularization & Dropout

- Regularization
  - Restrain weight in proper range
  - $L2\ Regularization = \sum_{i=1}^{n}(\hat{y} - y)^2$
  - $Cost = Cost' + \lambda \sum_{i=1}^{n} \widehat{w_n}^2$
    - Minimalize Cost means make weight close to zero ➔ restrain weight
    - Control with lambda
- Dropout
  - Randomly dropout some nodes in a layer in "training"
  - Not in Executing

```
❯ ./a.out
toal : 9897335391534825783 thread duration : 80.000000
total : 9897335391534825783 normal duration : 2.000000%
```

fibo0.cc

```cpp
#include <thread>
#include <stdio.h>
#include <time.h>

using namespace std;

long fiboArr[1000];

void fibo(int start, int end, long& result)
{
  result = 0;
  for(int i = start; i<end; ++i)
    result += fiboArr[i];
}

int main(void)
{
  long n,total,result0,result1,result2, result3;
  clock_t start, end;

  fiboArr[0] = 1, fiboArr[1] = 1;

  for(int i = 2; i<1000; ++i)
    fiboArr[i] = fiboArr[i-1] + fiboArr[i-2];

  n = 100;

  start = clock();
  thread t0(fibo,0,500,ref(result0));
  thread t1(fibo,500,1000,ref(result1));
  //thread t2(fibo,50,75,ref(result2));
  //thread t3(fibo,75,100,ref(result3));

  t0.join();
  t1.join();
  //t2.join();
  //t3.join();

  total = result0 + result1;// + result2 + result3;
  end = clock();

  printf("toal : %lu thread duration : %lf\n",total,double(end-start));

  start = clock();
  fibo(0,1000,ref(total));
  end = clock();
  printf("total : %lu normal duration : %lf",total,double(end-start));

  return 0;
}
```

NORMAL fibo0.cc