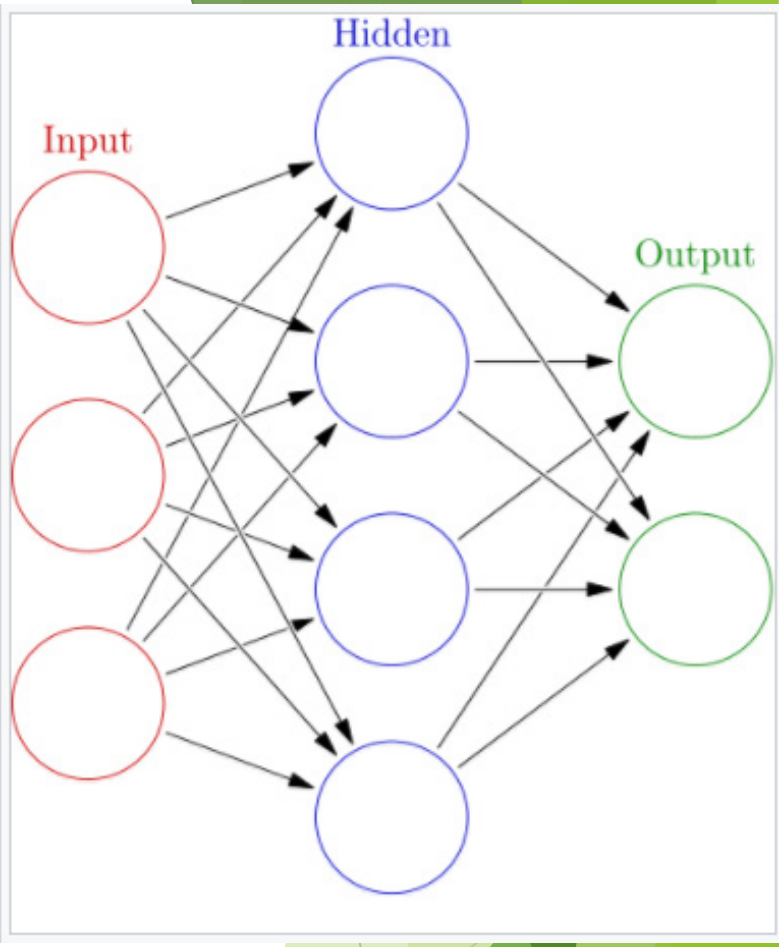
Back propagation

Outline

- Gradient descent
- Forward propagate and Back propagation
- Easy Example by tensorflow
- Homework

Gradient descent

- Neural network have many neural node
- Each neural node have its own weight and bias to compute its output
- Neural network output and Label(real answer) compute Loss function
- Know how far between output and Label(real answer) by Loss function



https://en.wikipedia.org/wiki/Artificial_neural_network

- Cost function is computed by output and real answer
- Output is computed by weight and bias
- Cost function is structured by weight and bias

direction of the negative gradient

$$\mathbf{a}_{n+1} = \mathbf{a}_n - \gamma \nabla F(\mathbf{a}_n)$$

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma_n \nabla F(\mathbf{x}_n),\ n \geq 0.$$

We have

$$F(\mathbf{x}_0) \geq F(\mathbf{x}_1) \geq F(\mathbf{x}_2) \geq \cdots,$$

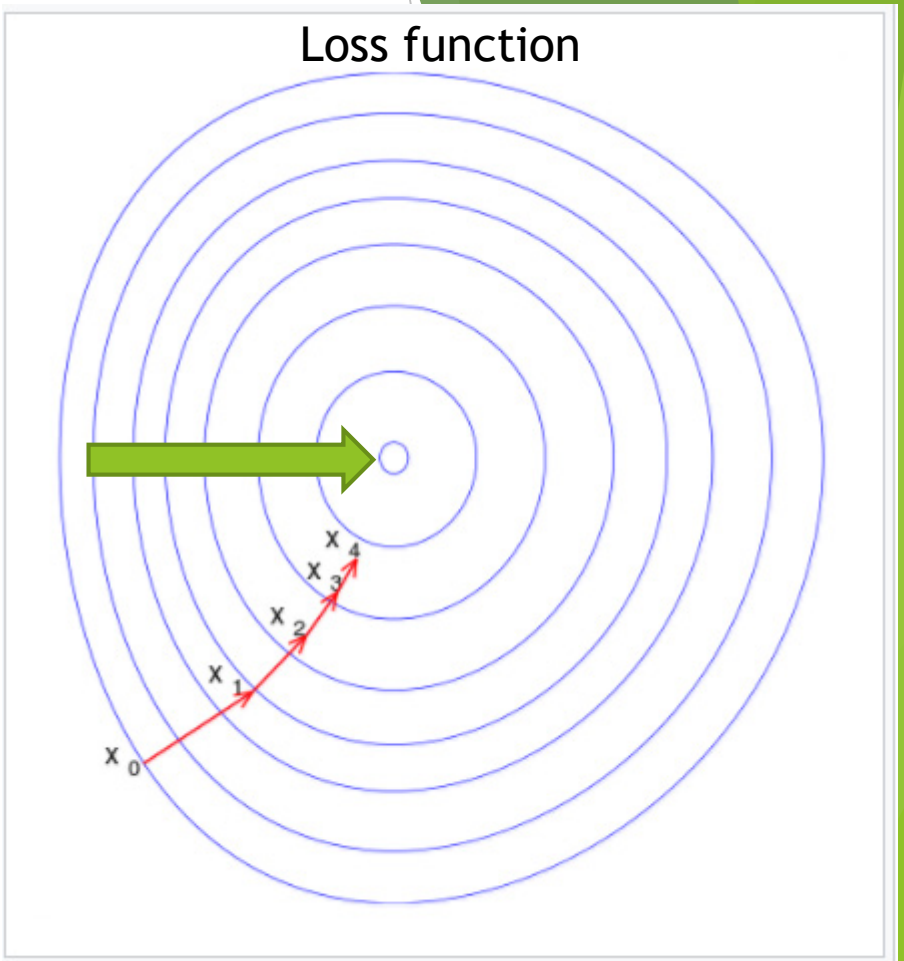
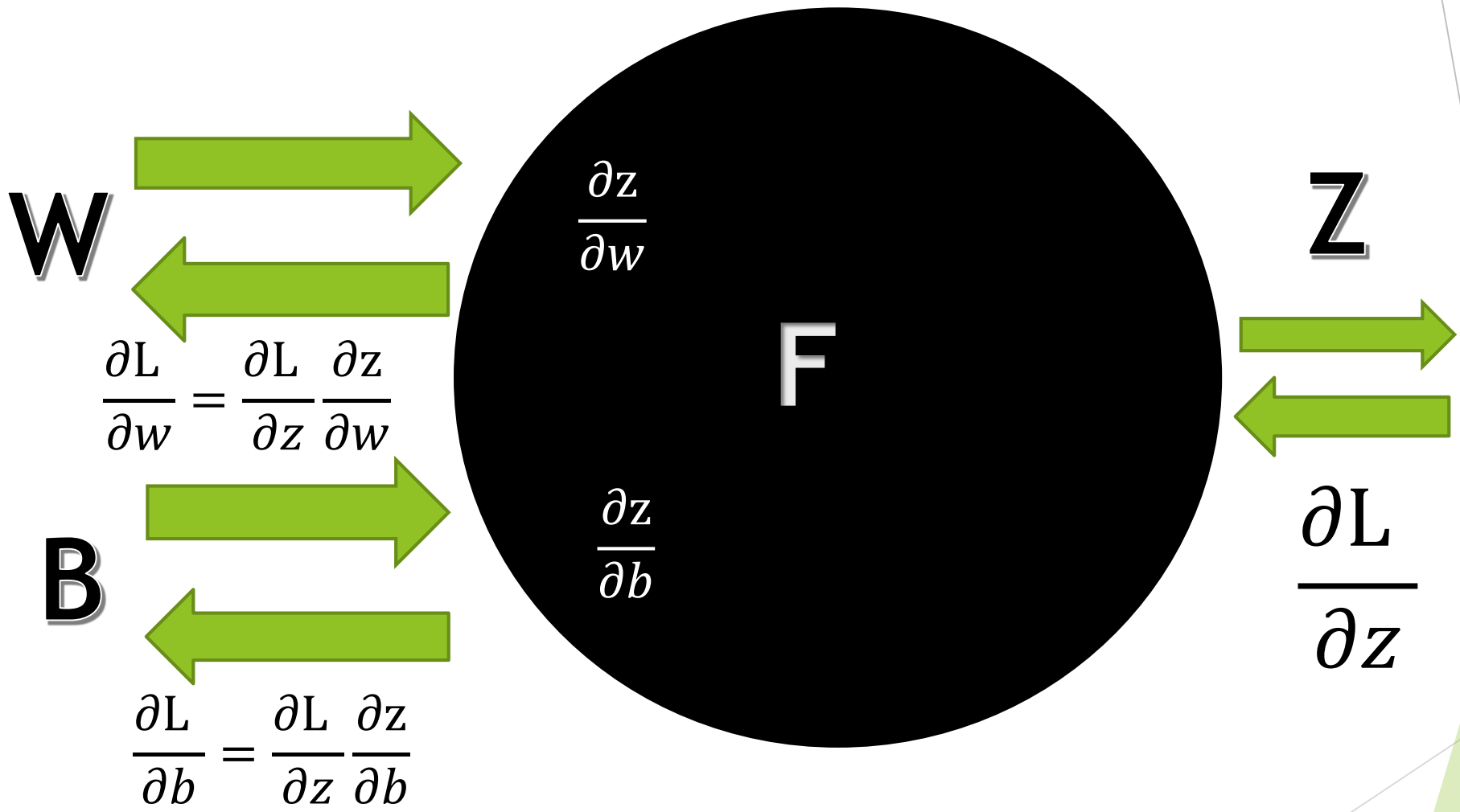


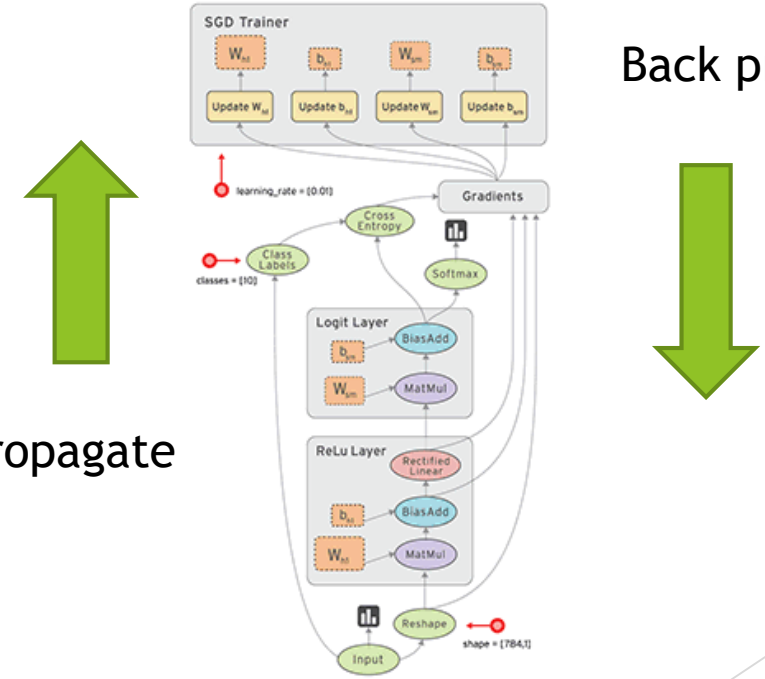
Illustration of gradient descent on a series of level sets.

https://en.wikipedia.org/wiki/Gradient_descen


W
$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial w}$
B
$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial b}$
$\frac{\partial z}{\partial w}$
F
$\frac{\partial z}{\partial b}$
Z
$\frac{\partial L}{\partial z}$

Forward propagate and back propagation

- Forward propagate: know neural network behaving and get the neural networks output
- Back propagation : derivative of error(Loss function) ,update weights to minima for error value



Back propagation

forward propagate

Easy example by tensorflow

- 3 x [1,3] input , 3 x [1] real answer
- hidden layer_1 have 3 input and 2 output
- hidden layer_2 have 2 input and 1 output
- Activation function : sigmoid
- loss function : Mean squared error
- Optimizer : Gradient Descent

Prepare data

```
#training data
train_1 = np.array( [[1.,2.,3.],
        [3.,4.,5.],
        [8.,5.,7.],
        [7.,1.,8.]] )
train_2 =  np.array( [[1.],
            [0.],
            [0.],
            [1.]] )
```

Prepare input

```
input_1 = tf.placeholder(tf.float32, shape = [None, 3])
input_2 = tf.placeholder(tf.float32, shape = [None, 1])
```

Prepare hidden layer

```
weight_1 = tf.get_variable(name='weight_1', shape = [3,2], dtype = tf.float32, initializer =
  tf.truncated_normal_initializer(mean=0.0, stddev=0.1) )
bias_1 = tf.get_variable(name='bias_1', shape = [2], dtype = tf.float32, initializer =
  tf.truncated_normal_initializer(mean=0.0, stddev=0.1) )
layer_1_output =  tf.add(tf.matmul( input_1, weight_1 ), bias_1)

weight_2 = tf.get_variable(name='weight_2', shape = [2,1], dtype = tf.float32, initializer =
  tf.truncated_normal_initializer(mean=0.0, stddev=0.1) )
bias_2 = tf.get_variable(name='bias_2', shape = [1], dtype = tf.float32, initializer =
  tf.truncated_normal_initializer(mean=0.0, stddev=0.1) )
layer_2_output = tf.sigmoid( tf.add(tf.matmul( layer_1_output, weight_2 ), bias_2) )
```

Prepare loss function

```
loss = tf.losses.mean_squared_error(train_2, layer_2_output)
```

Prepare optimizer

```
optimizer = tf.train.GradientDescentOptimizer(0.1)
```

Prepare How to train

```
train = optimizer.minimize(loss)
```

Prepare session to start train

```
with tf.Session() as sess:
    #initial
    init = tf.global_variables_initializer()
    sess.run( init )
    #train
    for step in range(201) :
        if step % 20 == 0:
            print ('loss : ', sess.run(loss, feed_dict = {input_1: train_1, input_2: train_2}))
            print ('predict : ', sess.run(layer_2_output, feed_dict = {input_1: train_1}))
        sess.run(train, feed_dict = {input_1: train_1, input_2: train_2})
```

```
#training data
train_1 = np.array( [[1.,2.,3.],
        [3.,4.,5.],
        [8.,5.,7.],
        [7.,1.,8.]] )
train_2 =  np.array( [[1.],
            [0.],
            [0.],
            [1.]] )
```

```
loss :  0.251956
predict :  [[ 0.46995696]
 [ 0.47192109]
 [ 0.47451538]
 [ 0.471793  ]]
loss :  0.247058
predict :  [[ 0.47125581]
 [ 0.46566859]
 [ 0.46073726]
 [ 0.4712811 ]]
loss :  0.23654
predict :  [[ 0.46862936]
 [ 0.44731638]
 [ 0.42845353]
 [ 0.47071627]]
loss :  0.0668569
predict :  [[ 0.57679796]
 [ 0.27093837]
 [ 0.08406196]
 [ 0.91137868]]
loss :  0.0565343
predict :  [[ 0.61291397]
 [ 0.26089114]
 [ 0.05567734]
 [ 0.92832458]]
loss :  0.0481612
predict :  [[ 0.64426953]
 [ 0.24730685]
 [ 0.03547321]
 [ 0.93932498]]
```

Homework

Data.txt

1 2 3 5 7 9 5 4 8 6 9 5 1 2 3 5 4 8 6 5 4 1 2 5 8 6 9 4 2 2 4 1↓
3 4 5 8 4 2 5 1 4 8 6 8 5 2 3 4 5 8 7 4 1 2 3 8 4 5 2 1 8 6 9 9↓
8 5 7 9 4 2 5 8 4 2 6 9 8 7 5 1 2 5 8 7 5 2 3 8 7 5 6 2 5 8 6 2↓
7 1 8 7 8 5 5 1 2 3 6 5 4 7 8 9 5 4 2 3 6 8 7 5 1 2 3 5 4 8 7 2↓
4 2 5 4 5 5 7 1 8 7 8 5 7 1 8 7 8 5 7 1 8 7 8 5 7 1 8 7 8 5 2 1↓

Answer.txt

10010

- Any number hidden layer you want
- Activation function : Anything you wanted
- loss function : Anything you wanted
- Optimizer : Anything you wanted
- Need demo

grade

- 0.005 : 1
- 0.001 : 2
- 0.0003 : 3
- 0.0006 : 4
- Use add layer function : +1

```
loss :  0.000501916
predict :  [[ 0.97440165]
 [ 0.01172936]
 [ 0.01904387]
 [ 0.97416872]
 [ 0.02620688]]
```