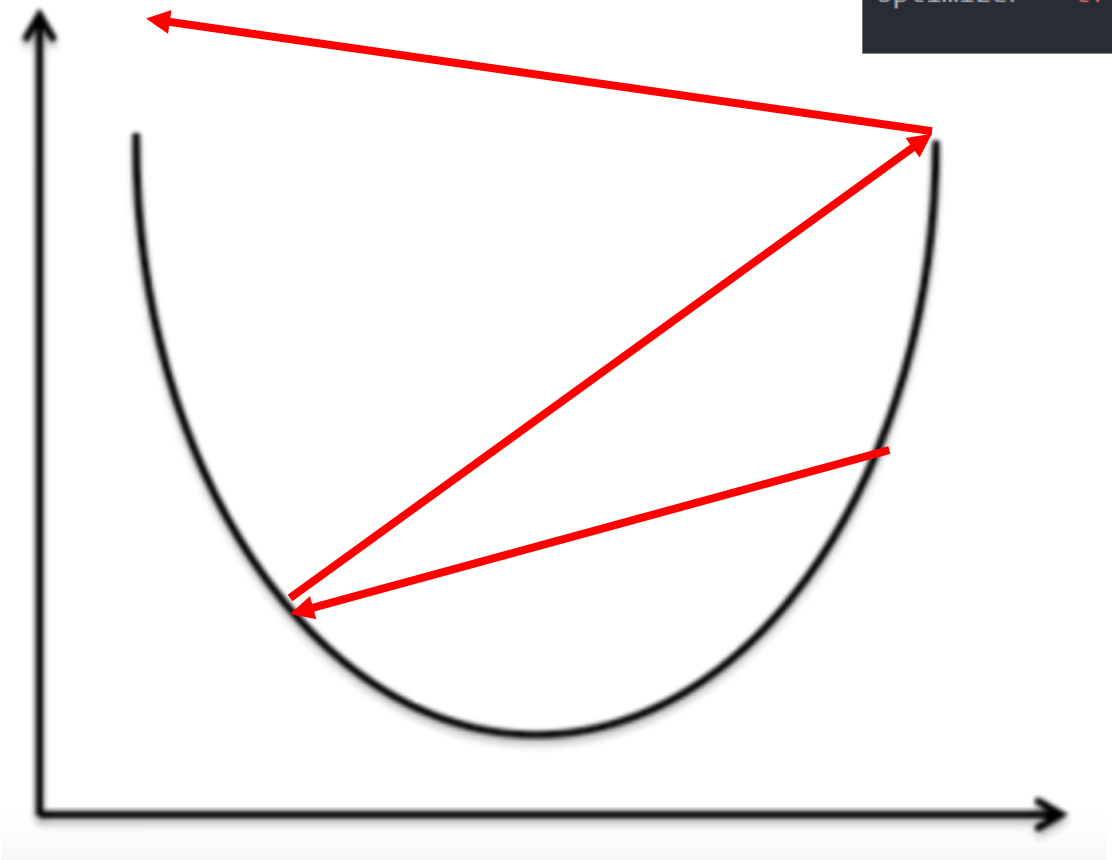
Deep Learning Study Week #3

발표자 : 명승주

Large learning rate : overshooting

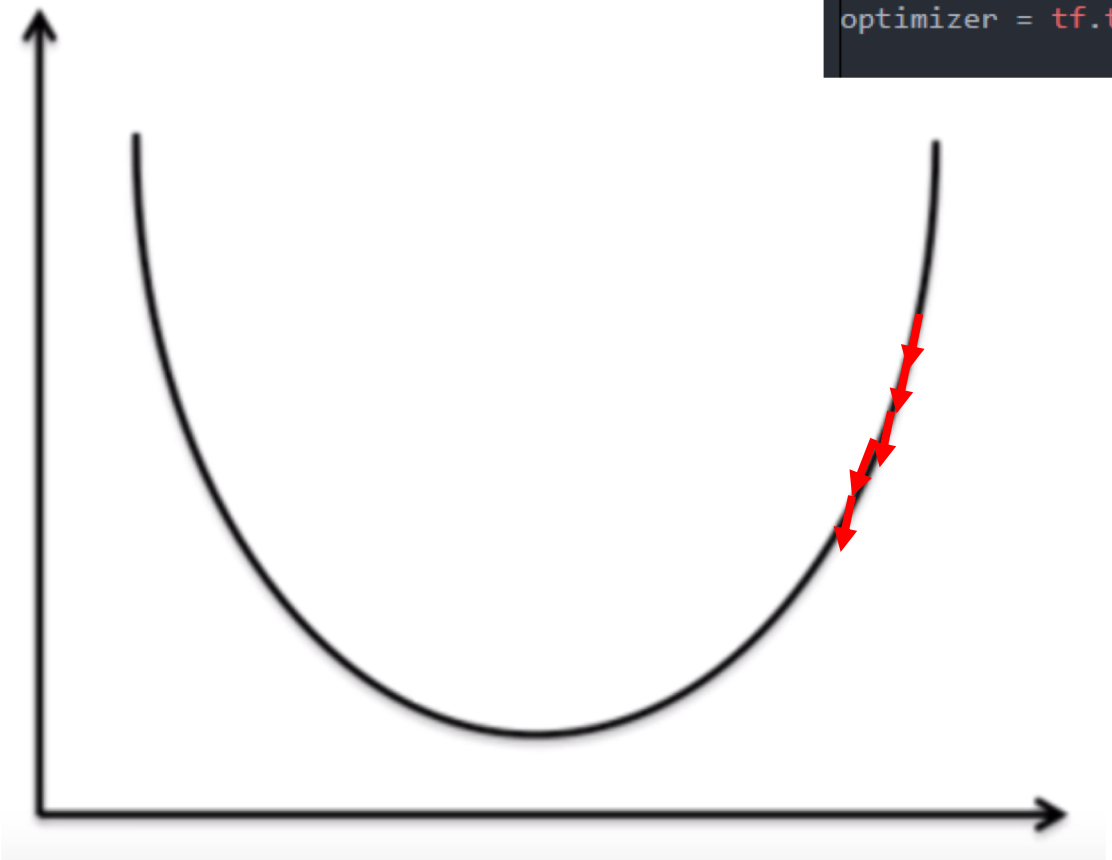


```
cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(hypothesis), axis = 1))
optimizer = tf.train.GradientDescentOptimizer(learning_rate = 2).minimize(cost)
```

```
 [nan nan nan]]
198 nan [[nan nan nan]
 [nan nan nan]
 [nan nan nan]]
199 nan [[nan nan nan]
 [nan nan nan]
 [nan nan nan]]
200 nan [[nan nan nan]
 [nan nan nan]
 [nan nan nan]]
Prediction:  [0 0 0]
Accuracy:  0.0
```

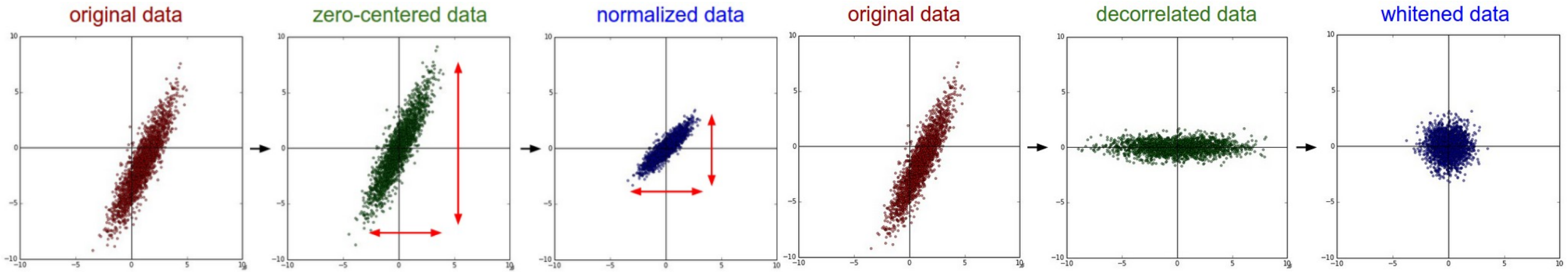
Small learning rate : local minimum, too much time

```
cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(hypothesis), axis = 1))
optimizer = tf.train.GradientDescentOptimizer(learning_rate = 0.00001).minimize(cost)
```



```
198 5.1813393 [[ 0.6154322   2.3503973  -1.075693  ]
 [ 0.06693207 -0.3310225   0.15146057]
 [ 0.8258102   1.9574785   0.23429139]]
199 5.1811275 [[ 0.61543465  2.3503911  -1.0756893 ]
 [ 0.06694818 -0.3310486   0.15147057]
 [ 0.82582635  1.9574536   0.23430014]]
200 5.180916 [[ 0.6154371   2.350385   -1.0756856 ]
 [ 0.06696429 -0.3310747   0.15148057]
 [ 0.8258425   1.9574287   0.23430888]]
Prediction:  [1 1 1]
Accuracy:  0.0
```

Data preprocessing : normalization(min max scaler)

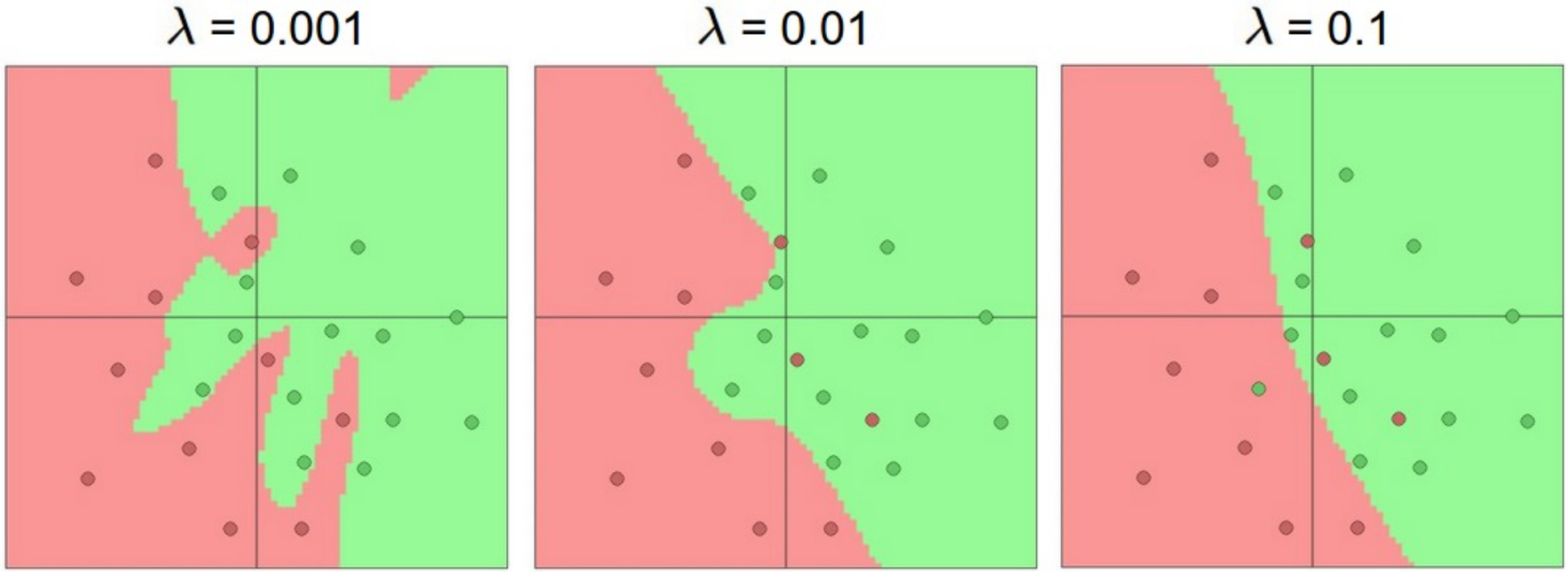



```
xy = np.array([[828.659973, 833.450012, 908100, 828.349976, 831.659973],
               [823.02002, 828.070007, 1828100, 821.655029, 828.070007],
               [819.929993, 824.400024, 1438100, 818.97998, 824.159973],
               [816, 820.958984, 1008100, 815.48999, 819.23999],
               [819.359985, 823, 1188100, 818.469971, 818.97998],
               [819, 823, 1198100, 816, 820.450012],
               [811.700012, 815.25, 1098100, 809.780029, 813.669983],
               [809.51001, 816.659973, 1398100, 804.539978, 809.559998]])
```

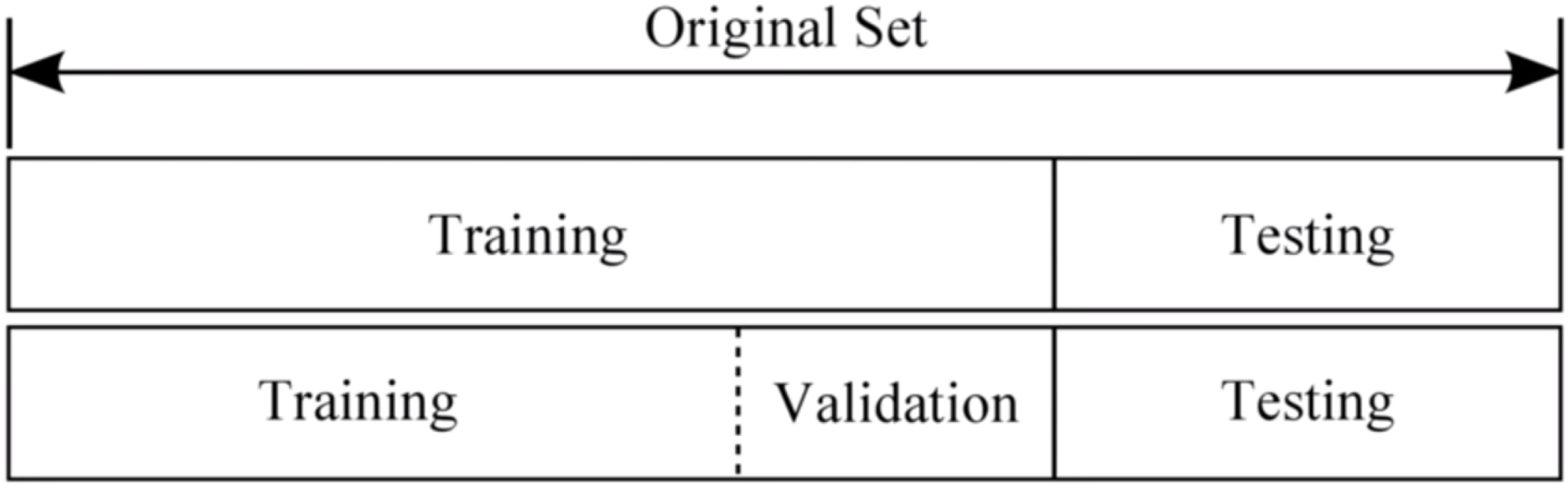
```
[[0.99999999 0.99999999 0.         1.         1.        ]
 [0.70548491 0.70439552 1.         0.71881782 0.83755791]
 [0.54412549 0.50274824 0.57608696 0.606468   0.6606331 ]
 [0.33890353 0.31368023 0.10869565 0.45989134 0.43800918]
 [0.51436    0.42582389 0.30434783 0.58504805 0.42624401]
 [0.49556179 0.42582389 0.31521739 0.48131134 0.49276137]
 [0.11436064 0.         0.20652174 0.22007776 0.18597238]
 [0.         0.07747099 0.5326087  0.         0.        ]]
```

Overfitting 해결 방법

1. Training set 증가
2. Feature 수 감소
3. Regularization (펴준다): cost 함수 뒤에 $\lambda \sum w^2$ 삽입



Training, validation and test sets



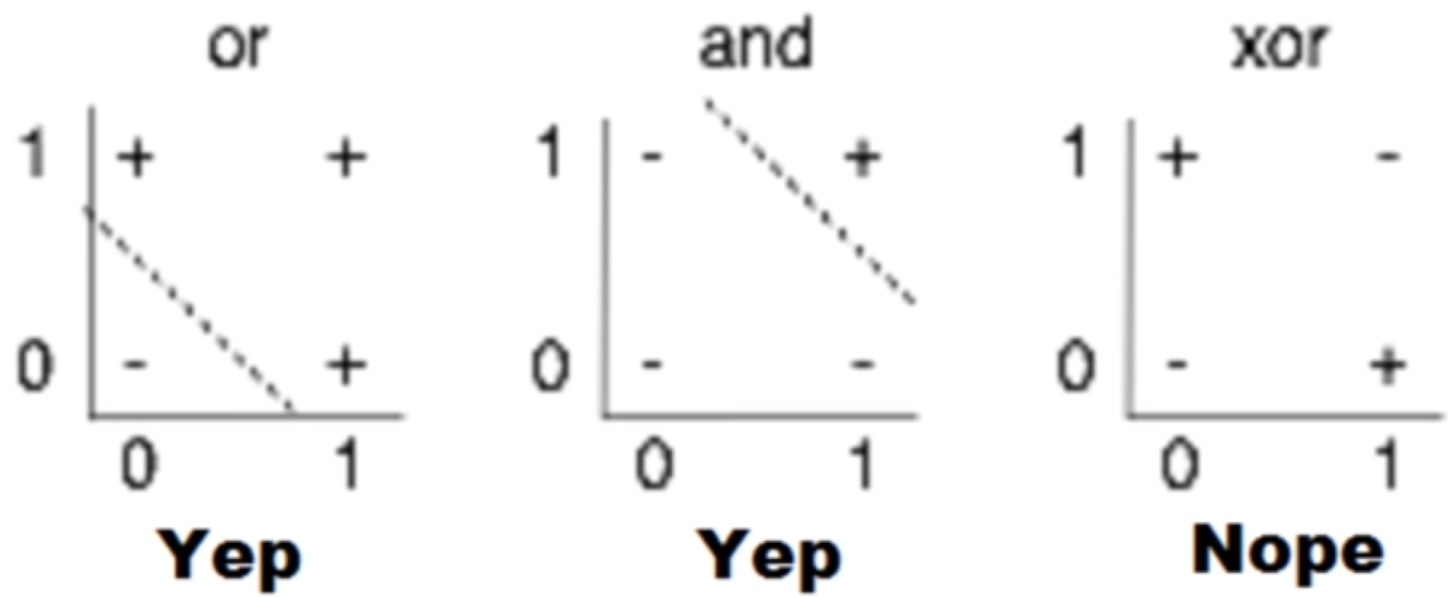
Training epoch / batch size

In the neural network terminology:

- one **epoch** = one forward pass and one backward pass of *all* the training examples
- **batch size** = the number of training examples in one forward/backward pass. The higher the batch size, the more memory space you'll need.
- number of **iterations** = number of passes, each pass using [batch size] number of examples. To be clear, one pass = one forward pass + one backward pass (we do not count the forward pass and backward pass as two different passes).

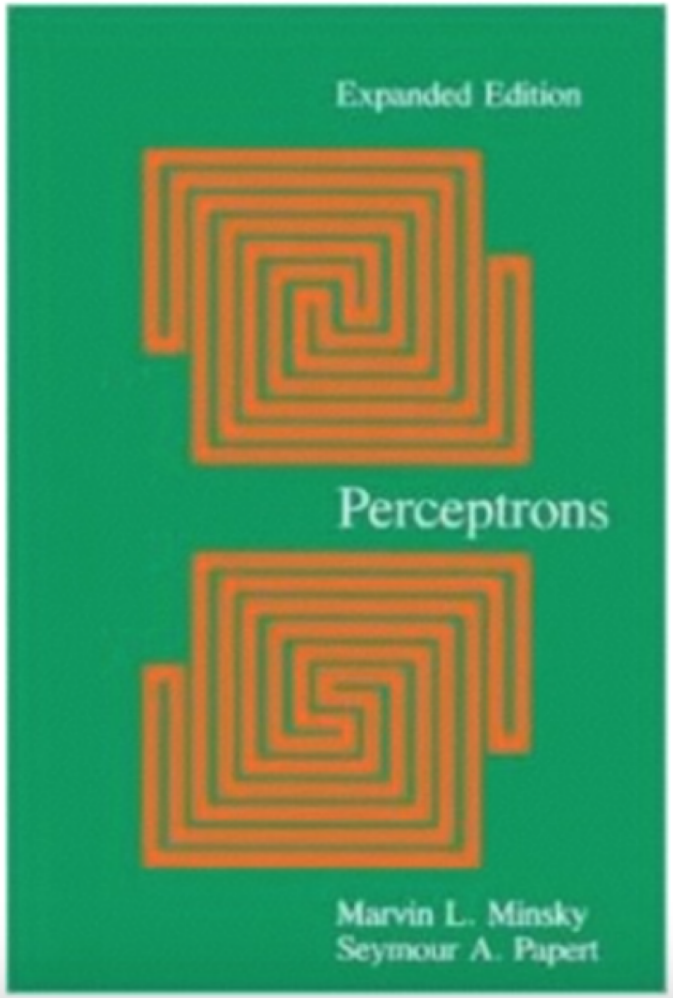
Example: if you have *1000 training examples*, and your *batch size is 500*, then it will take 2 iterations to complete 1 epoch.

(Simple) XOR problem: linearly separable?



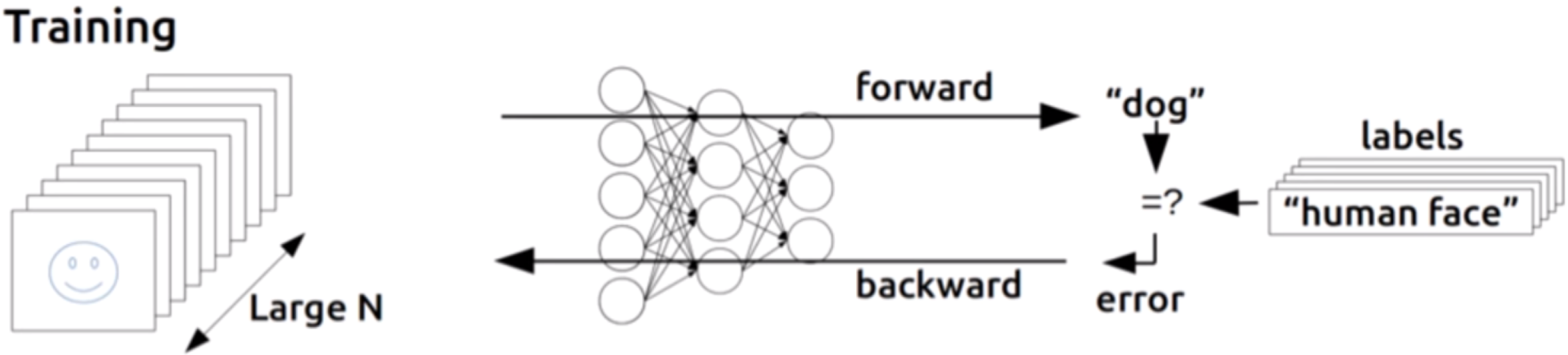
Perceptrons (1969)

by Marvin Minsky, founder of the MIT AI Lab

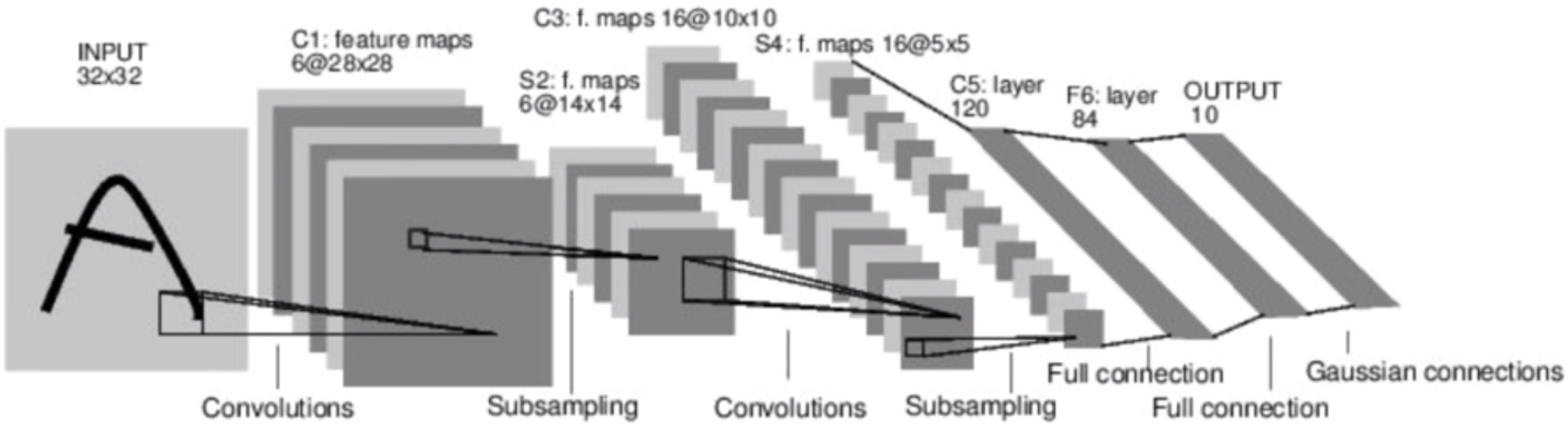


- We need to use MLP, multilayer perceptrons (multilayer neural nets)
- No one on earth had found a viable way to train MLPs good enough to learn such simple functions.

Backpropagation
(1974, 1982 by Paul Werbos, 1986 by Hinton)

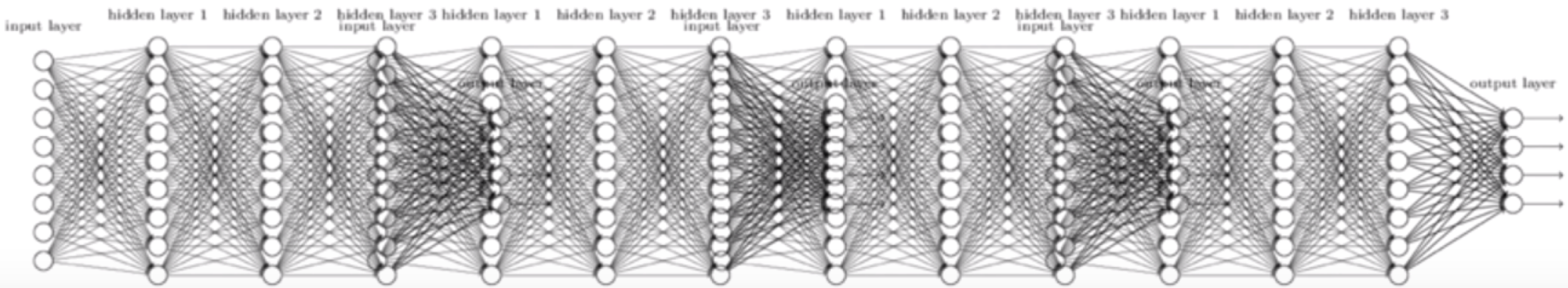


Convolutional Neural Networks



A BIG problem

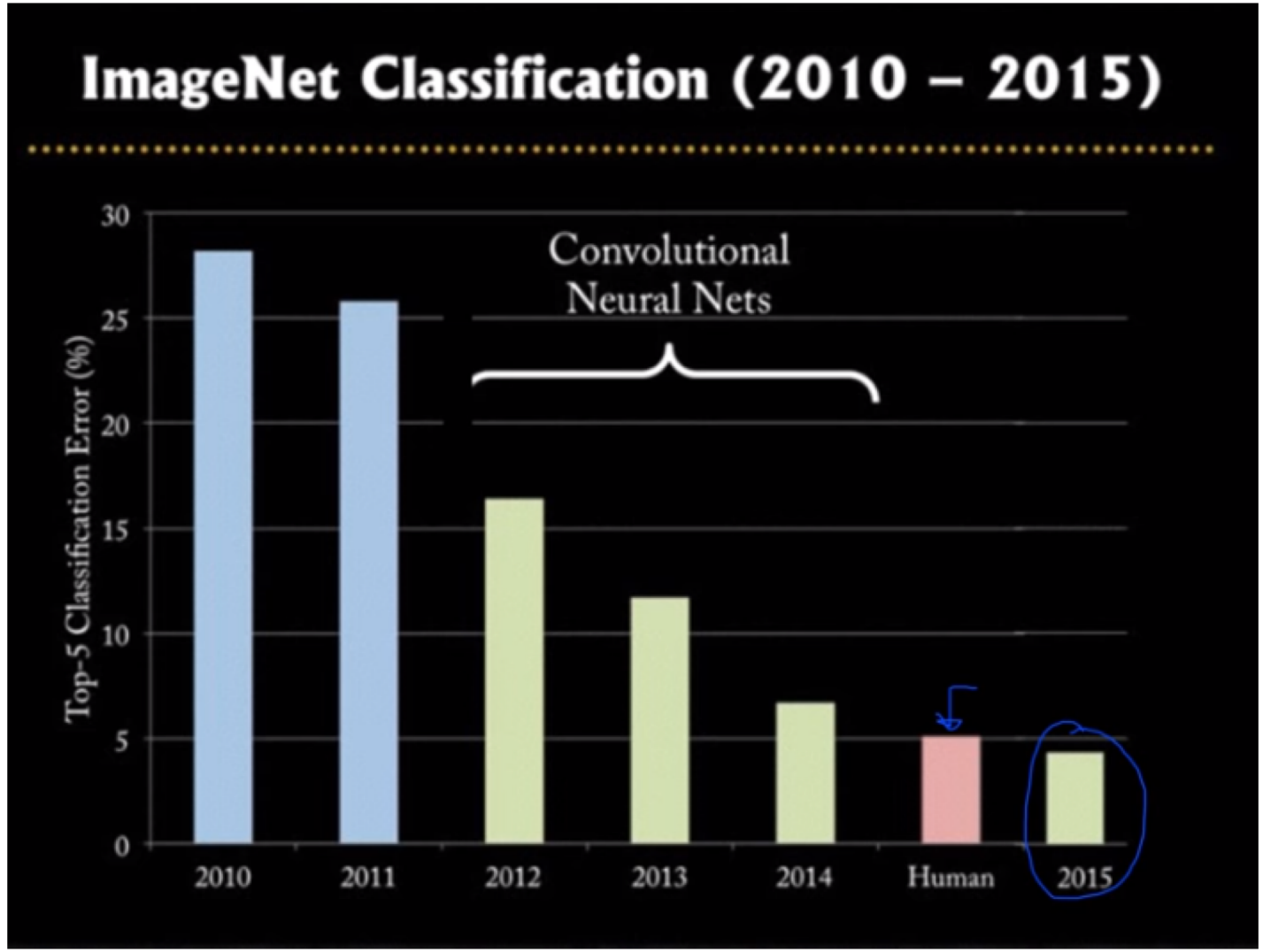
- Backpropagation just did not work well for normal neural nets with many layers
- Other rising machine learning algorithms: SVM, RandomForest, etc.
- **1995** "Comparison of Learning Algorithms For Handwritten Digit Recognition" by LeCun et al. found that this new approach worked better



Breakthrough

in 2006 and 2007 by Hinton and Bengio

- Neural networks with many layers really could be trained well, *if the weights are initialized in a clever way* rather than randomly.
- Deep machine learning methods are more efficient for difficult problems than shallow methods.
- Rebranding to ***Deep Nets***, ***Deep Learning***


ImageNet Classification (2010 – 2015)
Top-5 Classification Error (%)
30
25
20
15
10
5
0
Convolutional
Neural Nets
2010
2011
2012
2013
2014
Human
2015

Shape, Rank, Axis

```
t = tf.constant([[[[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]],
                 [[13, 14, 15, 16], [17, 18, 19, 20], [21, 22, 23, 24]]]])
tf.shape(t).eval()
```

```
array([1, 2, 3, 4], dtype=int32)
```

```
[
    [
        [
            [1,2,3,4],
            [5,6,7,8],
            [9,10,11,12]
        ],
        [
            [13,14,15,16],
            [17,18,19,20],
            [21,22,23,24]
        ]
    ]
]
```

Broadcasting


```
matrix1 = tf.constant([[1., 2.]])
matrix2 = tf.constant(3.)
(matrix1+matrix2).eval()
```

```
array([[ 4.,  5.]], dtype=float32)
```

```
matrix1 = tf.constant([[1., 2.]])
matrix2 = tf.constant([3., 4.])
(matrix1+matrix2).eval()
```

```
array([[ 4.,  6.]], dtype=float32)
```

```
matrix1 = tf.constant([[1., 2.]])
matrix2 = tf.constant([[3.],[4.]])
(matrix1+matrix2).eval()
```

```
array([[ 4.,  5.],
       [ 5.,  6.]], dtype=float32)
```

Reshape**

```
t = np.array([[[0, 1, 2],
               [3, 4, 5]],

              [[6, 7, 8],
               [9, 10, 11]]])
t.shape
```

```
(2, 2, 3)
```

```
tf.reshape(t, shape=[-1, 3]).eval()
```

```
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11]])
```

```
tf.reshape(t, shape=[-1, 1, 3]).eval()
```

```
array([[[ 0,  1,  2]],

       [[ 3,  4,  5]],

       [[ 6,  7,  8]],

       [[ 9, 10, 11]]])
```

Reshape (squeeze, expand)

```
tf.squeeze([[0], [1], [2]]).eval()
```

```
array([0, 1, 2], dtype=int32)
```

```
tf.expand_dims([0, 1, 2], 1).eval()
```

```
array([[0],
       [1],
       [2]], dtype=int32)
```

One hot

```
tf.one_hot([[0], [1], [2], [0]], depth=3).eval()
```

```
array([[[ 1.,  0.,  0.]],

       [[ 0.,  1.,  0.]],

       [[ 0.,  0.,  1.]],

       [[ 1.,  0.,  0.]]], dtype=float32)
```

```
t = tf.one_hot([[0], [1], [2], [0]], depth=3)
tf.reshape(t, shape=[-1, 3]).eval()
```

```
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.],
       [ 1.,  0.,  0.]], dtype=float32)
```

Zip

```
for x, y in zip([1, 2, 3], [4, 5, 6]):
    print(x, y)
```

```
1 4
2 5
3 6
```

```
for x, y, z in zip([1, 2, 3], [4, 5, 6], [7, 8, 9]):
    print(x, y, z)
```

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