Lecture 10

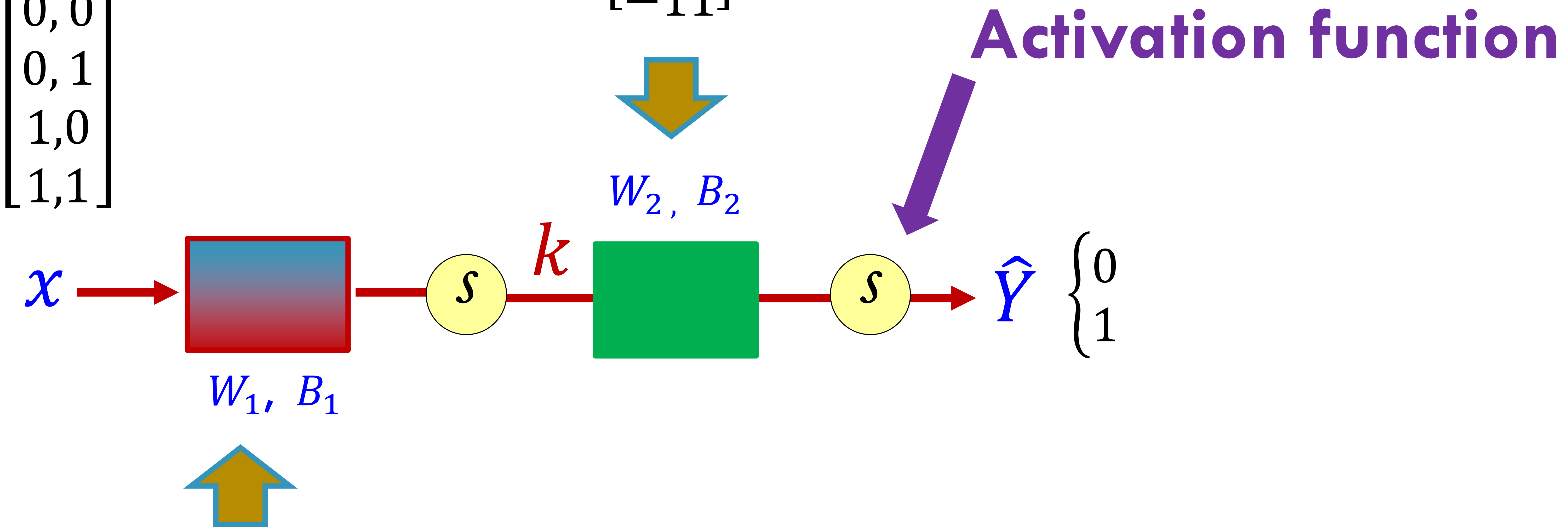
NN ReLu, Xavier, Dropout and Various NN

NN ReLu

NN for XOR

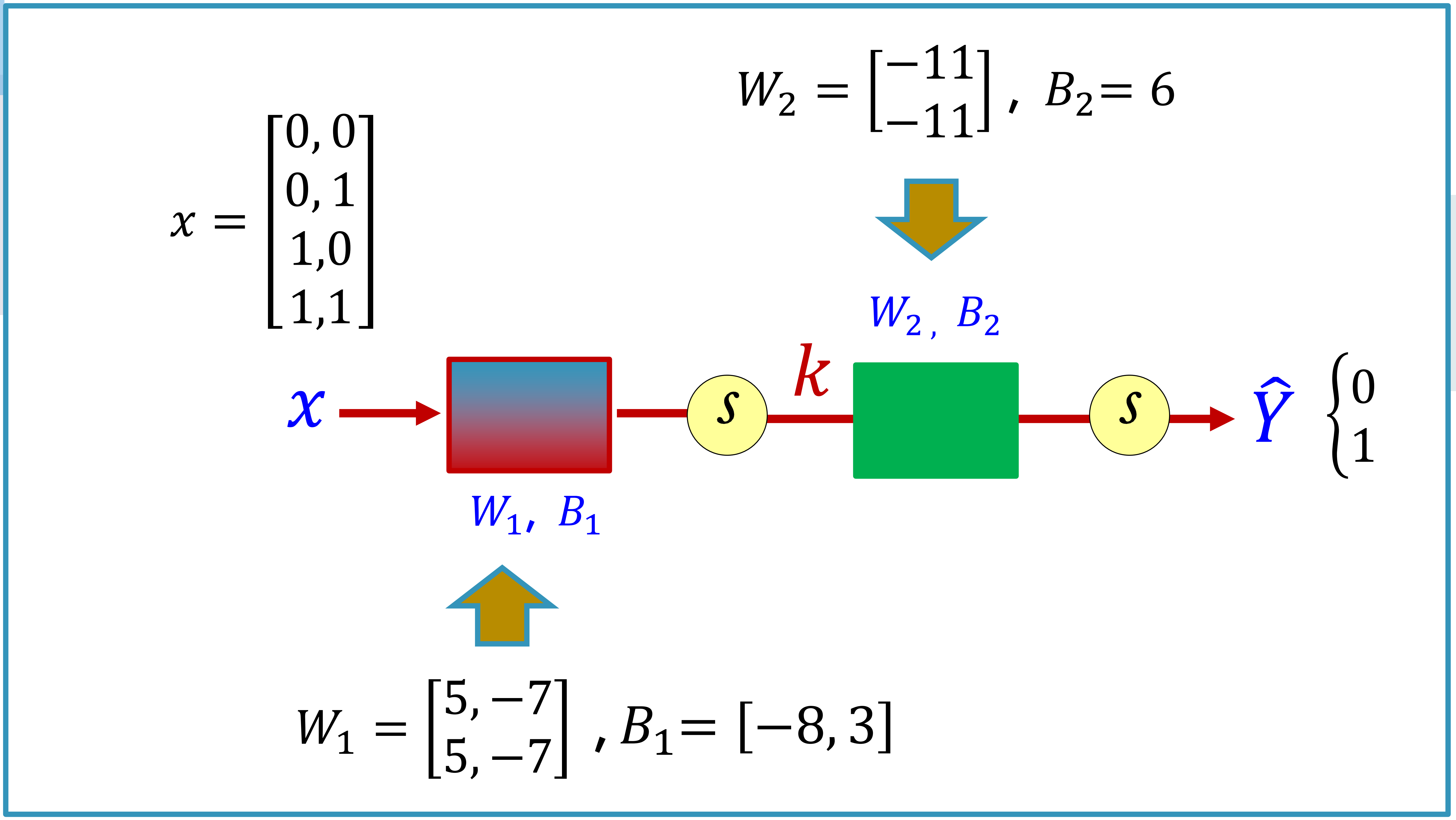
$$x = \begin{bmatrix} 0, 0 \\ 0, 1 \\ 1,0 \\ 1,1 \end{bmatrix}$$

$$W_2 = \begin{bmatrix} -11 \\ -11 \end{bmatrix},\ B_2 = 6$$



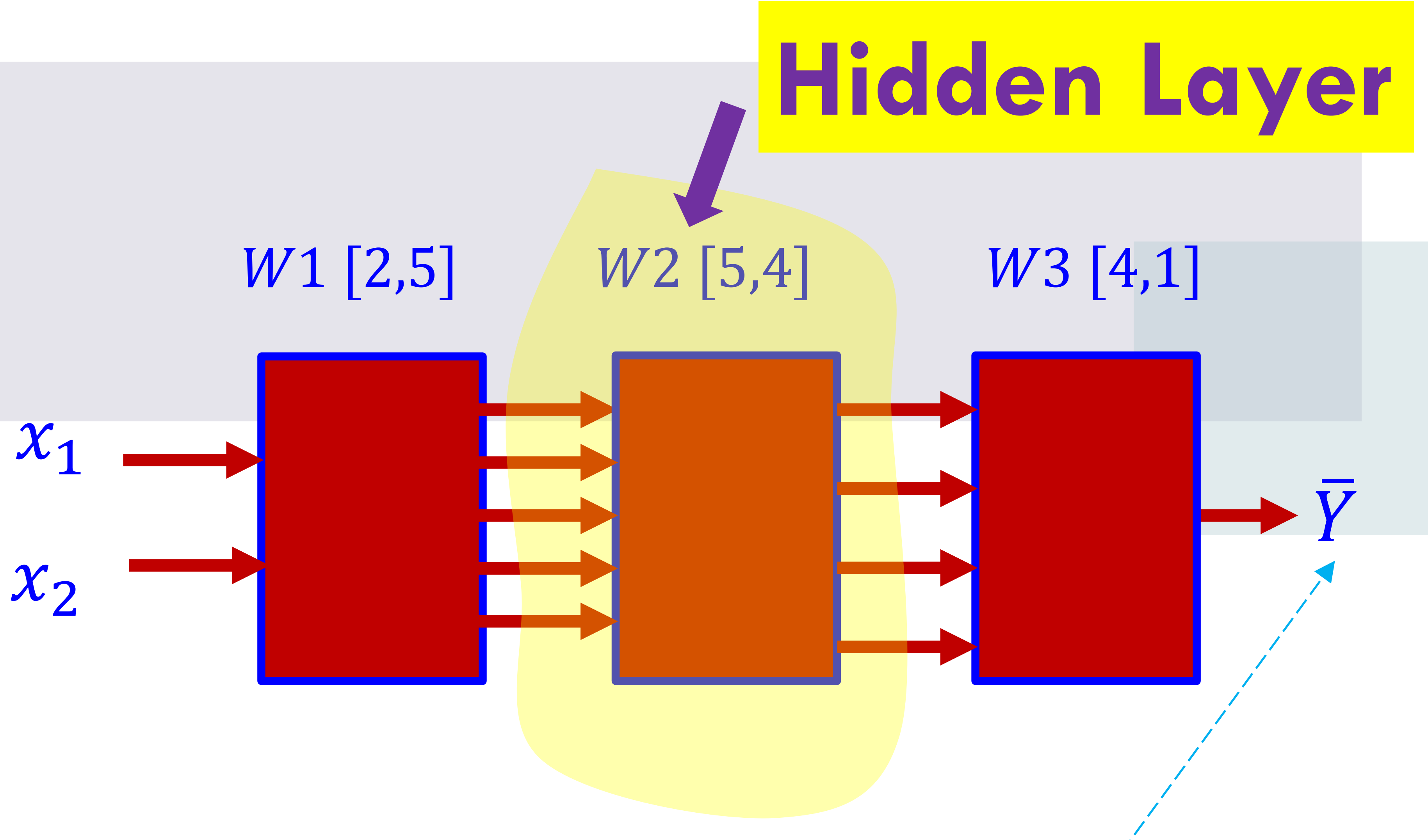
$$W_1 = \begin{bmatrix} 5, -7 \\ 5, -7 \end{bmatrix},\ B_1 = [-8, 3]$$

NN for XOR




```python
W1 = tf.Variable(tf.random_uniform([2, 2], -1.0, 1.0))
W2 = tf.Variable(tf.random_uniform([2, 1], -1.0, 1.0))

b1 = tf.Variable(tf.zeros([2]), name="Bias1")
b2 = tf.Variable(tf.zeros([1]), name="Bias2")

# Our hypothesis
L2 = tf.sigmoid(tf.matmul(X, W1) + b1)
hypothesis = tf.sigmoid(tf.matmul(L2, W2) + b2)
```

Let's go deep & Wide




```
W1 = tf.Variable(tf.random_uniform([2, 5], -1.0, 1.0))
W2 = tf.Variable(tf.random_uniform([5, 4], -1.0, 1.0))
W3 = tf.Variable(tf.random_uniform([4, 1], -1.0, 1.0))

b1 = tf.Variable(tf.zeros([5]), name="Bias1")
b2 = tf.Variable(tf.zeros([4]), name="Bias2")
b3 = tf.Variable(tf.zeros([1]), name="Bias2")

# Our hypothesis
L2 = tf.sigmoid(tf.matmul(X, W1) + b1)
L3 = tf.sigmoid(tf.matmul(L2, W2) + b2)
hypothesis = tf.sigmoid(tf.matmul(L3, W3) + b3)
```

0 or 1

9 hidden layers!

```
W1 = tf.Variable(tf.random_uniform([2, 5], -1.0, 1.0), name = "Weight1")

W2 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight2")
W3 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight3")
W4 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight4")
W5 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight5")
W6 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight6")
W7 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight7")
W8 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight8")
W9 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight9")
W10 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight10")

W11 = tf.Variable(tf.random_uniform([5, 1], -1.0, 1.0), name = "Weight11")

b1 = tf.Variable(tf.zeros([5]), name="Bias1")
b2 = tf.Variable(tf.zeros([5]), name="Bias2")
b3 = tf.Variable(tf.zeros([5]), name="Bias3")
b4 = tf.Variable(tf.zeros([5]), name="Bias4")
b5 = tf.Variable(tf.zeros([5]), name="Bias5")
b6 = tf.Variable(tf.zeros([5]), name="Bias6")
b7 = tf.Variable(tf.zeros([5]), name="Bias7")
b8 = tf.Variable(tf.zeros([5]), name="Bias8")
b9 = tf.Variable(tf.zeros([5]), name="Bias9")
b10 = tf.Variable(tf.zeros([5]), name="Bias10")

b11 = tf.Variable(tf.zeros([1]), name="Bias11")
```

9 hidden layers!

```python
W1 = tf.Variable(tf.random_uniform([2, 5], -1.0, 1.0), name = "Weight1")

W2 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight2")
W3 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight3")
W4 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight4")
W5 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight5")
W6 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight6")
W7 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight7")
W8 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight8")
W9 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight9")
W10 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight10")

W11 = tf.Variable(tf.random_uniform([5, 1], -1.0, 1.0), name = "Weight11")

b1 = tf.Variable(tf.zeros([5]), name="Bias1")
b2 = tf.Variable(tf.zeros([5]), name="Bias2")
b3 = tf.Variable(tf.zeros([5]), name="Bias3")
b4 = tf.Variable(tf.zeros([5]), name="Bias4")
b5 = tf.Variable(tf.zeros([5]), name="Bias5")
b6 = tf.Variable(tf.zeros([5]), name="Bias6")
b7 = tf.Variable(tf.zeros([5]), name="Bias7")
b8 = tf.Variable(tf.zeros([5]), name="Bias8")
b9 = tf.Variable(tf.zeros([5]), name="Bias9")
b10 = tf.Variable(tf.zeros([5]), name="Bias10")

b11 = tf.Variable(tf.zeros([1]), name="Bias11")
```

```python
# Our hypothesis
L1 = tf.sigmoid(tf.matmul(X, W1) + b1)
L2 = tf.sigmoid(tf.matmul(L1, W2) + b2)
L3 = tf.sigmoid(tf.matmul(L2, W3) + b3)
L4 = tf.sigmoid(tf.matmul(L3, W4) + b4)
L5 = tf.sigmoid(tf.matmul(L4, W5) + b5)
L6 = tf.sigmoid(tf.matmul(L5, W6) + b6)
L7 = tf.sigmoid(tf.matmul(L6, W7) + b7)
L8 = tf.sigmoid(tf.matmul(L7, W8) + b8)
L9 = tf.sigmoid(tf.matmul(L8, W9) + b9)
L10 = tf.sigmoid(tf.matmul(L9, W10) + b10)

hypothesis = tf.sigmoid(tf.matmul(L10, W11) + b11)
```

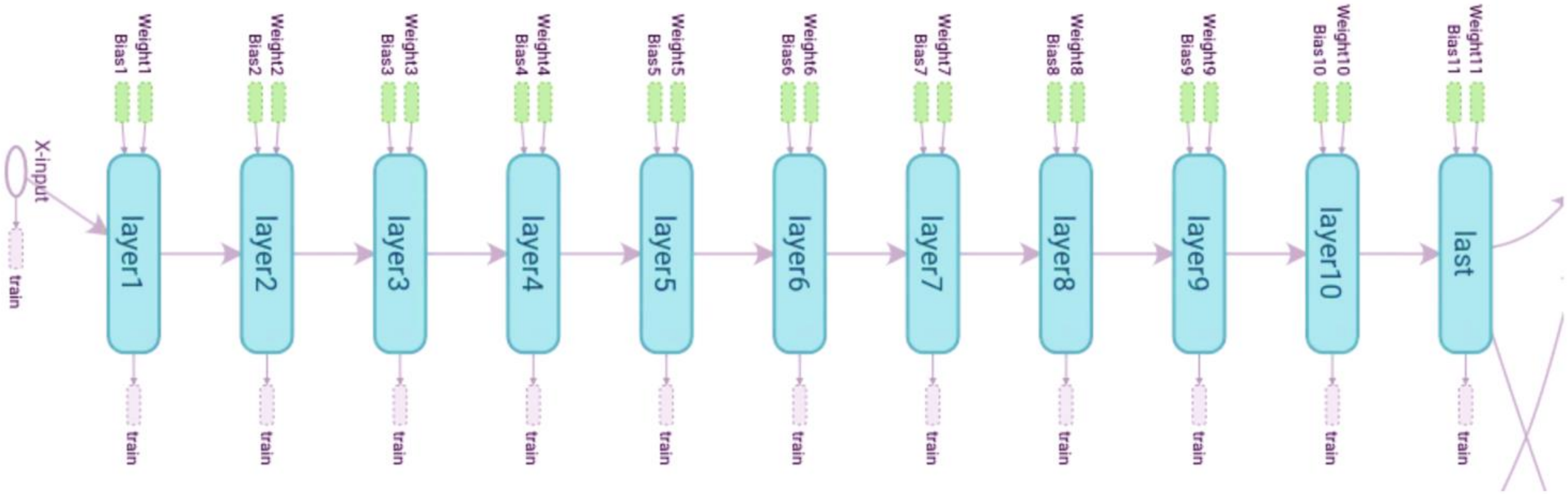
9 hidden layers!

```python
W1 = tf.Variable(tf.random_uniform([2, 5], -1.0, 1.0), name = "Weight1")

W2 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight2")
W3 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight3")
W4 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight4")
W5 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight5")
W6 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight6")
W7 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight7")
W8 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight8")
W9 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight9")
W10 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight10")

W11 = tf.Variable(tf.random_uniform([5, 1], -1.0, 1.0), name = "Weight11")

b1 = tf.Variable(tf.zeros([5]), name="Bias1")
b2 = tf.Variable(tf.zeros([5]), name="Bias2")
b3 = tf.Variable(tf.zeros([5]), name="Bias3")
b4 = tf.Variable(tf.zeros([5]), name="Bias4")
b5 = tf.Variable(tf.zeros([5]), name="Bias5")
b6 = tf.Variable(tf.zeros([5]), name="Bias6")
b7 = tf.Variable(tf.zeros([5]), name="Bias7")
b8 = tf.Variable(tf.zeros([5]), name="Bias8")
b9 = tf.Variable(tf.zeros([5]), name="Bias9")
b10 = tf.Variable(tf.zeros([5]), name="Bias10")

b11 = tf.Variable(tf.zeros([1]), name="Bias11")
```

Tensorboard

```python
# Our hypothesis
with tf.name_scope("layer1") as scope:
    L1 = tf.sigmoid(tf.matmul(X, W1) + b1)
with tf.name_scope("layer2") as scope:
    L2 = tf.sigmoid(tf.matmul(L1, W2) + b2)
with tf.name_scope("layer3") as scope:
    L3 = tf.sigmoid(tf.matmul(L2, W3) + b3)
with tf.name_scope("layer4") as scope:
    L4 = tf.sigmoid(tf.matmul(L3, W4) + b4)
with tf.name_scope("layer5") as scope:
    L5 = tf.sigmoid(tf.matmul(L4, W5) + b5)
with tf.name_scope("layer6") as scope:
    L6 = tf.sigmoid(tf.matmul(L5, W6) + b6)
with tf.name_scope("layer7") as scope:
    L7 = tf.sigmoid(tf.matmul(L6, W7) + b7)
with tf.name_scope("layer8") as scope:
    L8 = tf.sigmoid(tf.matmul(L7, W8) + b8)
with tf.name_scope("layer9") as scope:
    L9 = tf.sigmoid(tf.matmul(L8, W9) + b9)
with tf.name_scope("layer10") as scope:
    L10 = tf.sigmoid(tf.matmul(L9, W10) + b10)

with tf.name_scope("last") as scope:
    hypothesis = tf.sigmoid(tf.matmul(L10, W11) + b11)
```

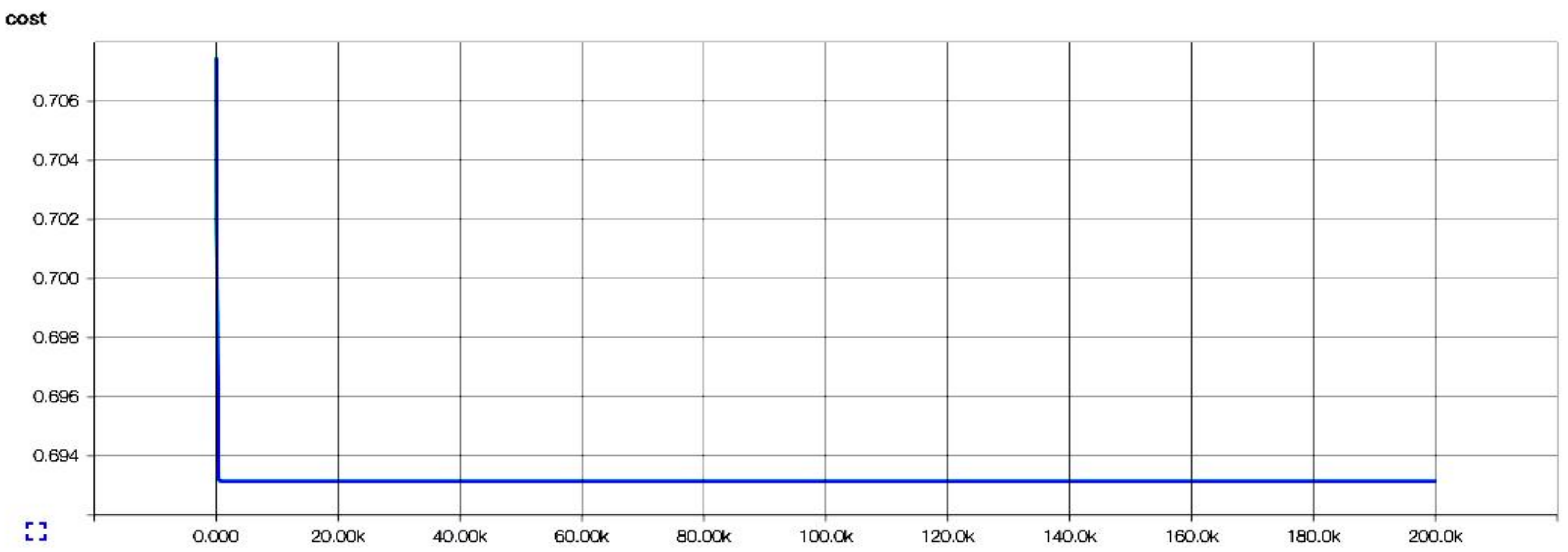
Tensorboard visualization

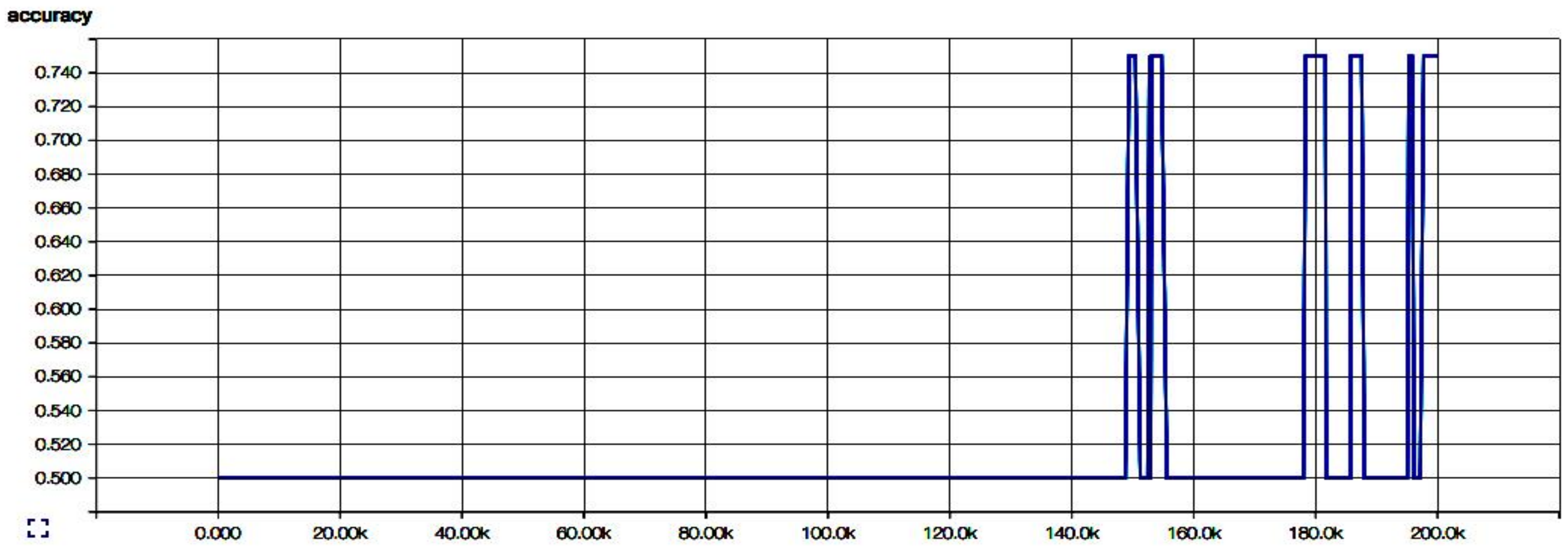


Poor results?

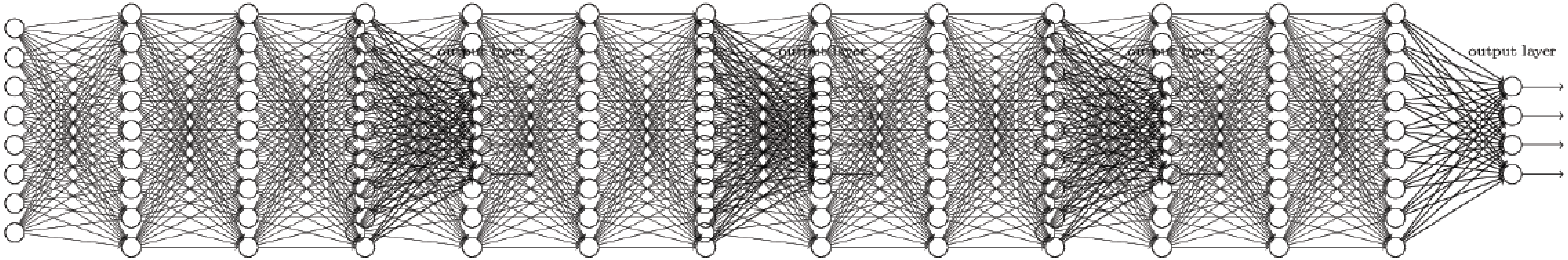
```
196000 [0.69314718, array([[ 0.49999988],
       [ 0.50000006],
       [ 0.49999982],
       [ 0.5       ]], dtype=float32)]
198000 [0.69314718, array([[ 0.49999988],
       [ 0.50000006],
       [ 0.49999982],
       [ 0.5       ]], dtype=float32)]
[array([[ 0.49999988],
       [ 0.50000006],
       [ 0.49999982],
       [ 0.5       ]], dtype=float32), array([[ 0.],
       [ 1.],
       [ 0.],
       [ 1.]], dtype=float32)]
Accuracy: 0.5
```

Tensorboard Cost & Accuracy

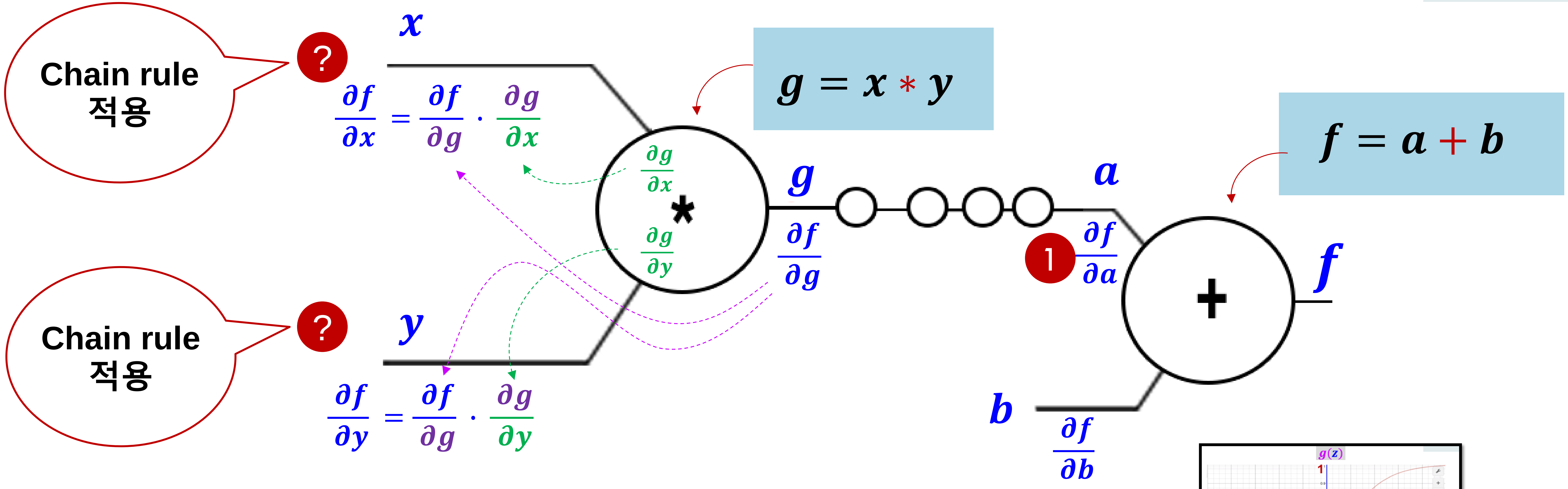


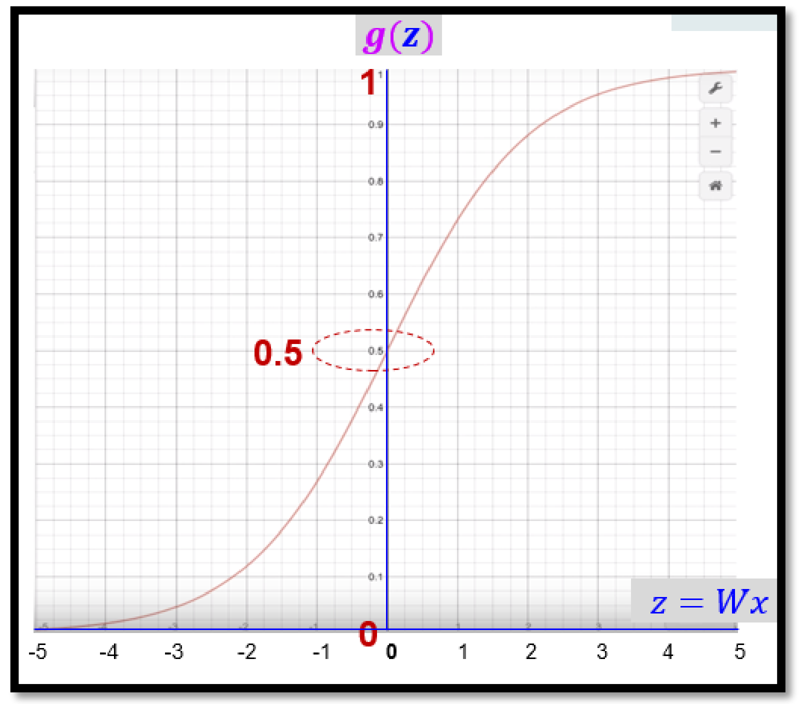


Backpropagation



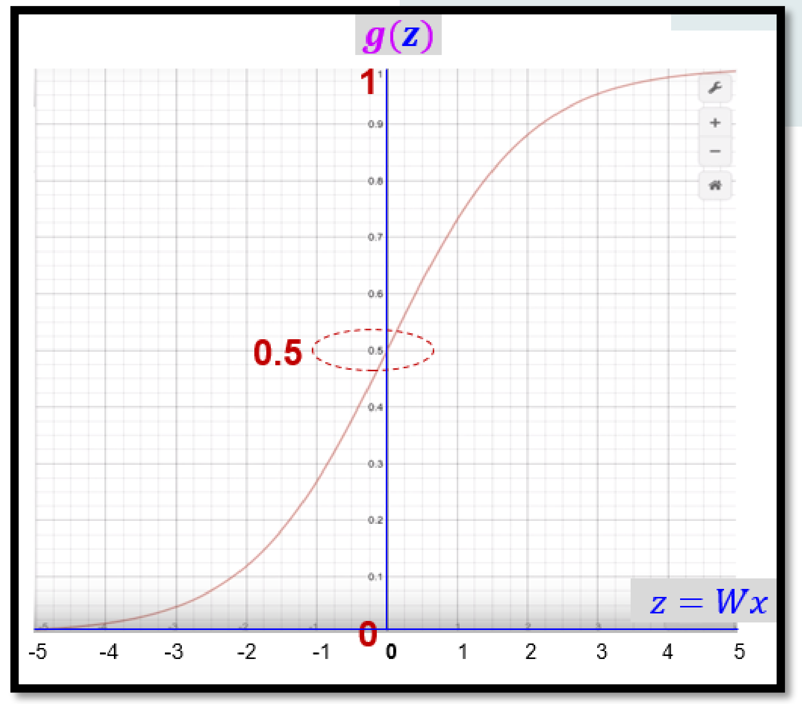
Backpropagation (chain rule)



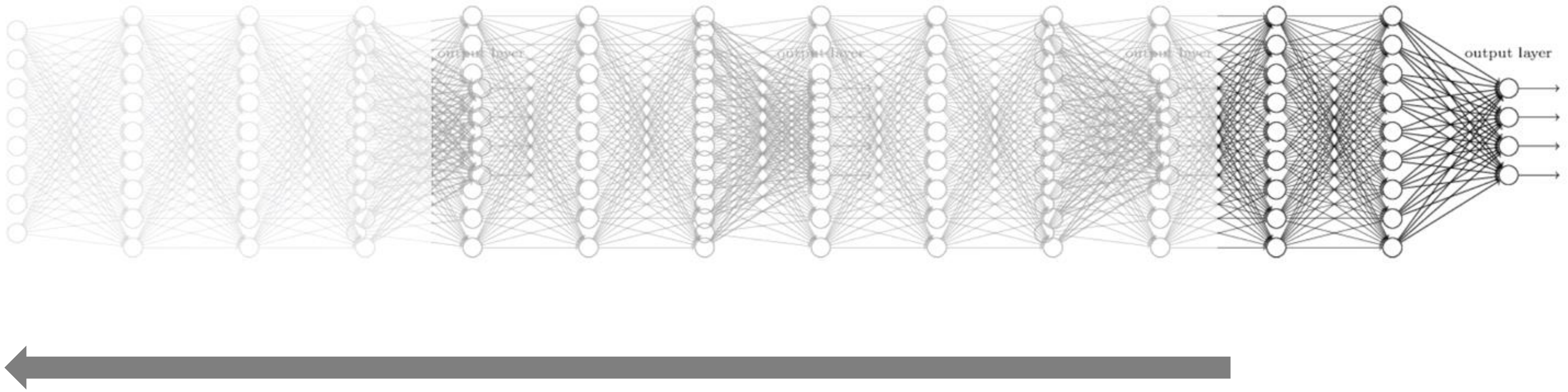


Backward 방향으로 1보다 작은 값들이 계속 곱해진다 !!

Vanishing gradient (NN winter2: 1986-2006)



Backward 방향으로 1보다 작은 값들이 계속 곱해진다 !!



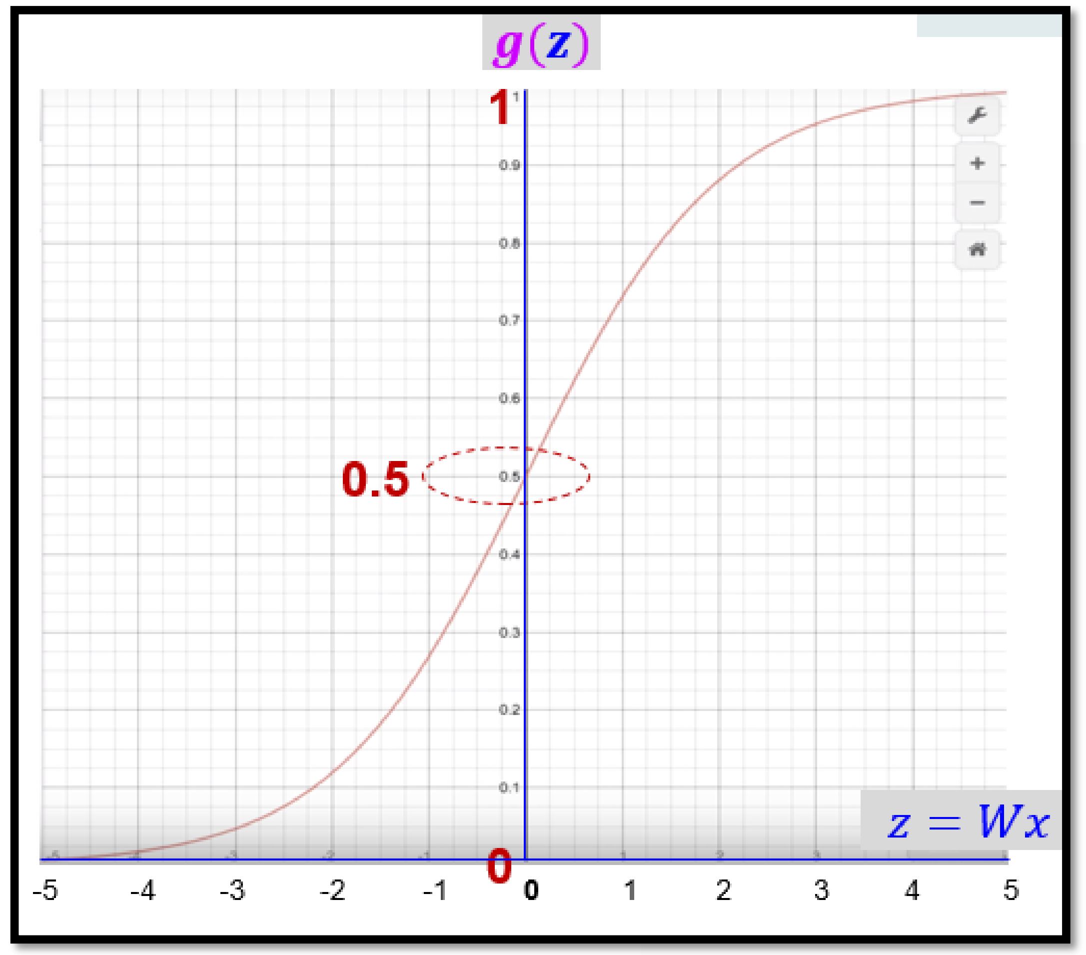
미분 값은 0에 가까운 값이 된다.

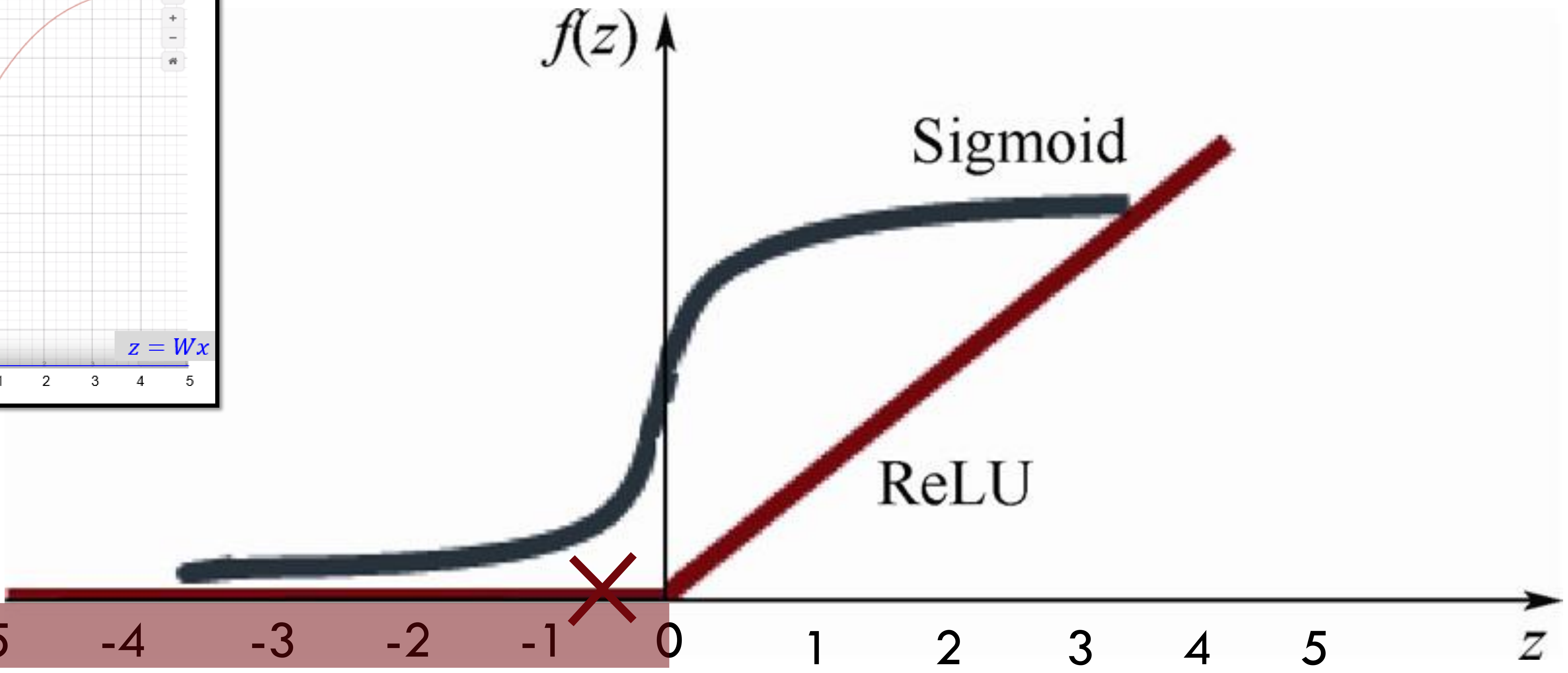
즉, NN은 deep 할수록 gradient가 사라지는 현상이 발생한다. → " Vanishing gradient "

Geoffrey Hinton's summary of findings up to today

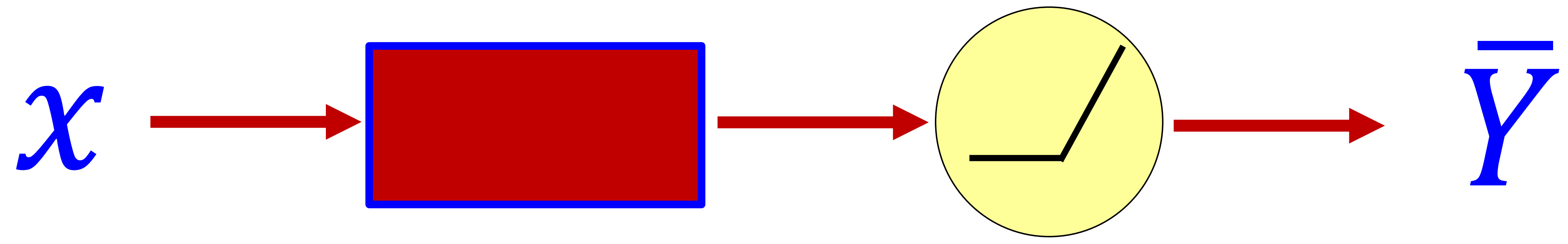
- Our labeled datasets were thousands of times too small.
- Our computers were millions of times too slow.
- We initialized the weights in a stupid way.
- We used the wrong type of non-linearity (sigmoid).

Sigmoid !





ReLU: Rectified Linear Unit



L1 = tf.sigmoid(tf.matmul(X, W1) + b1)

L1 = tf.nn.relu(tf.matmul(X, W1) + b1)

max (0, X) → 0 or X 의 값 중 하나가 출력됨

ReLU

Tensorboard

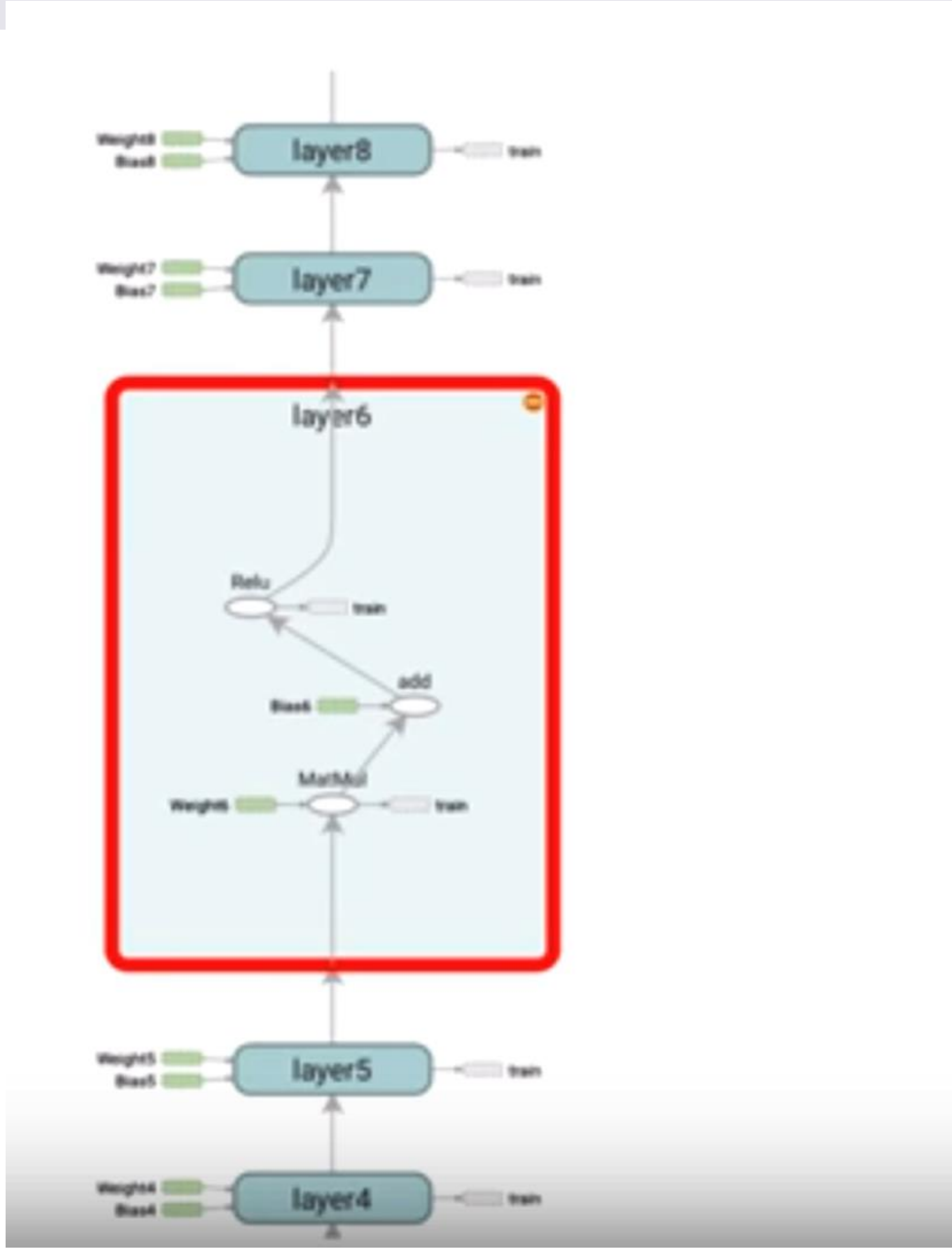
```
# Our hypothesis
with tf.name_scope("layer1") as scope:
    L1 = tf.nn.relu(tf.matmul(X, W1) + b1)
with tf.name_scope("layer2") as scope:
    L2 = tf.nn.relu(tf.matmul(L1, W2) + b2)
with tf.name_scope("layer3") as scope:
    L3 = tf.nn.relu(tf.matmul(L2, W3) + b3)
with tf.name_scope("layer4") as scope:
    L4 = tf.nn.relu(tf.matmul(L3, W4) + b4)
with tf.name_scope("layer5") as scope:
    L5 = tf.nn.relu(tf.matmul(L4, W5) + b5)
with tf.name_scope("layer6") as scope:
    L6 = tf.nn.relu(tf.matmul(L5, W6) + b6)
with tf.name_scope("layer7") as scope:
    L7 = tf.nn.relu(tf.matmul(L6, W7) + b7)
with tf.name_scope("layer8") as scope:
    L8 = tf.nn.relu(tf.matmul(L7, W8) + b8)
with tf.name_scope("layer9") as scope:
    L9 = tf.nn.relu(tf.matmul(L8, W9) + b9)
with tf.name_scope("layer10") as scope:
    L10 = tf.nn.relu(tf.matmul(L9, W10) + b10)

with tf.name_scope("last") as scope:
    hypothesis = tf.sigmoid(tf.matmul(L10, W11) + b11)
```

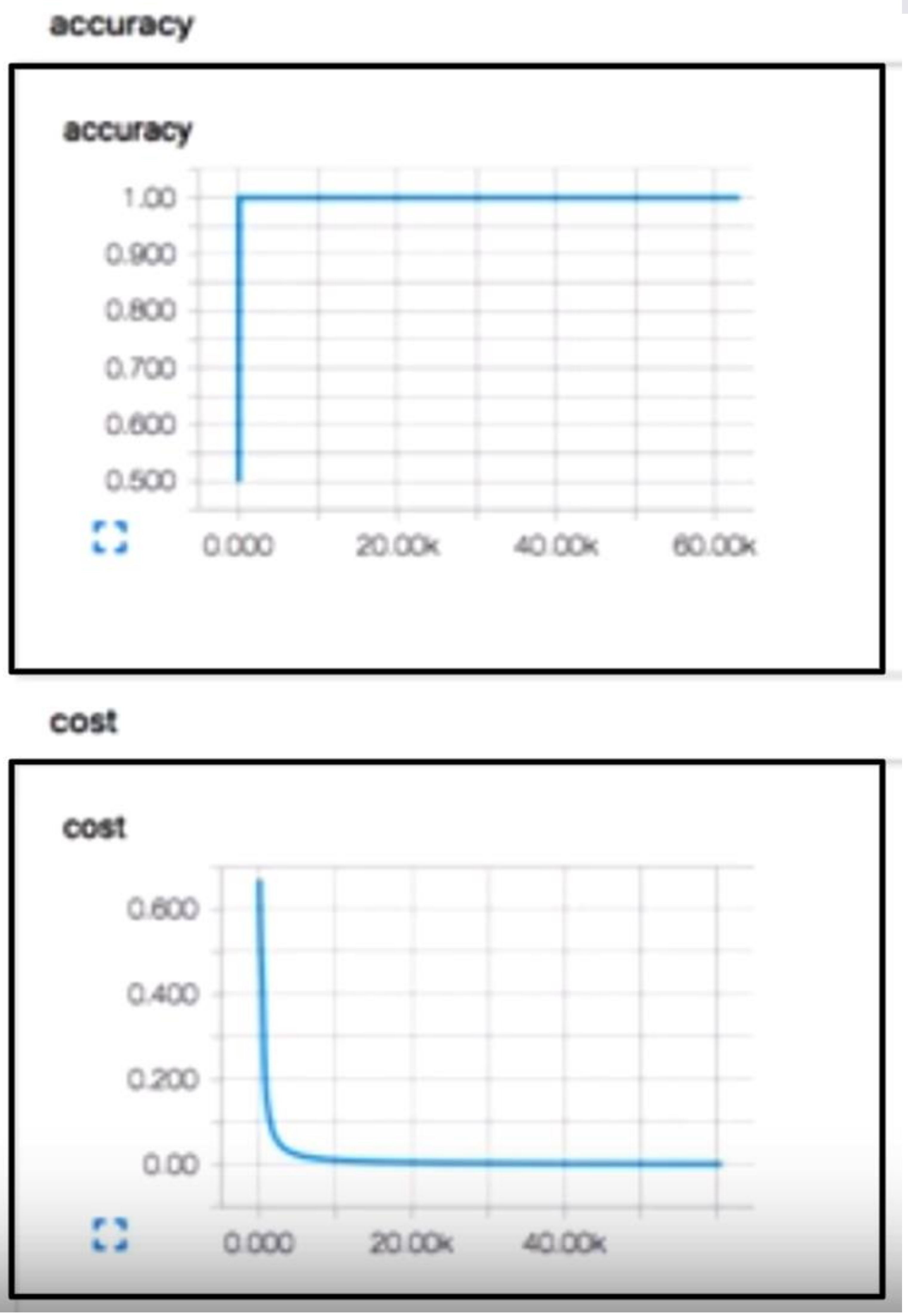
마지막 단에는
0 or 1 로 출력하기 위해서
sigmoid 사용

Works very well

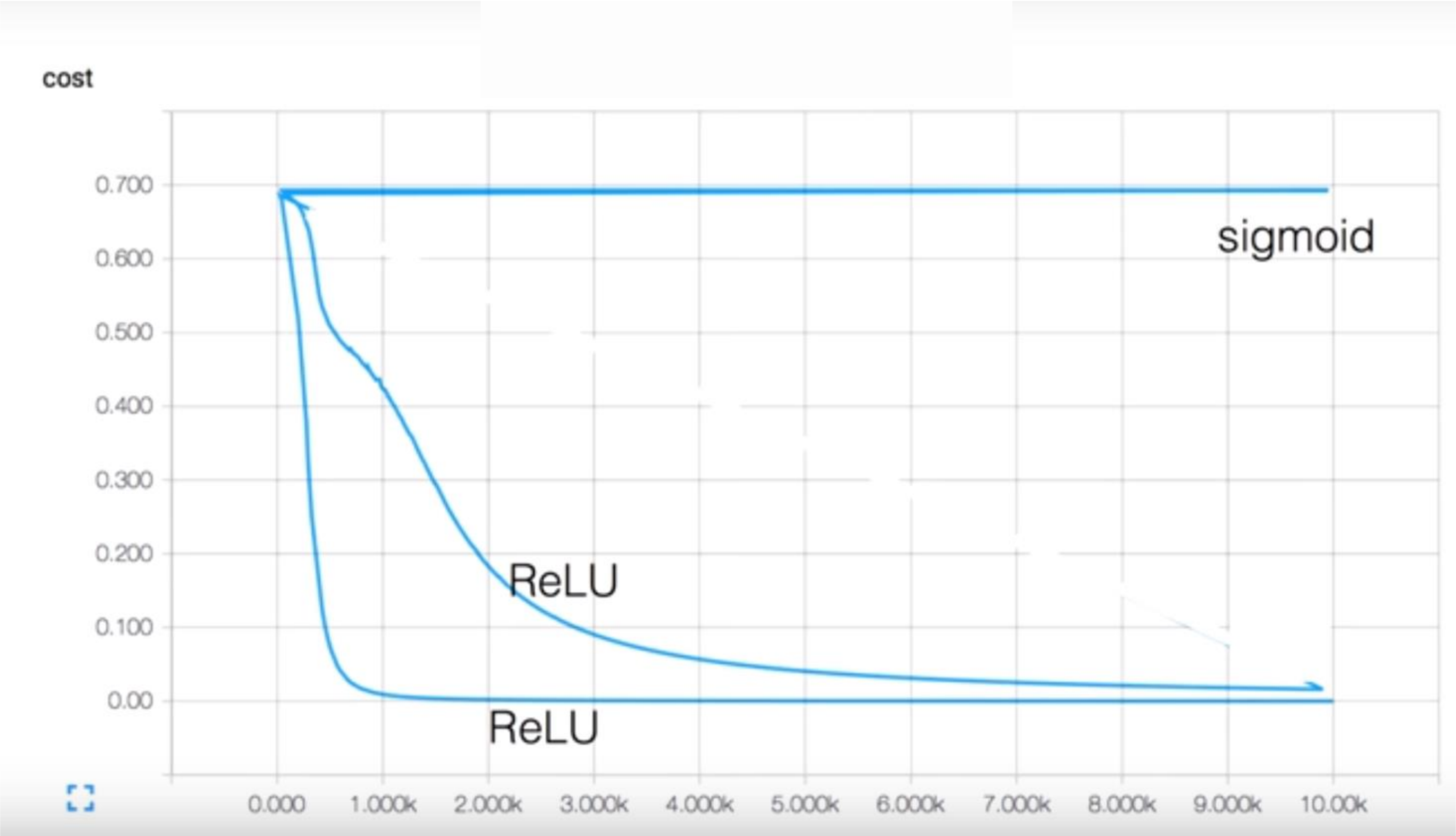
```
196000 [2.6226094e-06, array([[  2.59195826e-06],
       [  9.99999642e-01],
       [  9.99994874e-01],
       [  2.43454133e-06]], dtype=float32)]
198000 [2.607708e-06, array([[  2.55822852e-06],
       [  9.99999642e-01],
       [  9.99994874e-01],
       [  2.40260101e-06]], dtype=float32)]
[array([[  2.52509381e-06],
       [  9.99999642e-01],
       [  9.99994874e-01],
       [  2.37124709e-06]], dtype=float32), array([[ 0.],
       [ 1.],
       [ 1.],
       [ 0.]], dtype=float32)]
Accuracy: 1.0
```

Works very well





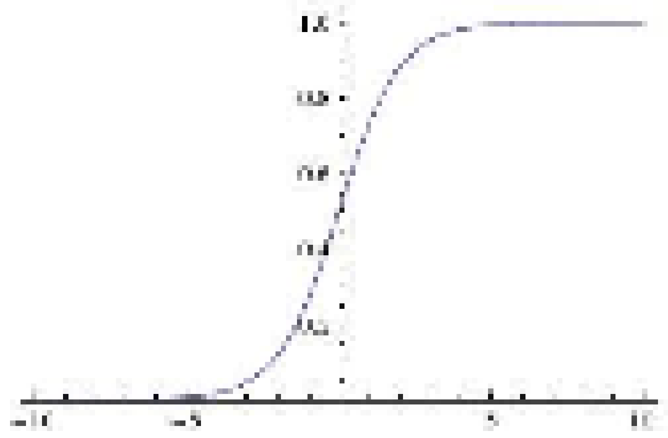
Cost function



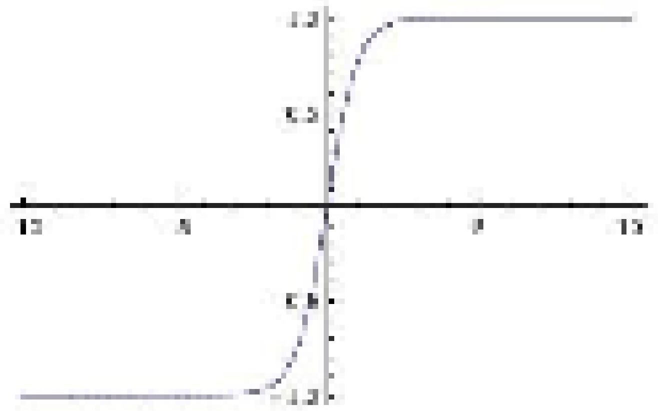
Activation Functions

Sigmoid

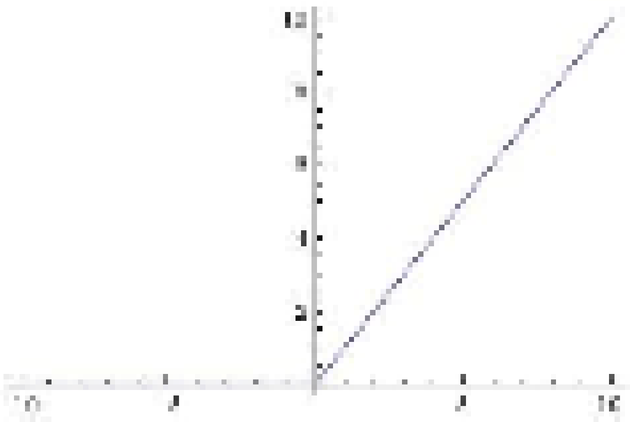
$\sigma(x) = 1/(1+e^{-x})$



tanh tanh(x)

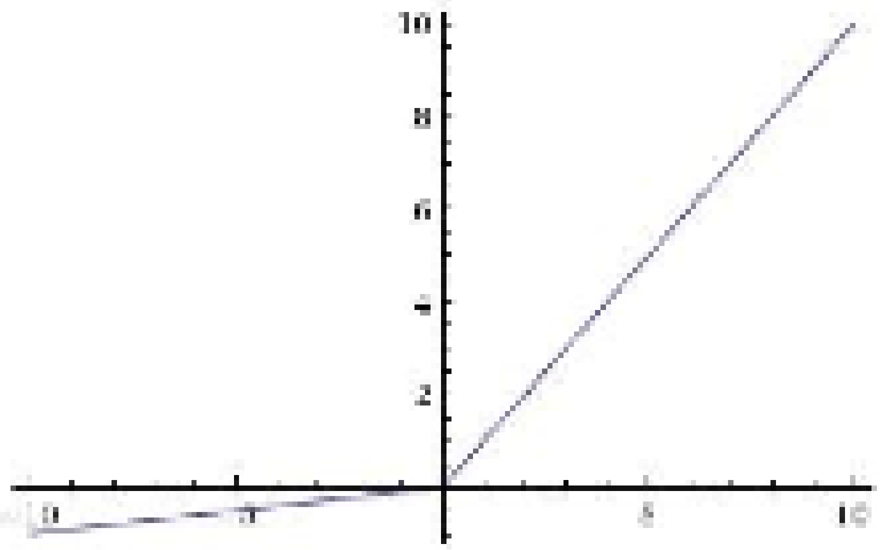


ReLU max(0,x)



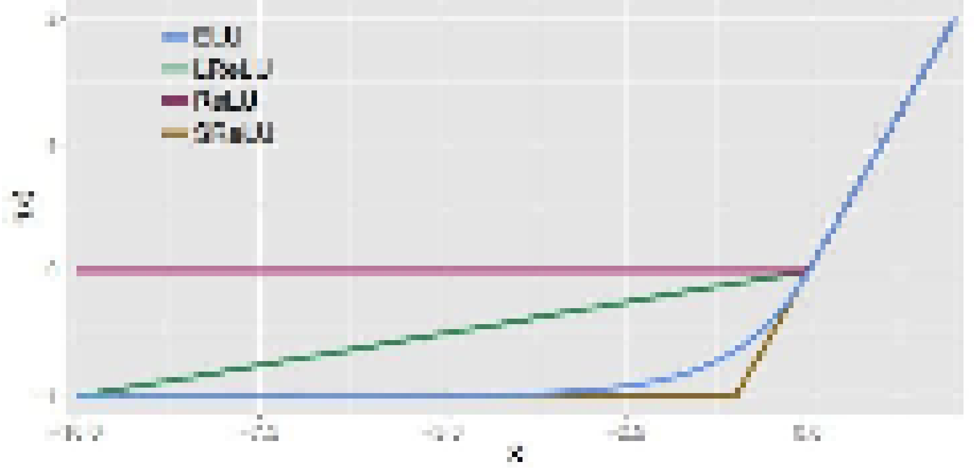
Leaky ReLU

max(0.1x, x)



Maxout $\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU $f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if } x \le 0 \end{cases}$



Activation functions on CIFAR-10

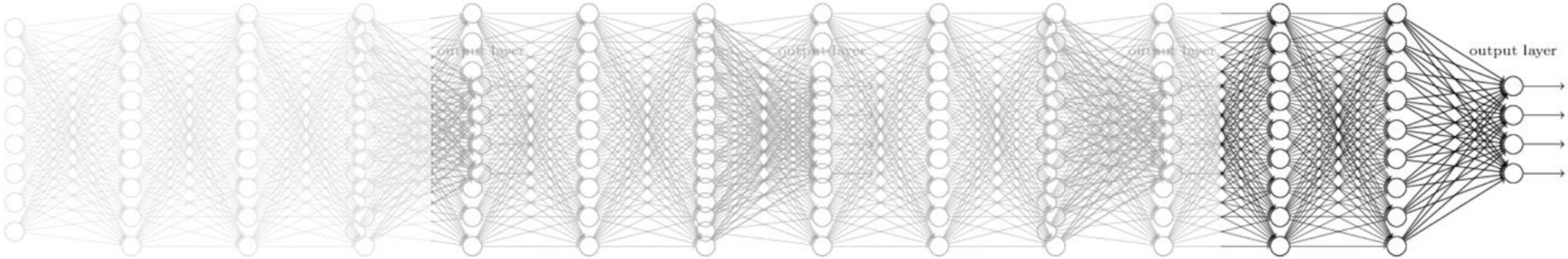
Accuracy

Init method	maxout	ReLU	VLReLU	tanh	Sigmoid
LSUV	**93.94**	**92.11**	92.97	89.28	n/c
OrthoNorm	93.78	91.74	92.40	89.48	n/c
OrthoNorm-MSRA scaled	–	91.93	**93.09**	–	n/c
Xavier	91.75	90.63	92.27	**89.82**	n/c
MSRA	n/c†	90.91	92.43	89.54	n/c

[Mishkin et al. 2015]

Initialize weights in a smart way

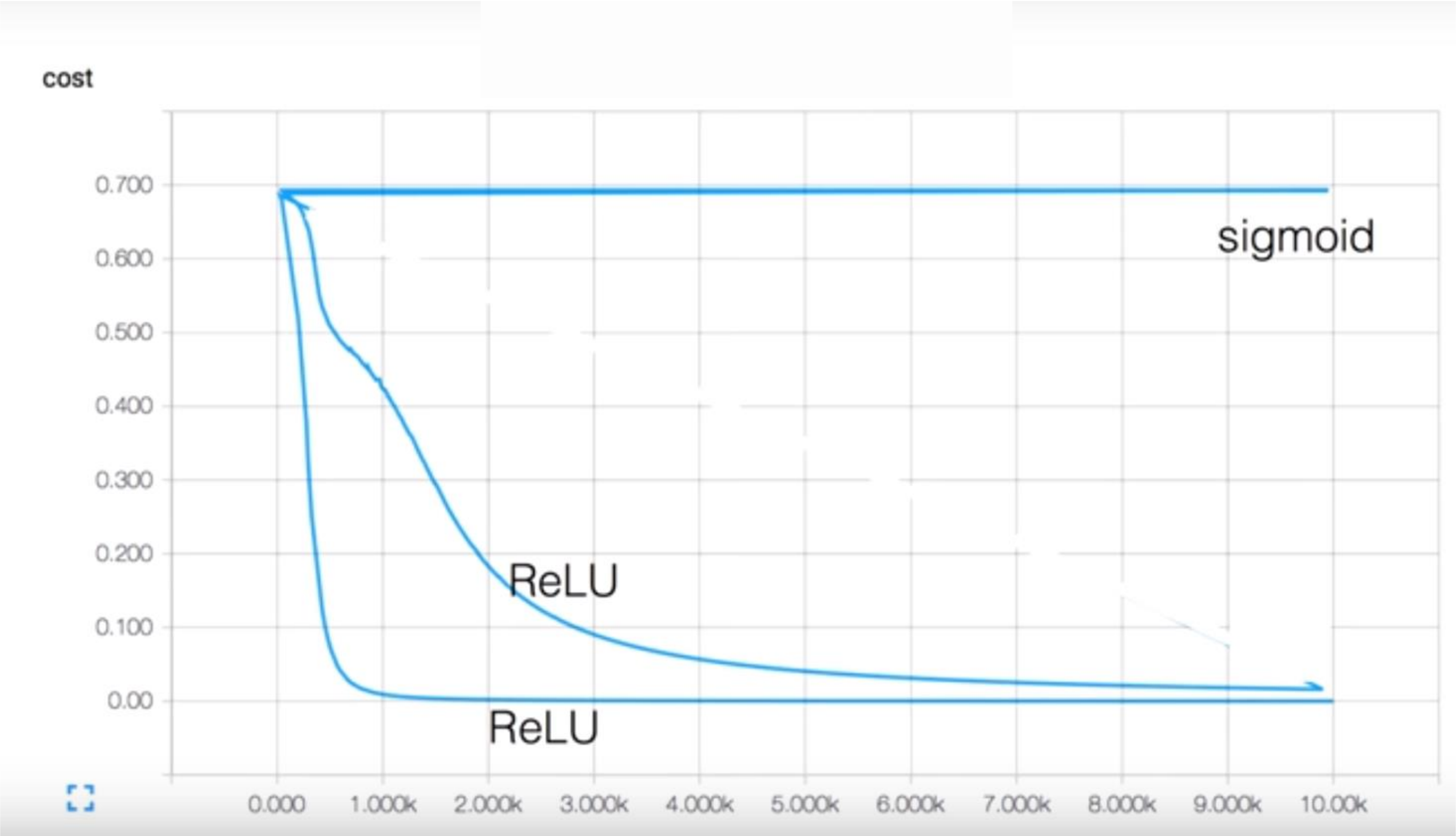
Vanishing gradient



Geoffrey Hinton's summary of findings up to today

- Our labeled datasets were thousands of times too small.
- Our computers were millions of times too slow.
- **We initialized the weights in a stupid way.**
- We used the wrong type of non-linearity.

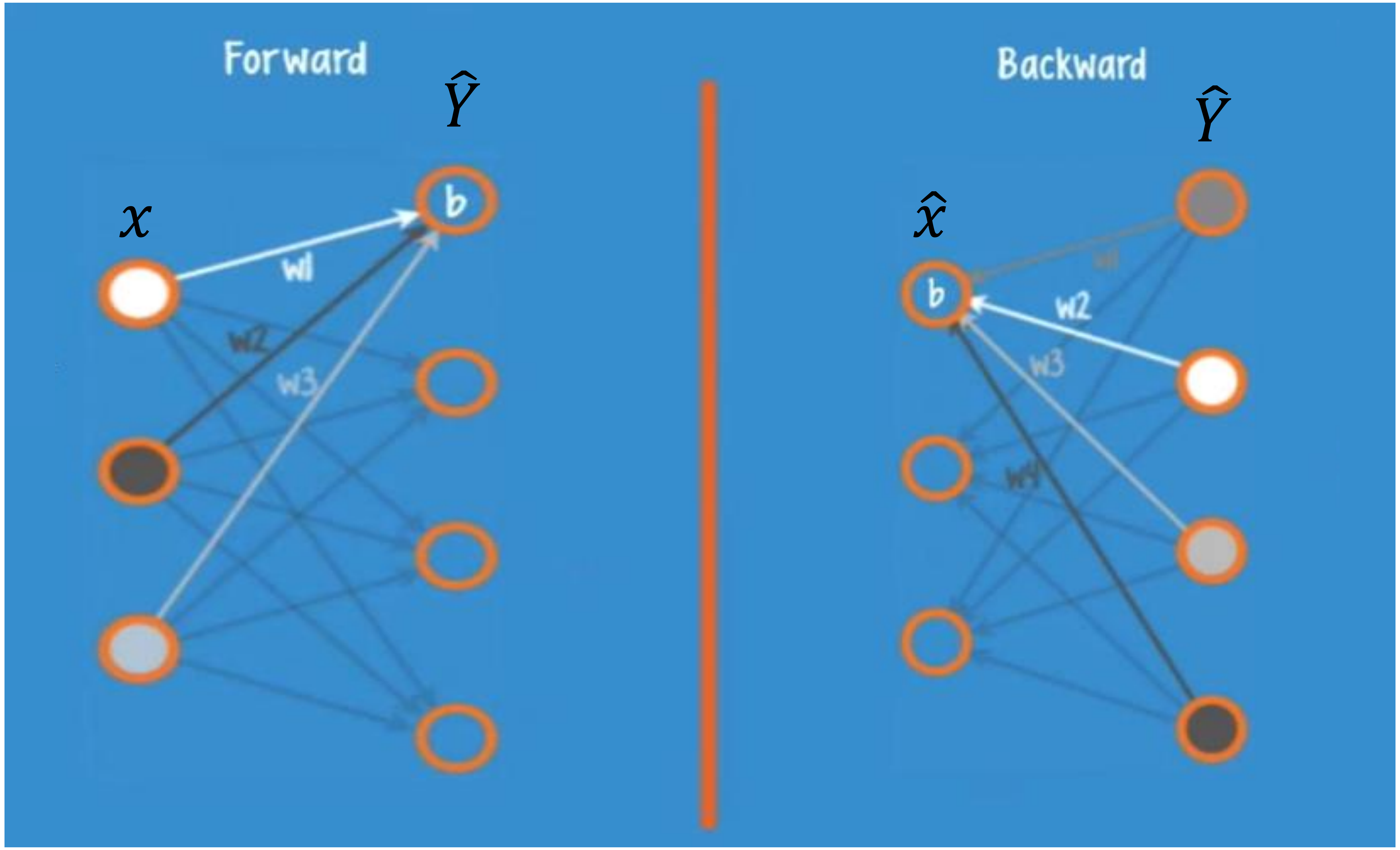
Cost function



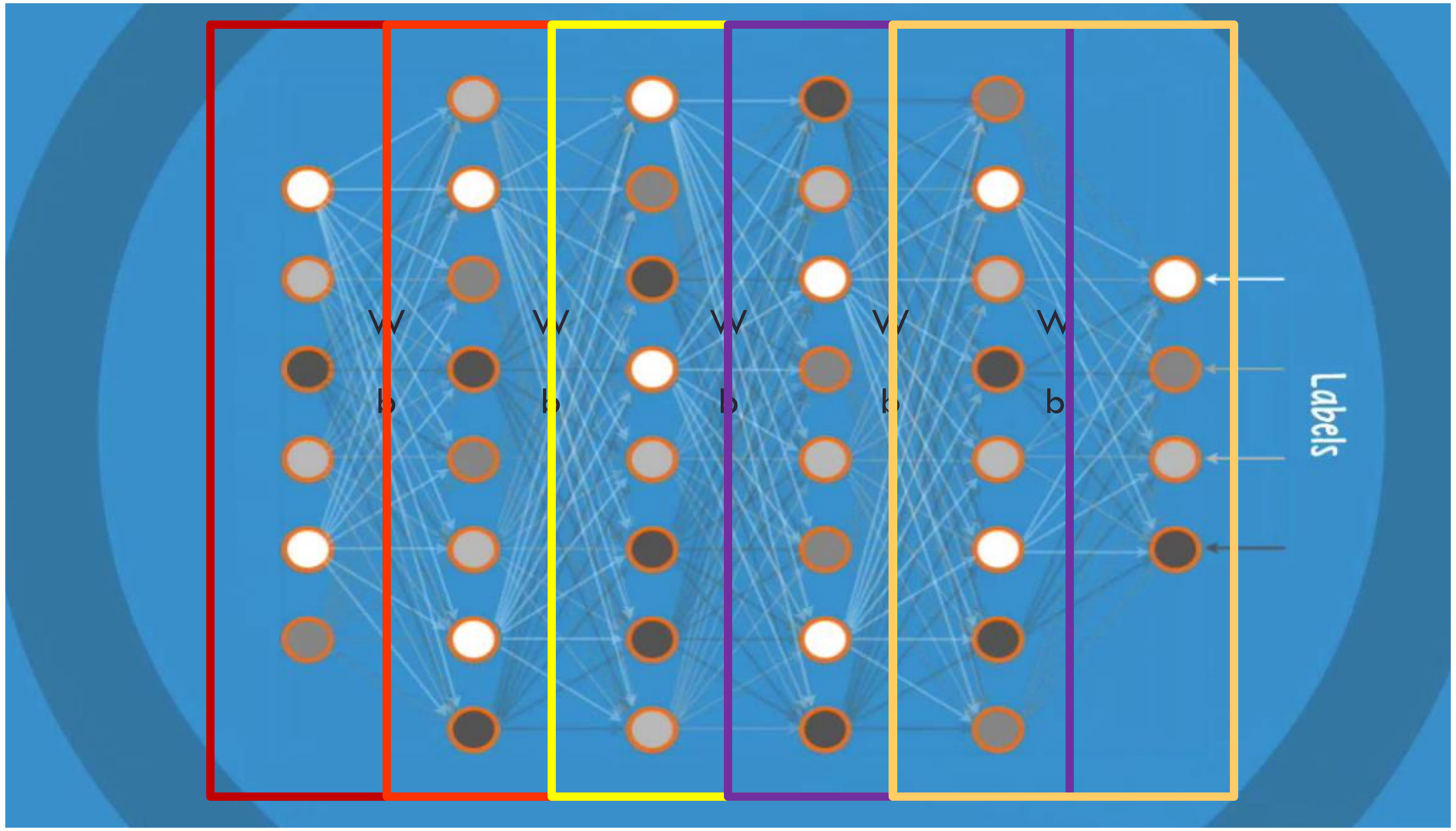
Need to set the initial weight values wisely

- Not all 0's
- 2006 Challenging issue :
  Hinton et al. (2006) "A Fast Learning Algorithm for Deep Belief Nets"
  - Restricted Boatman Machine (RBM)

How can we use RBM to initialize weights?



How can we use RBM to initialize weights?



How can we use RBM to initialize weights?

- Apply the RBM idea on adjacent two layers as a pre-training step
- Continue the first process to all layers
- This will set weights
- Example: Deep Belief Network
  - Weight initialized by RBM

Good news

- No need to use complicated RBM for weight initializations
- Simple methods are OK
  - Xavier initialization: X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in International conference on artificial intelligence and statistics, 2010
  - He's initialization: K. He, X. Zhang, S. Ren, and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," 2015

Xavier/He initialization

- **Makes sure the weights are 'just right', not too small, not too big**
- **Using number of input (fan_in) and output (fan_out)**

```
# Xavier initialization
# Glorot et al. 2010
W = np.random.randn(fan_in, fan_out)/np.sqrt(fan_in)
```

```
# He et al. 2015
W = np.random.randn(fan_in, fan_out)/np.sqrt(fan_in/2)
```

prettytensor implementation

```python
def xavier_init(n_inputs, n_outputs, uniform=True):
  """Set the parameter initialization using the method described.
  This method is designed to keep the scale of the gradients roughly the same
  in all layers.
  Xavier Glorot and Yoshua Bengio (2010):
           Understanding the difficulty of training deep feedforward neural
           networks. International conference on artificial intelligence and
           statistics.
  Args:
    n_inputs: The number of input nodes into each output.
    n_outputs: The number of output nodes for each input.
    uniform: If true use a uniform distribution, otherwise use a normal.
  Returns:
    An initializer.
  """
  if uniform:
    # 6 was used in the paper.
    init_range = math.sqrt(6.0 / (n_inputs + n_outputs))
    return tf.random_uniform_initializer(-init_range, init_range)
  else:
    # 3 gives us approximately the same limits as above since this repicks
    # values greater than 2 standard deviations from the mean.
    stddev = math.sqrt(3.0 / (n_inputs + n_outputs))
    return tf.truncated_normal_initializer(stddev=stddev)
```

Activation functions and initialization on CIFAR-10

Init method	maxout	ReLU	VLReLU	tanh	Sigmoid
LSUV	**93.94**	**92.11**	92.97	89.28	n/c
OrthoNorm	93.78	91.74	92.40	89.48	n/c
OrthoNorm-MSRA scaled	–	91.93	**93.09**	–	n/c
Xavier	91.75	90.63	92.27	**89.82**	n/c
MSRA	n/c†	90.91	92.43	89.54	n/c

[Mishkin et al. 2015]

Still an active area of research

- **We don’t know how to initialize perfect weight values, yet**
- **Many new algorithms**

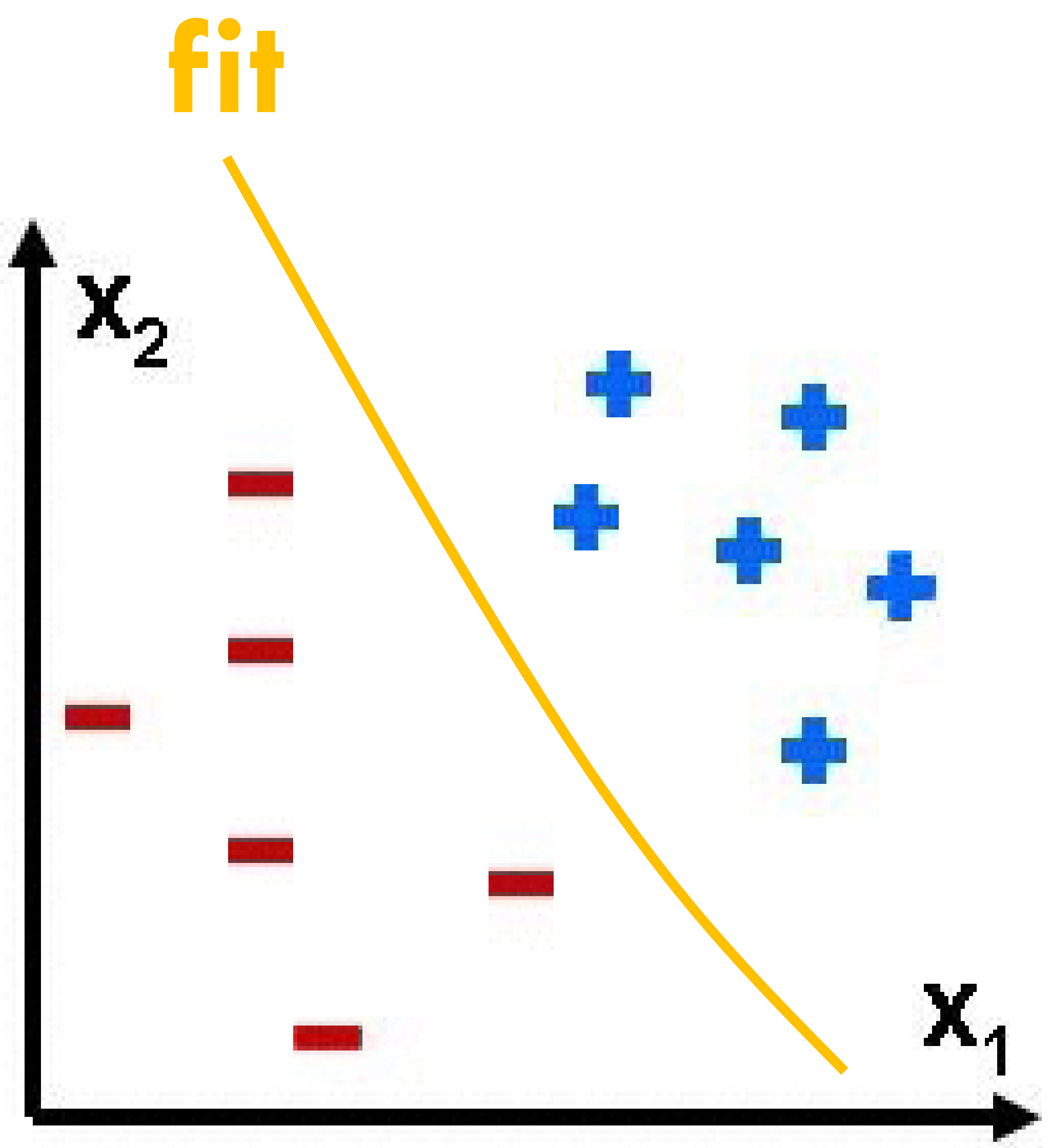
  …

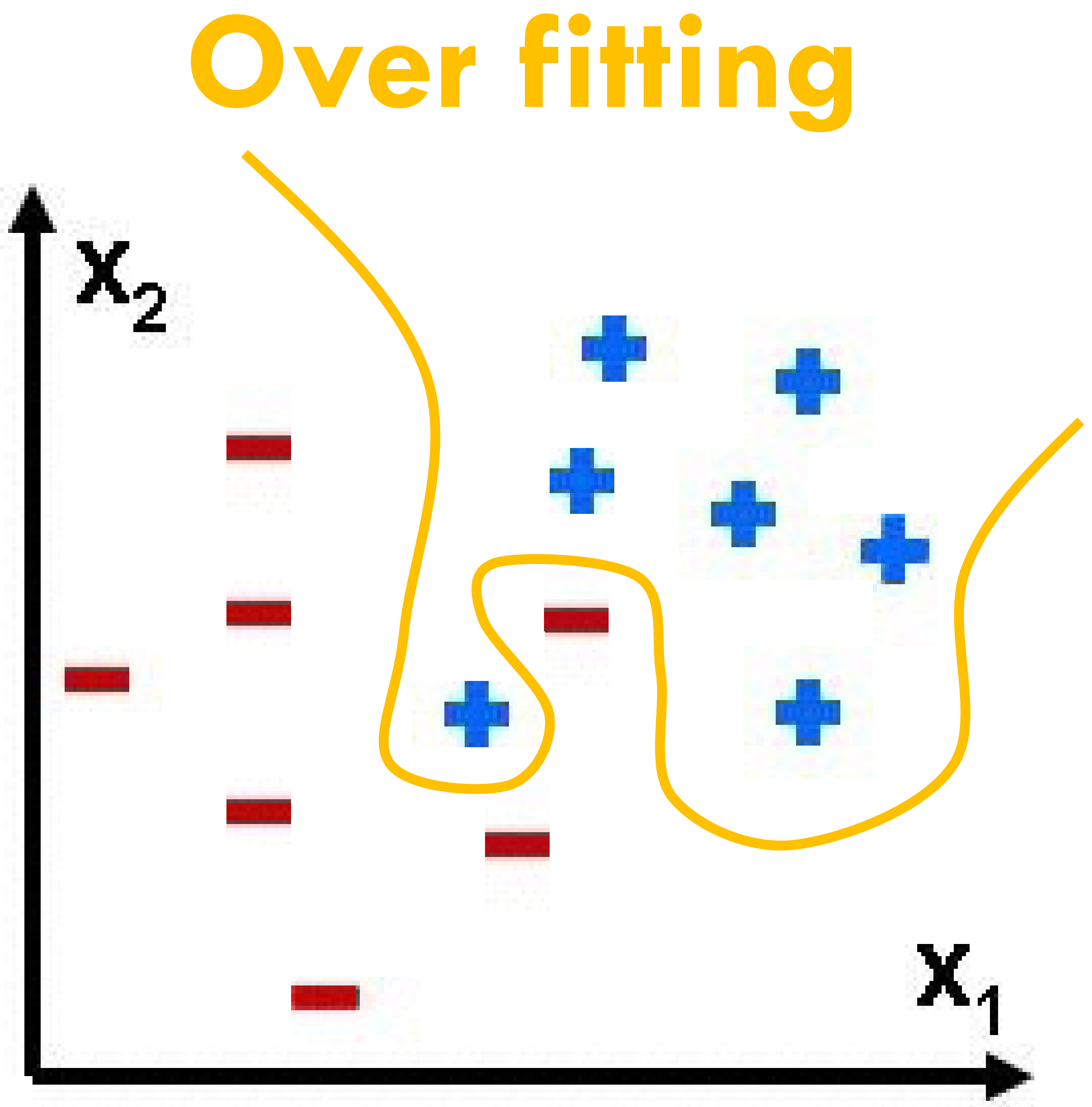
Geoffrey Hinton’s summary of findings up to today

- Our labeled datasets were thousands of times too small.
- Our computers were millions of times too slow.
- **We initialized the weights in a stupid way. → Xavier etc.**
- **We used the wrong type of non-linearity. → ReLu etc.**

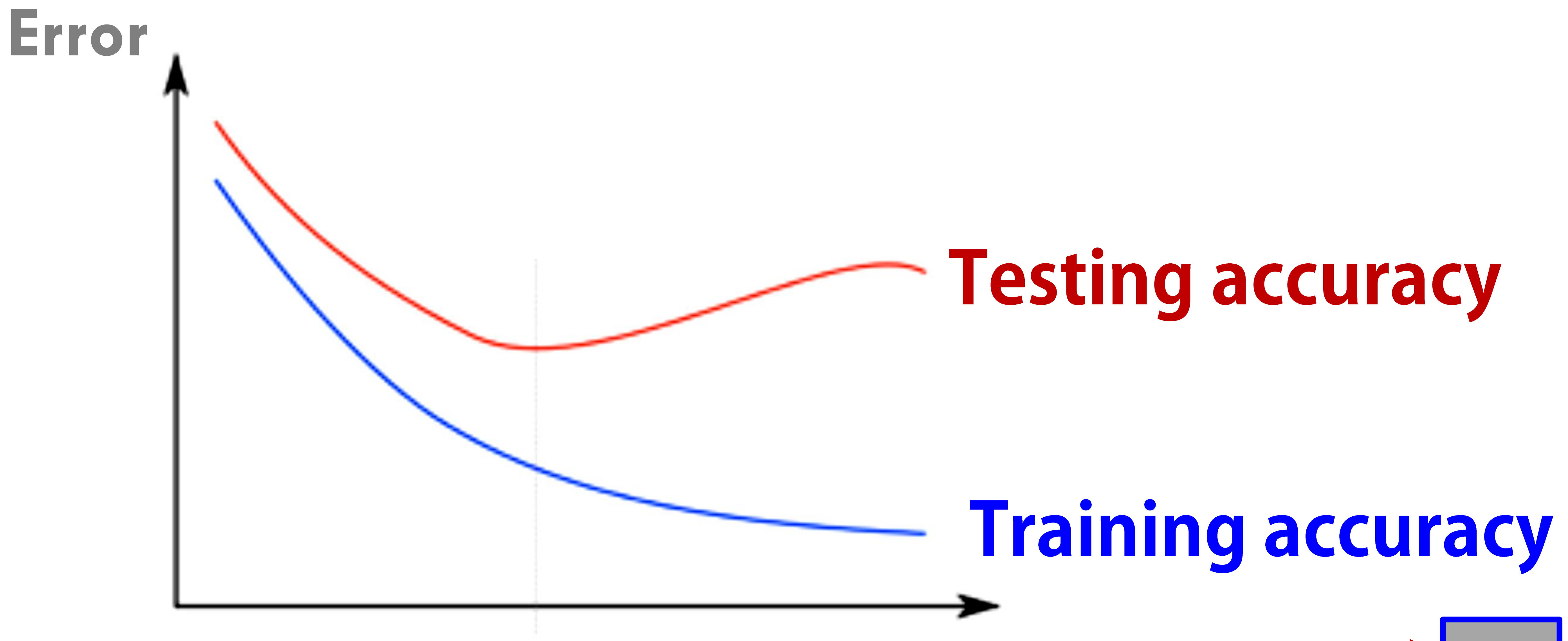
NN dropout and model ensemble

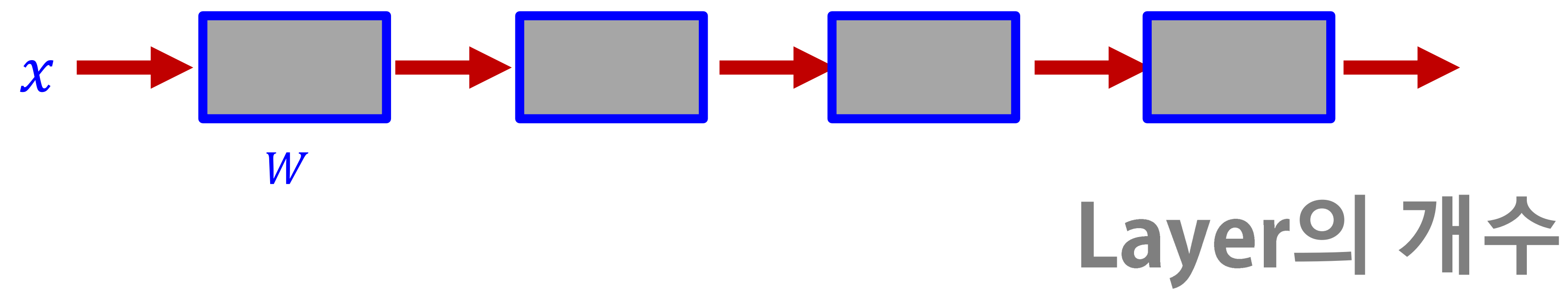
Overfitting





Am I overfitting?





Deep..

- Very high accuracy on the training data set (ex: 0.99)
- Poor accuracy on the test data set (ex: 0.85)

Solutions for overfitting

- More training data!
- Reduce the number of features
- Regularization

Regularization

- Let's not have too big numbers in the weight

Regularization strength

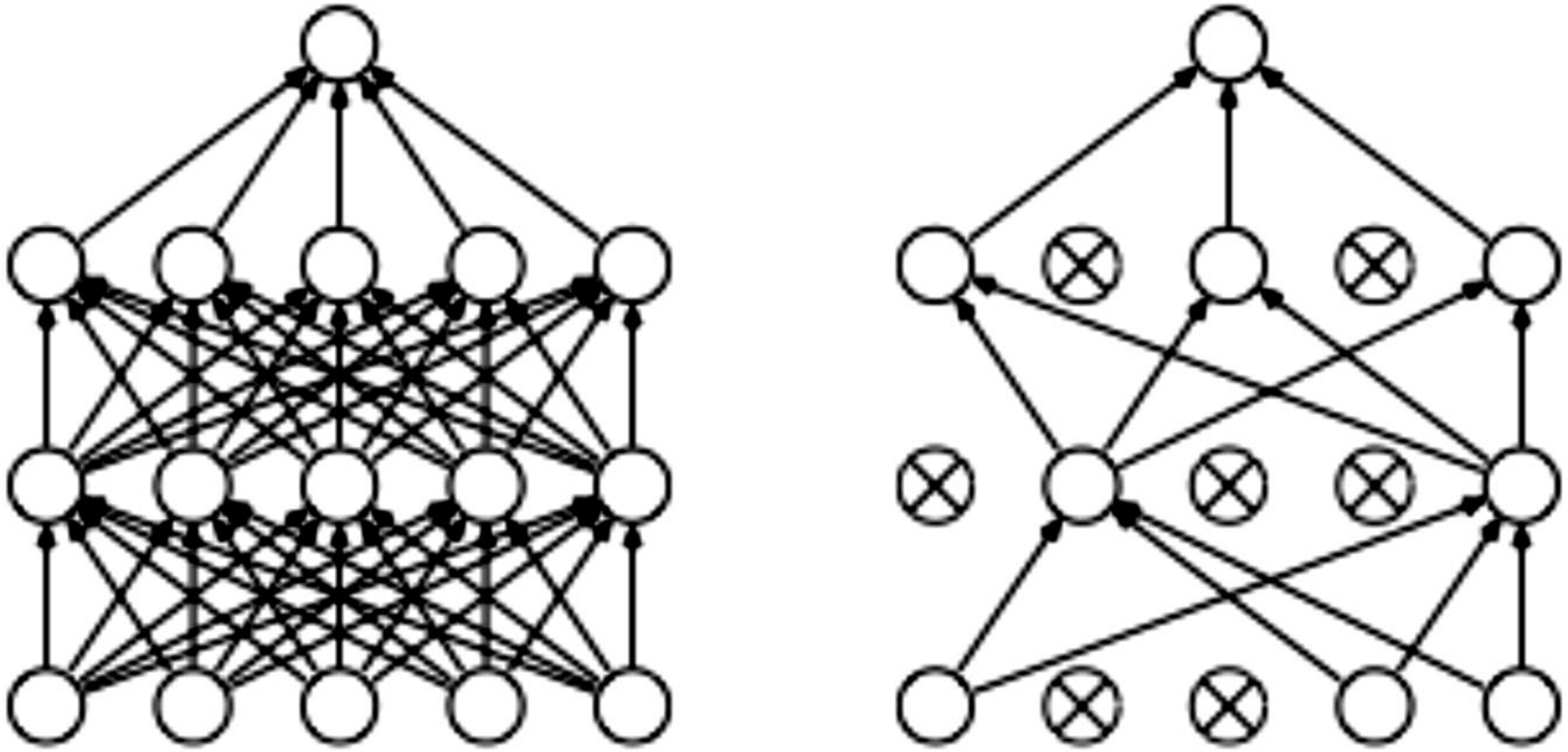
$$cost + \lambda \sum W^2$$

```
l2reg = 0.001 * tf.reduce_sum(tf.square(W))
```

L2 Regularization

Dropout : A Simple Way to Prevent Neural Networks from Overfitting

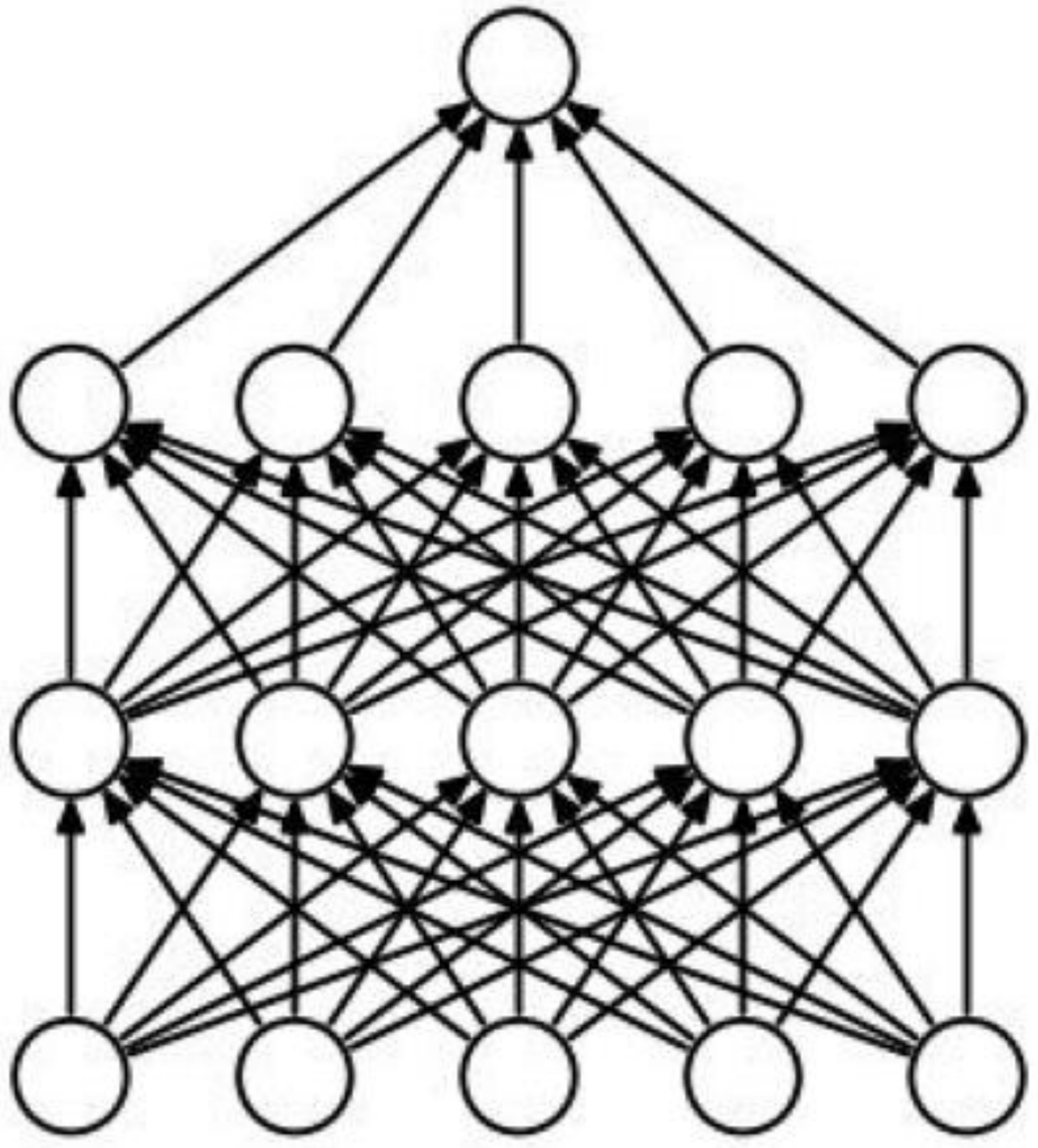
[Srivastava et al. 2014]



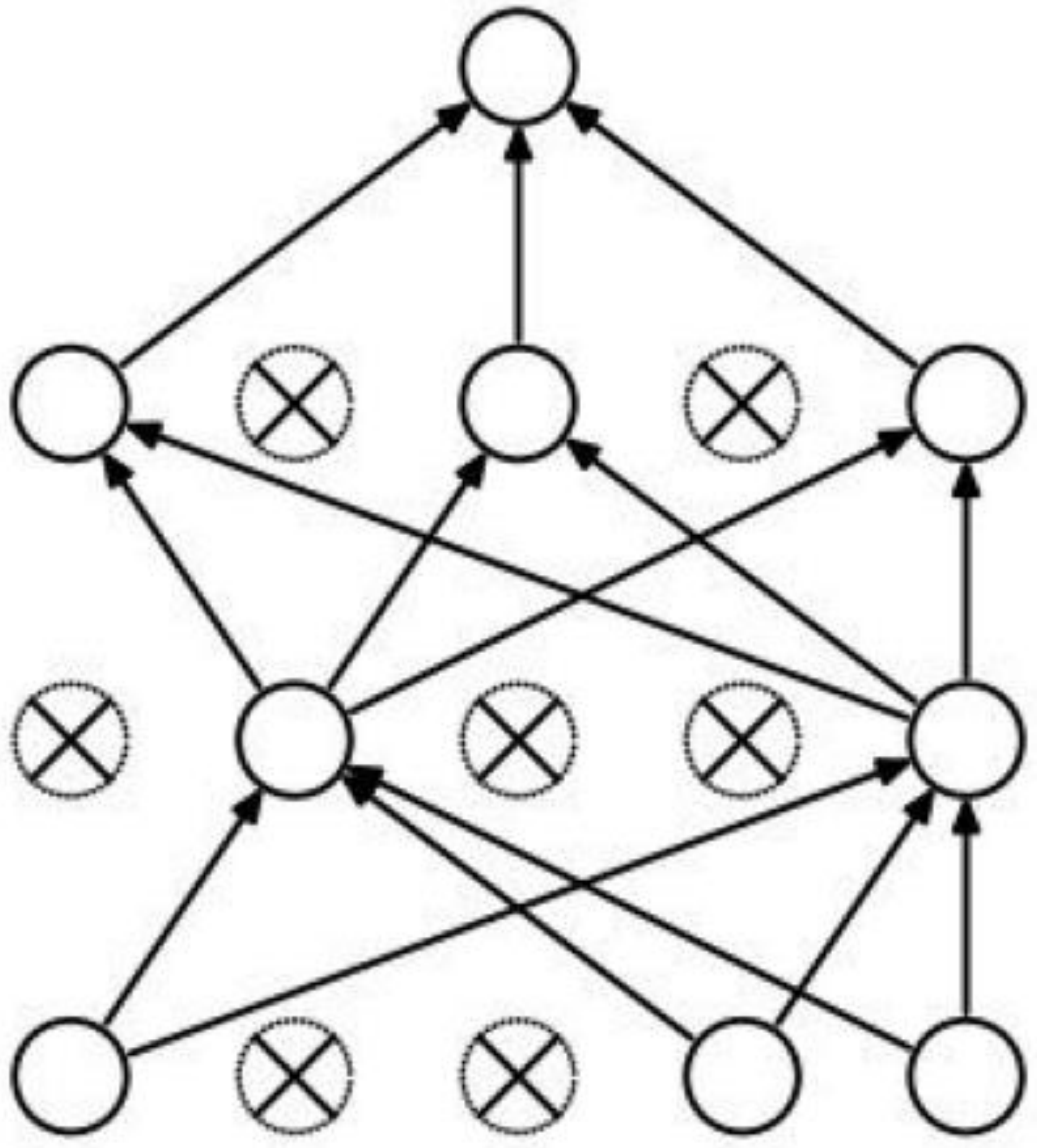
(a) Standard Neural Net (b) After applying dropout.

Regularization : Dropout

"randomly set some neurons to zero in the forward pass"



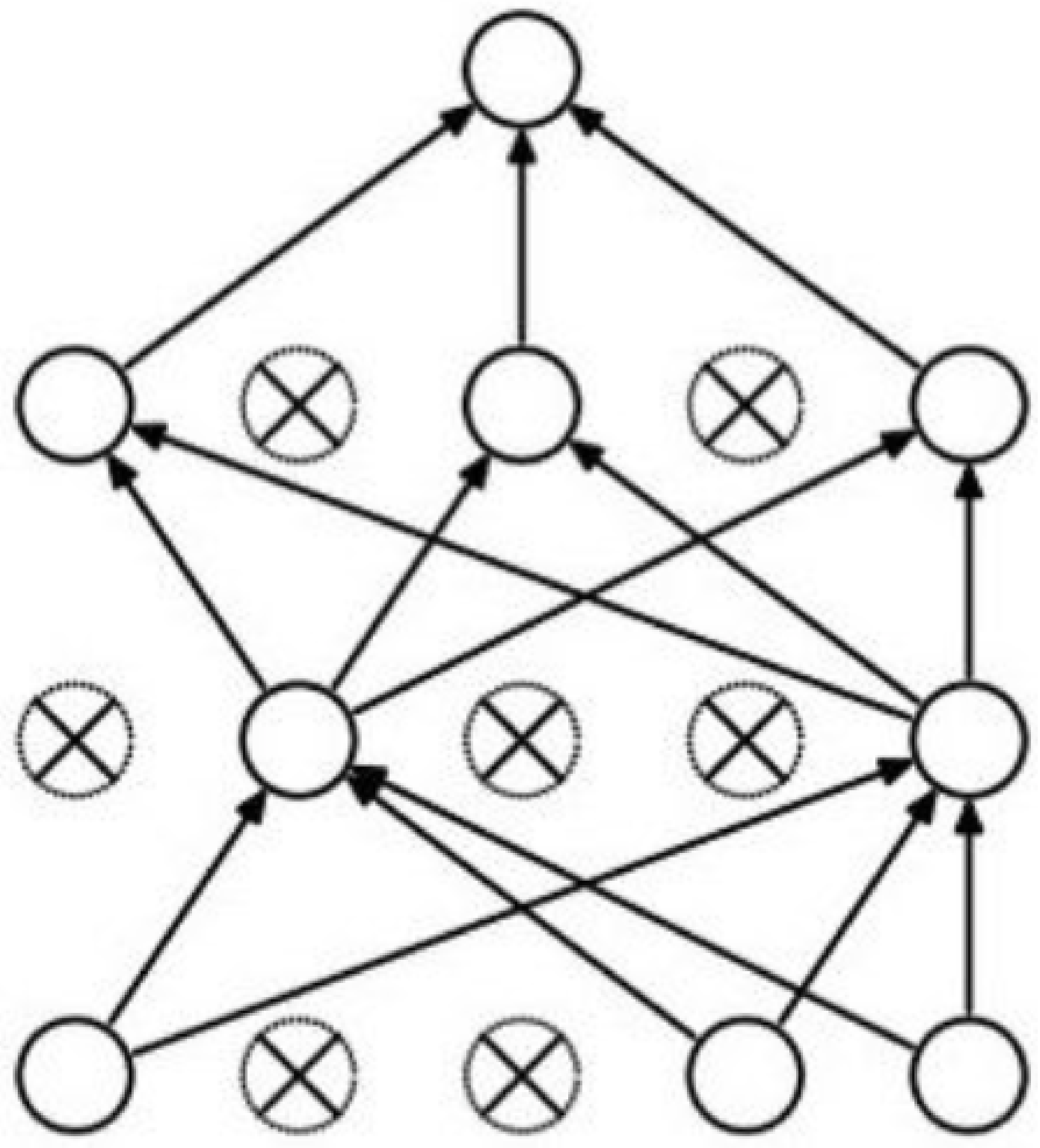
(a) Standard Neural Net



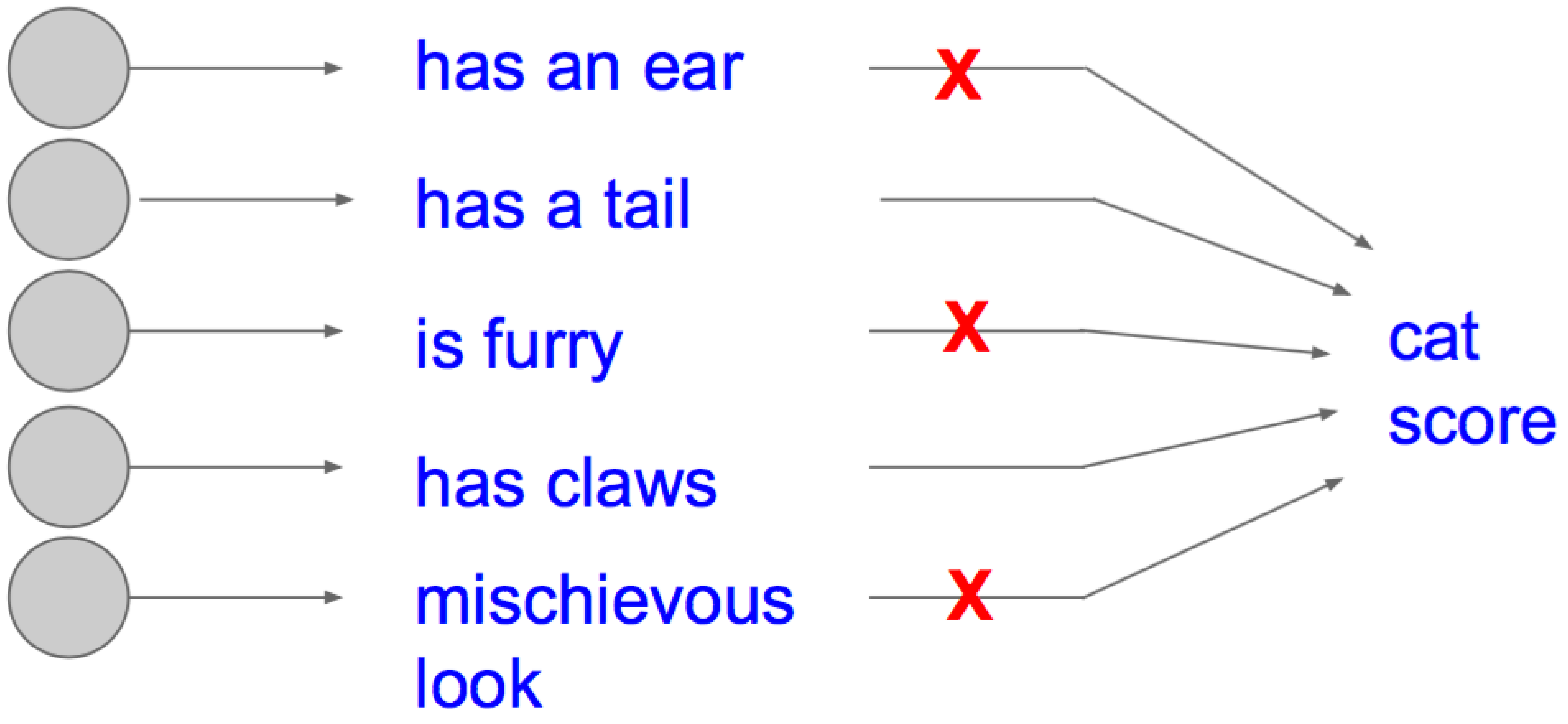
(b) After applying dropout.

[Srivastava et al., 2014]

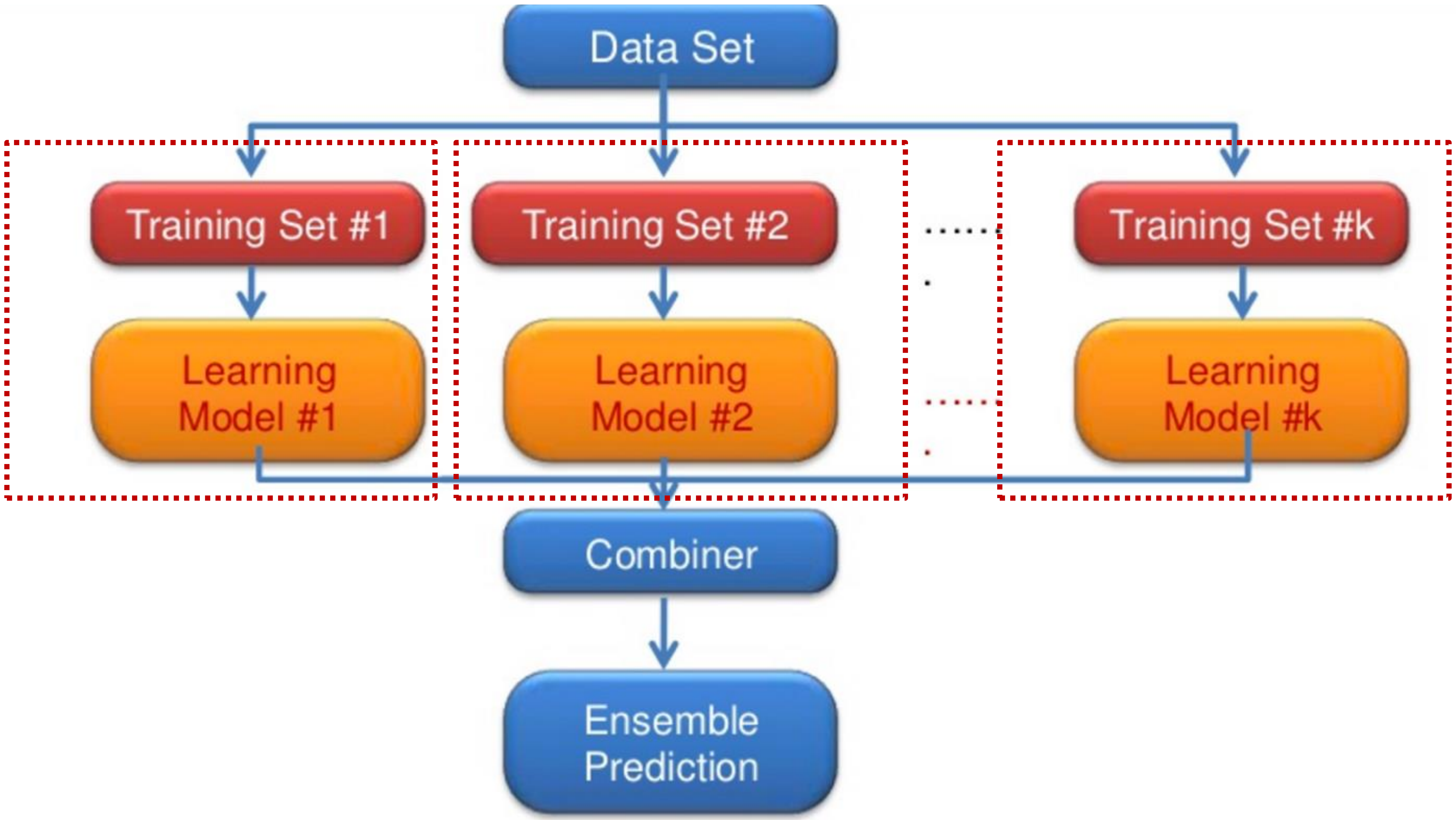
Wait a moment… How could this possibly be a good idea?



Forces the network to have a redundant representation.

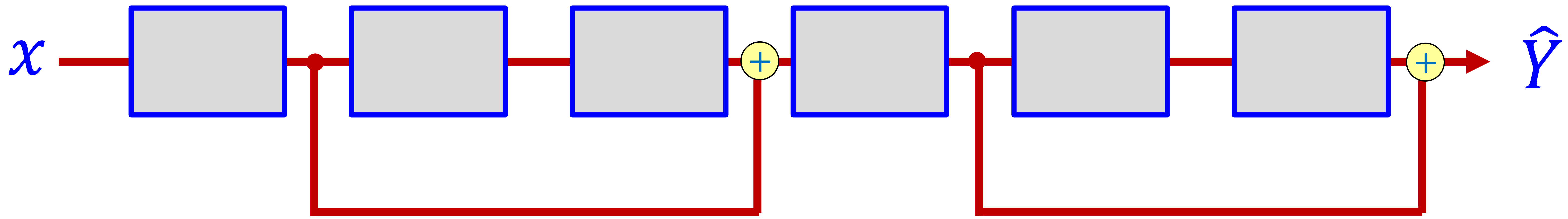


What is Ensemble?

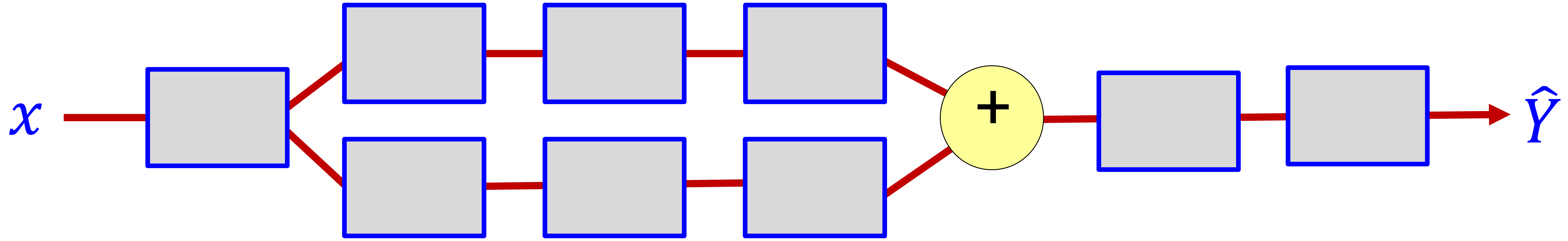


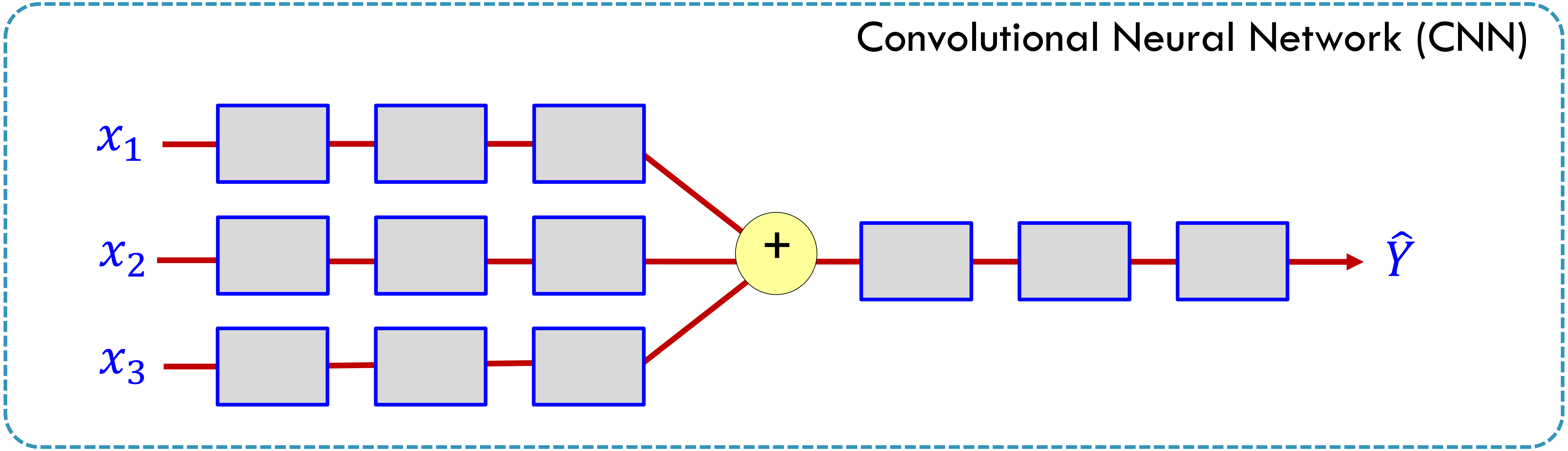
Various NN

Fast forward



Split & merge





Recurrent network (RNN)

Recurrent Neural Network (RNN)