Lecture 5.
Training Neural Networks I

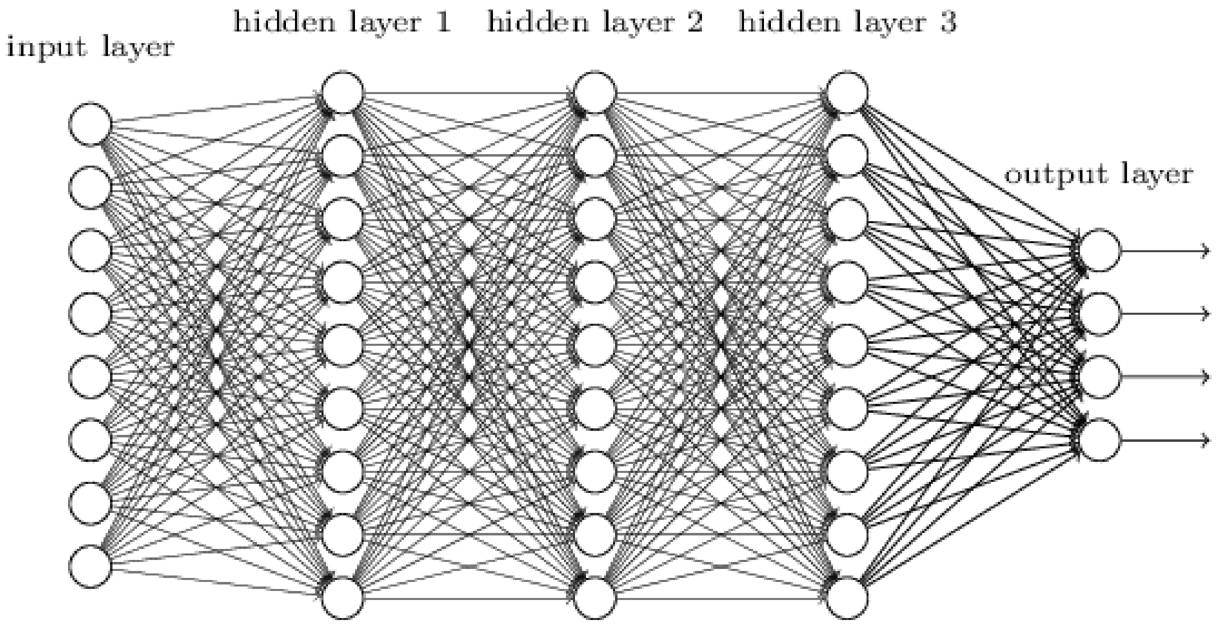
Review

1. Back to Image Classification

2. Convolutional Neural Network

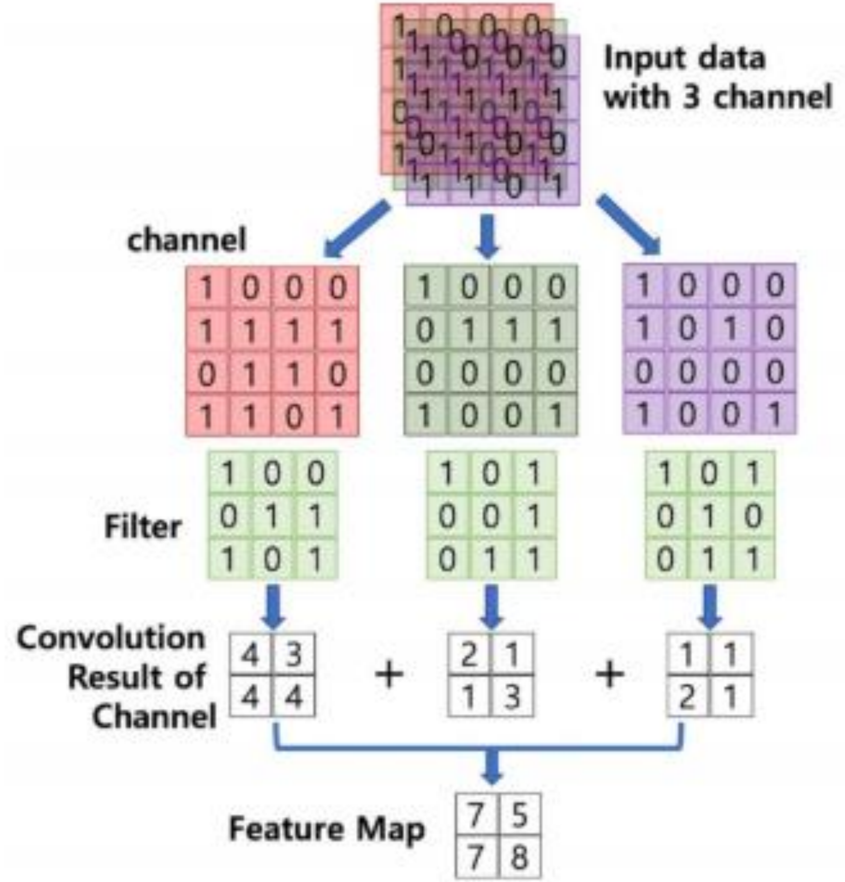
Review

MLP

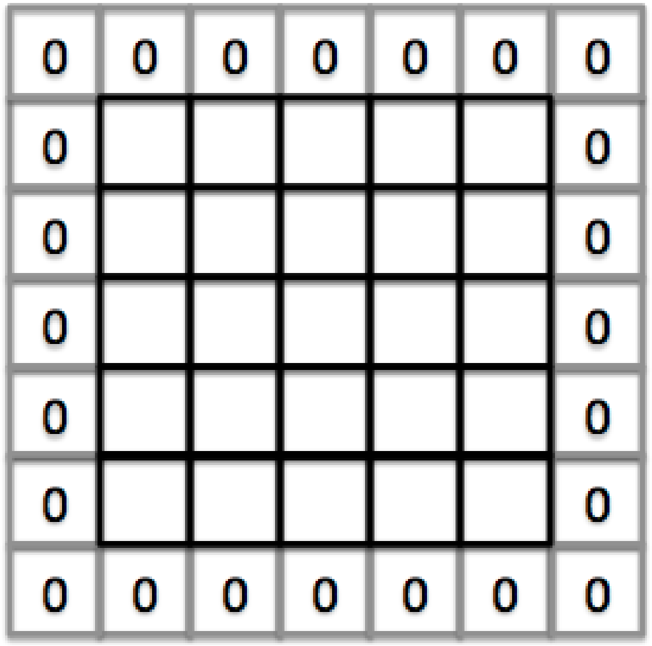

input layer
hidden layer 1
hidden layer 2
hidden layer 3
output layer

Review

$$O = \frac{I + 2Pa - F}{S} + 1$$

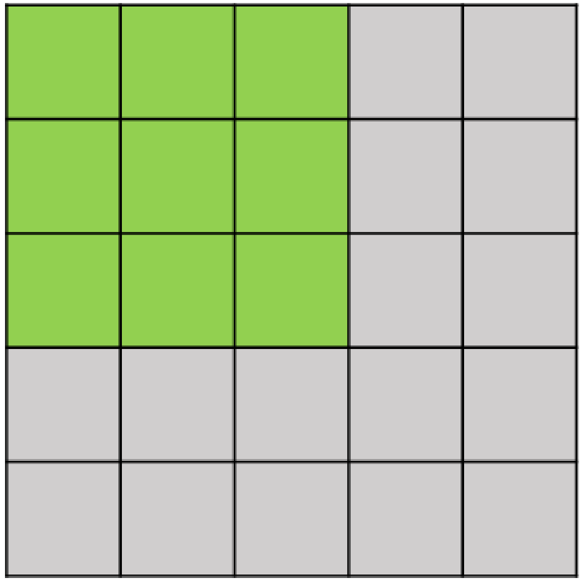


$O = Output\ size,\quad I = Input\ size,\quad Pa = Padding,$
$F = Filter\ size,\quad S = Stride$

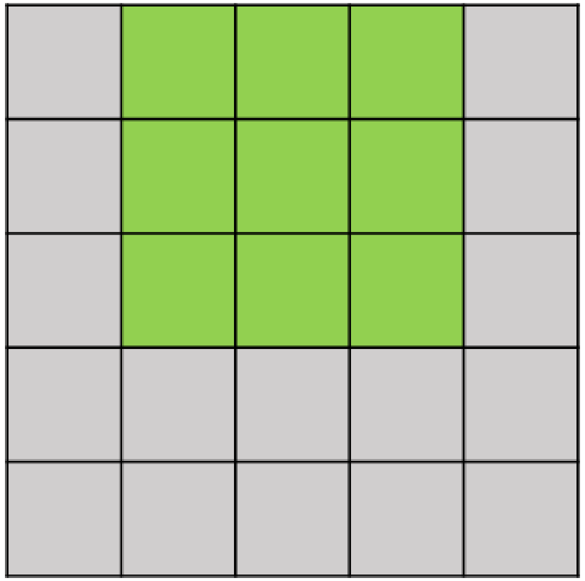


Review

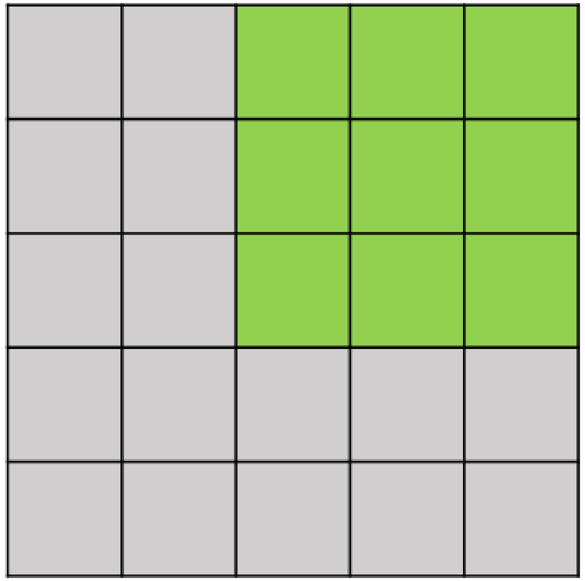
How much does a filter move per slide? **Stride**



Stride 1



Stride 2



Review

Activation Map

12	20	30	0
8	12	2	0
34	70	37	7
112	100	22	12

Max Pooling

20	30
112	37

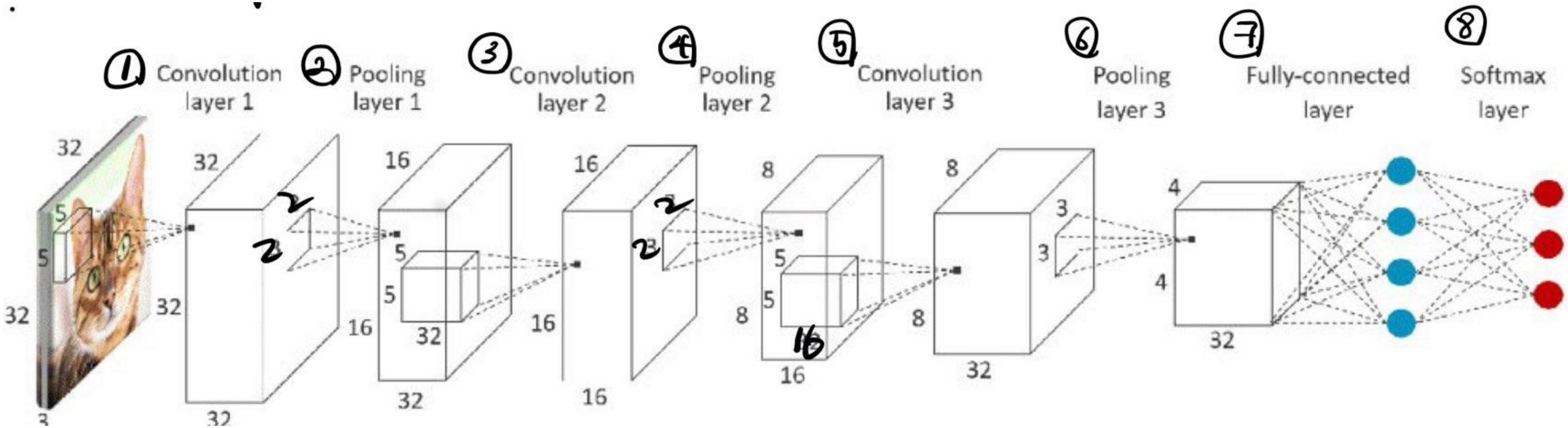
Average Pooling

13	8
79	18

$$O = \frac{I}{Po}$$

$O = Output\ size,\quad I = Input\ size,$
$Po = Pooling$

Review



① Convolution layer 1
② Pooling layer 1
③ Convolution layer 2
④ Pooling layer 2
⑤ Convolution layer 3
⑥ Pooling layer 3
⑦ Fully-connected layer
⑧ Softmax layer
32
32
3
5
5
32
32
32
2
2
16
16
5
5
32
32
16
16
16
2
2
8
8
5
5
16
16
8
8
32
3
3
4
4
32

Today's Content

1. Basic Things for Training
    - GPU (+TPU)
    - Activation functions
    - Weight Initialization
    - Data Normalization
    - Batch Normalization
    - Hyperparameter Search

2. Change Optimization Process
    - Gradient based method
    - Regularization

GPU란?

Graphic Process Unit의 약자

GPU란?

GPU는 병렬 계산에 특화 되어있음

CPU

GPU

GPU란?

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 64)        832
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, 16, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 16384)             0
_________________________________________________________________
dense (Dense)                (None, 10)                163850
=================================================================
Total params: 164,682
Trainable params: 164,682
Non-trainable params: 0
_________________________________________________________________
```

[9]

```
model.compile(optimizer=tf.keras.optimizers.Adam(0.01),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

hist = model.fit(train_data, train_labels, epochs=1, batch_size=32, validation_data=(val_data, val_labels))
```

```
Train on 40000 samples, validate on 10000 samples
40000/40000 [==============================] - 47s 1ms/sample - loss: 1.6668 - acc: 0.4205 - val_loss: 1.6619 - val_acc: 0.4252
```

[11]

```
model.compile(optimizer=tf.keras.optimizers.Adam(0.01),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

hist = model.fit(train_data, train_labels, epochs=1, batch_size=32, validation_data=(val_data, val_labels))
```

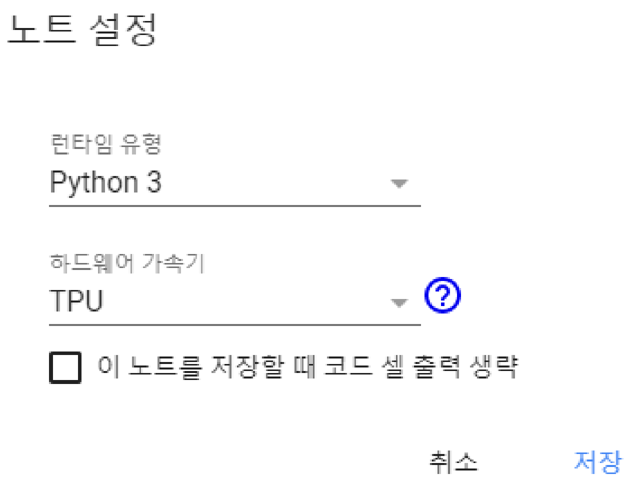
```
Train on 40000 samples, validate on 10000 samples
40000/40000 [==============================] - 10s 240us/sample - loss: 1.9463 - acc: 0.3175 - val_loss: 1.8910 - val_acc: 0.3235
```

* TPU란?

구글에서 개발한 Tensor Process Unit

뉴럴 네트워크 연산에 대해 GPU보다 15~30배 빠르다..!

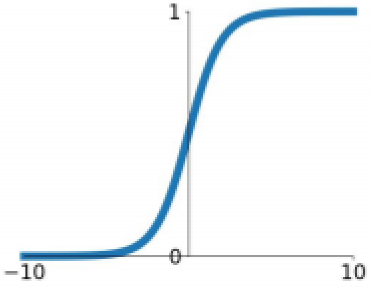
클라우드로 TPU를 제공한다.



Activation Function

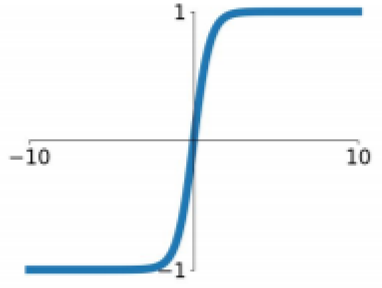
Sigmoid

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



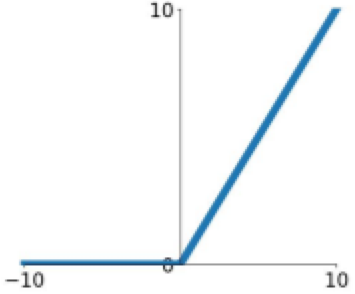
tanh

$\tanh(x)$



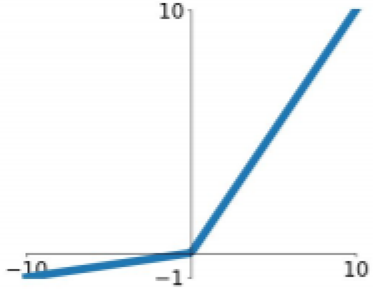
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

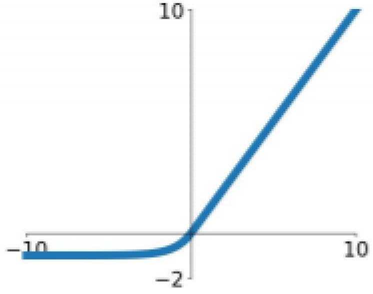


Maxout

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

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Activation Function

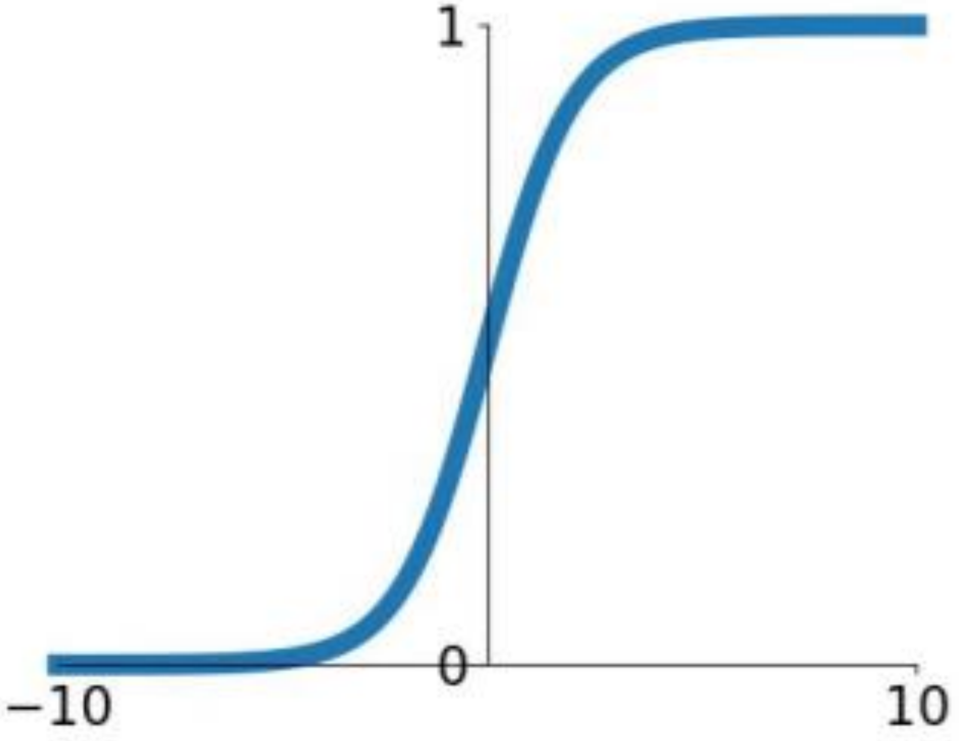
Sigmoid

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

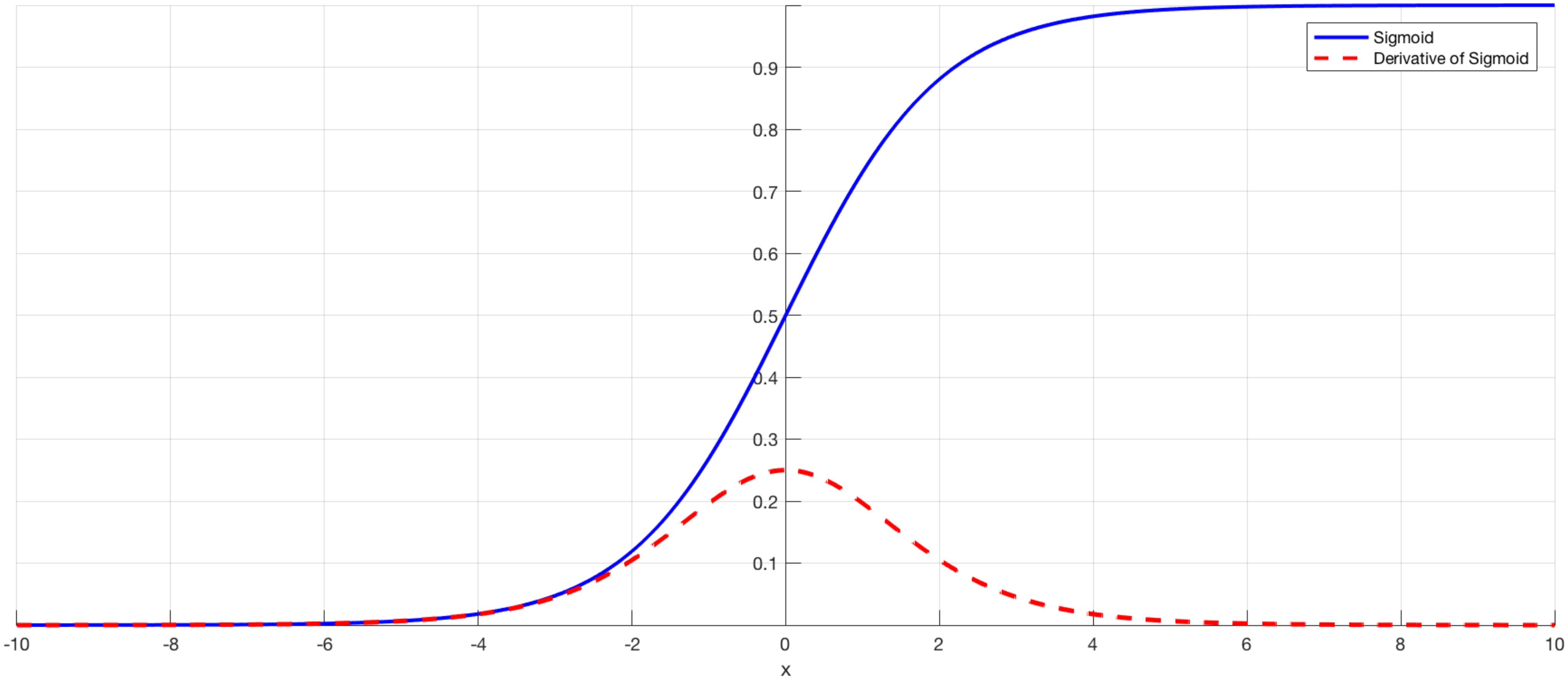
초기 머신러닝에 적용한 활성화 함수

문제점 3가지

1. Kill gradient
2. Not zero-centered
3. exp() is compute expensive



Kill gradient



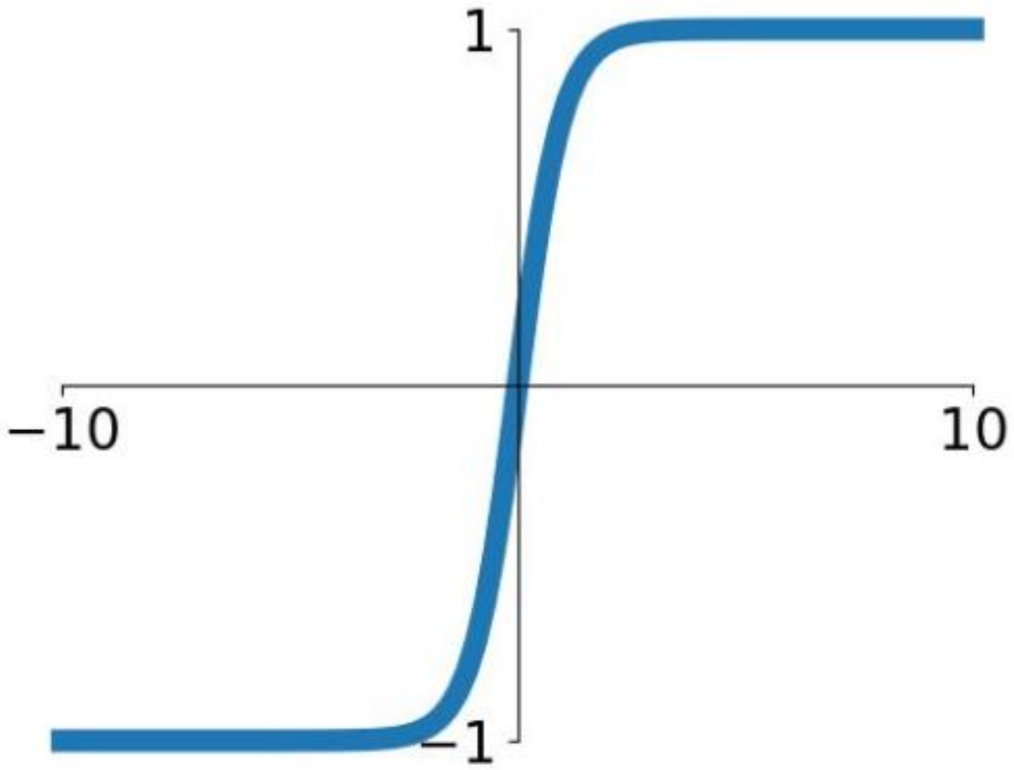
Zero-mean problem

Activation Function

tanh

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

- -1부터 1사이
- Zero-centered!
- Still kill gradient..



Activation Function

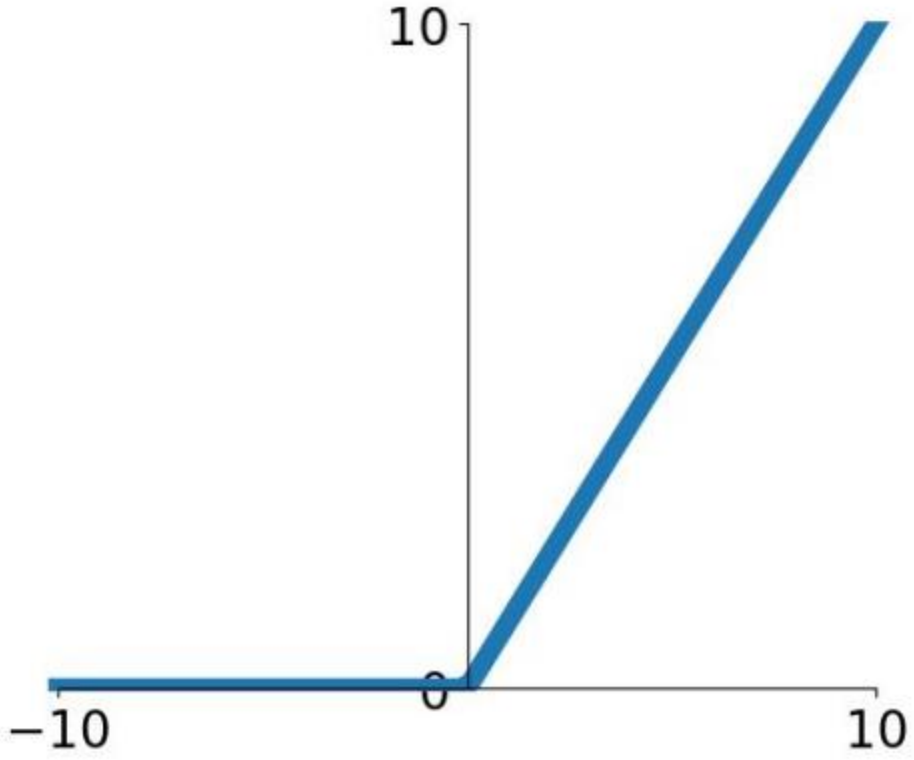
ReLU

$$f(x) = max(0, x)$$

가장 널리 사용되는 활성화함수

장점 3가지

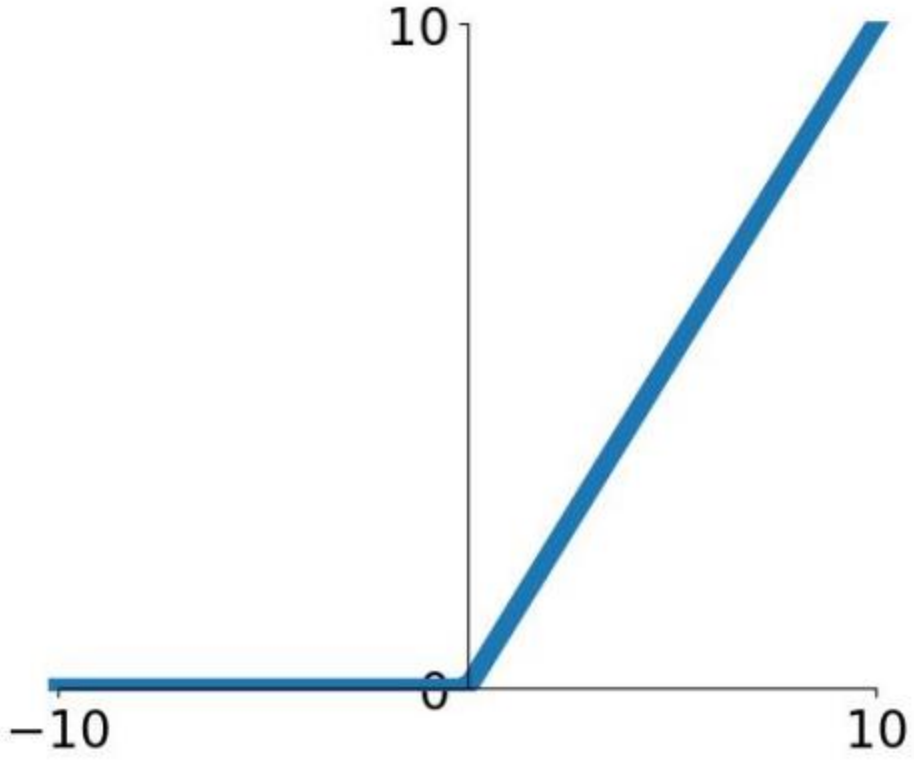
1. Not kill gradient (in +region)
2. 계산이 편하고 빠름
3. 훨씬 빠르게 수렴 (6배)



Activation Function

ReLU

$$f(x) = max(0, x)$$



단점 2가지

1. Not zero-centered
2. 'Dead' ReLU problem

'Dead' ReLU problem

Activation Function

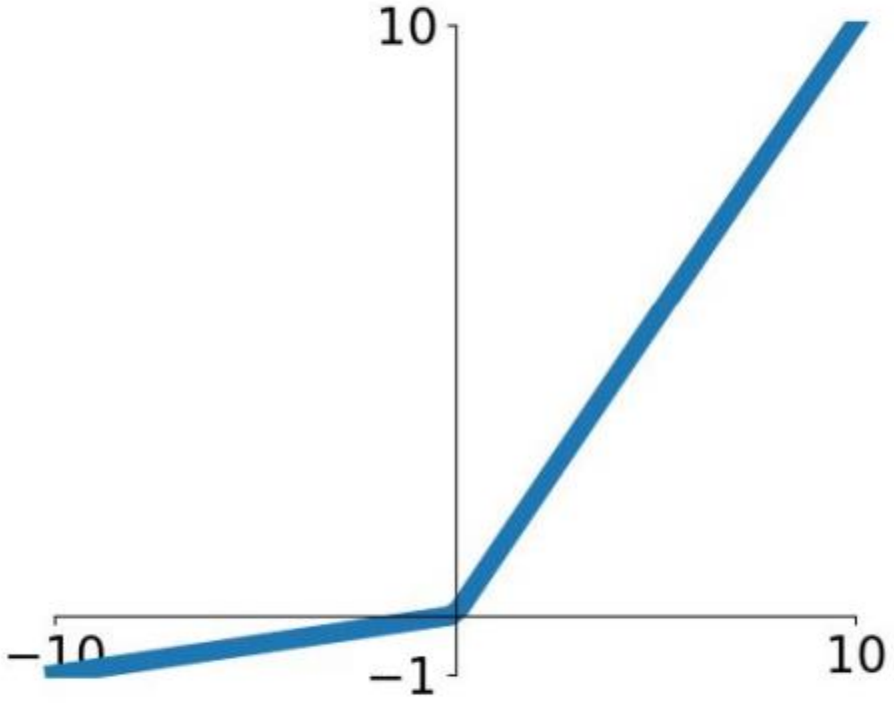
Leaky ReLU

$$f(x) = max(0.01x, x)$$

- ReLU의 변형
- Not die!

PReLU

$$f(x) = max(\alpha x, x)$$

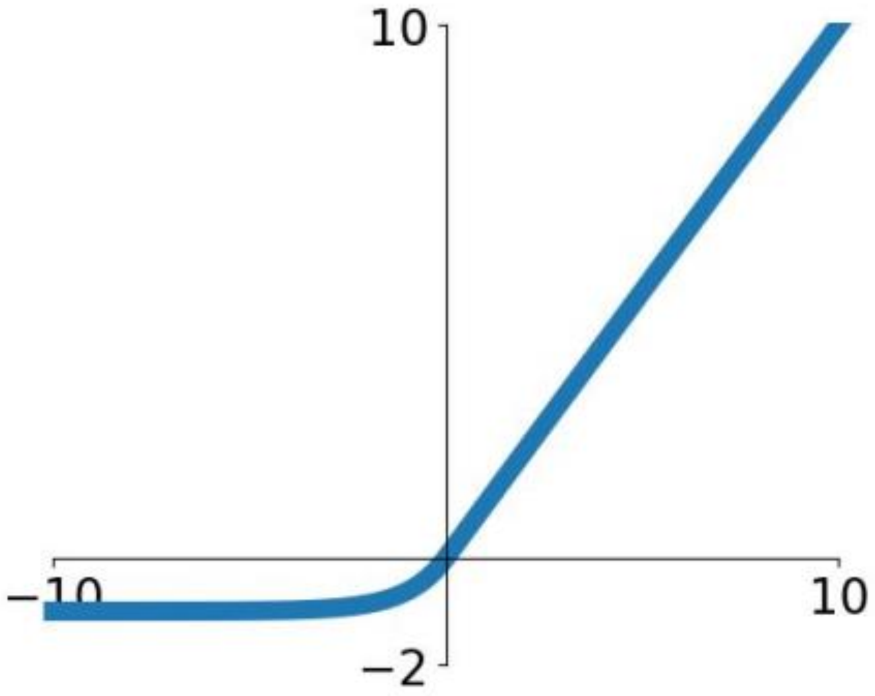


Activation Function

ELU

$$f(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha(e^x - 1), & \text{if } x \leq 0 \end{cases}$$

- Robustness to noise
- Compute expensive
- ReLU 와 Leaky ReLU 의 중간



Activation Function

Maxout

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

- "Neuron"
- Nonlinearity by max
- Generalize ReLUs
- Double # of parameters : **Big Problem**

Summary of activation function

그래서 뭐 쓸까요?

1. 일단 ReLU
2. Leaky ReLU / ELU / PReLU / Maxout 은 실험정신으로 사용
3. tanh도 시도해볼만 하지만 성능향상을 기대하긴 어려움
4. Sigmoid는 pass

Weight Initialization

Weight initialzation은 어떻게 할까?

Idea 1. weight = 0

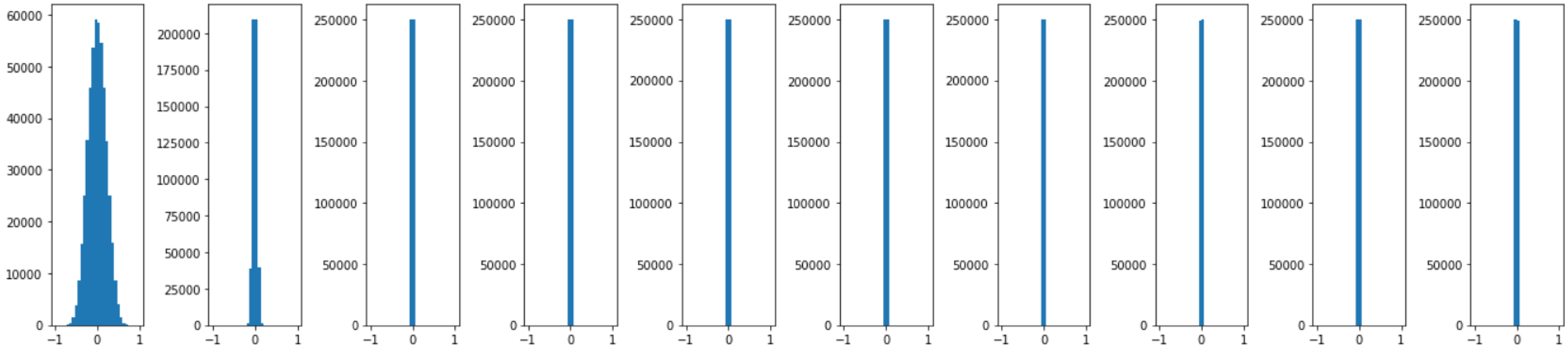
$$np.zeros(size = (D, H))$$

Idea 2. weight = small random distribution

$$0.01 * np.random.randn(size = (D, H))$$

```
W = np. random.randn(fan_in, fan_out) * 0.01
```

Idea 2. weight = small random distribution

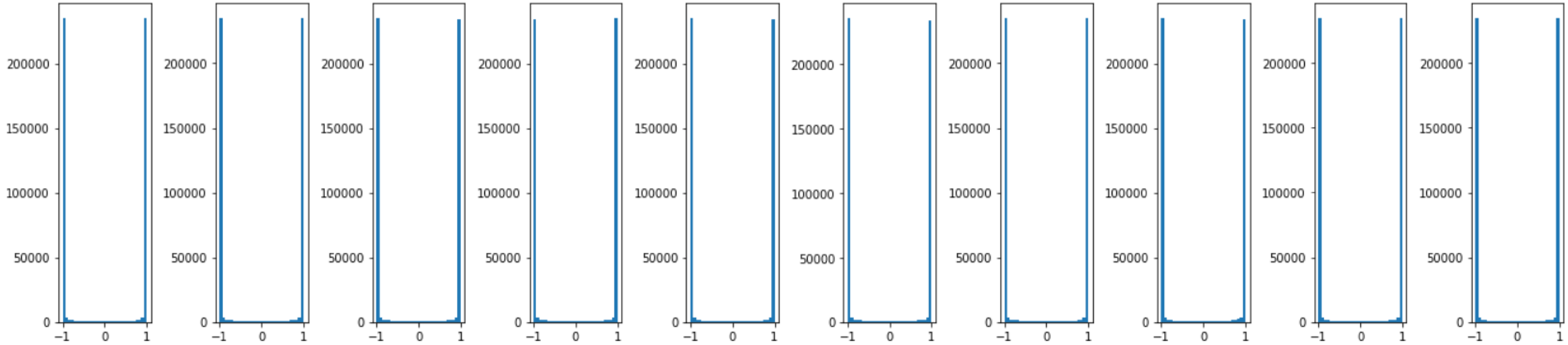


Idea 3. weight = big random distribution

$$1 * np.random.randn(size=(D,H))$$

```
W = np. random.randn(fan_in, fan_out) * 1.0
```

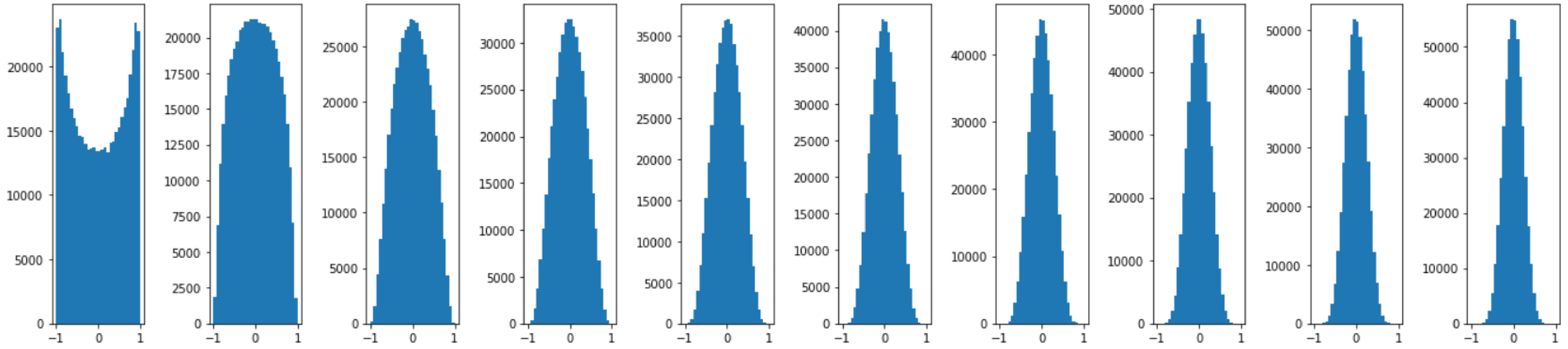
Idea 3. weight = big random distribution



Xavier initialize

$$np.random.randn(size = (D, H))/np.sqrt(D)$$

```
W = np. random.randn(fan_in, fan_out) / np.sqrt(fan_in)
```



Preview on Next Class(es)

1. Basic Things for Training
    - ~~GPU (+TPU)~~
    - ~~Activation functions~~
    - ~~Weight Initialization~~
    - Data Normalization
    - Batch Normalization
    - Hyperparameter Search

2. Change Optimization Process
    - Gradient based method
    - Regularization