# Lecture 7.
# Training Neural Networks II

# Review

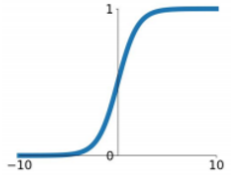
## GPU란?

Graphic Process Unit의 약자


## Activation Function

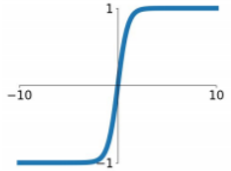
**Sigmoid**

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



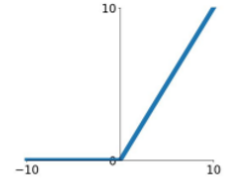

**tanh**

$\tanh(x)$



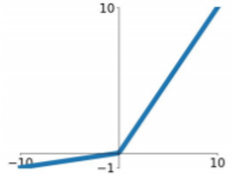

**ReLU**

$\max(0, x)$




**Leaky ReLU**

$\max(0.1x, x)$

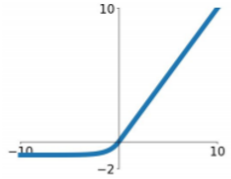



**Maxout**

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

**ELU**

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

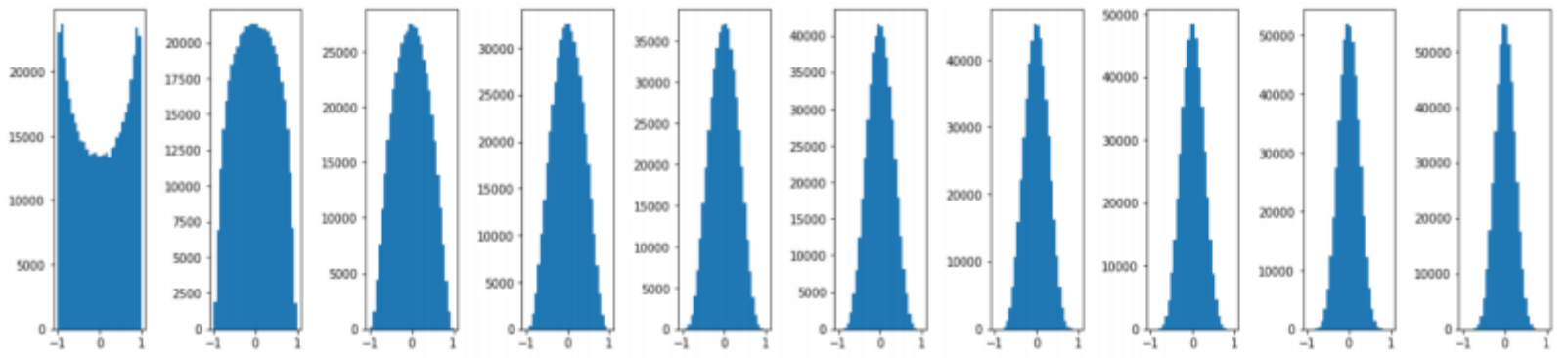


# Review

Xavier initialize

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

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



# Today's Contents

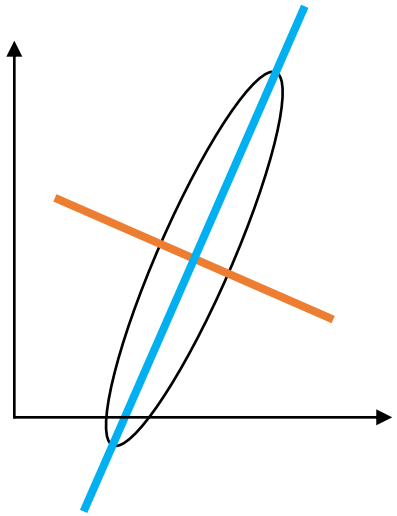
1. Basics for Training
    - Data Normalization
    - Batch Normalization
    - Hyperparameter Search

2. Change Optimization Process
    - Gradient based method
    - Regularization

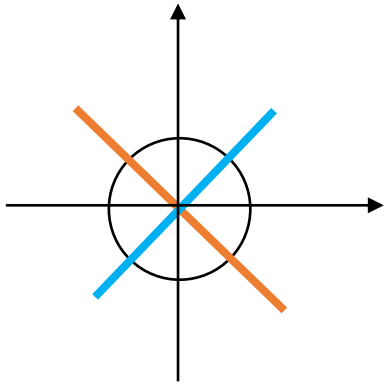
# Data Normalization

Why?

Before normalization



After normalization



# CIFAR-10 (Before Normalization)

```
Train on 40000 samples, validate on 10000 samples
Epoch 1/10
40000/40000 [==============================] - 8s 190us/sample - loss: 2.6906 - acc: 0.3472 - val_loss: 1.5718 - val_acc: 0.4412
Epoch 2/10
40000/40000 [==============================] - 7s 183us/sample - loss: 1.4218 - acc: 0.4926 - val_loss: 1.3552 - val_acc: 0.5186
Epoch 3/10
40000/40000 [==============================] - 7s 182us/sample - loss: 1.2132 - acc: 0.5728 - val_loss: 1.3401 - val_acc: 0.5425
Epoch 4/10
40000/40000 [==============================] - 7s 182us/sample - loss: 1.0239 - acc: 0.6409 - val_loss: 1.2987 - val_acc: 0.5665
Epoch 5/10
40000/40000 [==============================] - 7s 183us/sample - loss: 0.8357 - acc: 0.7096 - val_loss: 1.4056 - val_acc: 0.5687
Epoch 6/10
40000/40000 [==============================] - 7s 184us/sample - loss: 0.6655 - acc: 0.7715 - val_loss: 1.6159 - val_acc: 0.5672
Epoch 7/10
40000/40000 [==============================] - 7s 183us/sample - loss: 0.5271 - acc: 0.8209 - val_loss: 1.7894 - val_acc: 0.5591
Epoch 8/10
40000/40000 [==============================] - 7s 182us/sample - loss: 0.4105 - acc: 0.8633 - val_loss: 2.2164 - val_acc: 0.5533
Epoch 9/10
40000/40000 [==============================] - 7s 185us/sample - loss: 0.3605 - acc: 0.8806 - val_loss: 2.4291 - val_acc: 0.5584
Epoch 10/10
40000/40000 [==============================] - 8s 188us/sample - loss: 0.3013 - acc: 0.9037 - val_loss: 2.5739 - val_acc: 0.5547
```

# CIFAR-10 (After Normalization)

```
Train on 40000 samples, validate on 10000 samples
Epoch 1/10
40000/40000 [==============================] - 9s 231us/sample - loss: 1.2820 - acc: 0.5416 - val_loss: 1.0695 - val_acc: 0.6196
Epoch 2/10
40000/40000 [==============================] - 7s 180us/sample - loss: 0.8911 - acc: 0.6865 - val_loss: 0.8893 - val_acc: 0.6866
Epoch 3/10
40000/40000 [==============================] - 7s 180us/sample - loss: 0.6864 - acc: 0.7598 - val_loss: 0.8371 - val_acc: 0.7143
Epoch 4/10
40000/40000 [==============================] - 7s 180us/sample - loss: 0.4900 - acc: 0.8286 - val_loss: 0.8445 - val_acc: 0.7215
Epoch 5/10
40000/40000 [==============================] - 7s 180us/sample - loss: 0.2988 - acc: 0.8984 - val_loss: 0.9551 - val_acc: 0.7258
Epoch 6/10
40000/40000 [==============================] - 7s 179us/sample - loss: 0.1615 - acc: 0.9468 - val_loss: 1.1369 - val_acc: 0.7181
Epoch 7/10
40000/40000 [==============================] - 7s 178us/sample - loss: 0.0940 - acc: 0.9698 - val_loss: 1.3044 - val_acc: 0.7150
Epoch 8/10
40000/40000 [==============================] - 7s 179us/sample - loss: 0.0776 - acc: 0.9746 - val_loss: 1.3674 - val_acc: 0.7242
Epoch 9/10
40000/40000 [==============================] - 7s 179us/sample - loss: 0.0634 - acc: 0.9787 - val_loss: 1.5953 - val_acc: 0.7187
Epoch 10/10
40000/40000 [==============================] - 7s 178us/sample - loss: 0.0578 - acc: 0.9804 - val_loss: 1.5720 - val_acc: 0.7145
```

# Data Normalization

Why?

```
Epoch 10/10
40000/40000 [==============================] - 8s 188us/sample - loss: 0.3013 - acc: 0.9037 - val_loss: 2.5739 - val_acc: 0.5547

Epoch 10/10
40000/40000 [==============================] - 7s 178us/sample - loss: 0.0578 - acc: 0.9804 - val_loss: 1.5720 - val_acc: 0.7145
```

Converges much faster!

# Data Normalization

$$\frac{x - x_{min}}{x_{max} - x_{min}}$$

데이터를 0에서 1사이의 값으로 변환!

```
data, test_data = data / 255.0, test_data / 255.0
```

# Data Normalization

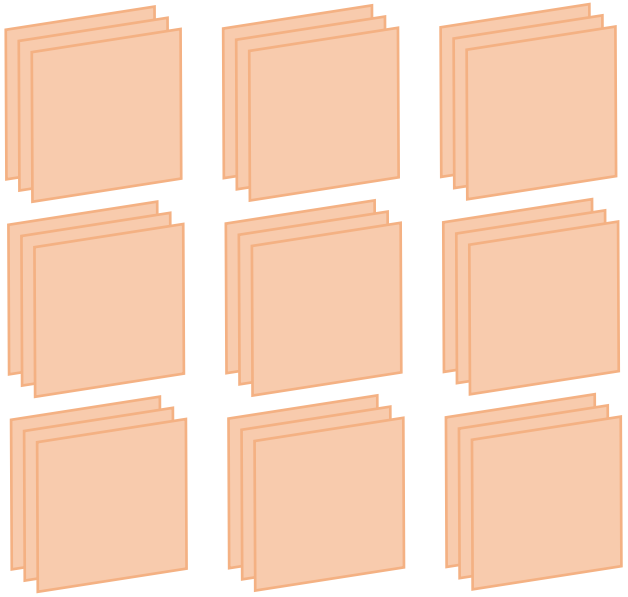
$$\frac{x - \bar{x}}{\sigma}$$

데이터를 평균이 0, 표준편차가 1인 분포로 만듦!

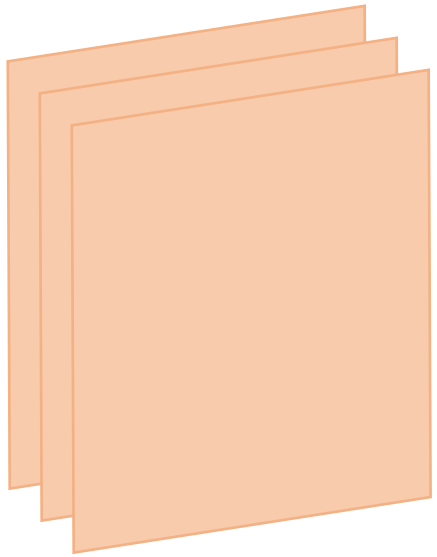
```
data -= np.mean(data)
data /= np.std(data)
```

# Data Normalization (Image)

GCN



LCN



# Batch Normalization

**Input:** Values of $x$ over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: $\gamma$, $\beta$

**Output:** $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

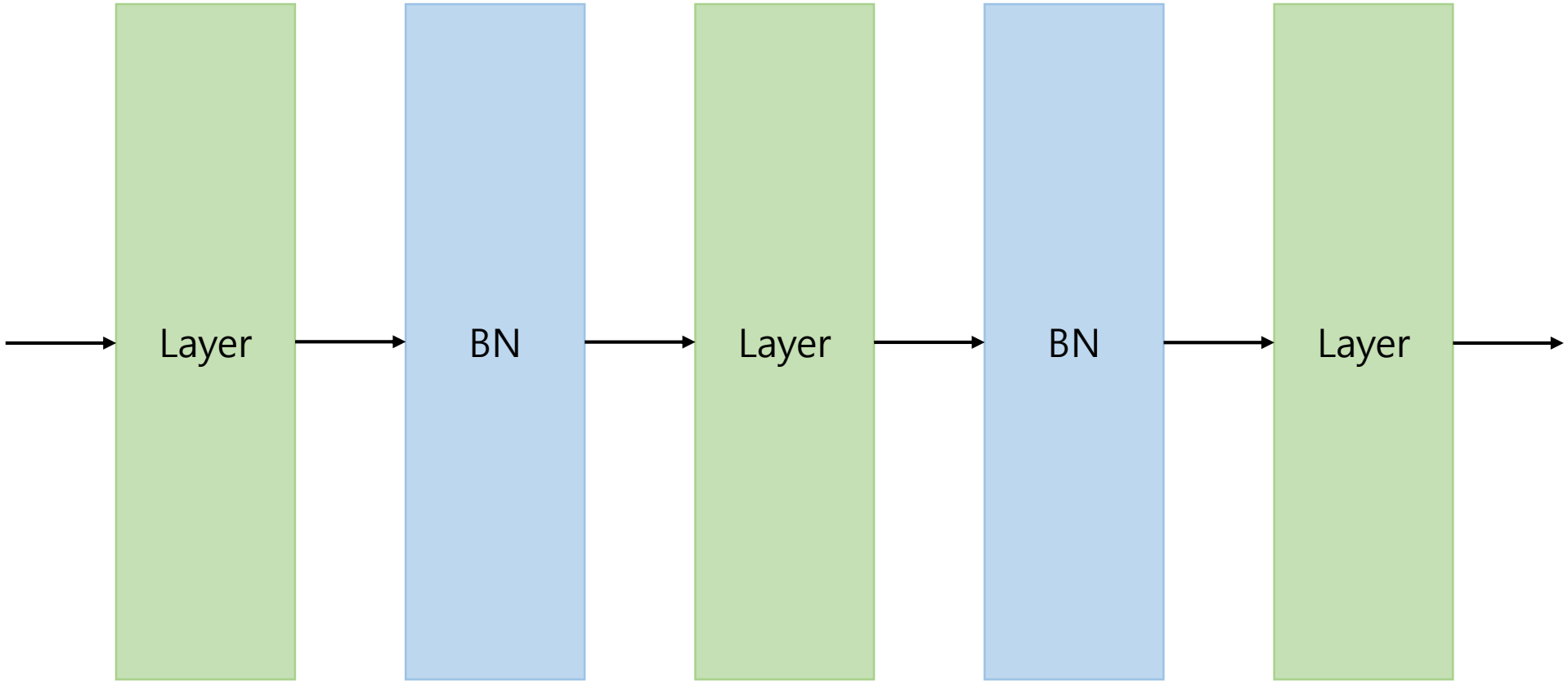
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

# Batch Normalization



# Batch Normalization – CIFAR-10

| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d (Conv2D) | (None, 32, 32, 64) | 832 |
| batch_normalization (BatchNo | (None, 32, 32, 64) | 256 |
| max_pooling2d (MaxPooling2D) | (None, 16, 16, 64) | 0 |
| conv2d_1 (Conv2D) | (None, 16, 16, 64) | 16448 |
| batch_normalization_1 (Batch | (None, 16, 16, 64) | 256 |
| max_pooling2d_1 (MaxPooling2 | (None, 8, 8, 64) | 0 |
| flatten (Flatten) | (None, 4096) | 0 |
| dense (Dense) | (None, 1024) | 4195328 |
| dense_1 (Dense) | (None, 10) | 10250 |

Total params: 4,223,370
Trainable params: 4,223,114
Non-trainable params: 256

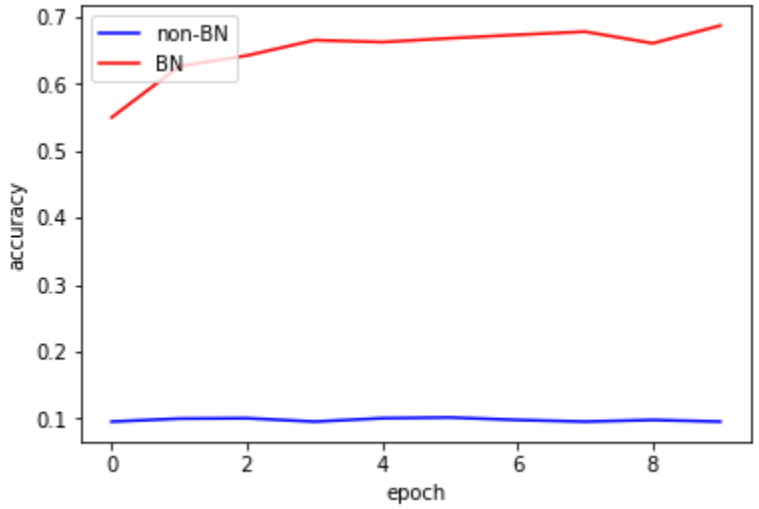
VS

| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d_6 (Conv2D) | (None, 32, 32, 64) | 832 |
| max_pooling2d_6 (MaxPooling2 | (None, 16, 16, 64) | 0 |
| conv2d_7 (Conv2D) | (None, 16, 16, 64) | 16448 |
| max_pooling2d_7 (MaxPooling2 | (None, 8, 8, 64) | 0 |
| flatten_3 (Flatten) | (None, 4096) | 0 |
| dense_6 (Dense) | (None, 1024) | 4195328 |
| dense_7 (Dense) | (None, 10) | 10250 |

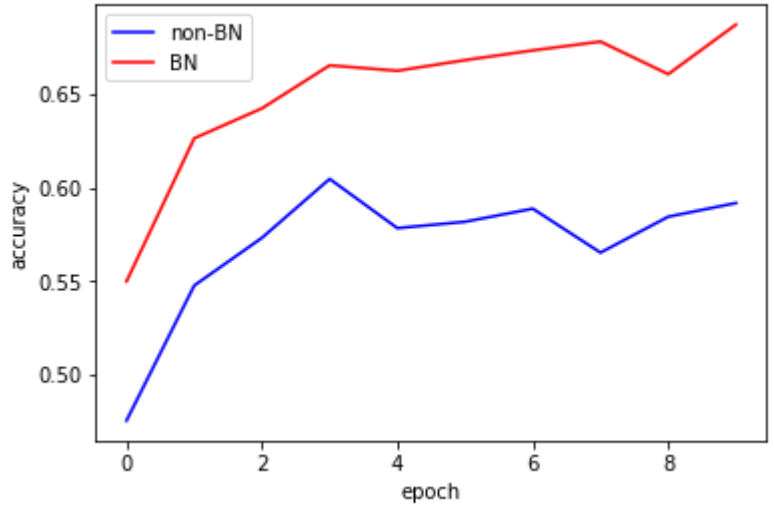
Total params: 4,222,858
Trainable params: 4,222,858
Non-trainable params: 0

# Batch Normalization

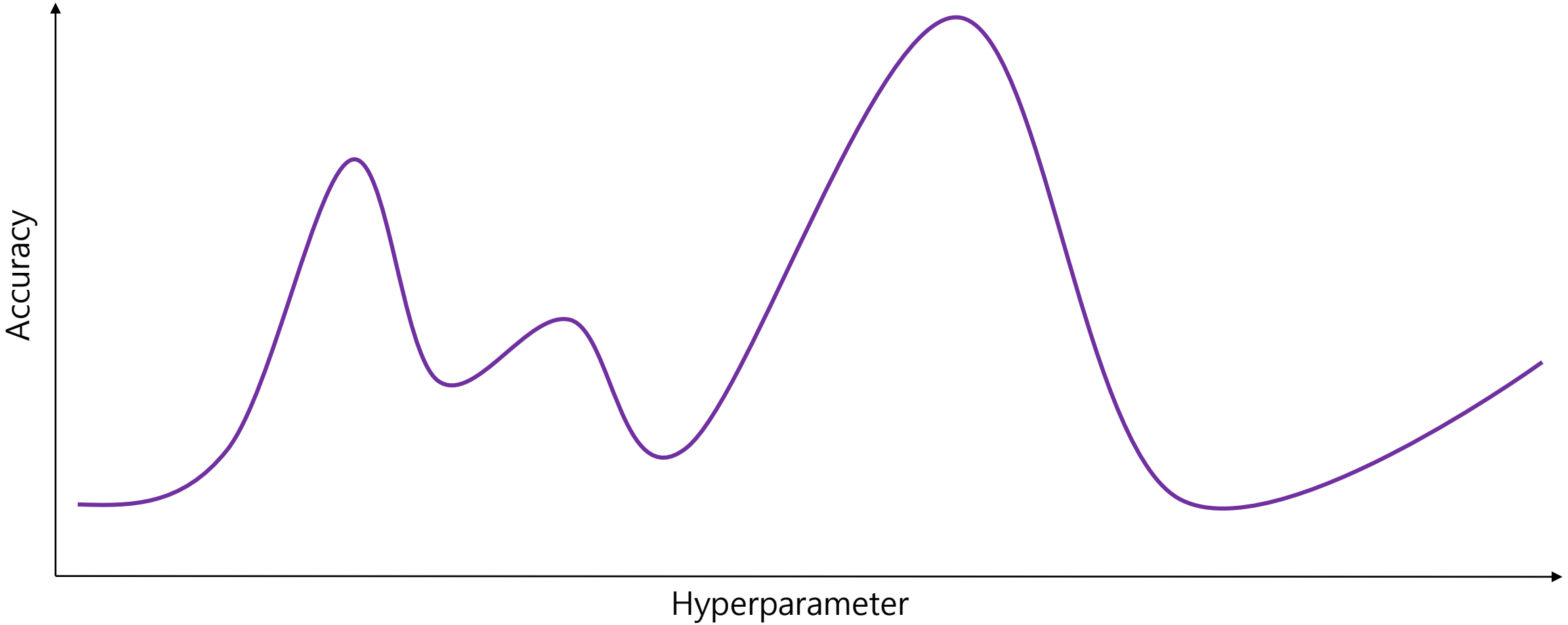
Learning rate = 0.005



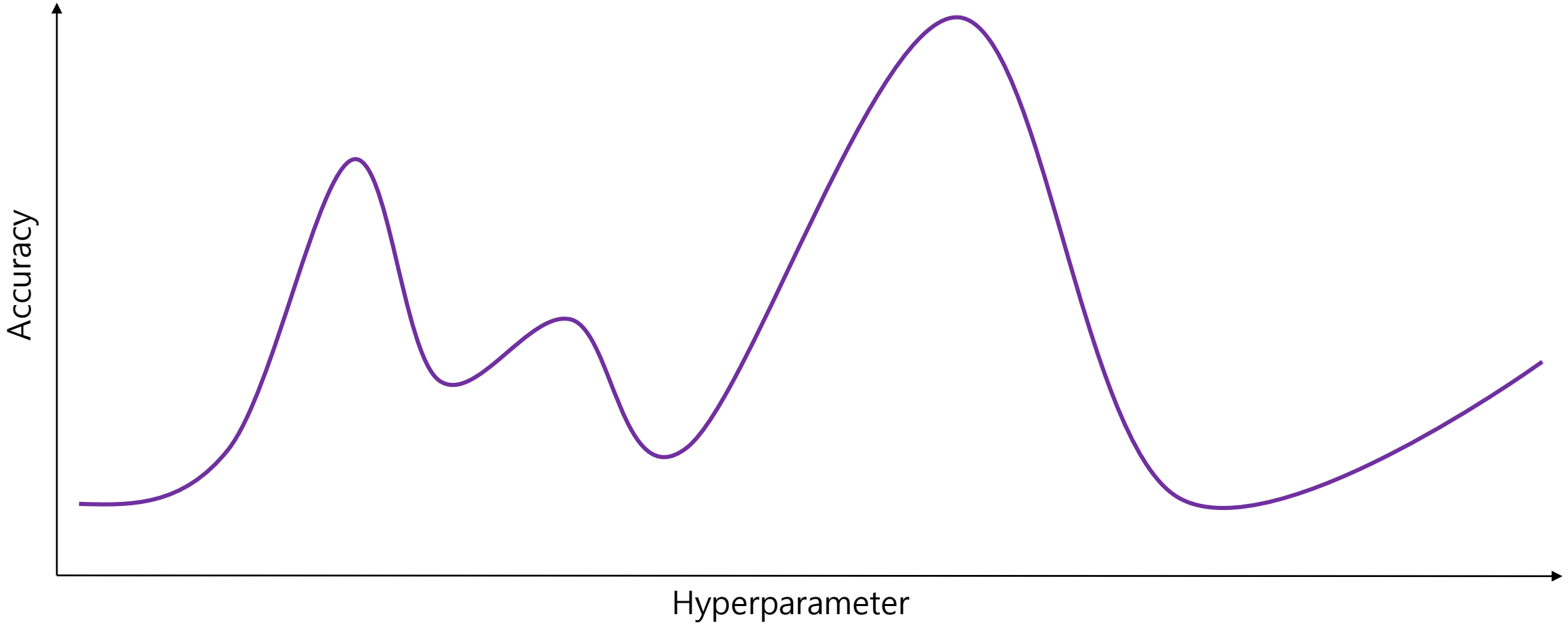

Learning rate = 0.001



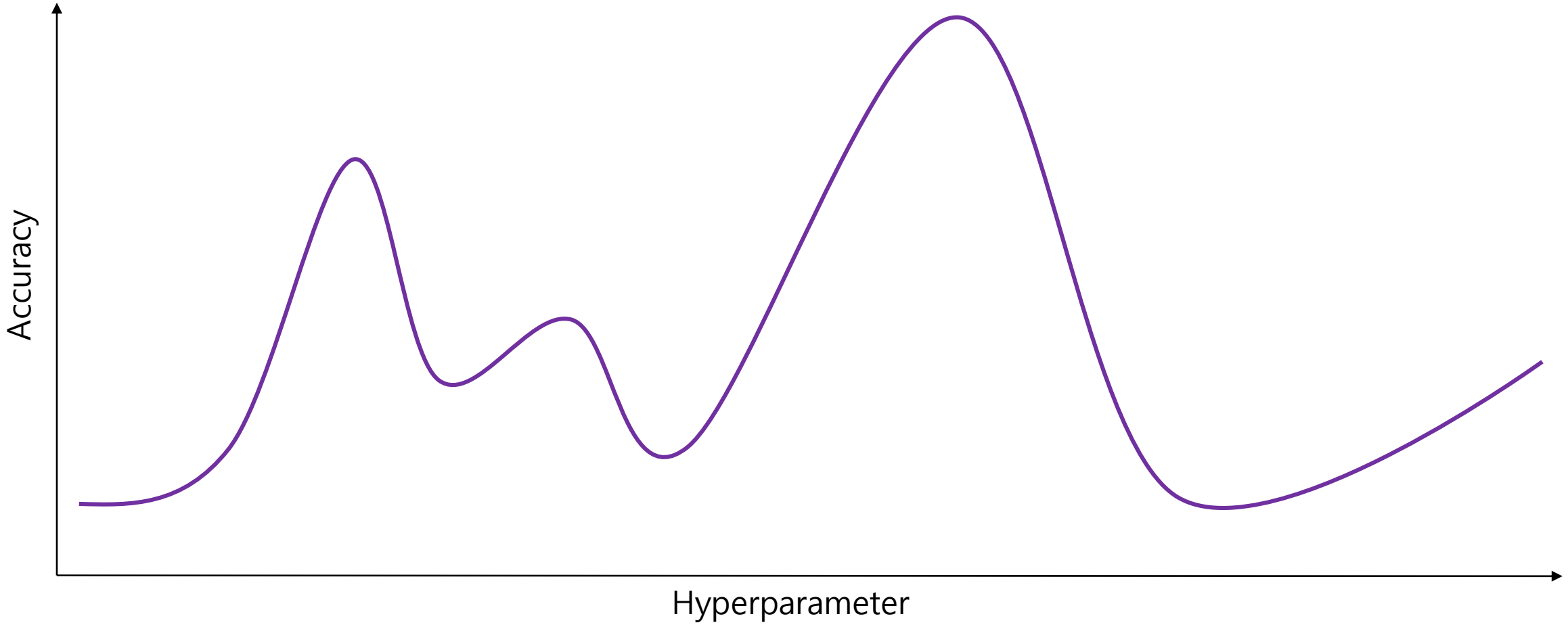
# Hyperparameter Search



# Grid Search



# Random Search



# * AutoML

요약: Machine Learning으로 Machine Learning 하기!

참고: http://research.sualab.com/introduction/practice/2019/02/19/bayesian-optimization-overview-1.html

# Image Augmentation

Crop, Scale, Flip, Translate, Rotate, add a nosie

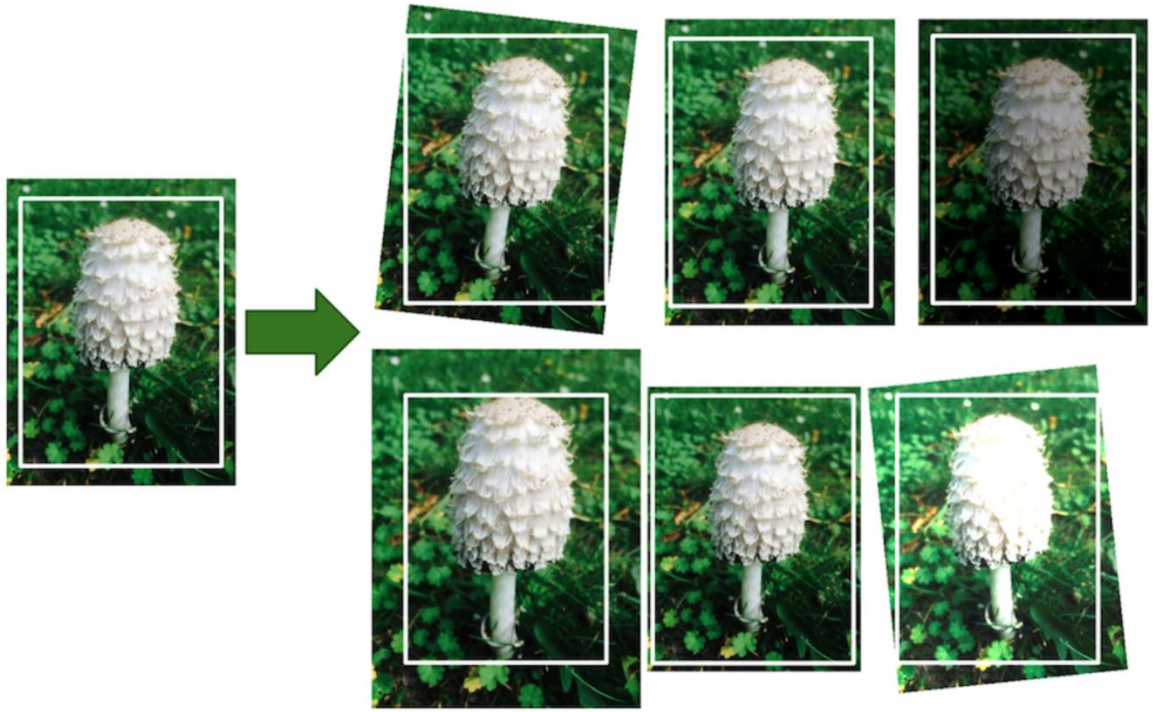
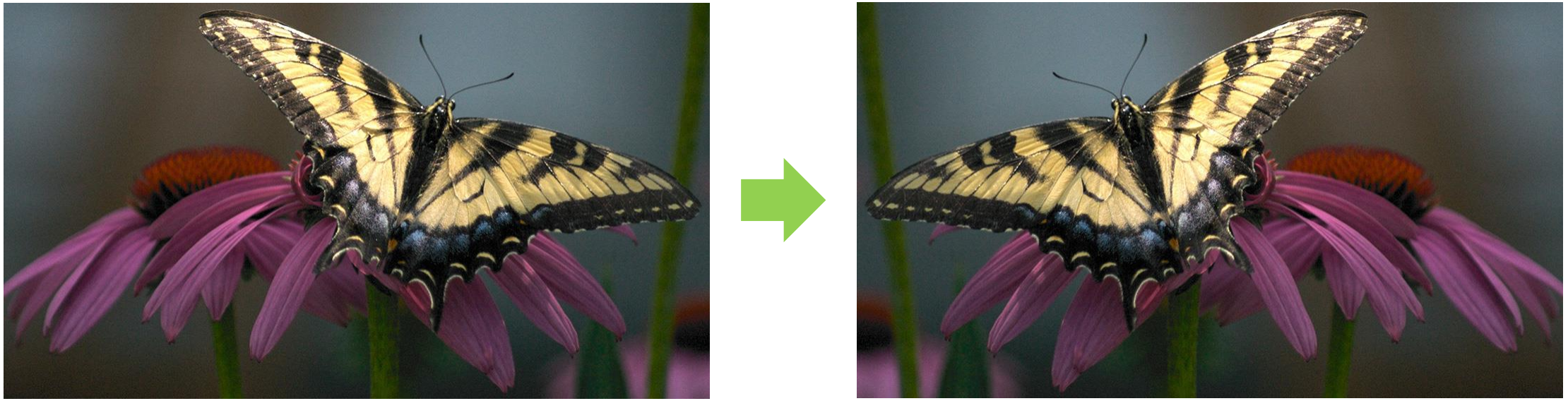


Image source: https://github.com/oryondark/-/blob/master/DeepLearning_master/Augmentation_Tutorial/Data%20Augmentation.md

# Crop

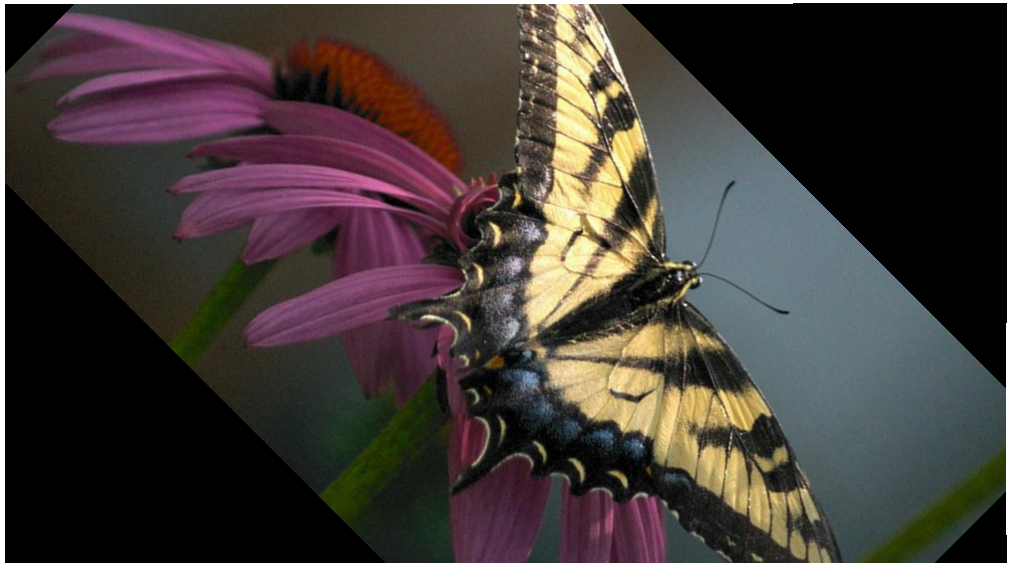
# Scale

# Flip



# Translation

# Rotate



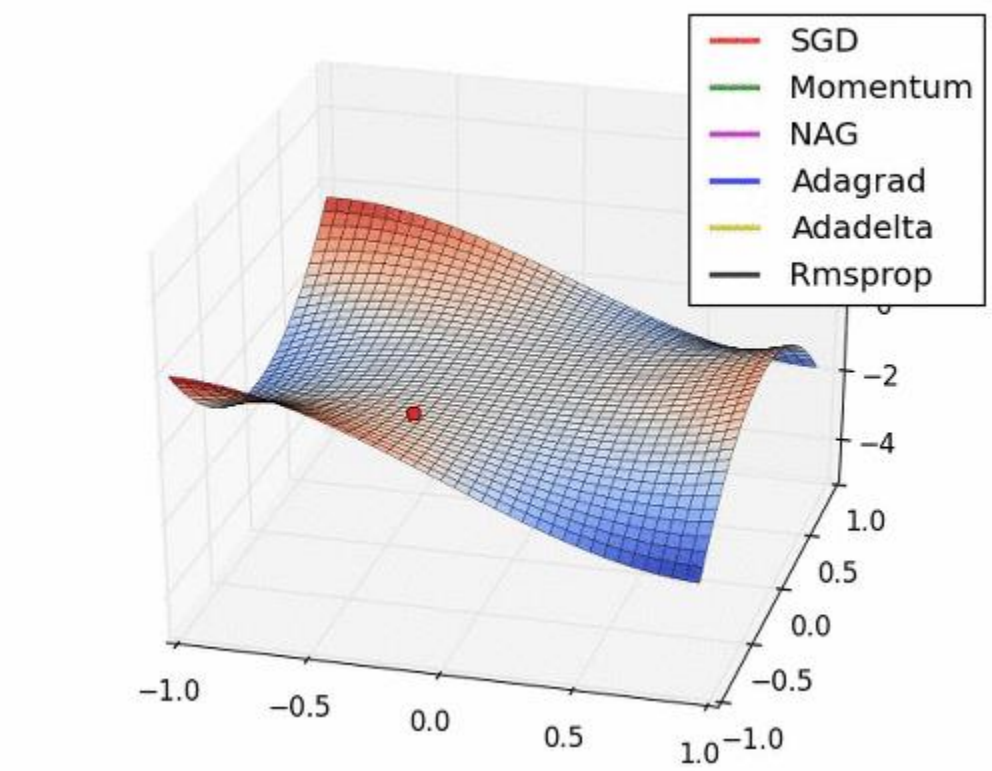
# Add a Noise

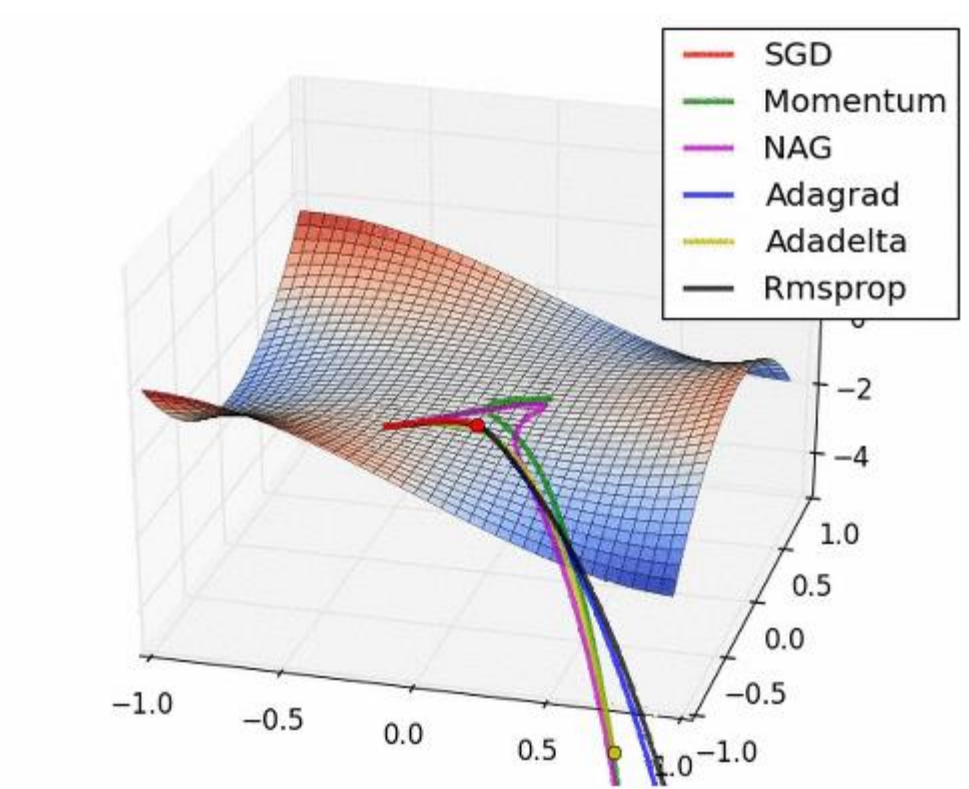
# Image Augmentation

Make 2 ~ 3 more images

1. Small dataset ➔ Large dataset
2. Prevent overfitting
3. BUT, take a right way!!

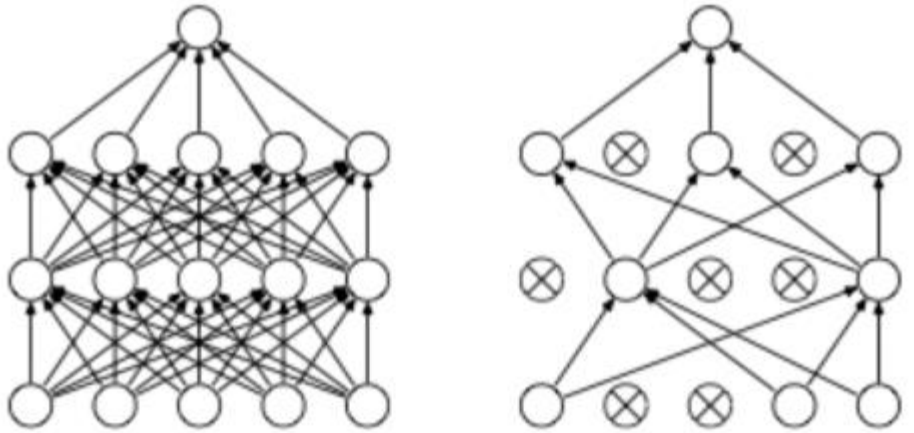
# Preview on Next Class



# Preview on Next Class

Dropout: A Simple Way to Prevent Neural Networks from Overfitting [Srivastava et al. 2014]



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