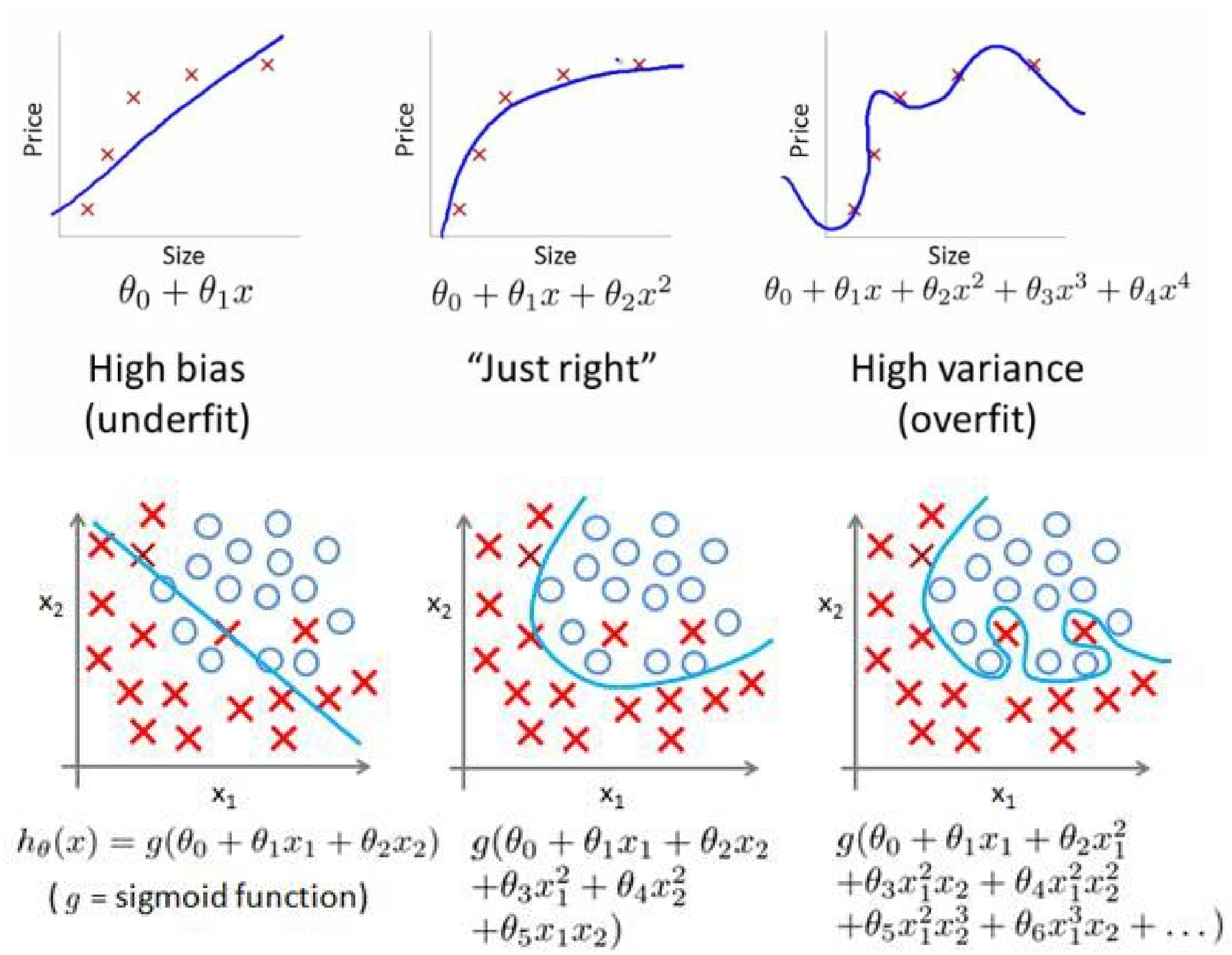
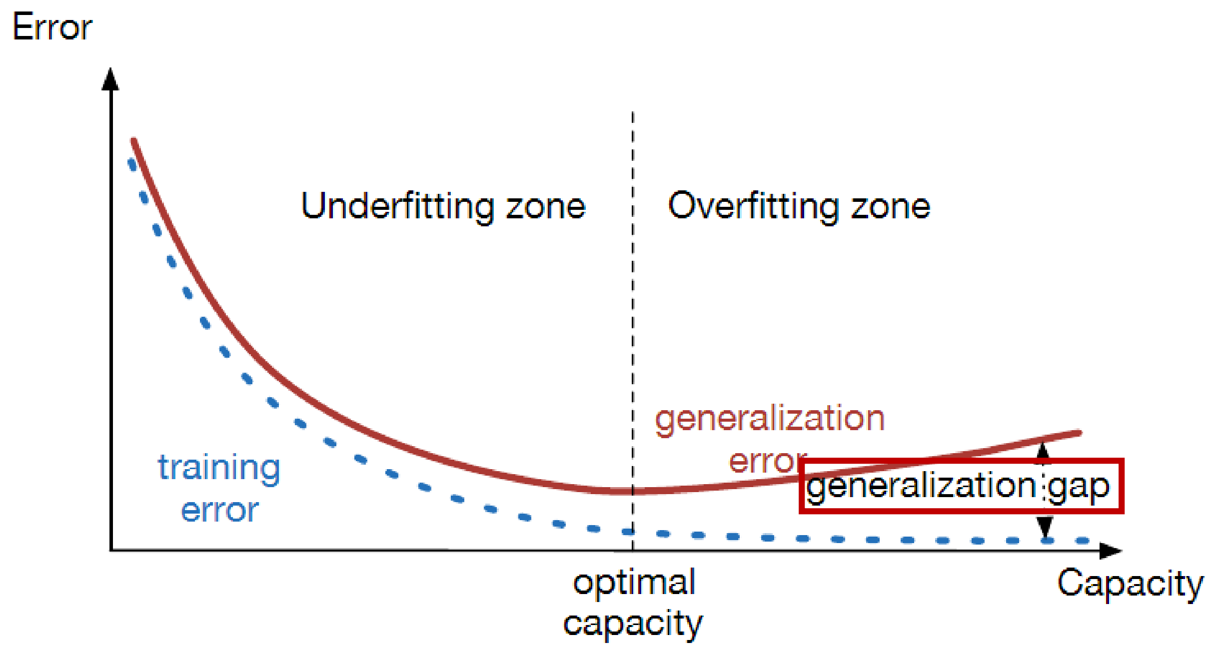
# Training Neural Networks II

Fast Campus
Start Deep Learning with TensorFlow

# Underfitting vs Overfitting



# Underfitting vs Overfitting




- test error

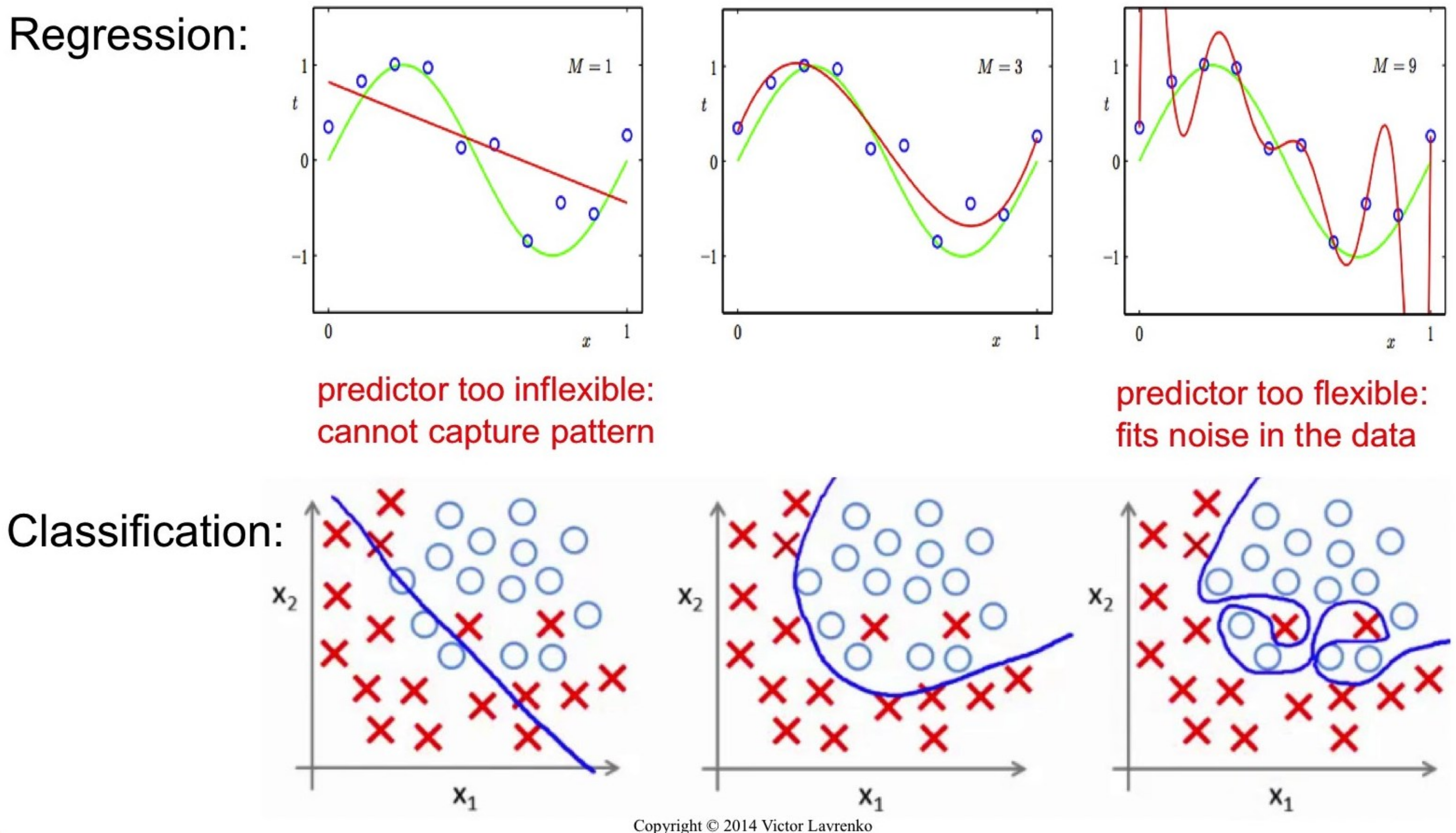
= train error + generalization gap

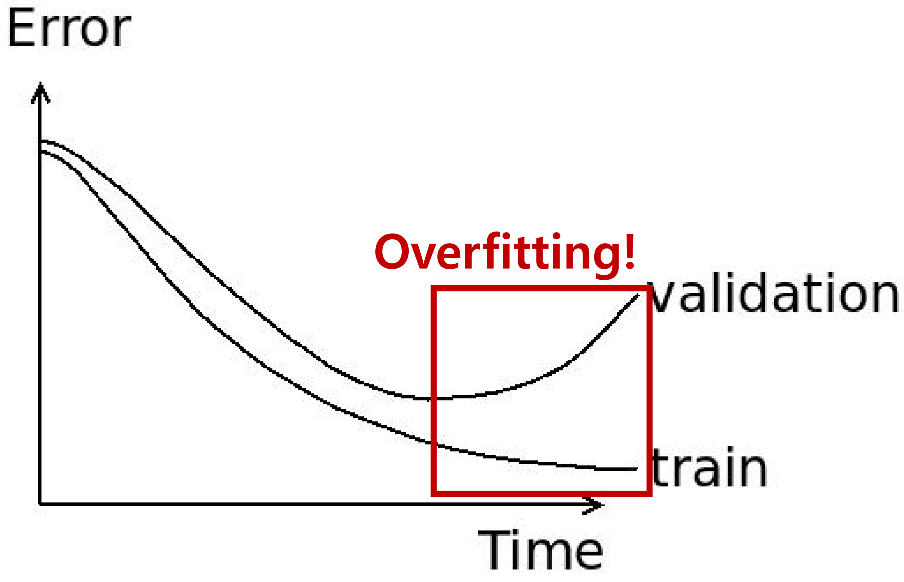
- Model capacity가 작으면 optimal capacity에 도달 자체가 불가능
- Model capacity가 너무 크면 overfitting이 일어남 (generalization gap이 커짐)

- Capacity는 크게하고 generalization gap을 줄이는 방법을 찾아보자 → deep learning!

# Fight Against Overfitting

- Data가 많지 않아서 발생
- 학습한 data에만 최적화되어서, 학습하지 않은 data(test data)에 대한 추론 성능이 악화되는 현상



# Regularization Method

- Dropout
- Weight Decay(L2 Regularization)

$$E(w) = E_0(w) + \boxed{\frac{1}{2}\lambda \sum_i w_i^2}$$

**L2 regularization** $R(W) = \sum_k \sum_l W_{k,l}^2$

L1 regularization $R(W) = \sum_k \sum_l |W_{k,l}|$

Elastic net (L1 + L2) $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

- Batch Normalization
  - Benefits of BN
    - Increase learning rate
    - Remove dropout
    - Reduce L2 weight decay
    - Remove LRN

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

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

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

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

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

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

# Add Term to Loss

$$L = \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1) + \boxed{\lambda R(W)}$$

In common use:

**L2 regularization** $R(W) = \sum_k \sum_l W_{k,l}^2$ (Weight decay)

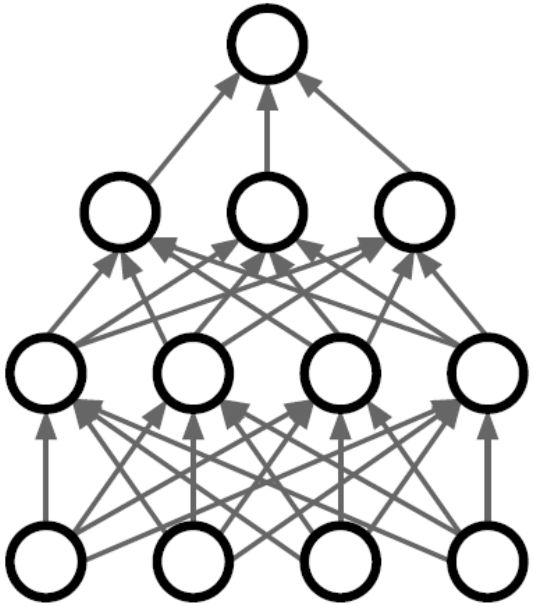
L1 regularization $R(W) = \sum_k \sum_l |W_{k,l}|$

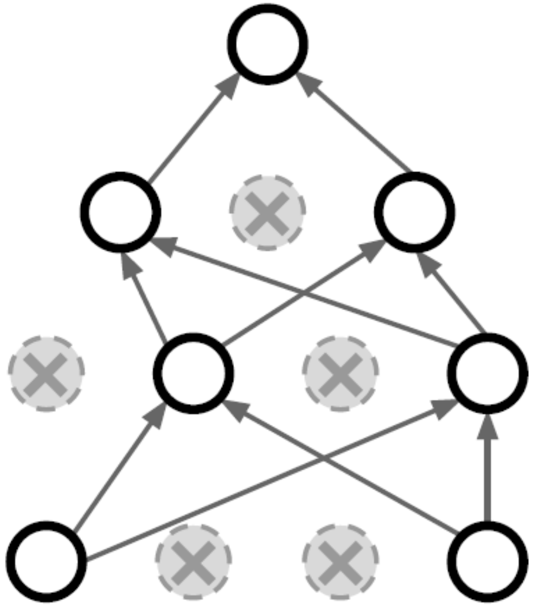
Elastic net (L1 + L2) $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

Slide Credit : Stanford CS231n

# Dropout

- In each forward pass, randomly set some neurons to zero
- Probability of dropping is a hyperparameter; 0.5 is common





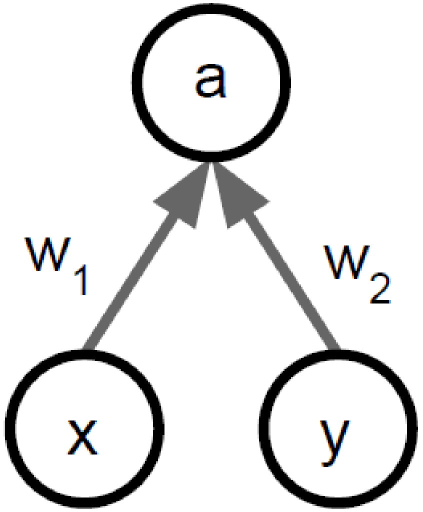
Dropout is training a large **ensemble** of models (that share parameters).

Each binary mask is one model

An FC layer with 4096 units has 24096 ~ 101233 possible masks!
Only ~ 1082 atoms in the universe...

# Dropout: Test Time

- Dropout makes the output random!
- Want to average out the randomness at test time




Consider a single neuron.

At test time we have: $E[a] = w_1x + w_2y$

During training we have: $$E[a] = \frac{1}{4}(w_1x + w_2y) + \frac{1}{4}(w_1x + 0y) + \frac{1}{4}(0x + 0y) + \frac{1}{4}(0x + w_2y) = \frac{1}{2}(w_1x + w_2y)$$

At test time, **multiply** by dropout probability

# Batch Normalization

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

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

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

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

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

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

After normalization, allow the network to squash the range if it wants to

- Improves gradient flow through the network
- Allows higher learning rates
- Reduces the strong dependence on initialization
- Acts as a form of regularization in a funny way, and slightly reduces the need for dropout, maybe

Note, the network can learn:

$$\gamma^{(k)} = \sqrt{\mathrm{Var}[x^{(k)}]}$$

$$\beta^{(k)} = \mathrm{E}[x^{(k)}]$$

to recover the identity mapping.

Slide Credit : Stanford CS231n

# Batch Normalization: Test Time

**Input:** Values of $x$ over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: $\gamma$, $\beta$
**Output:** $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

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

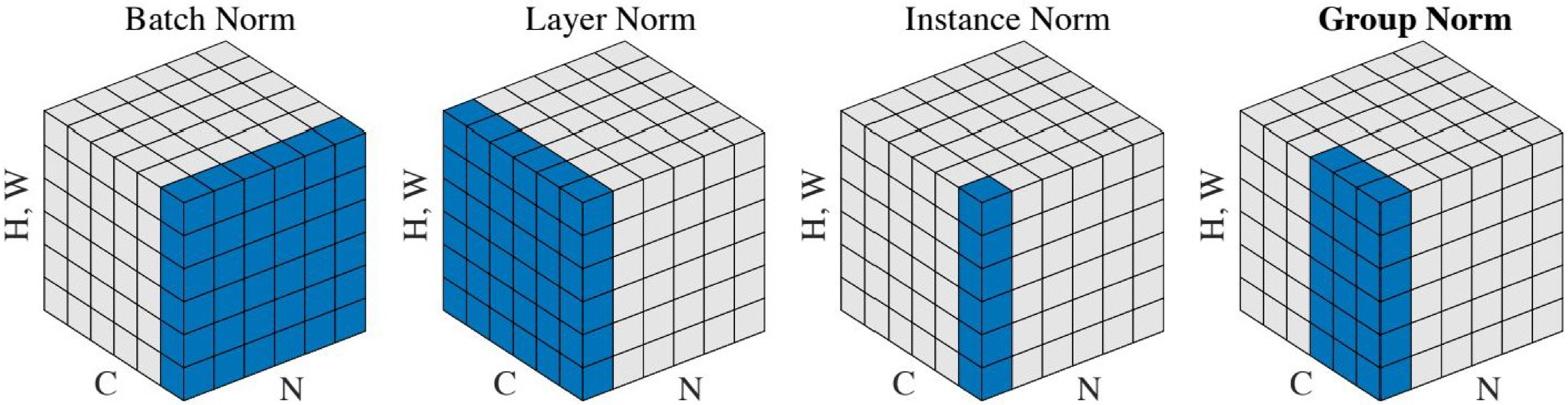
$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

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

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

- The mean/std are not computed based on the batch. Instead, a single fixed empirical mean of activations during training is used.
- It can be estimated during training with running averages

# Other Normalizations




Slide Credit : Stanford CS231n

# Regularization: A Common Pattern

- Training : Add some kind of randomness

Output (label)

Input (image)

$$\boxed{y} = f_W(\boxed{x}, \boxed{z})$$

Random mask

- Testing : Average out randomness

$$y = f(x) = E_z\big[f(x, z)\big] = \int p(z) f(x, z) dz$$

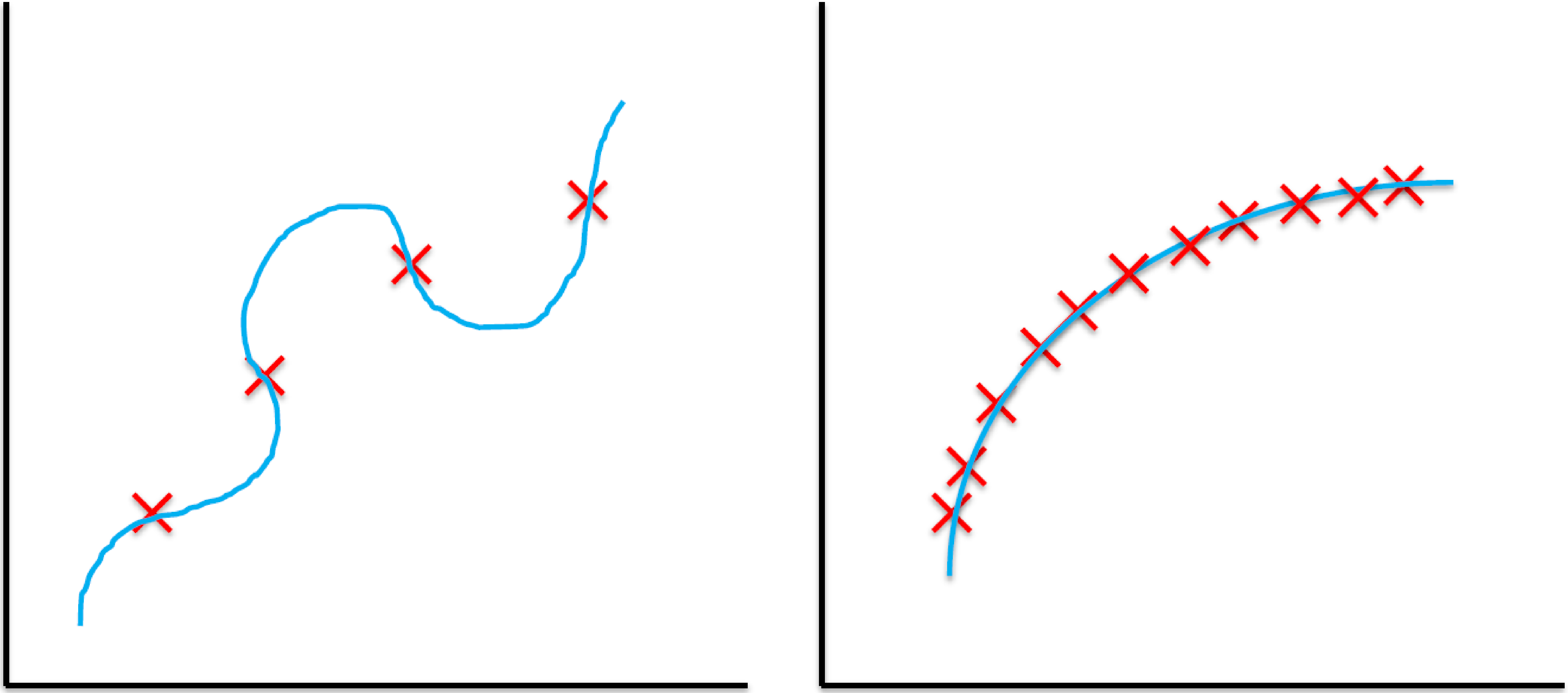
# Regularization: A Common Pattern

- Dropout
  - Training : Randomly set some neurons to zero
  - Testing : Average out the randomness  by multiplying dropout probability

- Batch Normalization
  - Training : Normalize using stats from random mini-batches
  - Testing : Use fixed stats to normalize

Slide Credit : Stanford CS231n

# Data Augmentation



# Data Augmentation

원본

Flip(LR)

Flip(UD)

Translation

Rotate

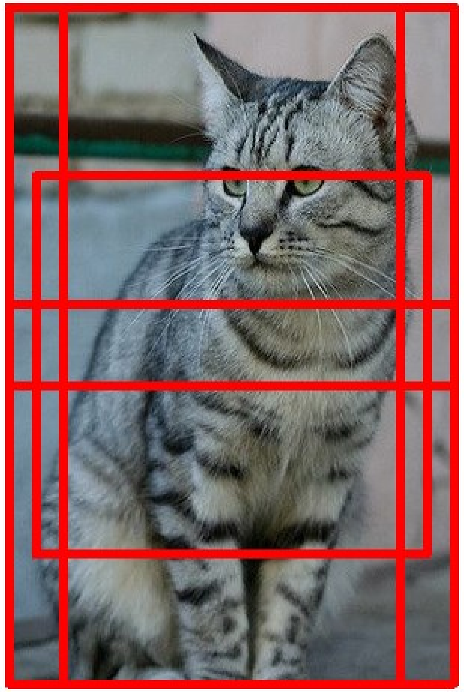
CROP

# Data Augmentation - Example

- Training : sample random crops / scales

ResNet:

1. Pick random L in range [256, 480]
2. Resize training image, short side = L
3. Sample random 224x224 patch



- Testing : average a fixed set of crops

ResNet:

1. Resize image at 5 scales: {224, 256, 384, 480, 640}
2. For each size, use 10 224x224 crop: 4 corners + center, + flips

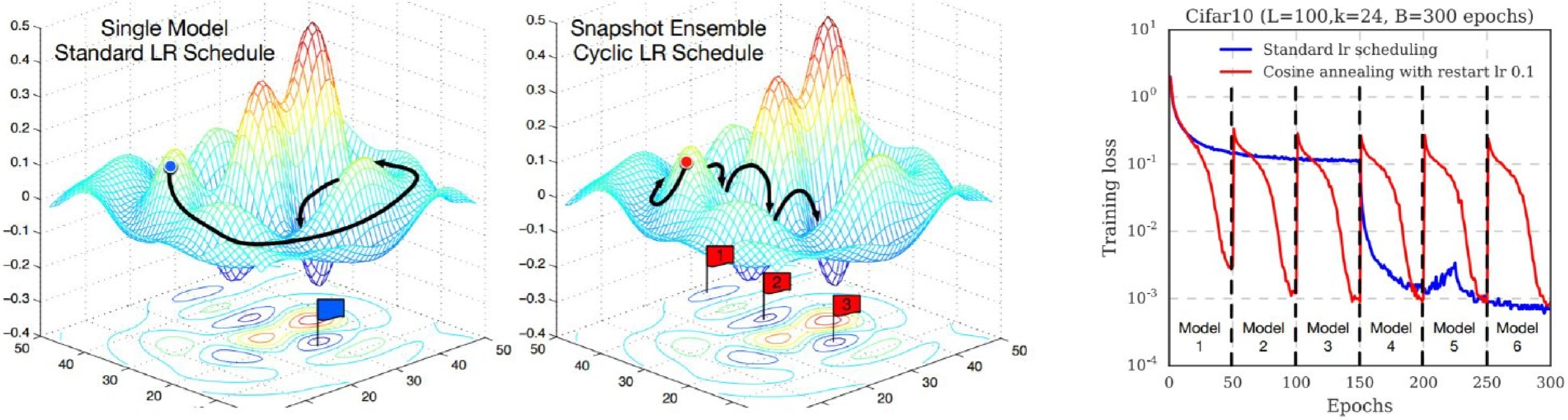
# Model Ensembles

1. Train multiple independent models
2. At test time average their results
   - Take average of predicted probability distributions, then choose argmax

- Enjoy 2% extra performance!

# Model Ensembles: Tips and Tricks

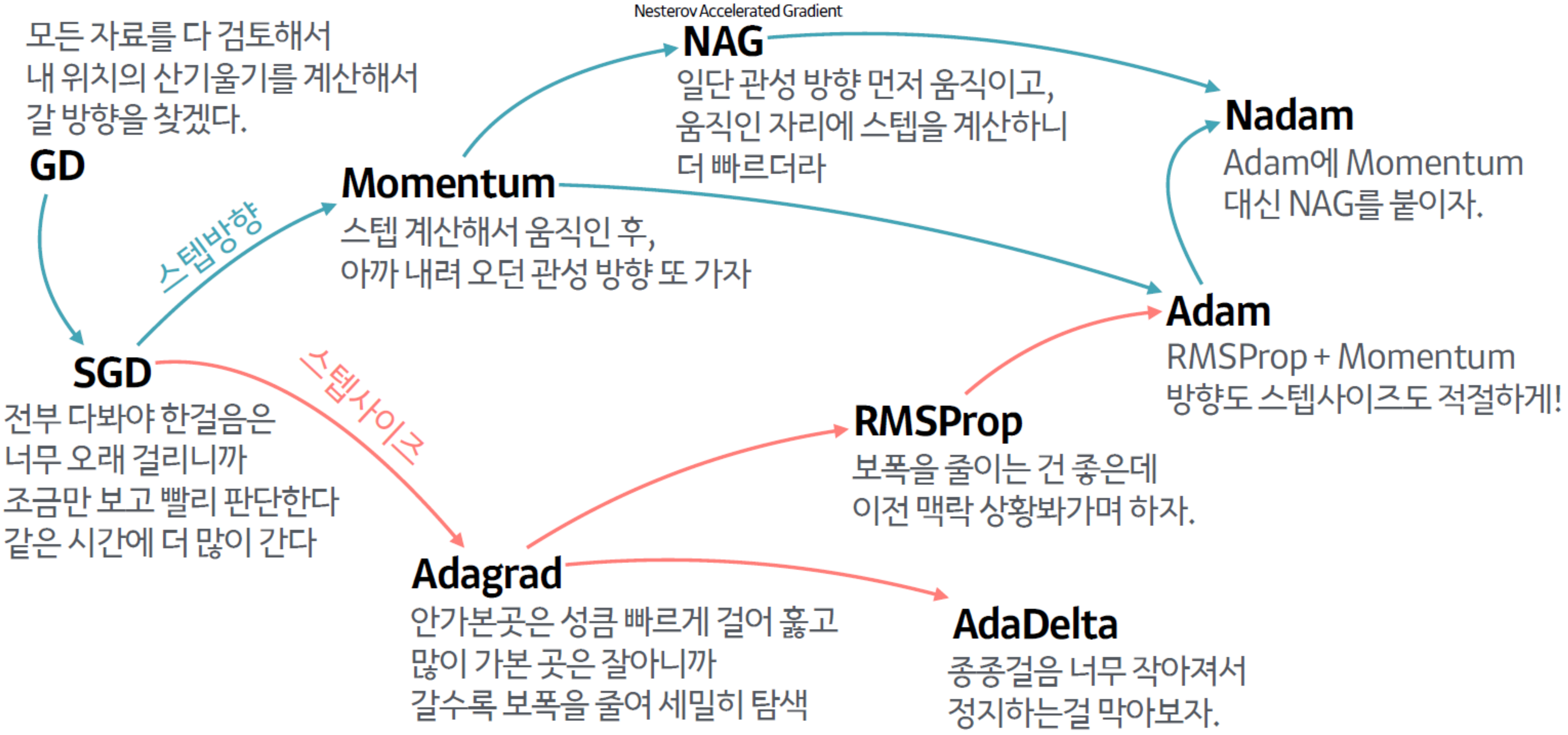
- Instead of training independent models, use multiple snapshots of a single model during training!




Slide Credit : Stanford CS231n

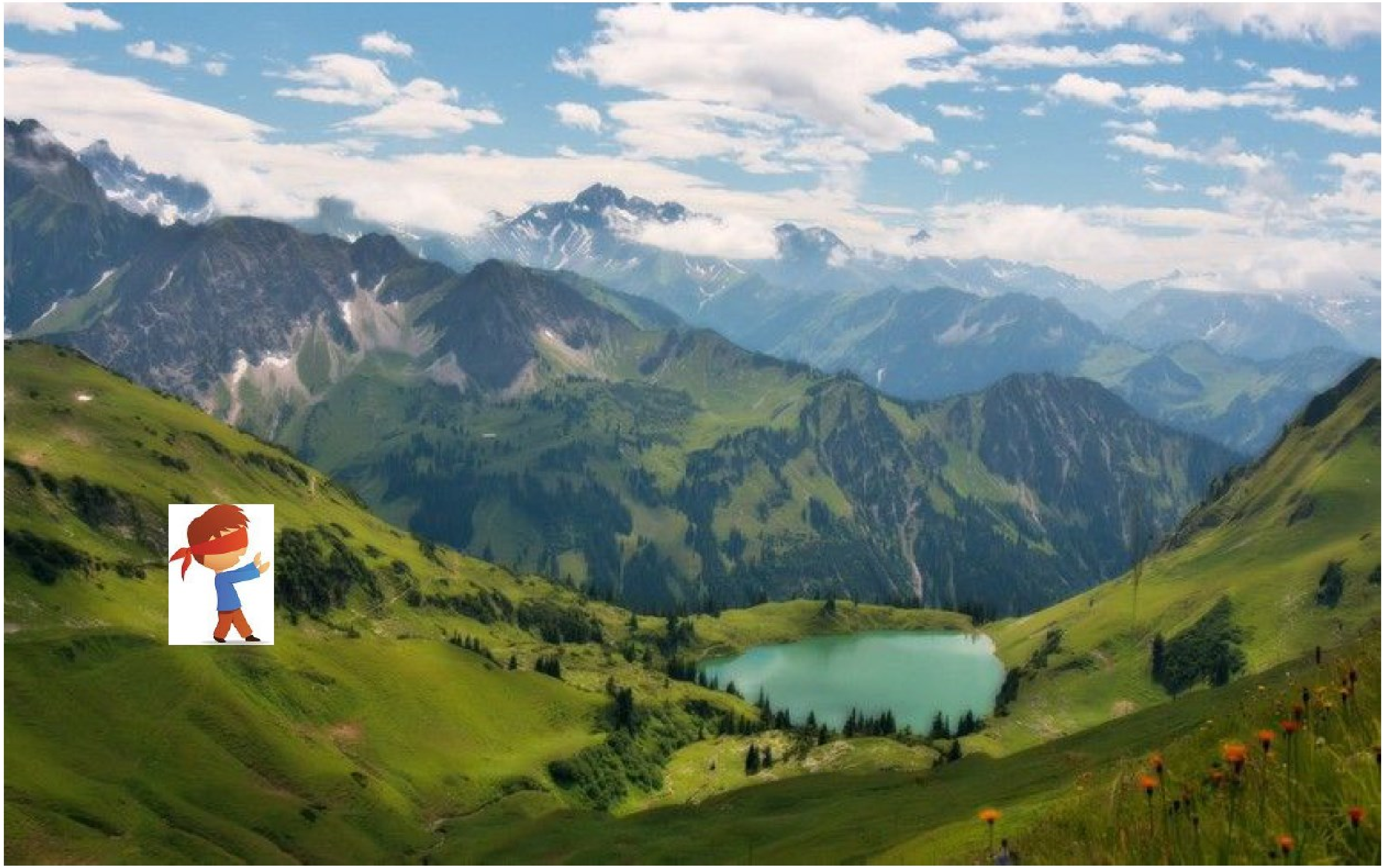
# Now we will study

- Optimization Methods




Slide Credit : 하용호@Kakao

# Gradient Descent



Slide Credit : Stanford CS231n

# Recap : Gradient Descent

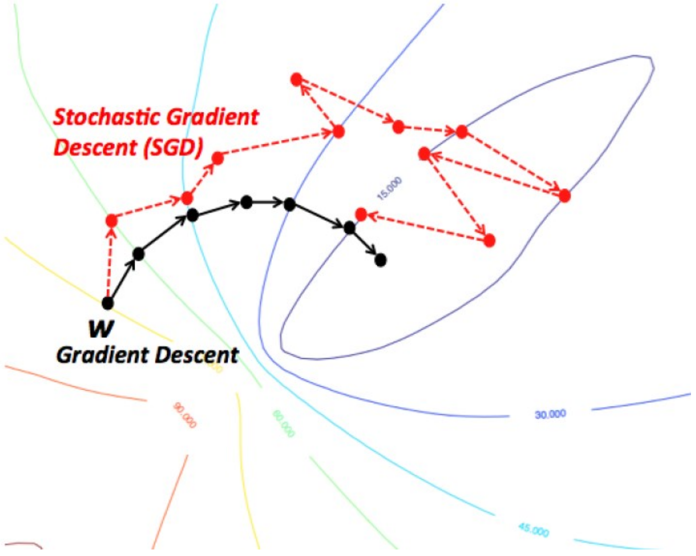
- Batch gradient descent

$$\theta_t = \underbrace{\boxed{\theta_{t-1}}}_{} - \boxed{\eta} \cdot \boxed{\nabla_\theta J(\theta)}$$

Previous parameter

Learning Rate

Gradient




- Stochastic gradient descent

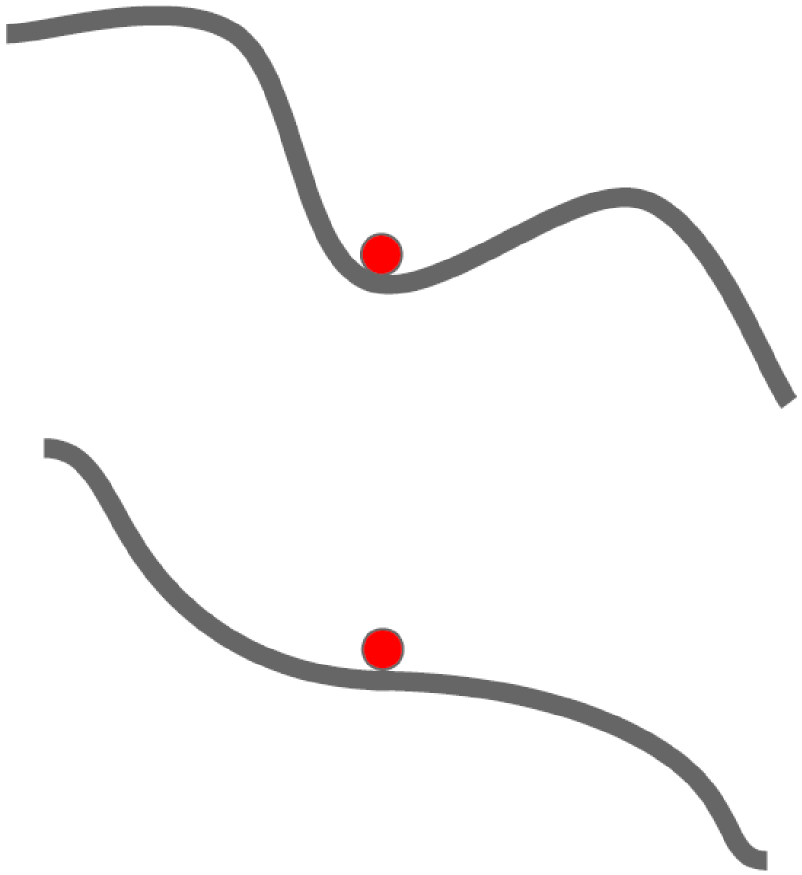
$$\theta_t = \theta_{t-1} - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)})$$

- Mini-batch gradient descent

$$\theta_t = \theta_{t-1} - \eta \cdot \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)})$$

# Problems of Gradient Descent

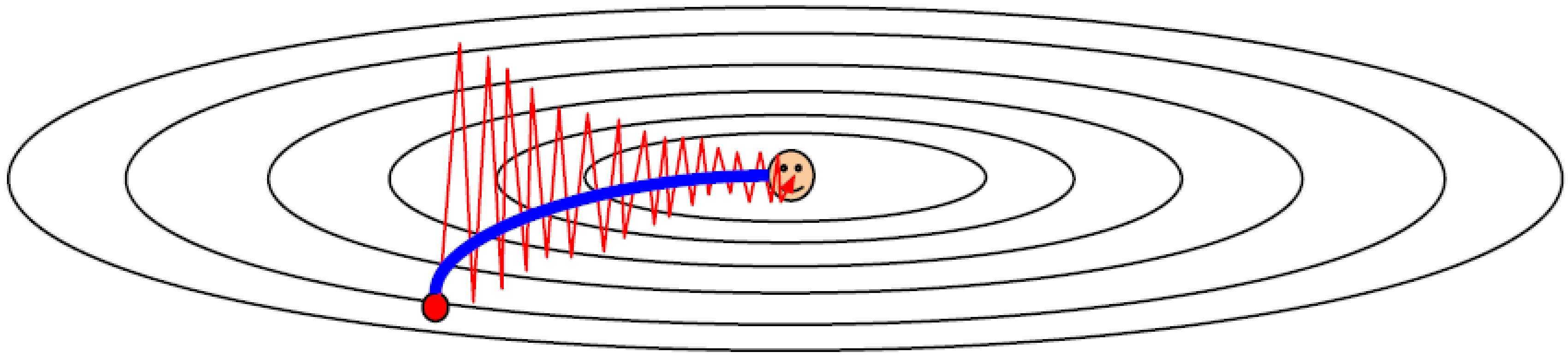
- (It can) stuck at local minima or saddle point(zero gradient)



Picture Credit : Stanford CS231n

# Problems of Gradient Descent

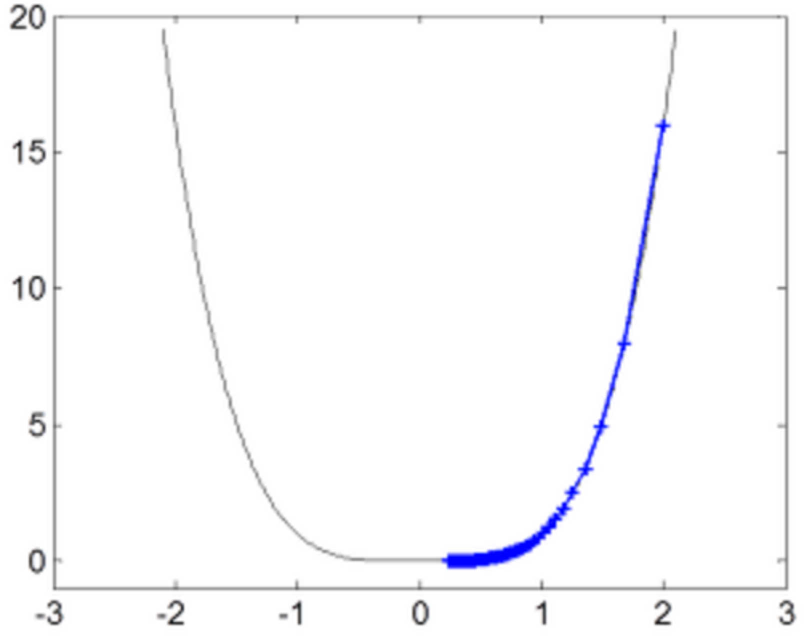
- Poor Conditioning



Picture Credit : Stanford CS231n

# Problems of Gradient Descent

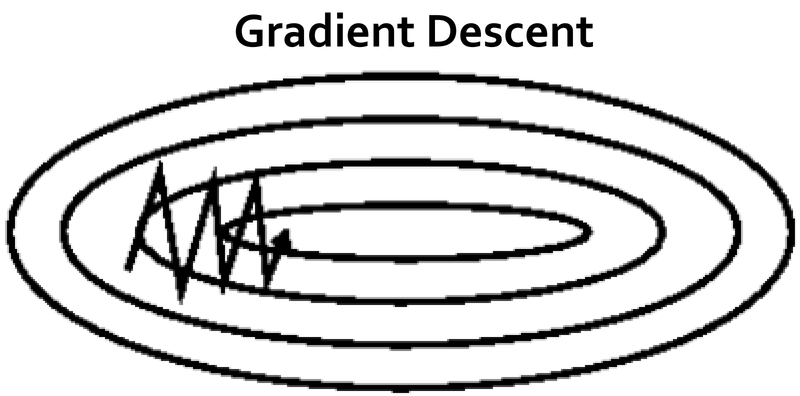
- Slow….
  - The closer to the optimal point, the smaller the gradient becomes.

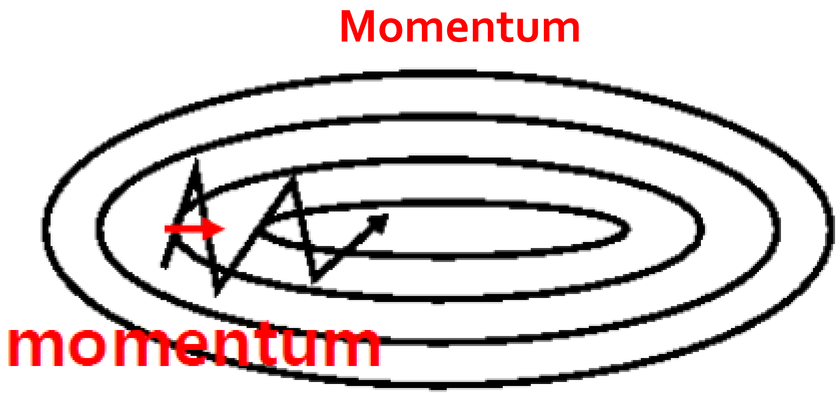


# Momentum

- Let's move with inertia in the direction that we moved earlier.

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$$
$$\theta_t = \theta_{t-1} - v_t$$

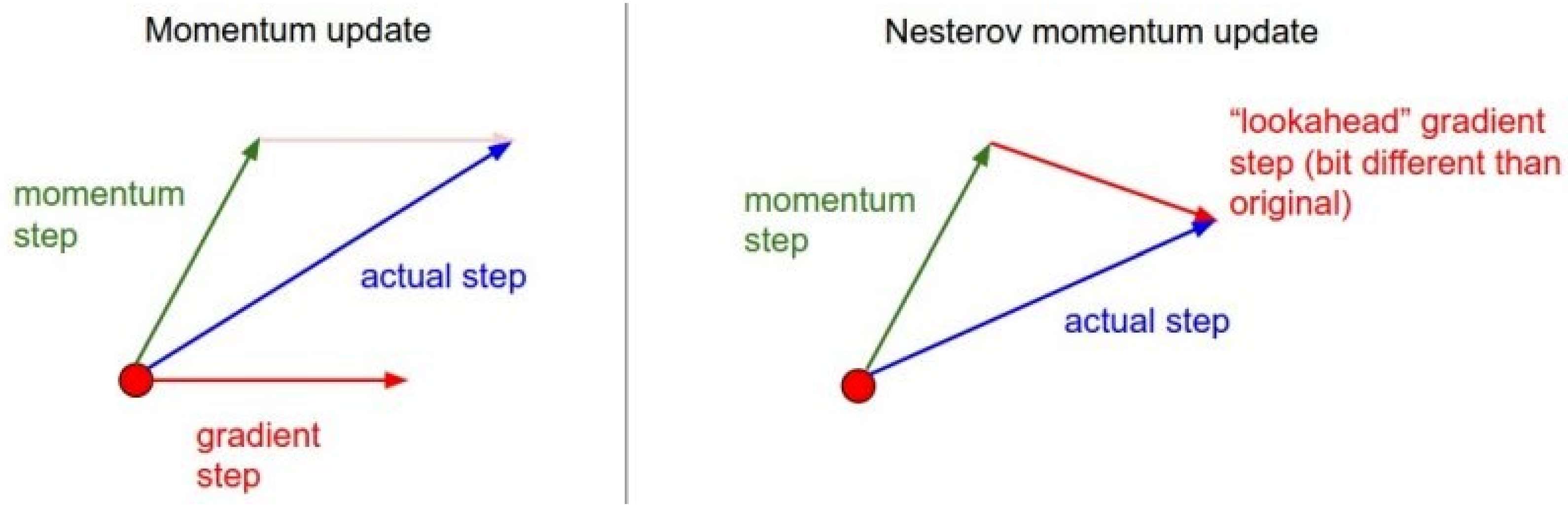


# NAG(Nesterov Accelerated Gradient)

- Move in the direction we were previously moving, and then calculate the gradient from where we moved.

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1})$$
$$\theta_t = \theta_{t-1} - v_t$$



# Adagrad(Adaptive Gradient)

- It adapts the learning rate to the parameters, performing larger updates for infrequent and smaller updates for frequent parameters.

Element-wise Product

$$G_t = G_{t-1} + \boxed{(\nabla_\theta J(\theta_t))^2}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \cdot \nabla_\theta J(\theta_t)$$

- As the learning continues, the G value continues to increase, so the step size becomes too small to learn

# RMSprop

- Use exponentially decaying average of squared gradients

$$G_t = \gamma G_{t-1} + (1-\gamma)(\nabla_\theta J(\theta_t))^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \cdot \nabla_\theta J(\theta_t)$$

# Adam(Adaptive Moment Estimation)

- Adaptive Moment Estimation (Adam) stores both exponentially decaying average of past gradients and squared gradients
- Combination of Momentum and RMSprop
- Compensate the initial momentum biased towards zero

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

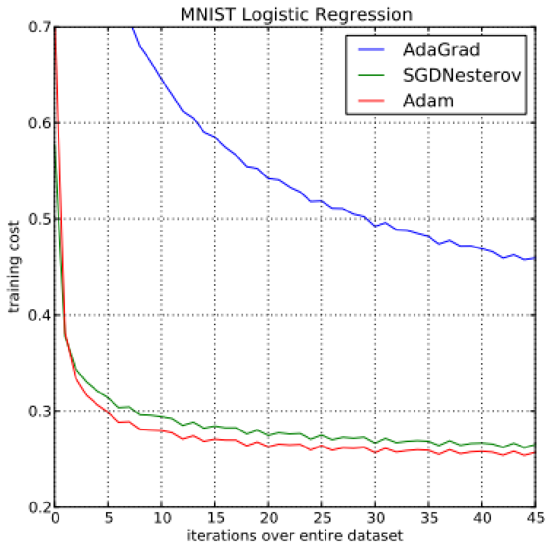
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

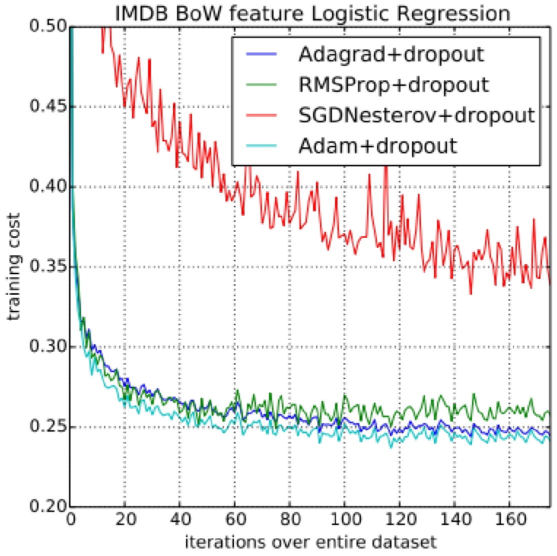
$$\widehat{m_t} = {}^{m_t}/_{1-\beta_1^t} \quad \widehat{v_t} = {}^{v_t}/_{1-\beta_2^t}$$

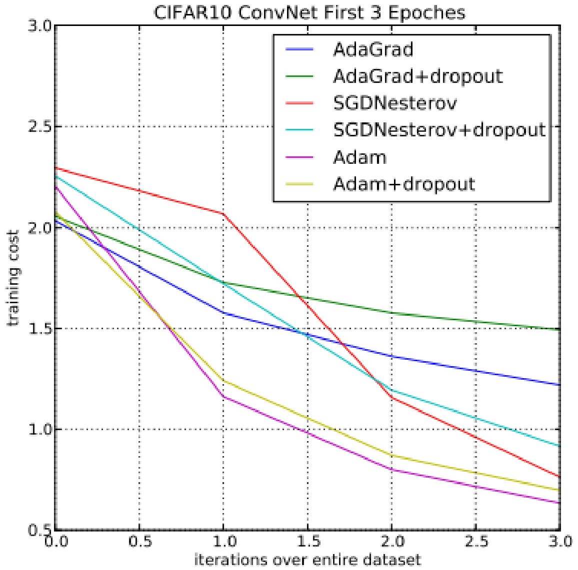
$$\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{\widehat{v_t}} + \epsilon} \widehat{m_t}$$

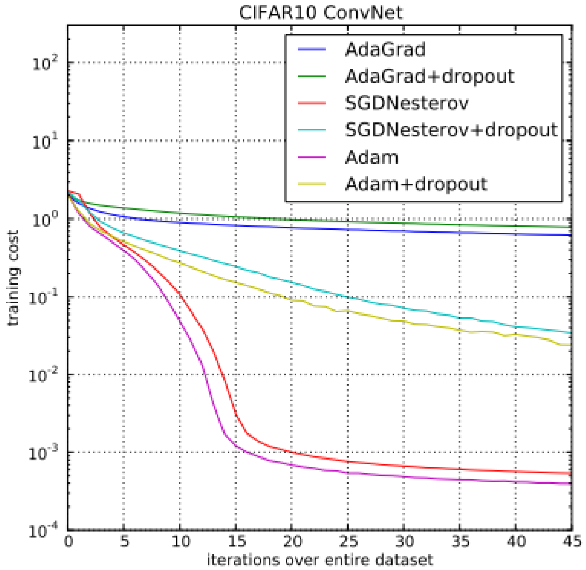
# Adam

- Adam is a very popular method



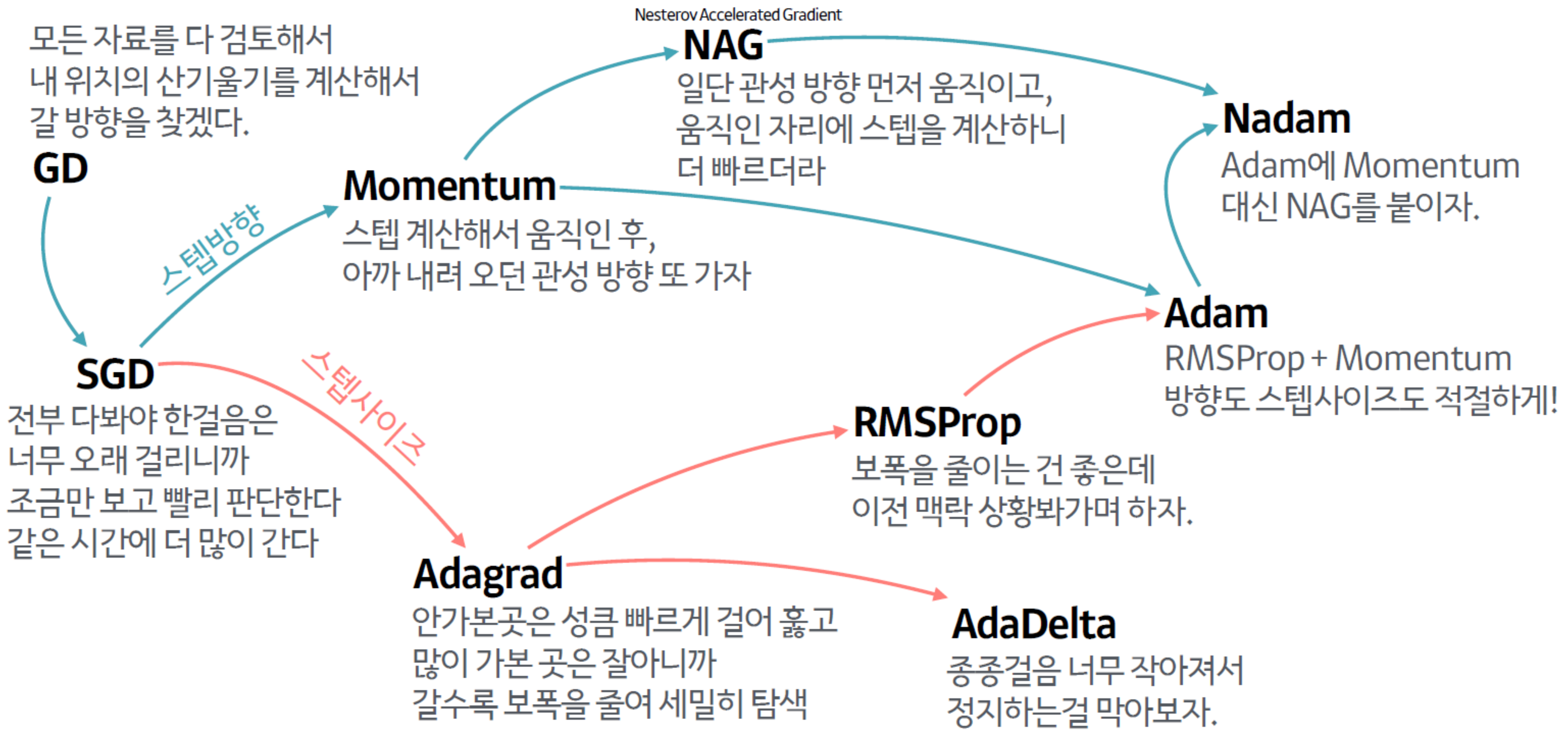



# We Studied

- Optimization Methods



# Optimizers in Tensorflow

- Just use these APIs!
  - `tf.train.Optimizer`
  - `tf.train.GradientDescentOptimizer`
  - `tf.train.AdadeltaOptimizer`
  - `tf.train.AdagradOptimizer`
  - `tf.train.AdagradDAOptimizer`
  - `tf.train.MomentumOptimizer`
  - `tf.train.AdamOptimizer`
  - `tf.train.FtrlOptimizer`
  - `tf.train.ProximalGradientDescentOptimizer`
  - `tf.train.ProximalAdagradOptimizer`
  - `tf.train.RMSPropOptimizer`