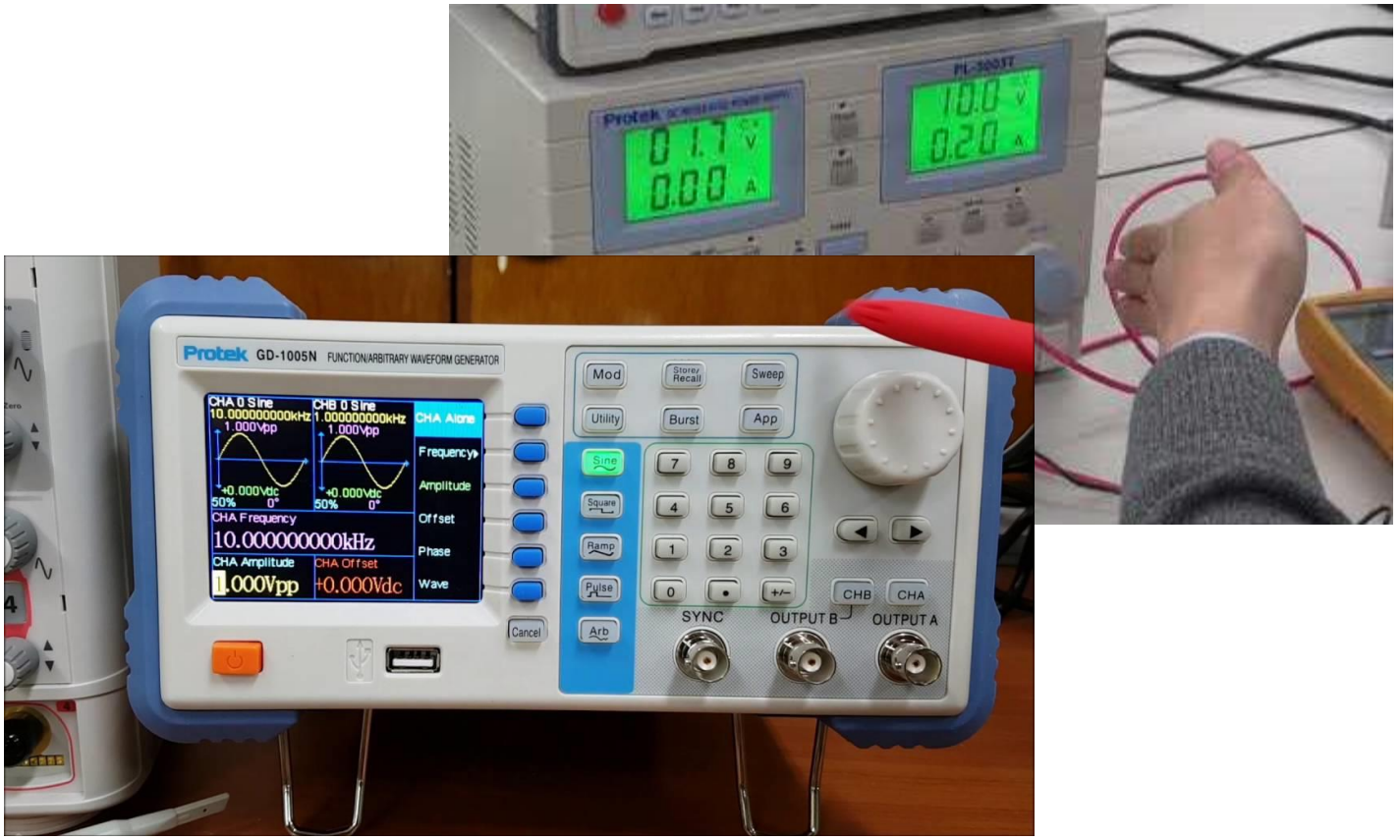
4. Neural networks 설계 고려사항

Ha-Jin Yu, Dept. of Computer Science, University of Seoul
서울시립대학교 컴퓨터과학부 유하진
2019

HJYU@UOS.AC.KR

학습목표

- Deep neural networks를 설계하고 학습하기 위하여 tool을 사용할 때 고려해야 할 다양한 조건들을 알아본다.



참고 자료

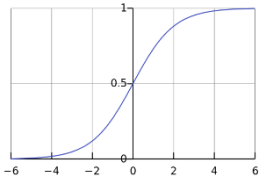
- https://www.deeplearningbook.org/
  - Chapter 8 Optimization for Training Deep Models
- http://cs231n.stanford.edu/syllabus.html
  - Lecture 8 April 25 Training Neural Networks, part II
- Youtube: Lecture Collection | Convolutional Neural Networks for Visual Recognition (Spring 2017) Stanford University School of Engineering
  - Lecture 7 | Training Neural Networks II

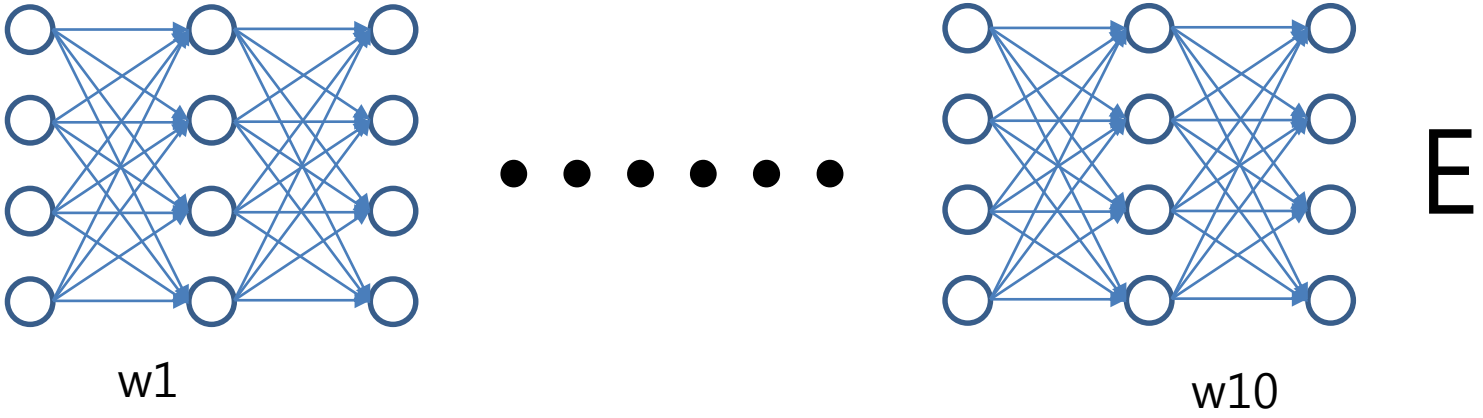
목차

- Vanishing gradient problem
- Restricted Boltzmann Machine(RBM)
- Deep Belief Networks
- Objectives
- Optimizers
- Regularization
- Initialization
- Output Units
- Dropout
- Summary : Neural networks 설계 고려사항

기울기 소실 문제 (vanishing gradient problem)

- Deep neural network 에서 문제
  - Back propagation 중 작은 기울기가 계속 곱해져서 0에 수렴
- 해결방법
  - Bottom-up layerwise unsupervised pre-training
  - ReLU
  - Gradient Flow

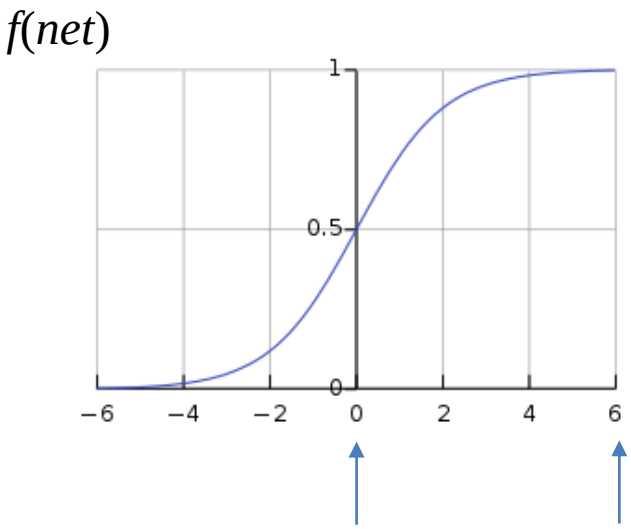




기울기 소실 문제 해결

Vanishing gradient problem

$$\Delta w_{ij} = c f'(net_j) x_i \sum_k \delta_k w_{jk} = c\delta_j x_i$$



$f'(net_j)$ → 0

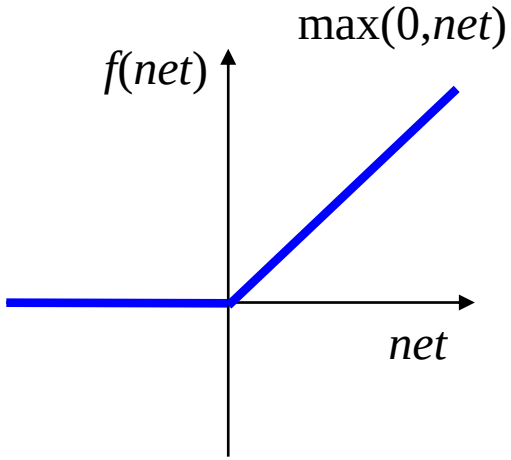
Δw_{ij} → 0

Update → 0

기울기 = 1

기울기 → 0

해결 방법



Rectified linear unit (ReLU)

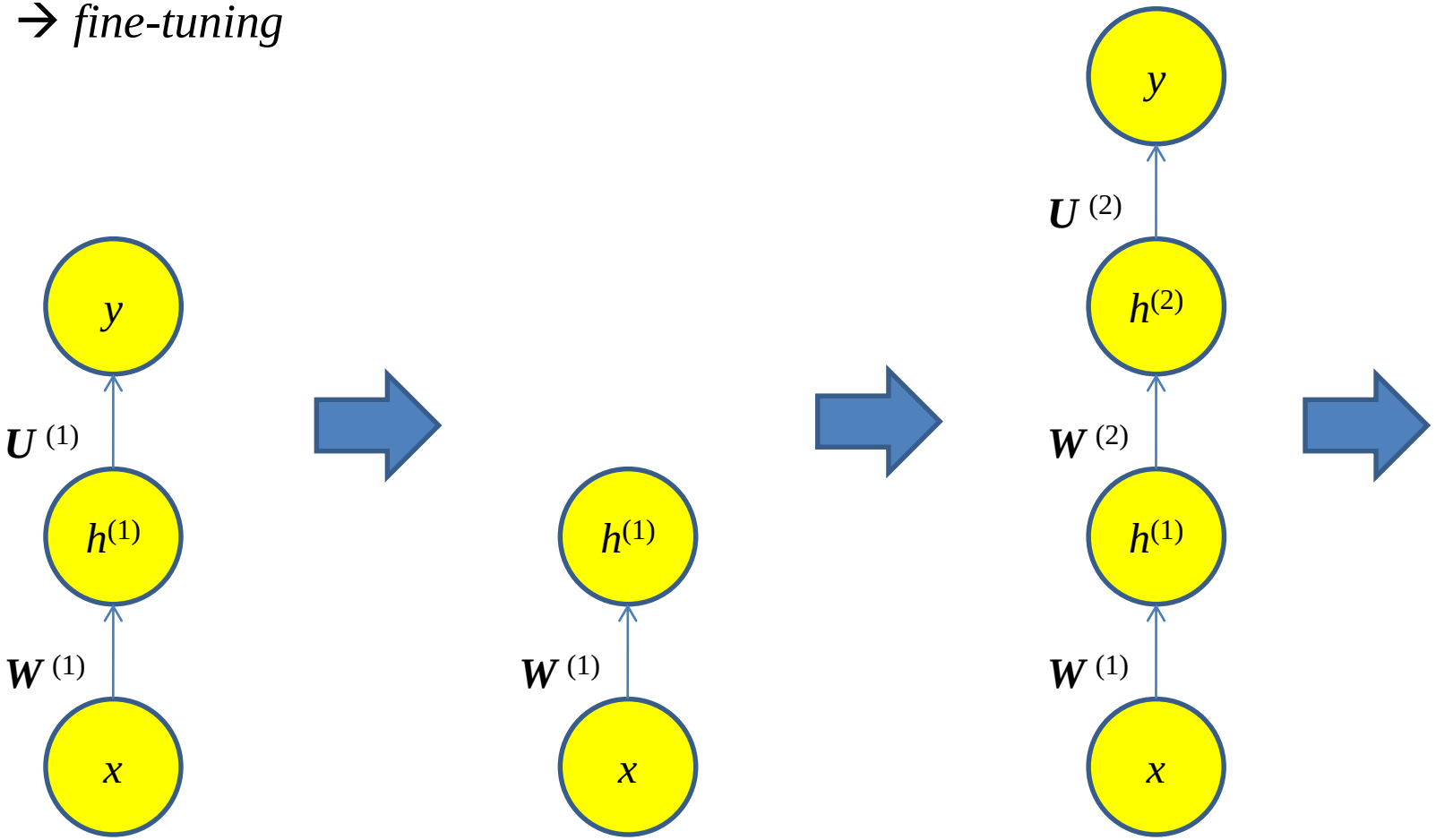
$f'(net) = 1 \quad \text{if } net > 0$

$f'(0) = 0.5$

- $f'(net) = 0$ 에 가까운 값이 층마다 계속 곱해짐.

Greedy layerwise supervised pretraining

- 작은 부분을 차례로 학습
  → *fine-tuning*



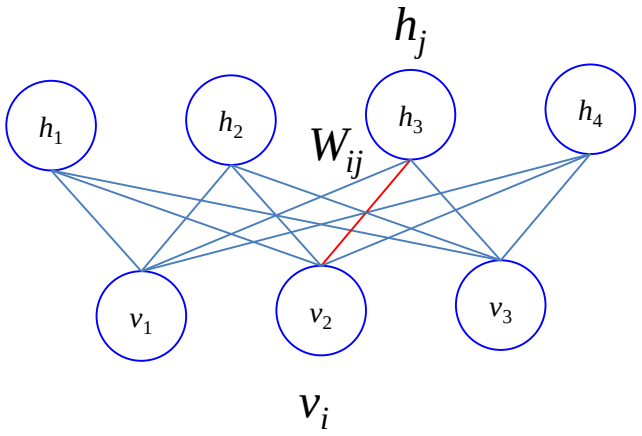
기울기 소실 문제 해결

Restricted Boltzmann Machine(RBM)

- Stochastic : data distribution 학습 가능
- Generative : 데이터 생성 가능

Hidden units **h**

Visible units **v**



- Energy:

$$E(\boldsymbol{v},\boldsymbol{h}) = -\boldsymbol{b}^{\mathrm{T}}\boldsymbol{v} - \boldsymbol{c}^{\mathrm{T}}\boldsymbol{h} - \boldsymbol{v}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{h}$$

- 생성 확률:

$$P(\boldsymbol{v},\boldsymbol{h}) = \frac{1}{Z}\exp(-E(\boldsymbol{v},\boldsymbol{h}))$$

$$E(\boldsymbol{v},\boldsymbol{h}) = -\sum_j b_j v_j - \sum_i h_i c_i - \sum_{i,j} v_i W_{ij} h_j$$

$$p(\mathbf{h} \mid \mathbf{v}) = \Pi_i p(\mathrm{h}_i \mid \mathbf{v})$$

$$p(\mathbf{v} \mid \mathbf{h}) = \Pi_i p(\mathrm{v}_i \mid \mathbf{h}).$$

$$P(\mathrm{h}_i = 1 \mid \mathbf{v}) = \sigma\left(\mathbf{v}^\top \boldsymbol{W}_{:,i} + b_i\right),$$

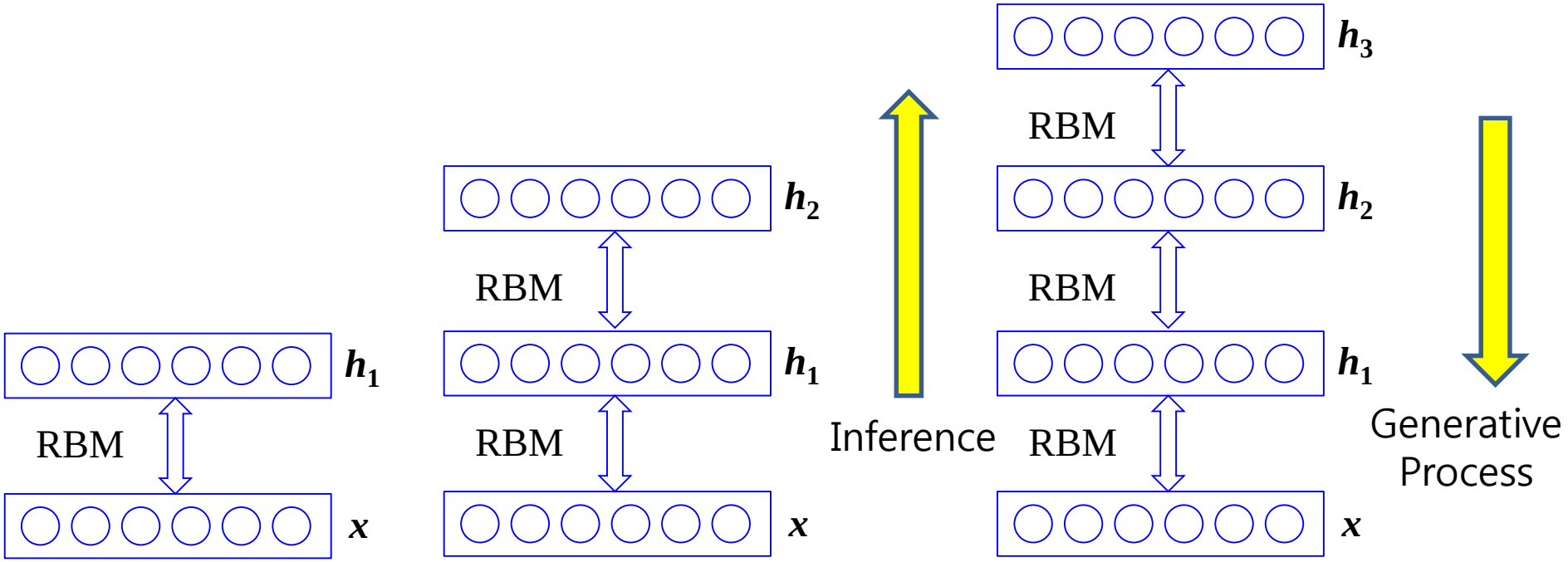
$$P(\mathrm{h}_i = 0 \mid \mathbf{v}) = 1 - \sigma\left(\mathbf{v}^\top \boldsymbol{W}_{:,i} + b_i\right)$$

$$\frac{\partial}{\partial W_{i,j}} E(\mathbf{v}, \mathbf{h}) = -\mathrm{v}_i \mathrm{h}_j.$$

기울기 소실 문제 해결

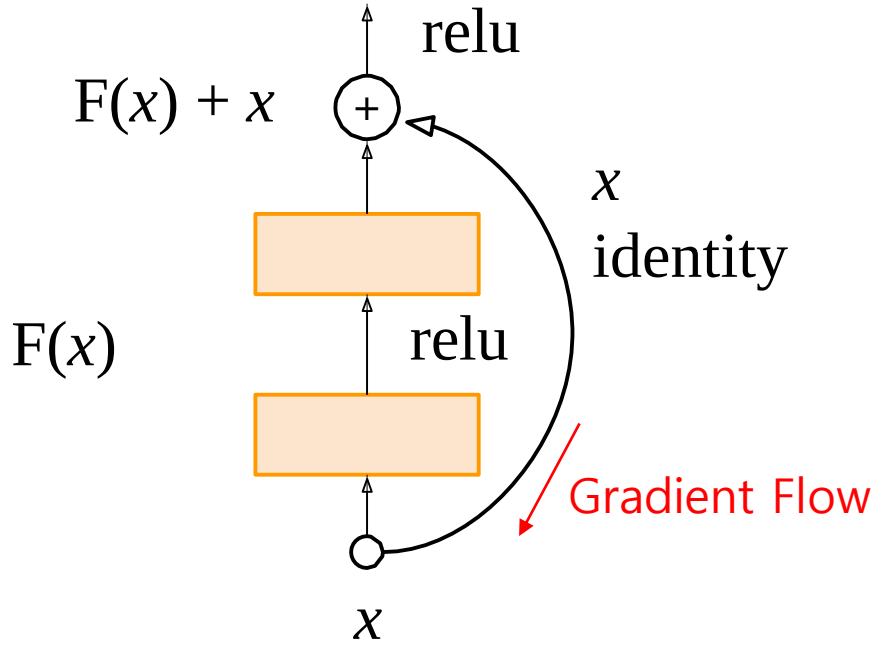
Deep Belief Networks (Hinton et al., 2006)

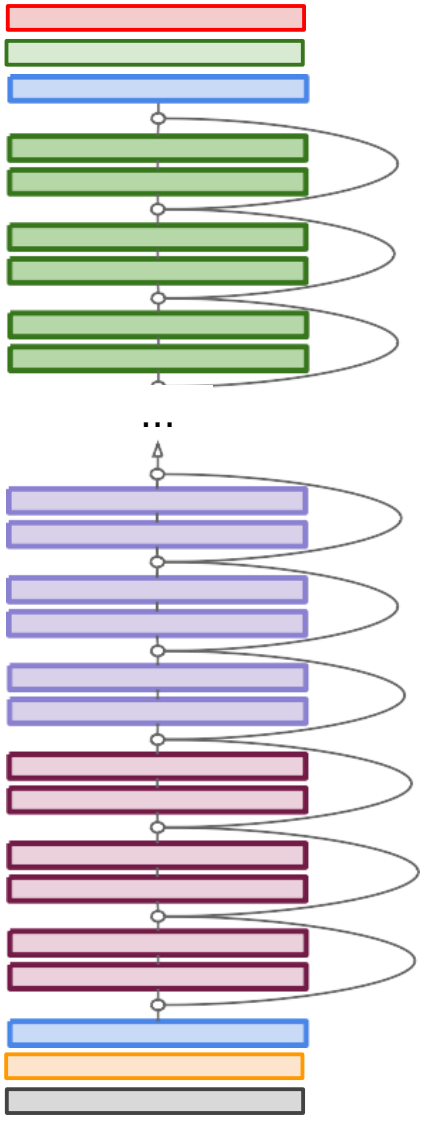
- Greedy layer-wise training
- 나중에 Supervised fine-tuning
- Deep learning의 시작 → 지금은 많이 사용되지 않음



기울기 소실
문제 해결

Residual Net



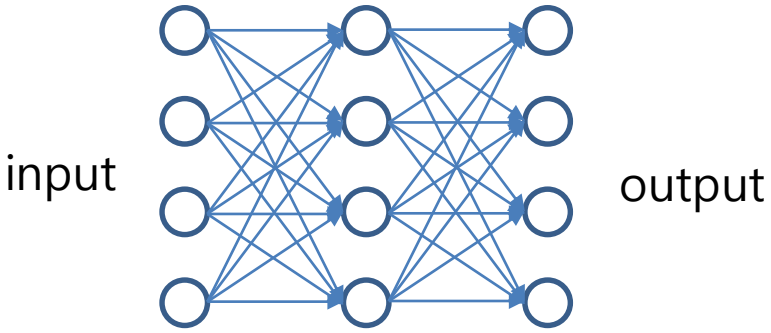


- Gradient Flow 를 위한 통로를 만들어 깊은 층 구현 가능

Summary : Neural networks 설계 고려사항

- Network topology : 어떤 모양?
- Activation function : 출력의 형태?
  - linear or non-linear?
- Objectives : 훈련의 목적?
  - Classification or regression ?

  = Loss function, Error
- Optimizers : 훈련의 방법?
  - Weights update
- Regularization
  - overfitting 을 어떻게 방지할까?



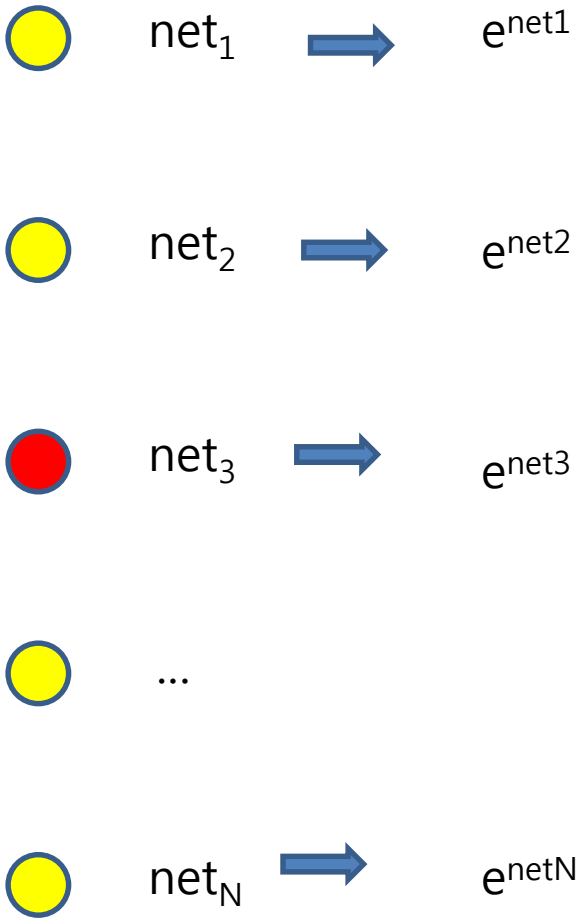
Output 형태

- **One-hot vector**

$$\mathbf{d}_n = [\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,]^\top$$

- **Softmax function**
  - 출력을 확률로 표현

$$p(C_k|X) = y_k = \frac{e^{net_j}}{\sum_{j=1}^{K} e^{net_j}}$$



Objective functions

Objectives (목적함수)

- Mean squared error

$$MSE = \frac{1}{2}(t - f(x))^2$$

- Cross entropy = negative log likelihood

$$E(\mathbf{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K} d_{nk} \log y_k(\mathbf{x}_n; \mathbf{w})$$

- Binary cross entropy:

$$C = -\frac{1}{n}\sum_x [y \ln f(x) + (1-y)\ln(1-f(x))]$$

- Categorical cross entropy:

$$C = -\frac{1}{n}\sum_x \sum_y [y \ln f(x) + (1-y)\ln(1-f(x))]$$

문제의 유형에 따라 사용되는 활성화 함수 및 오차함수의 종류

문제의 유형	Output Activation function	Objectives
Regression	Linear	Mean squared error
Binary classification	Logistic function (sigmoid)	Binary cross entropy
Multi class classification	Softmax	Categorical cross entropy

참고도서: 딥 러닝 제대로 시작하기

회기 (regression)

- 출력이 연속값을 갖는 함수를 대상으로 훈련 data 를 잘 재현하는 함수를 찾는 것.

- Squared error

$$\| \mathbf{d} - \mathbf{y}(\mathbf{x}, \mathbf{w}) \|^2$$

d: desired output (target)

y(x,w) : 신경망의 output

N : number of training samples

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \| \mathbf{d}_n - \mathbf{y}(\mathbf{x}_n, \mathbf{w}) \|^2$$

- 응용 분야: 매출액 추정, 생산 속도에 따른 불량률 추정, …, ∞

Binary Classification

- Two classes : $d = 0$, $d = 1$
- 훈련 데이터 $\{(\mathbf{x}_n, \mathbf{d}_n)\}$, $n=1, \ldots, N$

if d = 1 then output p(1|x)
if d = 0 then output p(0|x)

$$p(d=1|\mathbf{x}) \approx y(\mathbf{x}, \mathbf{w})$$

$$p(d|\mathbf{x}) = p(d=1|\mathbf{x})^d \, p(d=0|\mathbf{x})^{1-d}$$

- Maximum likelihood estimation

  likelihood 를 가장 크게 하는 w를 구한다.

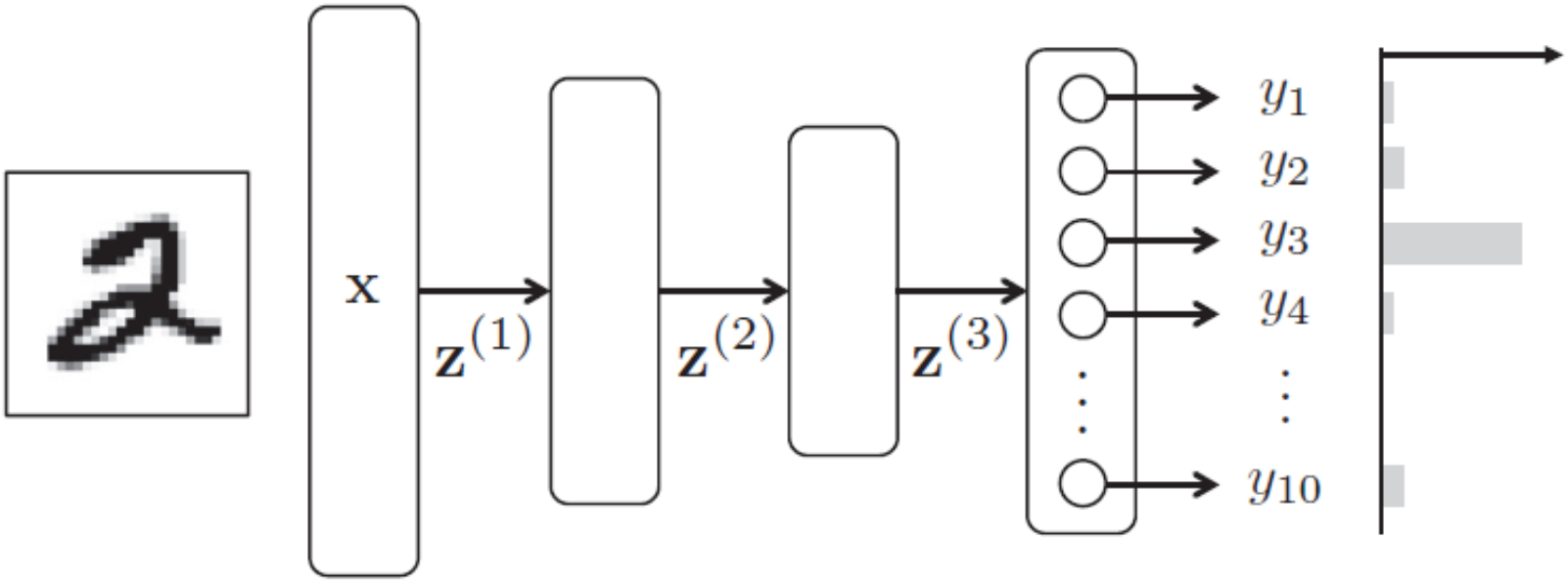
$$f(u) = \frac{1}{1+e^{-u}}$$

$$L(\mathbf{w}) \equiv \prod_{n=1}^{N} p(d_n|\mathbf{x}_n;\mathbf{w}) = \prod_{n=1}^{N} \{y(\mathbf{x}_n;\mathbf{w})\}^{d_n} \{1-y(\mathbf{x}_n;\mathbf{w})\}^{1-d_n}$$

  - 곱셈을 반복하면 underflow 가 생기므로 log 로 변환

$$E(\mathbf{w}) = -\sum_{n=1}^{N} [d_n \log y(\mathbf{x}_n, \mathbf{w}) + (1-d_n) \log \{1-y(\mathbf{x}_n, \mathbf{w})\}]$$

Multi-class Classification



- Classification 문제의 예. 숫자 필기 인식.
- 입력 이미지 **x**가 **0**에서 **9**까지의 숫자 중 어느 것이지를 판별한다.
- Softmax 층은 **0**에서 **9**까지의 숫자 중 입력 이미지가 어느 것에 가까운지를 나타내는 p_1, ..., p_{10}의 수치로 출력한다.

출처: 딥 러닝 제대로 시작하기

다클래스 분류의 오차 함수

- Classes: $\boldsymbol{C}_1, \ldots, \boldsymbol{C}_K$
- 목표 출력 (binary): $\mathbf{d}_n = [d_{n1}, \ldots, d_{nK}]^T$ ex) $\mathbf{d}_n = [0\,0\,1\,0\,0\,0\,0\,0\,0\,0]^\top$ ($\boldsymbol{C}_3$)

One-hot vector

- 사후 확률:

$$p(\mathbf{d}|\mathbf{x}) = \prod_{k=1}^{K} p(\mathcal{C}_k|\mathbf{x})^{d_k}$$

- 훈련 데이터 $\{(\mathbf{x}_n, \mathbf{d}_n)\}$, $n=1, \ldots, N$ 에 대한 likelihood

$$L(\mathbf{w}) = \prod_{n=1}^{N} p(\mathbf{d}_n|\mathbf{x}_n;\mathbf{w}) = \prod_{n=1}^{N}\prod_{k=1}^{K} p(\mathcal{C}_k|\mathbf{x}_n)^{d_{nk}} = \prod_{n=1}^{N}\prod_{k=1}^{K} (y_k(\mathbf{x};\mathbf{w}))^{d_{nk}}$$

- Cross entropy, negative log likelihood

$$E(\mathbf{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K} d_{nk} \log y_k(\mathbf{x}_n;\mathbf{w})$$

기타 Objectives (목적함수)

- Mean absolute error / mae
- Mean absolute percentage error / mape
- Mean squared logarithmic error / msle
- Hinge = $\max(0, 1 - t \cdot f(x))$
- Squared hinge
- Sparse categorical cross entropy
- Kullback leibler divergence / kld
- Poisson: mean of (predictions - targets * log(predictions))
- Cosine proximity: the opposite (negative) of the mean cosine proximity between predictions and targets.

t : desired output

Optimizers

Optimizers – weight update 방법

- Stochastic gradient descent
- Momentum
- Learning rate decay
- Rmsprop
- Adagrad
- Adadelta
- Adam
- Adamax
- Nadam
- Nesterov

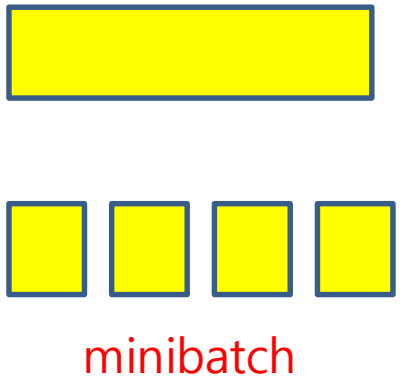
Batch and Minibatch Algorithms

1. Oneline : Sample 하나 마다 update
2. *batch* or *deterministic* gradient methods :
   - 모든 training sample 에 대하여 계산되는 error function 사용
   - 모든 input에 대하여 Δw를 계산하여 평균 이용 → 많은 시간 소요

$$E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w})$$

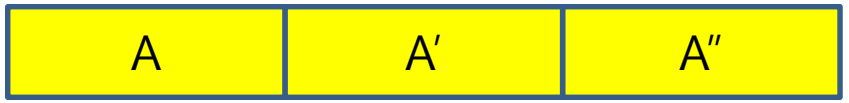
3. *minibatch* or *minibatch stochastic* methods
   - sample 의 일부만을 사용하여 파라미터 update

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \epsilon \nabla E_n$$



Stochastic gradient descent의 장점

- 훈련 데이터에 중복성이 있을 때 효율이 향상되고 학습이 빨리 진행된다



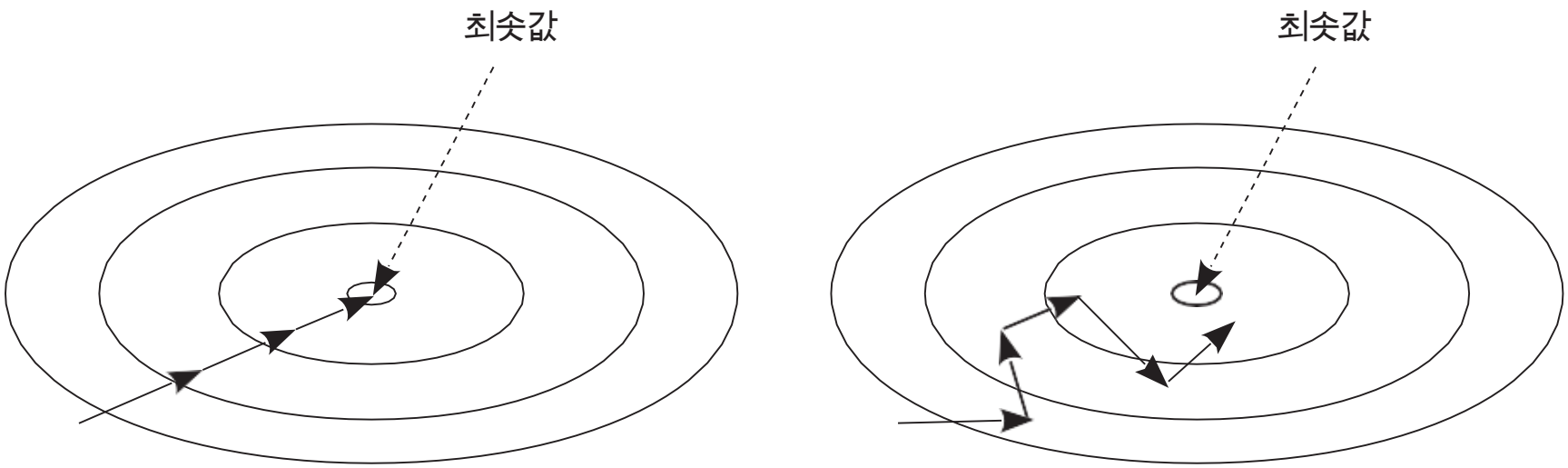
전체 데이터

Stochastic gradient descent (sgd)

- Local minimum 에 빠질 위험이 줄어든다
  - 반복할 때마다 random sample selection
- Update 의 크기가 작은 상태로 학습이 진행
  - 학습의 경과 관찰이 용이
- Online 학습
  - 데이터의 수집과 최적화가 동시에 진행

Stochastic gradient descent

- 기울기 벡터가 정확하게 계산되지 않음
- Avoid local minimum



모든 데이터를 사용한 경사 하강법 미니 배치에 의한 확률적 경사 하강법

그림 2-24 Stochastic gradient descent 방법으로 최솟값으로 향하는 모습

Algorithm 8.1 Stochastic gradient descent (SGD) update at training iteration *k*

Require: Learning rate ε.

Require: Initial parameter $\boldsymbol{\theta}$

while stopping criterion not met do

Sample a minibatch of m examples from the training set $\{\boldsymbol{x}(1), \ldots, \boldsymbol{x}(m)\}$ with corresponding targets $y^{(i)}$.

Compute gradient estimate: $\boldsymbol{g} \leftarrow 1/m \, \nabla_{\theta} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), y^{(i)})$

Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \varepsilon \boldsymbol{g}$

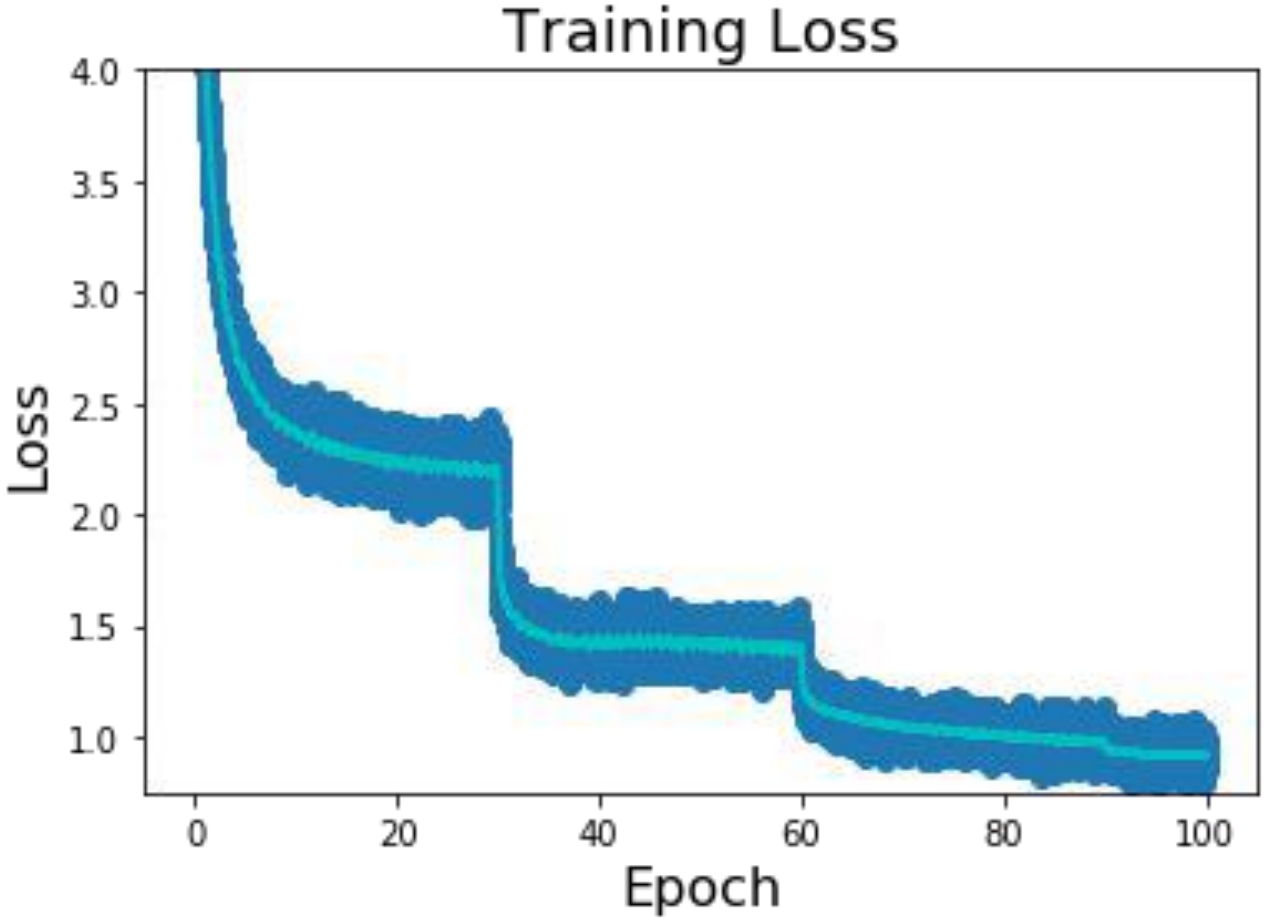
end while

학습률(ϵ)의 결정 방법

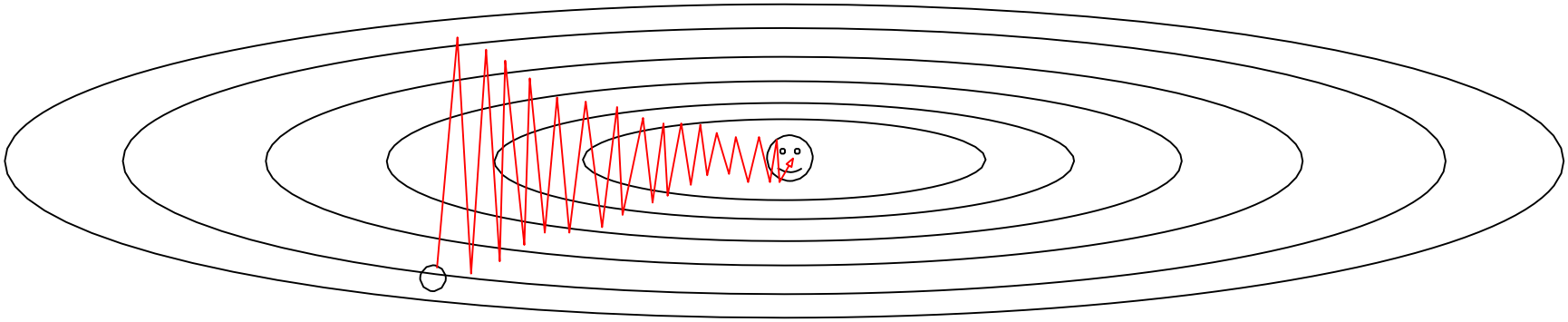
- Learning Rate Decay : 학습 초기에 값을 크게 설정했다가 학습의 진행과 함께 학습률을 점점 줄여가는 방법
  - 학습률을 파라미터 업데이트 회수에 비례해서 작아지도록

    $\epsilon = \epsilon_0/(\alpha t)$
  - 처음에 상수 $\epsilon = \epsilon_0$ 을 설정하고 어느 정도 학습이 진행되면 그 값을 1/10로 줄여
- 다른 예: 각 층마다 서로 다른 값을 사용
  - 각 층의 weight 업데이트 속도가 비슷하게 되도록
    - 예)
      - 출력에 가까운 층 : 작게
      - 입력에 가까운 층 : 크게

Learning Rate Decay



Problems with SGD

- 

Momentum

- 가속도 (gradient 가 속도를 조절하는 역할)
- 이전 weight 변화를 어느정도 유지

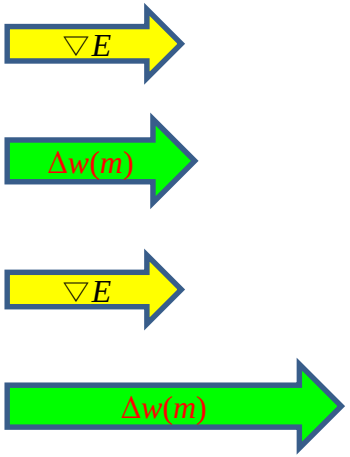
$$\Delta w^{(t)} = \mu \Delta w^{(t-1)} - \epsilon \nabla E_t$$

Momentum constant

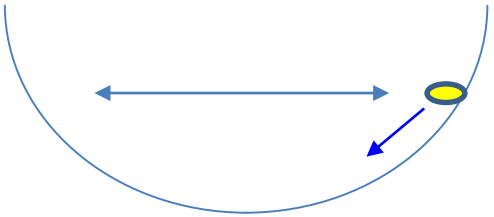
이전 변화량 (속도 v)

Learning rate

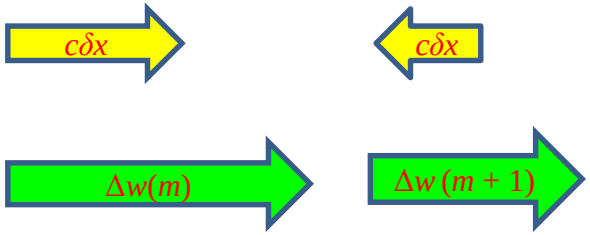
현재 기울기



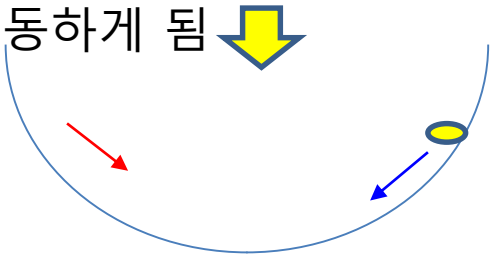
- weight의 업데이트 값에 이전 업데이트 값의 일정 비율을 더함.
- $\mu = 0.5 \sim 0.9$
- ϵ : learning rate



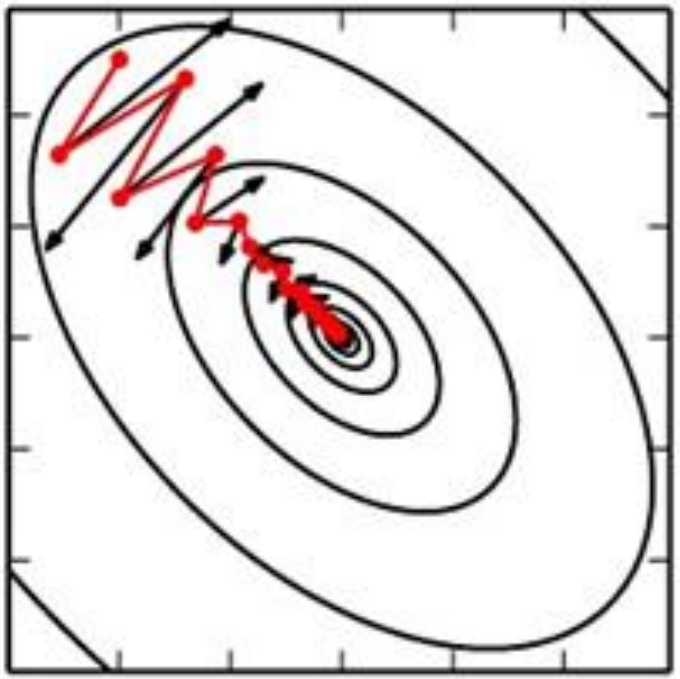
- Weight 변화가 oscillating 하는 것 방지

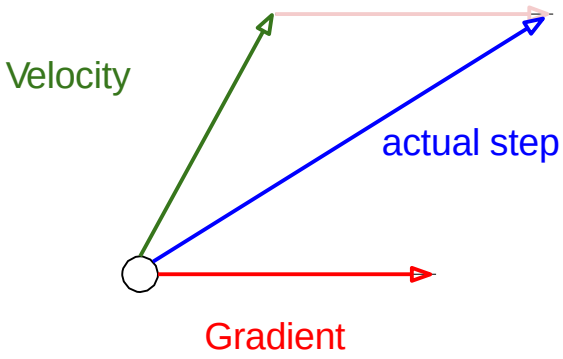


문제: 골짜기를 지그재그로 이동하게 됨



The effect of momentum





출처 https://www.deeplearningbook.org/

Nesterov Momentum (Sutskever *et al.* 2013)

- a variant of the momentum algorithm
- inspired by Nesterov's accelerated gradient method (Nesterov, 1983, 2004).
- The update rules :

$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \nabla_{\boldsymbol{\theta}} \left[\frac{1}{m} \sum_{i=1}^{m} L\left(\boldsymbol{f}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta} + \alpha \boldsymbol{v}), \boldsymbol{y}^{(i)} \right) \right]$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v},$$

- The difference between Nesterov momentum and standard momentum is
  - where the gradient is evaluated.
- With Nesterov momentum the gradient is evaluated
  - after the current velocity is applied.
- Thus one can interpret Nesterov momentum as attempting to add a *correction factor* to the standard method of momentum.
- The complete Nesterov momentum algorithm is presented in Algorithm 8.3.

Nesterov momentum

Require: Learning rate ϵ, momentum parameter α.

Require: Initial parameter θ, initial velocity v.

while stopping criterion not met **do**

Sample a minibatch of m examples from the training set $\{x^{(1)}, \ldots, x^{(m)}\}$ with corresponding labels $y^{(i)}$.

Apply interim update: $\tilde{\theta} \leftarrow \theta + \alpha v$

Compute gradient (at interim point): $g \leftarrow \frac{1}{m} \nabla_{\tilde{\boldsymbol{\theta}}} \sum_i L(f(\boldsymbol{x}^{(i)}; \tilde{\boldsymbol{\theta}}), \boldsymbol{y}^{(i)})$

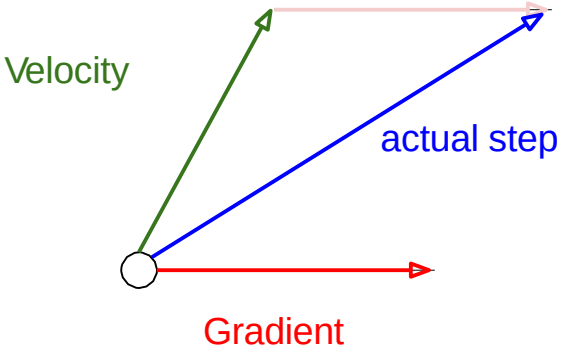
Compute velocity update: $v \leftarrow \alpha v - \epsilon g$

Apply update: $\theta \leftarrow \theta + v$

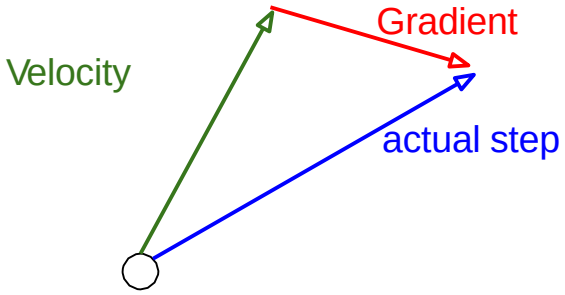
end while

Nesterov momentum

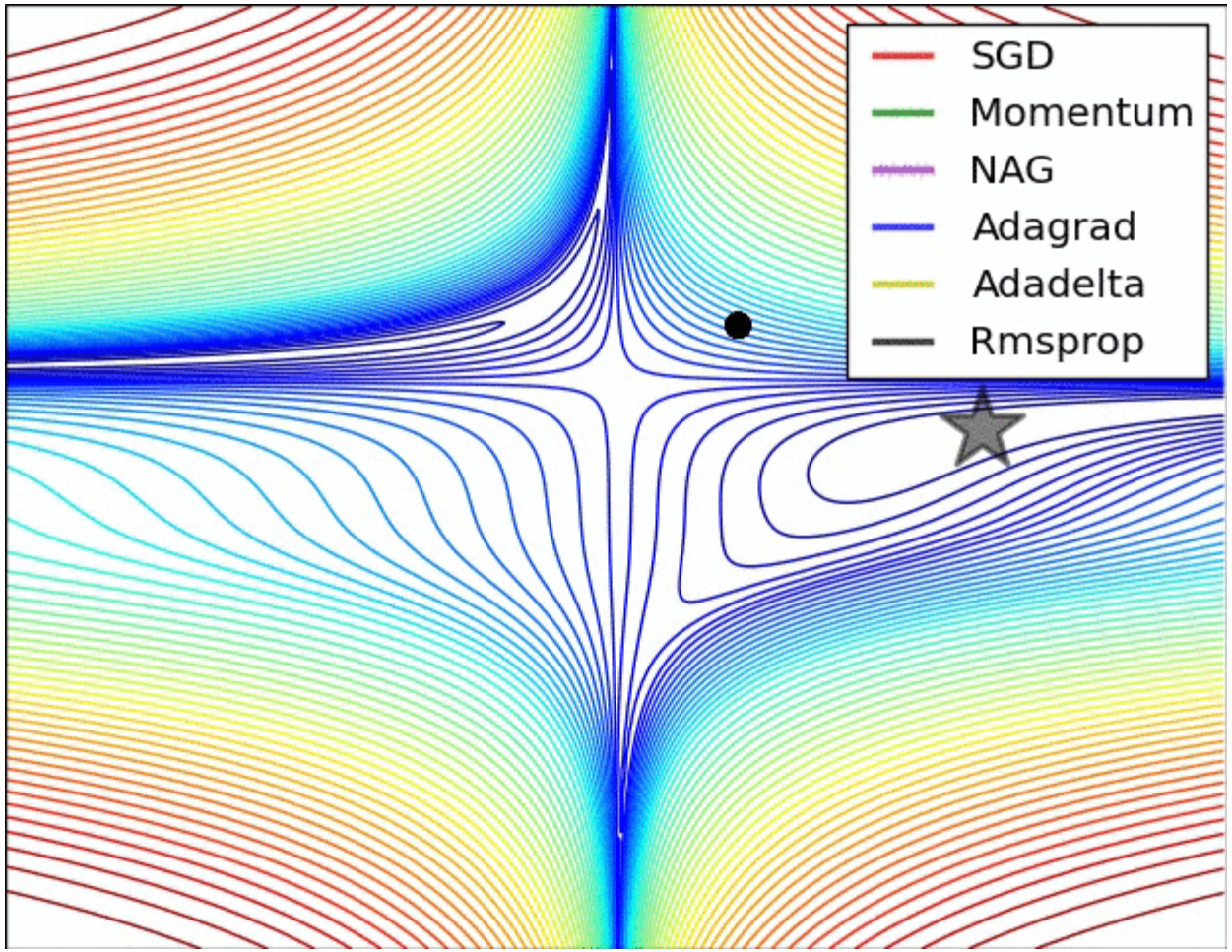
Momentum update:



Nesterov Momentum

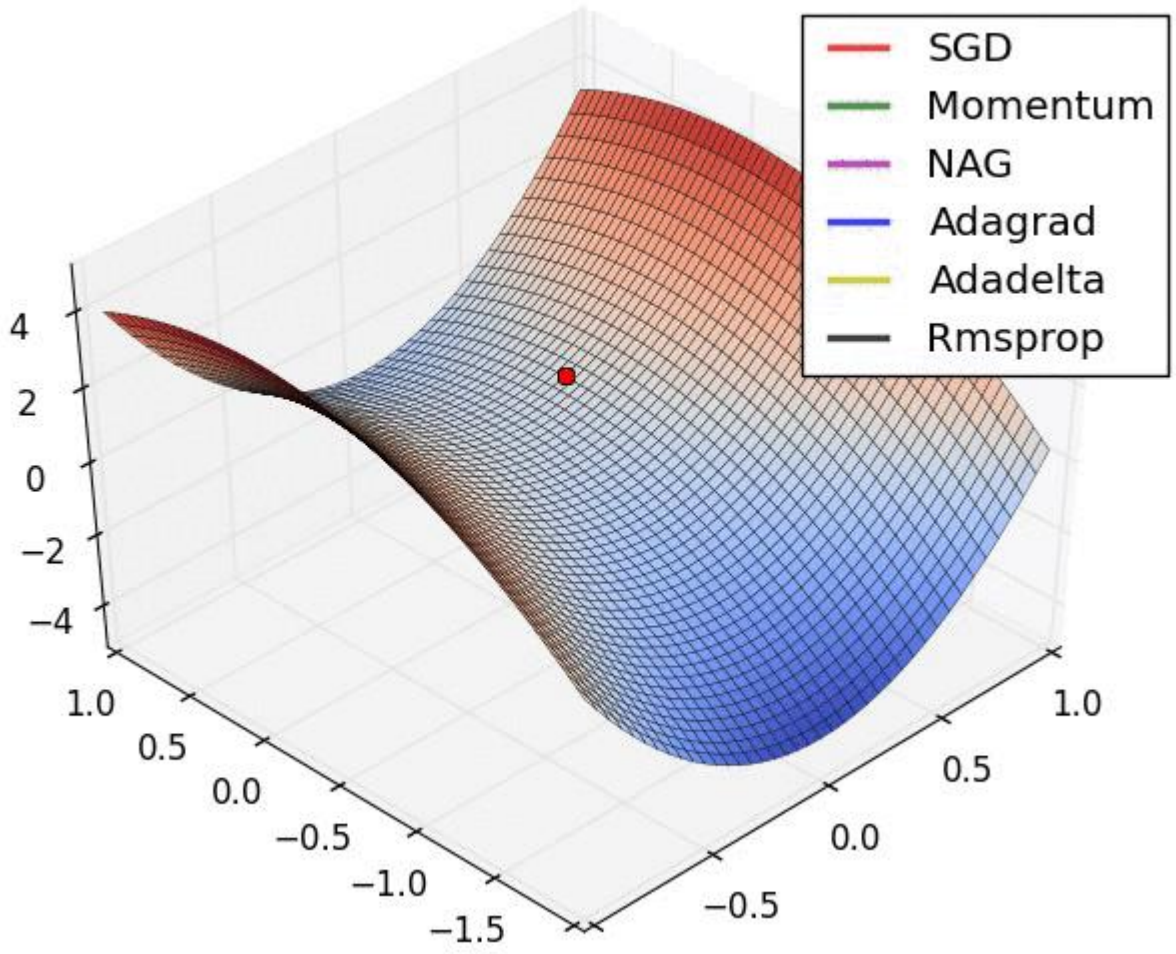


SGD optimization on loss surface contours



http://ruder.io/optimizing-gradient-descent/

SGD optimization on saddle point



http://ruder.io/optimizing-gradient-descent/

AdaGrad

- 자주 나타나는 기울기의 성분보다 드물게 나타나는 기울기 성분을 더 중시해서 파라미터를 업데이트
- 자주 나오는 파라미터 → learning rate 적게
  - 드믈게 나오는 파라미터로는 크게
- 오차함수의 기울기: $\boldsymbol{g_t} \equiv \nabla \boldsymbol{E_t}$
- $g_{t,i}$: 기울기 함수의 벡터 성분
- 업데이트 양의 성분:

$$-\frac{\epsilon}{\sqrt{\sum_{t'=1}^{t} g_{t',i}^2}} g_{t,i}$$

$\boldsymbol{\epsilon}$: learning rate

The AdaGrad algorithm

Global learning rate ε

Initial parameter θ

Small constant δ, (10^{-7})

$\boldsymbol{x}^{(i)} = [\, x_0, x_1, \ldots, x_N]$

$\boldsymbol{\theta} = [\, w_0, w_1, \ldots, w_N]$

$\Delta\boldsymbol{\theta} = [\Delta w_0, \Delta w_1, \ldots, \Delta w_N]$

$r = [\, r_0, r_1, \ldots, r_N]$

Initialize gradient accumulation variable $r = 0$

while stopping criterion not met **do**

Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)} \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $y^{(i)}$.

Compute gradient: $g \leftarrow 1/m \nabla_{\theta i} L(f(x^{(i)}; \theta), y^{(i)})$

Accumulate squared gradient: $\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$

Compute update: $\Delta\boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta + \sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$

Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$

end while

(Division and square root applied element-wise)

$\boldsymbol{g} \leftarrow 1/m \nabla_{\theta} L =$
$[\partial L/\partial w_0, \ldots \partial L/\partial w_1, \ldots, \partial L/\partial w_N]$

$f(x^{(i)}; \theta)$ = network output

$L(f(x^{(i)}; \theta), y^{(i)})$: Loss

Ex) $L = \frac{1}{2}\sum_i (t_i - O_i)^2$

⊙: 벡터의 성분별 곱

$r \leftarrow \rho r + (1 - \rho) g \odot g$

rmsprop

Adam (adaptive moments)

출처 https://www.deeplearningbook.org/

Rmsprop (Root Mean Square Propagation)

Require: Global learning rate ϵ, decay rate ρ, momentum coefficient α.
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.
Initialize accumulation variable $\boldsymbol{r} = \boldsymbol{0}$
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
Compute interim update: $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$
Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\tilde{\boldsymbol{\theta}}} \sum_i L(f(\boldsymbol{x}^{(i)}; \tilde{\boldsymbol{\theta}}), \boldsymbol{y}^{(i)})$
Accumulate gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g}$
Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \frac{\epsilon}{\sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$. ($\frac{1}{\sqrt{\boldsymbol{r}}}$ applied element-wise)
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
end while

- AdaGrad 의 문제: 시간이 흐를수록 update 양이 작아짐.

→ exponentially decaying average to discard history from the extreme past

출처 https://www.deeplearningbook.org/

The Adam algorithm

Require: Step size ϵ (Suggested default: 0.001)

Require: Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1)$. (Suggested defaults: 0.9 and 0.999 respectively)

Require: Small constant δ used for numerical stabilization. (Suggested default: 10^{-8})

Require: Initial parameters $\boldsymbol{\theta}$

Initialize 1st and 2nd moment variables $\boldsymbol{s} = \mathbf{0}$, $\boldsymbol{r} = \mathbf{0}$

Initialize time step $t = 0$

while stopping criterion not met **do**

Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.

Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

$t \leftarrow t + 1$

Update biased first moment estimate: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1)\boldsymbol{g}$

Update biased second moment estimate: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2)\boldsymbol{g} \odot \boldsymbol{g}$

Correct bias in first moment: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1 - \rho_1^t}$

Correct bias in second moment: $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1 - \rho_2^t}$

Compute update: $\Delta\boldsymbol{\theta} = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}} + \delta}$ (operations applied element-wise)

Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$

end while

https://www.deeplearningbook.org/

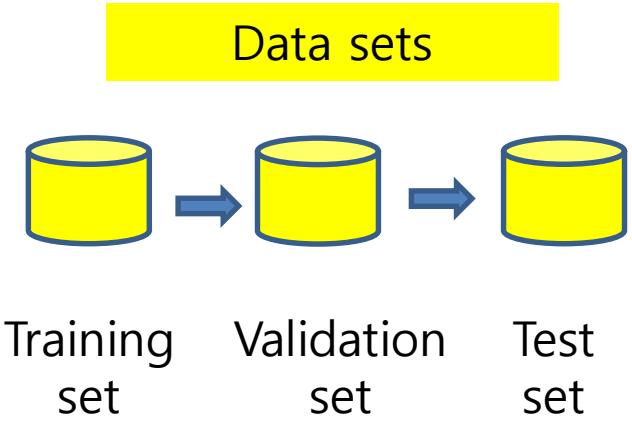
The Adam algorithm

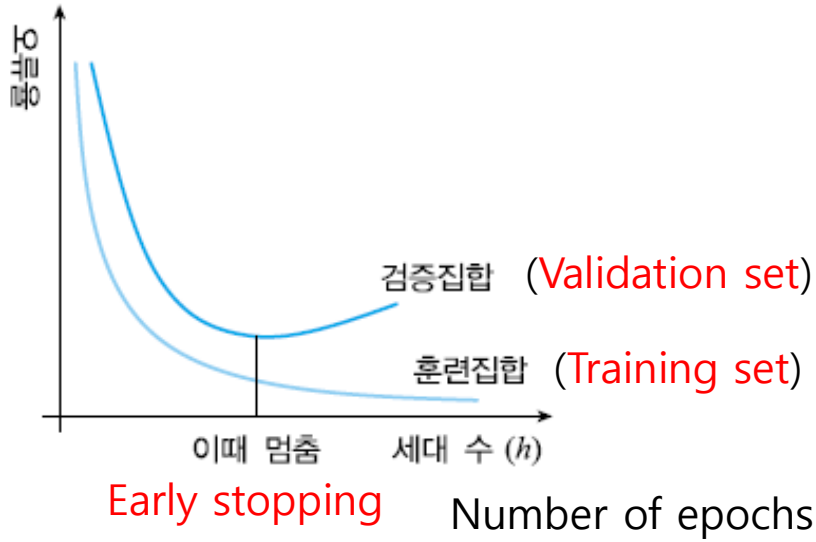
- adaptive moments.
- a variant on the combination of RMSProp and momentum.
- Bias correction for the fact that first and second moment estimates start at zero
- Adam is generally regarded as being fairly robust to the choice of hyperparameters.

Generalization

Generalization

- **Generalization :** training 에 사용되지 않은 data 에 대한 성능
- **Overfitting (과적합):**
  training error 에 비하여 generalization error 가 커지는 경우
  ← 모델 파라미터에 비하여 학습 데이터가 너무 적은 경우





Regularization

- Regularization :
  Overfitting 을 방지하도록 loss function에 complexity penalty 추가
- Weight decay
  - L2 Regularization
    - 목적 함수 J($\boldsymbol{w}$) = MSE + $\lambda \boldsymbol{w}^{\top}\boldsymbol{w}$ ← *regularizer*
    - λ : controls the strength of our preference for smaller weights.
  - L1 regularization (LASSO)
    - 목적 함수 J($\boldsymbol{w}$) = MSE + $\lambda ||\boldsymbol{w}||_1$

$$\| w \|_1 = \sum_{j=1}^{D} w_j$$

Initialization

- **he_normal**: initialization scaled by fan_in (He et al., 2014)

$$Var(W) = \frac{1}{n_{in}}$$

  - W = the initialization distribution for the neuron in question
  - n_{in} = the number of neurons feeding into it.
  - Gaussian or uniform.

- Glorot & Bengio's paper

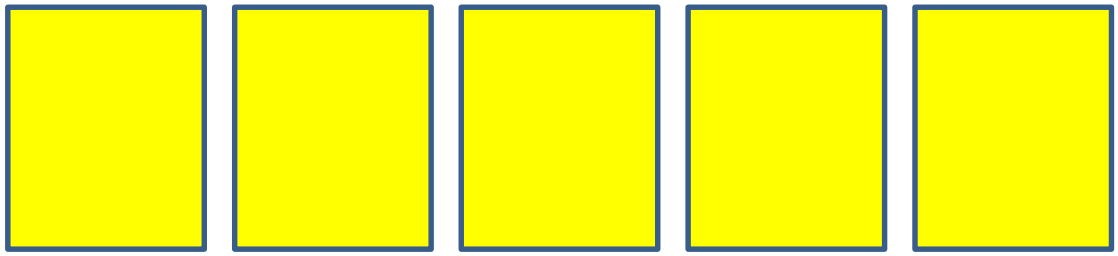
$$Var(W) = \frac{2}{n_{in} + n_{out}}$$

Model Ensembles

1. Train multiple independent models
2. At test time average their results

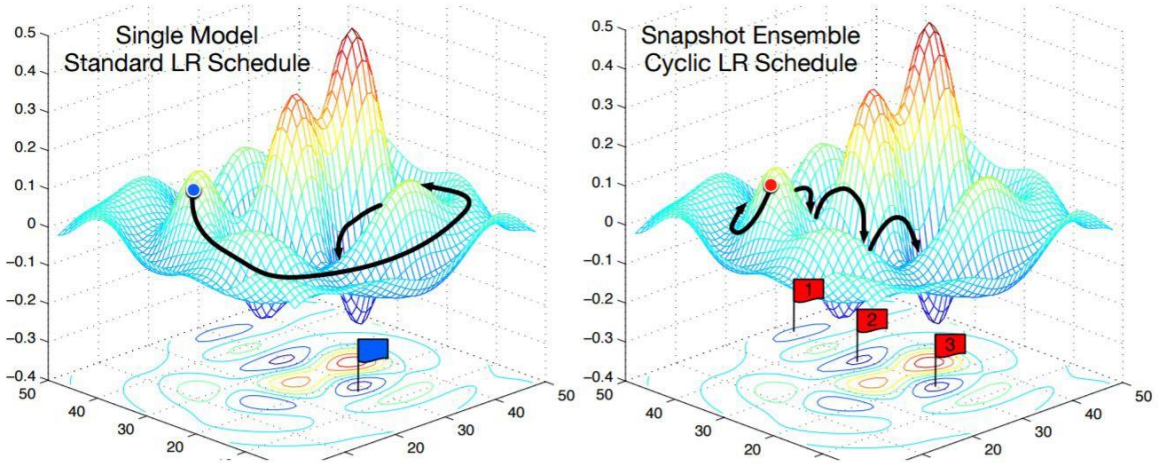
   (다양한 방법으로 조합)

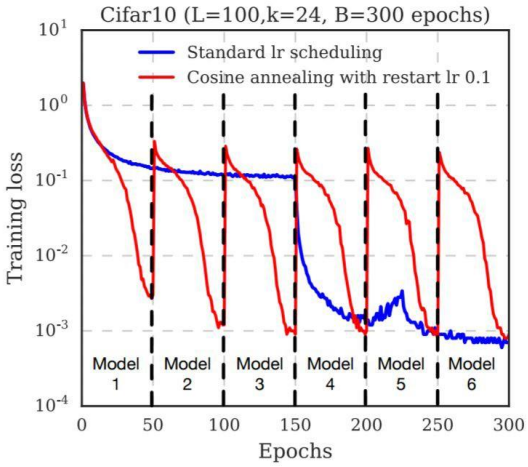
→ 큰 성능 향상은 없지만 거의 대부분 조금씩 (2%) 향상됨.



Model Ensembles: new idea (2017)

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

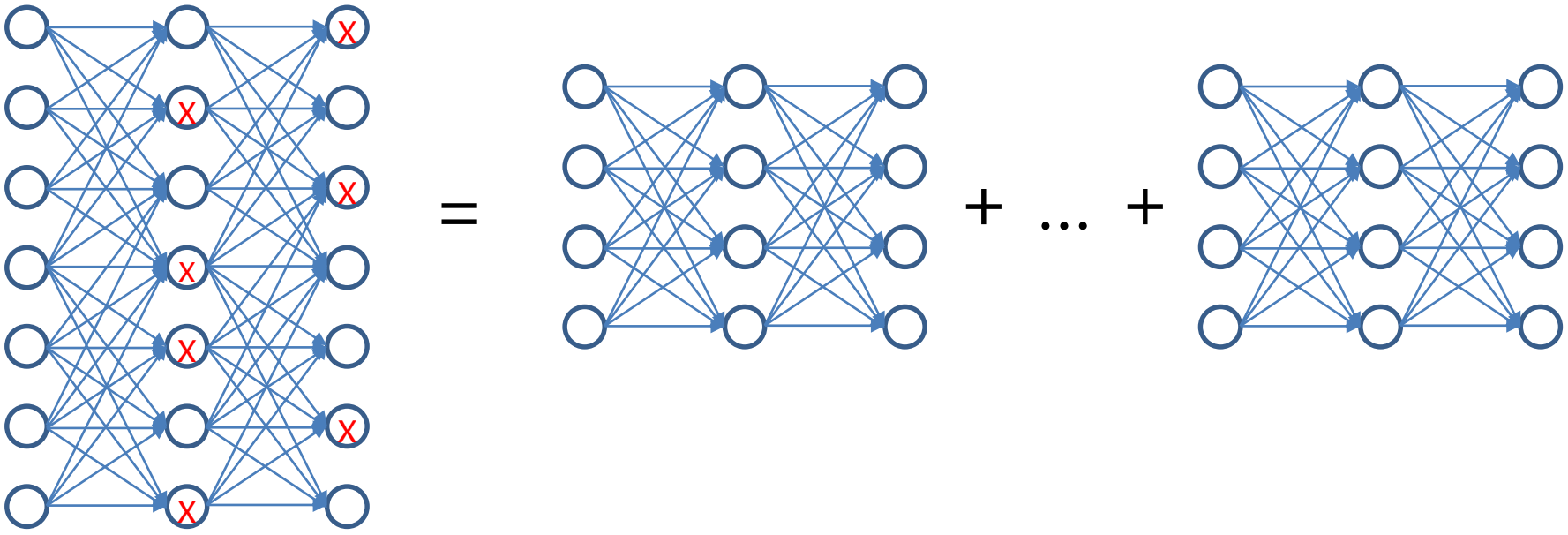




- Learning rate을 크게했다 작게했다 반복.

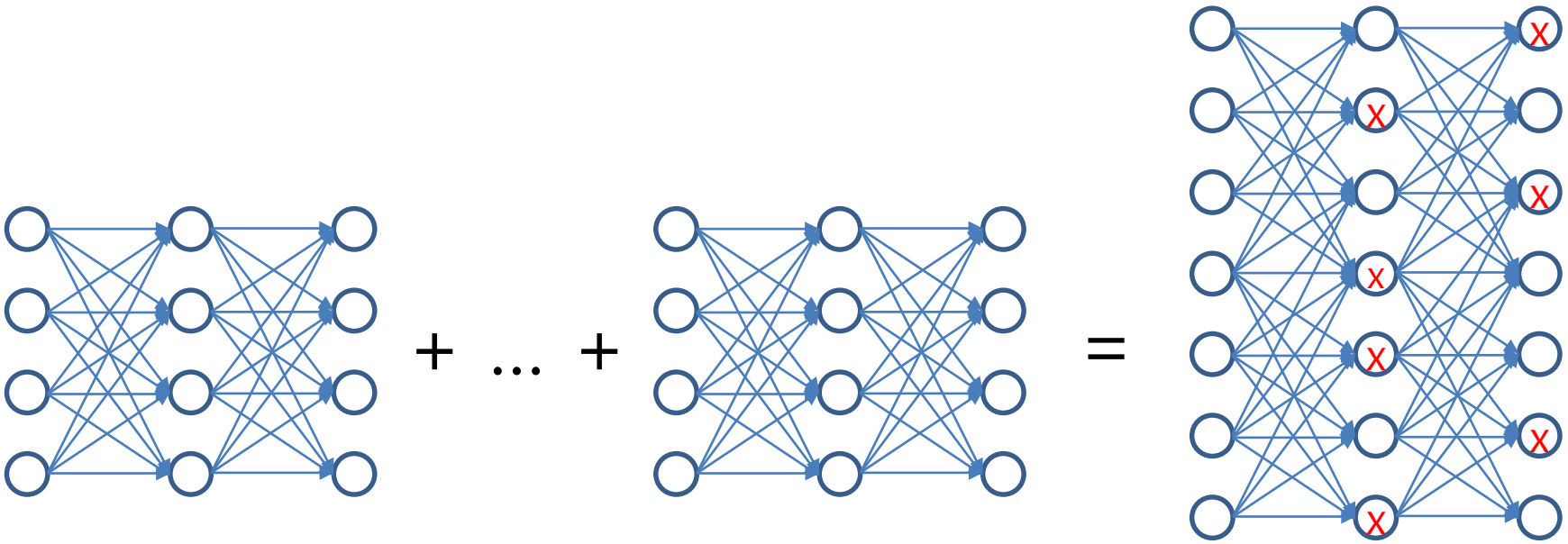
Dropout

- 문제: Overfitting 해결
- Dropout : Training example 마다 일부 hidden unit 을 없는 것으로 계산
- 다층 신경망의 유닛 중 일부를 확률적으로 선택하여 학습
- Randomly, 확률 $p = 0.5$ 적용
- 결국 2^n 개의 서로 다른 모델을 사용하는 효과



Ensemble Learning (예: adaboost)

- 작은 model을 많이 만들어서 각 output의 조합을 사용하면 성능이 높다



ReLU 도 비슷한 효과

데이터 확장 (data augmentation)

- 훈련 데이터의 부족 → overfitting 의 원인
- Data augmentation :
  이미 확보한 샘플 데이터를 일정하게 가공하여 양을 늘임
- 예
  - 이미지의 평행이동, 대칭이동, 회전 등의 기하학적 변환
  - 픽셀의 명암값, 색에 변동을 가한 결과 이미지
  - 가우스 분포를 따르는 랜덤 노이즈 추가

Summary : Neural networks 설계 고려사항

- Network topology : 어떤 모양?
  - Feed forward, feed backward
- Activation function : 출력의 형태?
  - linear or non-linear?
- Objectives : 훈련의 목적?
  - Classification or regression ?

  = Loss function, Error
- Optimizers : 훈련의 방법?
  - Weights update
- Regularization : overfitting 을 어떻게 방지할까?