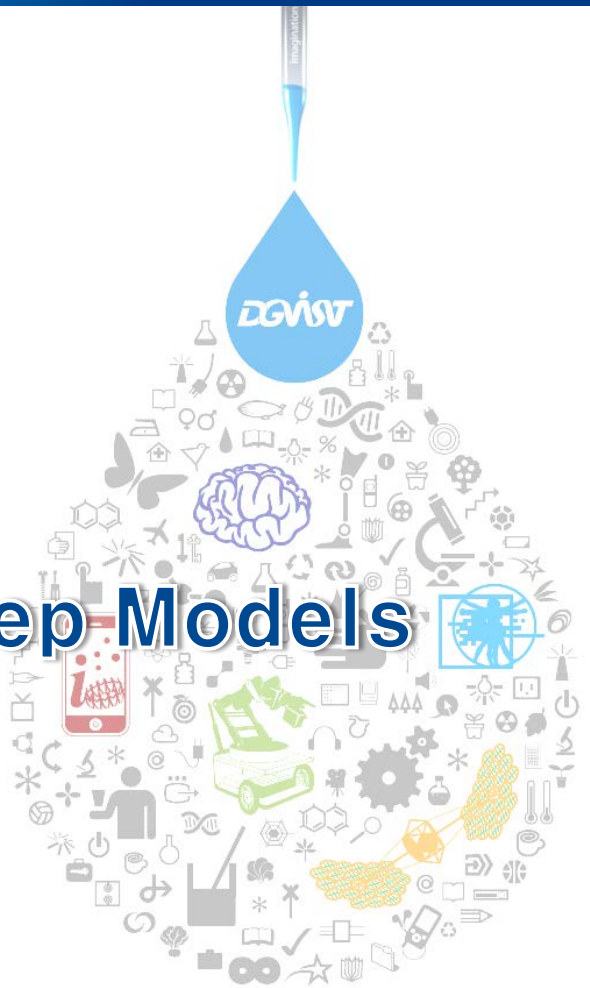
InfoLab



Chapter 8.
Optimization for Training Deep Models

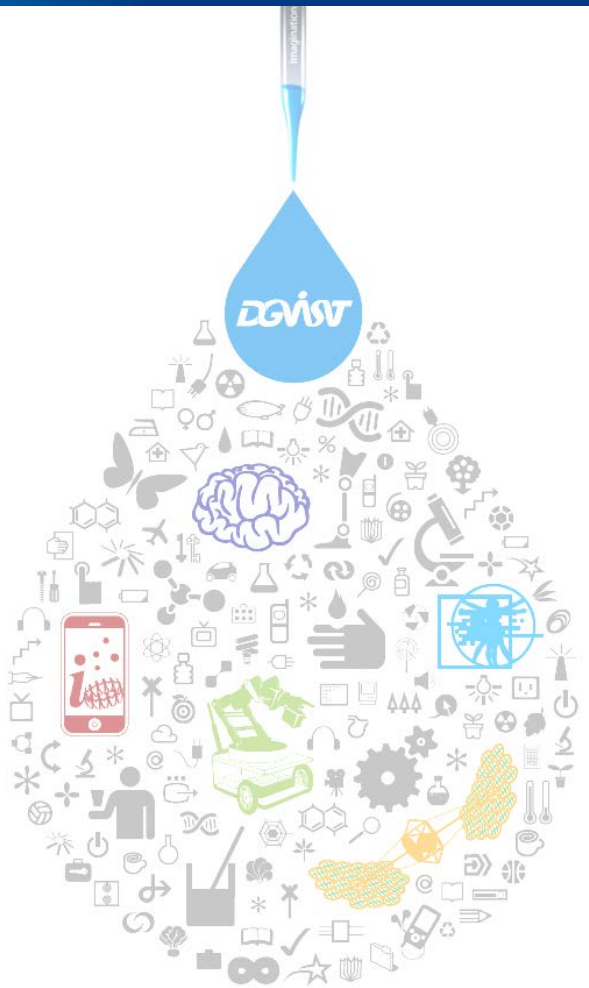
2017-08-31

Jinwook Kim

Contents

InfoLab

How Learning Differs from Pure Optimization



Empirical Risk Minimization

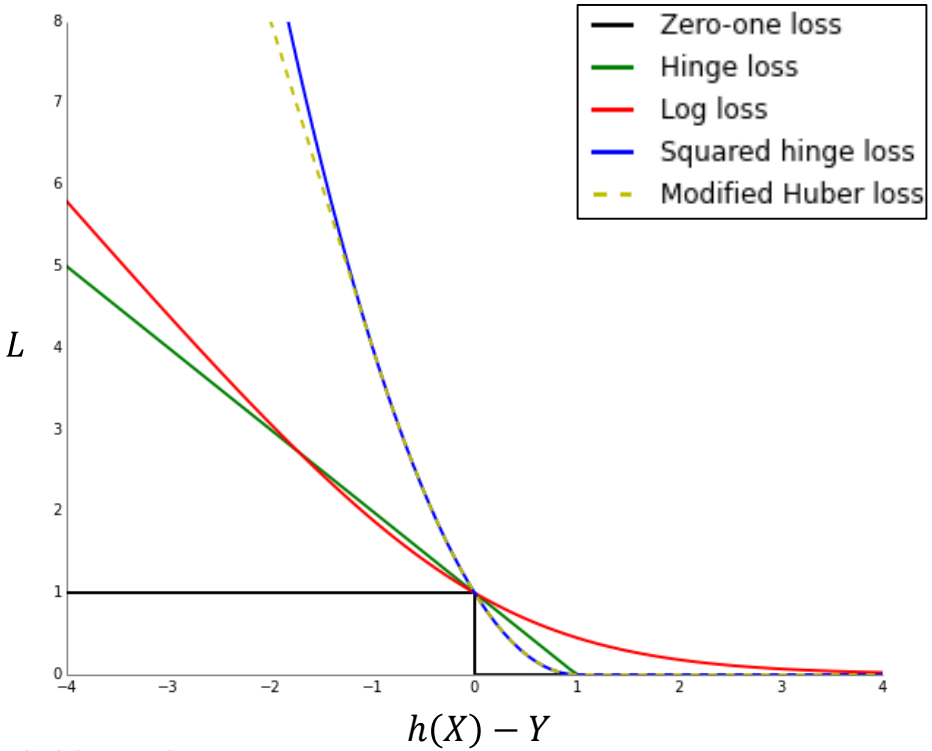
- **Machine learning usually acts indirectly**
- **The goal of learning is to reduce the expected generalization error (risk)**
- **However, learning algorithms reduce cost functions (empirical risk)**
  - minimizing the expected loss on the **training dataset**
  - in the hope that the indirect optimization will improve the performance

Surrogate Loss Functions

- **Sometimes the loss function is difficult to be optimized**
  - e.g., 0-1 loss

$$L(h(\boldsymbol{x}), Y) = \begin{cases} 1 & h(\boldsymbol{x}) < Y \\ 0 & \text{otherwise} \end{cases}$$

- **In the case, we can use a surrogate loss for the optimization**
  - differentiable
  - improve robustness



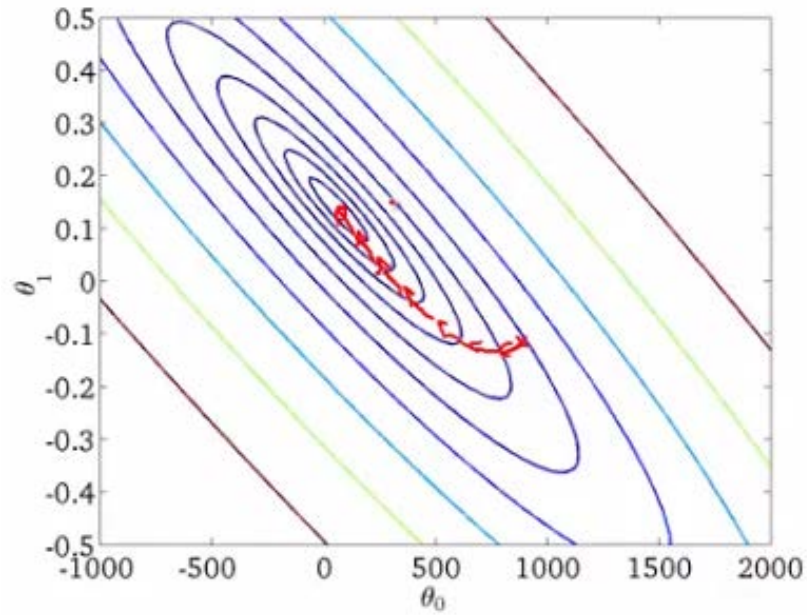
P. L. Bartlett, M. I. Jordan, and J. D. McAuliffe, “Convexity , Classification , and Risk Bounds,” J. Am. Stat. Assoc., pp. 1–36, 2003.

Minibatch Algorithms

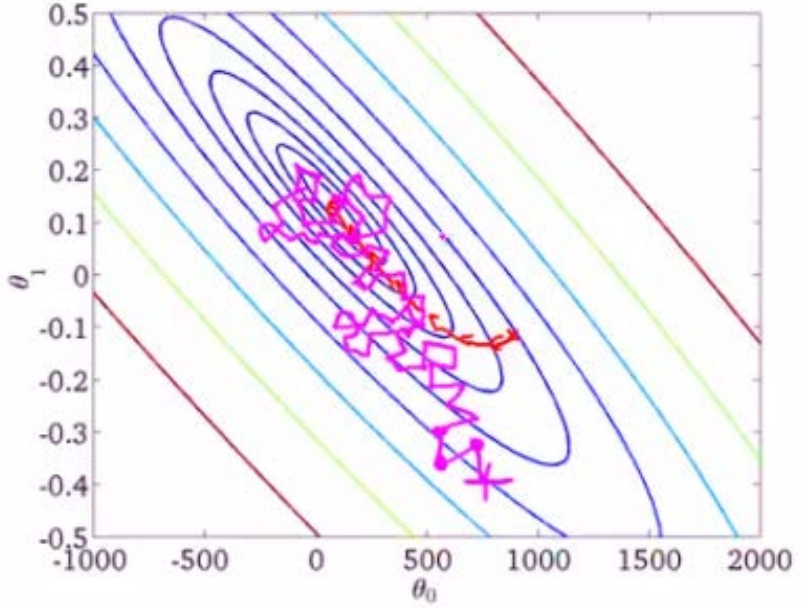
- **Optimization in machine learning typically computes loss and updates parameters iteratively**
  - e.g., gradient descent
- **Using the entire training set (batch algorithm) per every iteration is computationally infeasible**
- **Minibatch algorithm uses only b examples in each iteration**
  - m: the size of entire training examples
  - b: minibatch size ($1 \leq b < m$)

Batch vs. Minibatch

- Each iteration in minibatch algorithm may have poor optimization performance than batch algorithm
- However, after many iterations, the minibatch algorithm generally converges to optimal state



(a) Batch gradient descent optimization

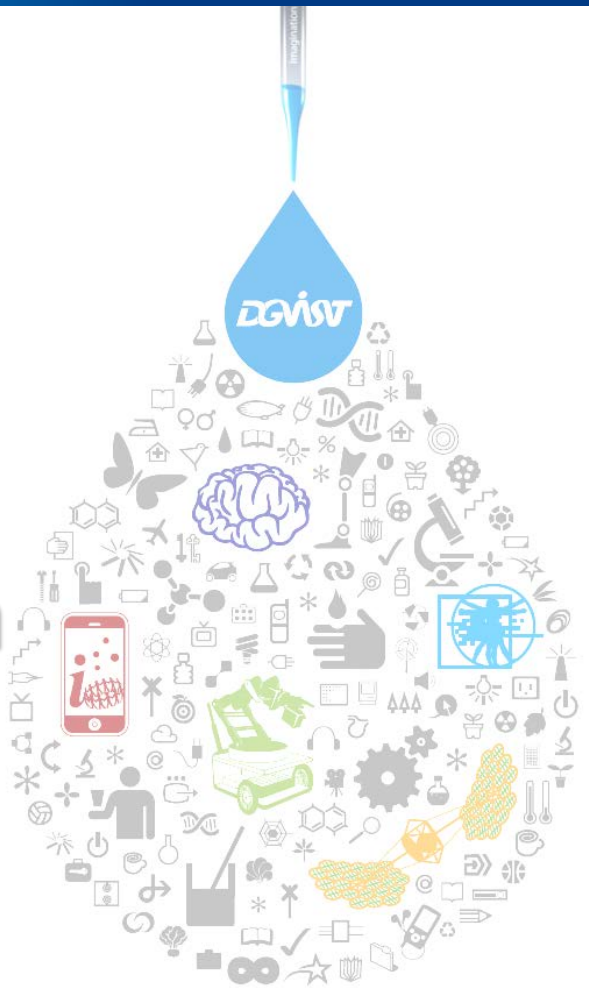


(b) Minibatch gradient descent optimization

http://www.holehouse.org/mlclass/17_Large_Scale_Machine_Learning.html

InfoLab

Challenges in Neural Network Optimization



DGIST 대구경북과학기술원
Daegu Gyeongbuk Institute of Science & Technology

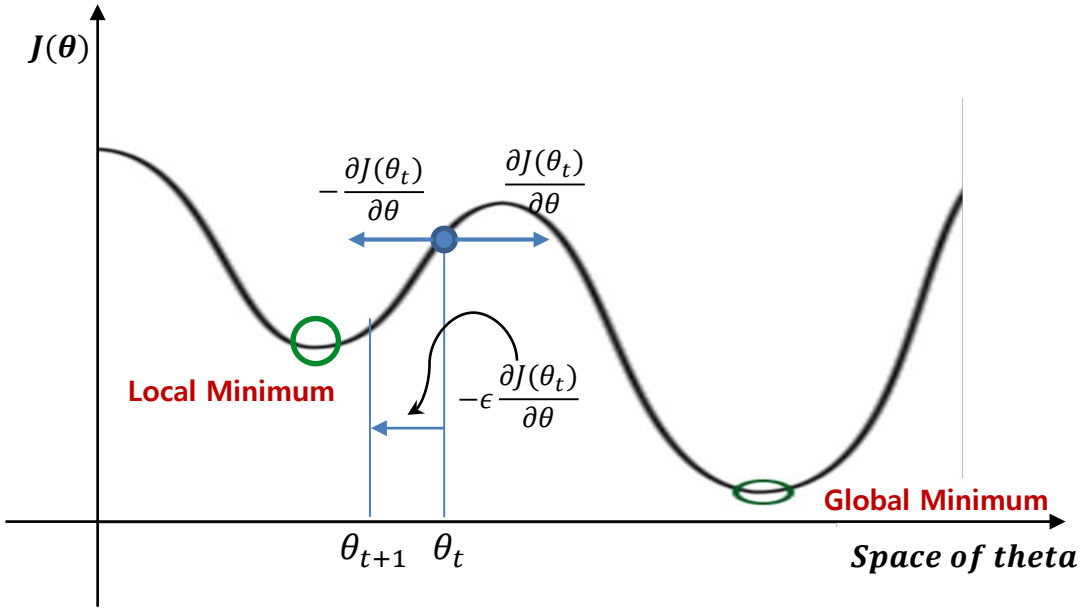
Limitation of Gradient Descent

- **Issue of local minimum**

$$Objective: \ \ min \ J(\theta)$$

$$\theta_{t+1} = \theta_t - \epsilon \frac{\partial J(\theta)}{\partial \theta}$$

$(\epsilon : Learning\ rate)$



- **If the starting point for gradient descent was chosen inappropriately, cannot reach global minimum**

Saddle Point

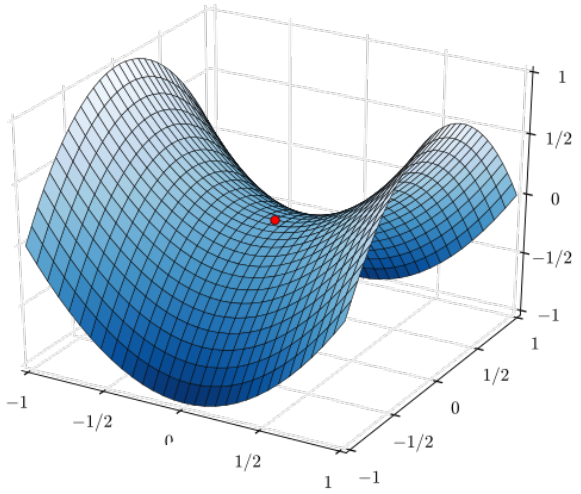
- **For many high-dimensional non-convex functions, local minima are in fact rare cases**
  - local minima should satisfy,

n: # of parameters in model

$$\frac{\partial J}{\partial \theta_i} = 0 \, , i = 1, \dots n$$

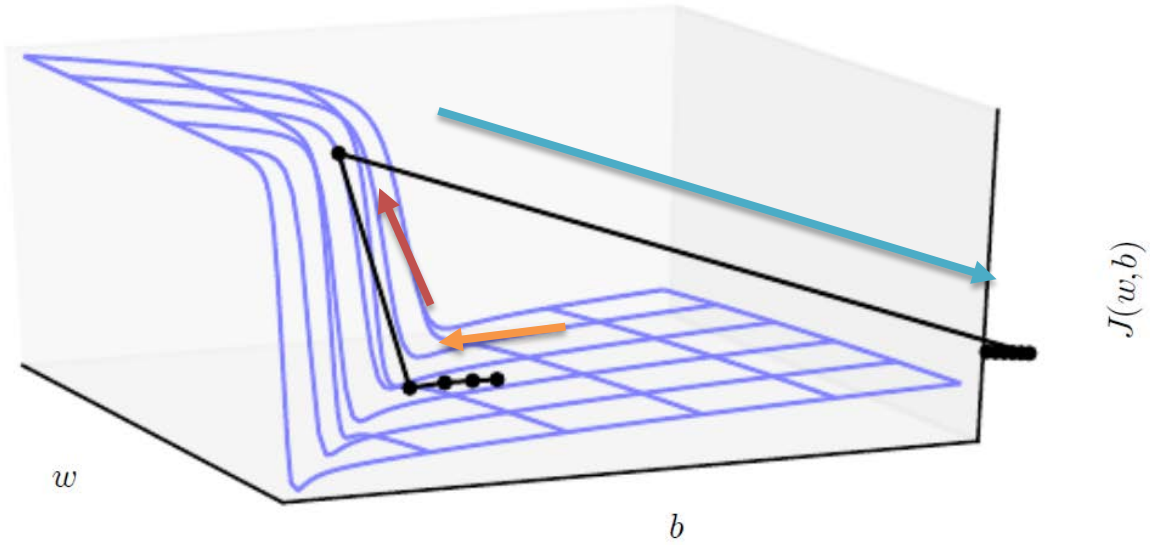
$$\frac{\partial^2 J}{\partial^2 \theta_i} > 0 \, , i = 1, \dots n$$

- **Most of zero-gradient points in deep neural networks are saddle points**



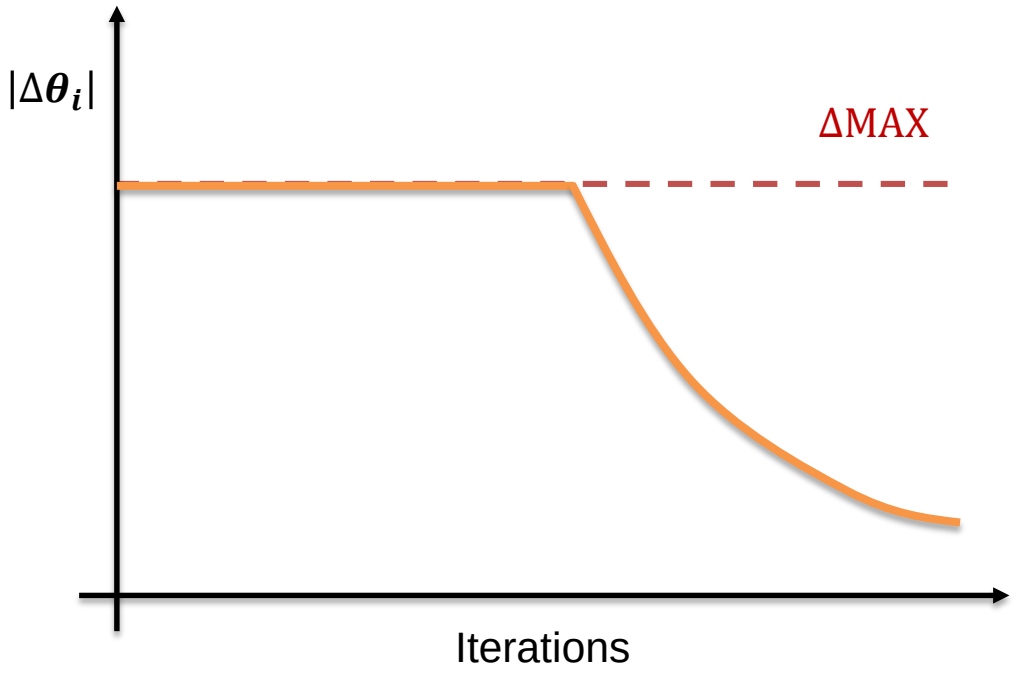
Cliffs and Exploding Gradient

- **Cliff is one common issue in the objective functions for deep neural networks**
  - ➢ resulting from the multiplication of several parameters
- **The gradient update step can move the parameter extremely far (exploding gradient)**
  - ➢ losing most of optimization works that has been done



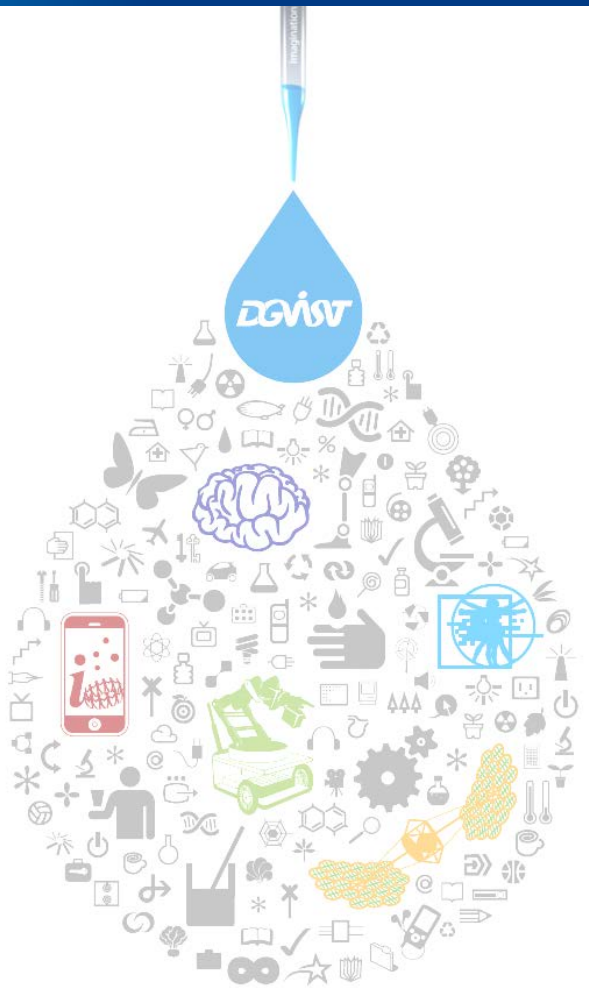
Gradient Clipping for Handling Cliffs

$$\Delta\boldsymbol{\theta_i} \leftarrow \Delta\boldsymbol{\theta_i} \frac{\Delta\mathbf{MAX}}{\boldsymbol{max}(|\Delta\boldsymbol{\theta_i}|, \ \Delta\mathbf{MAX})}$$



InfoLab

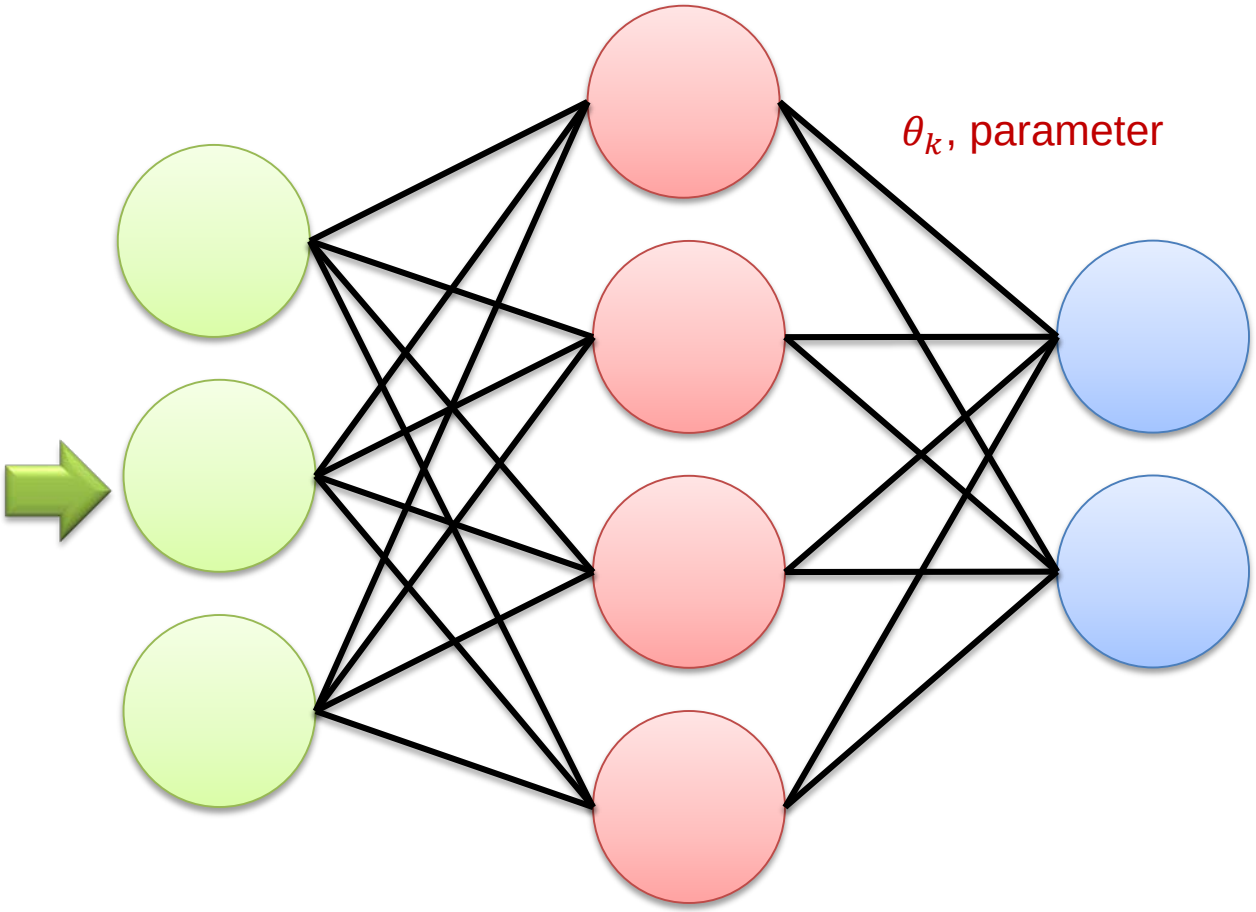
Basic Algorithms



DGIST 대구경북과학기술원
Daegu Gyeongbuk Institute of Science & Technology

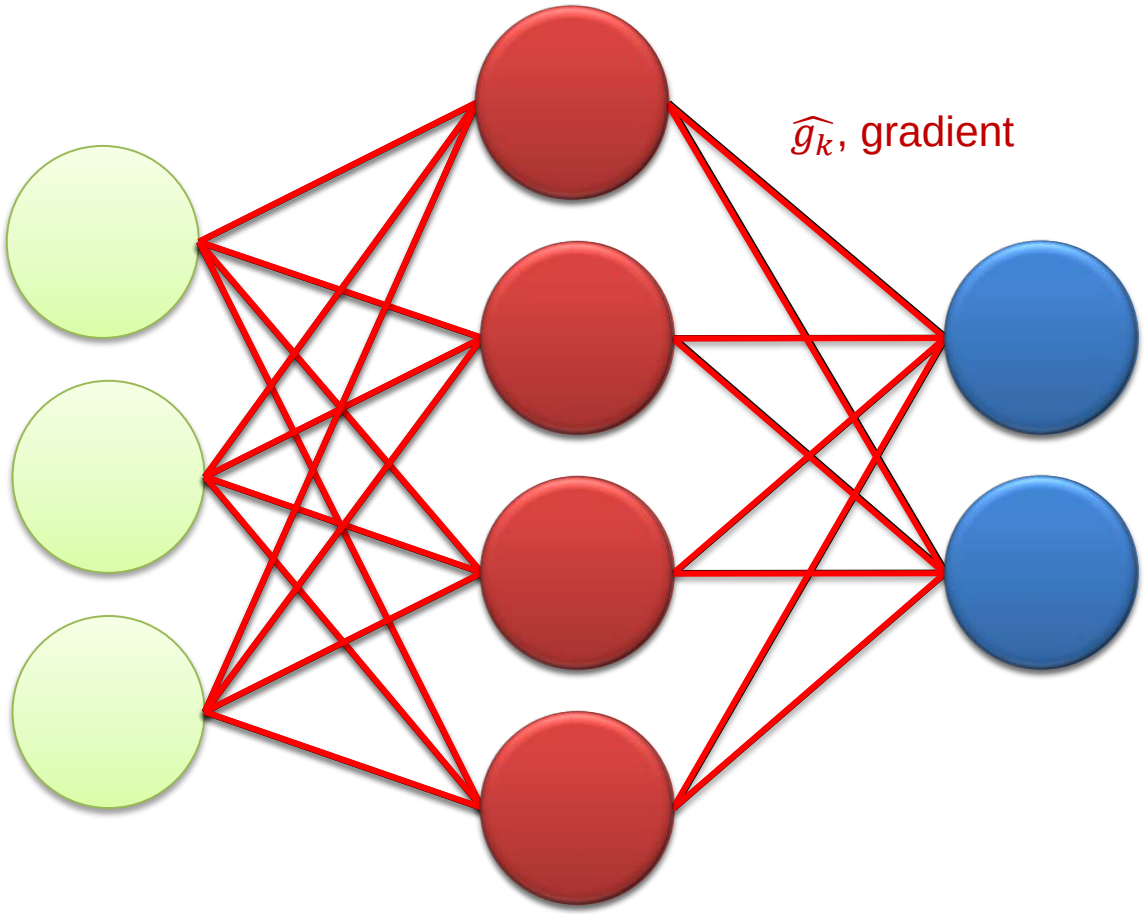
Forward Propagation (Review)

Data
(minibatch)



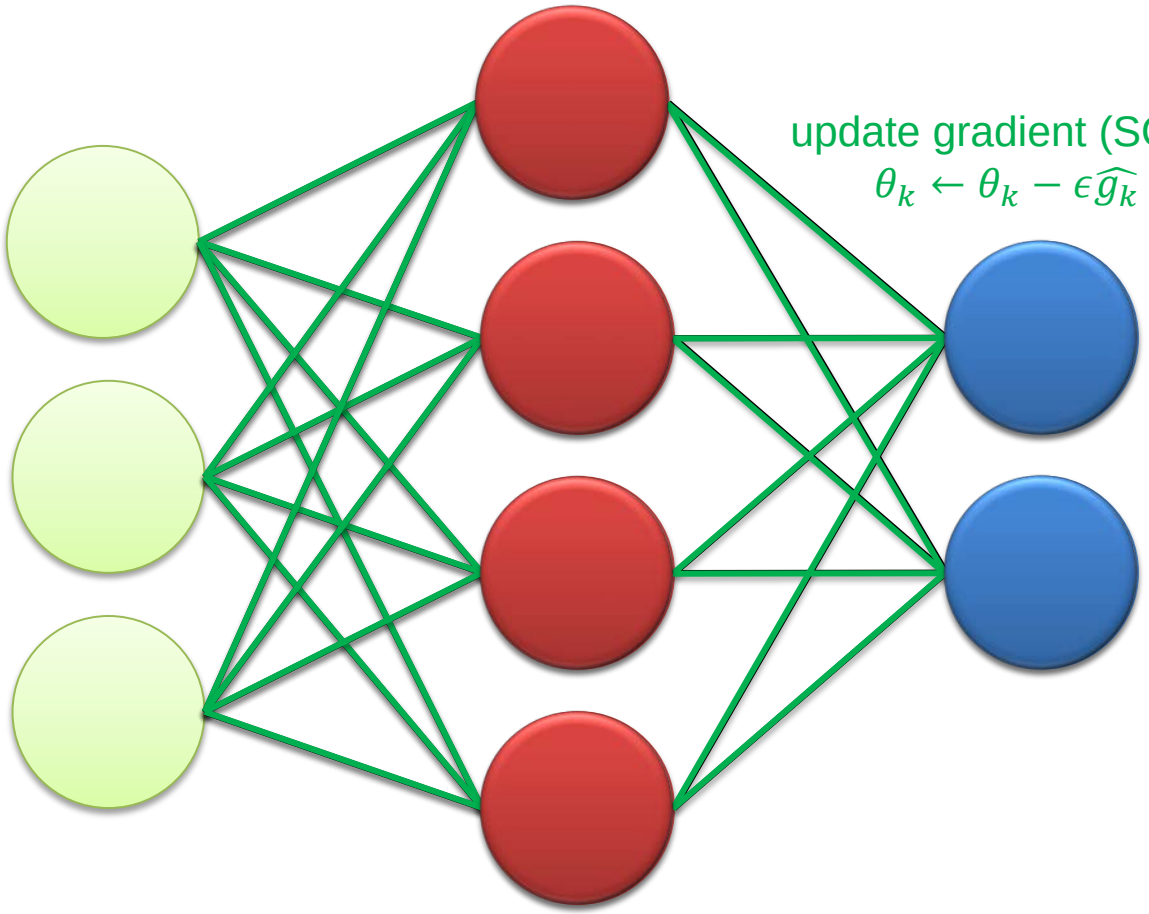
Back Propagation (Review)

Data
(minibatch)



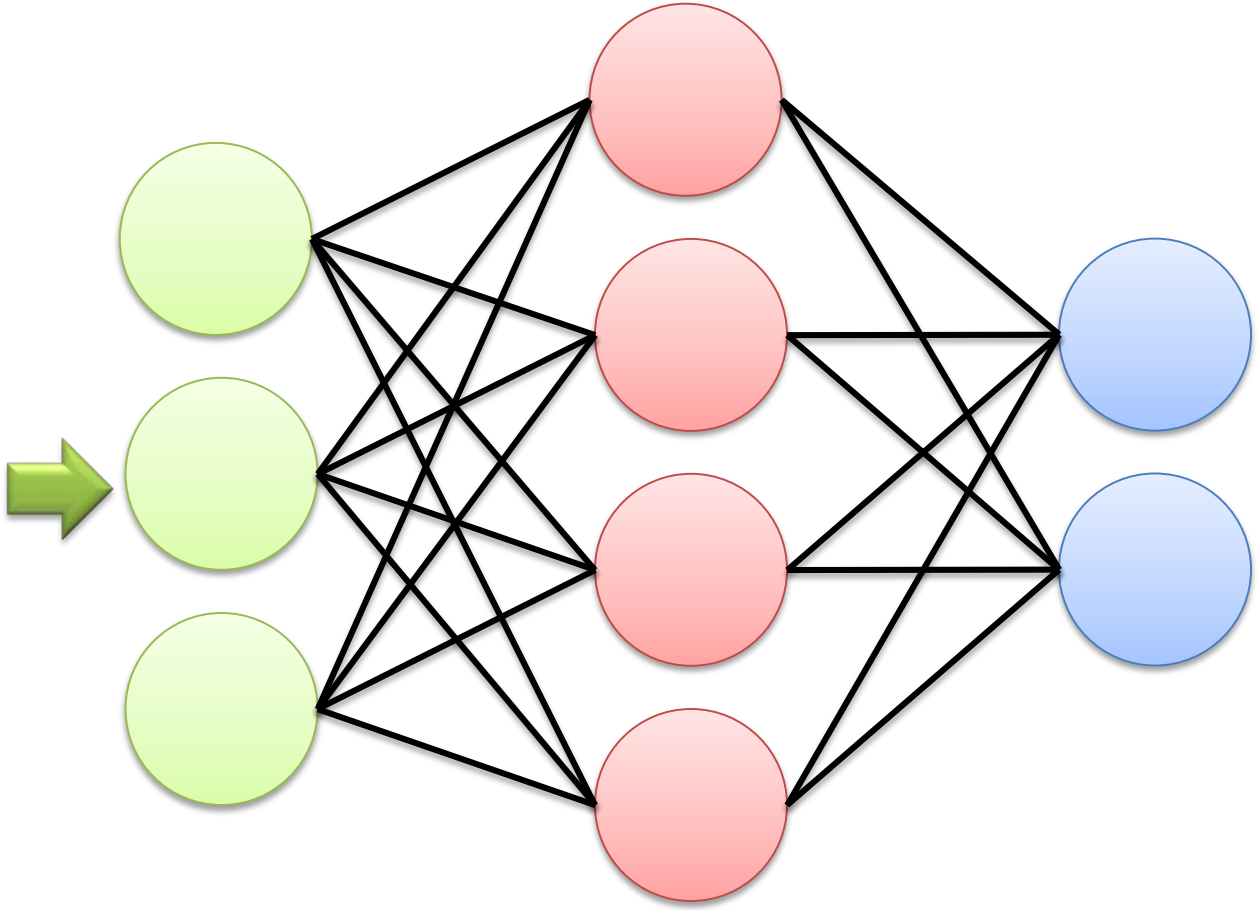
Parameter Update (Review)

Data
(minibatch)



Next Iteration

Data
(minibatch)



Loss, L

Stochastic Gradient Descent (SGD)

- **Minibatch of the training set:**

  data $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ **with targets** $\{y^{(1)}, \ldots, y^{(m)}\}$

- **Gradient:**

$$\hat{\boldsymbol{g}} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_{i}^{m} L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), y^{(i)})$$

- **Apply update:**

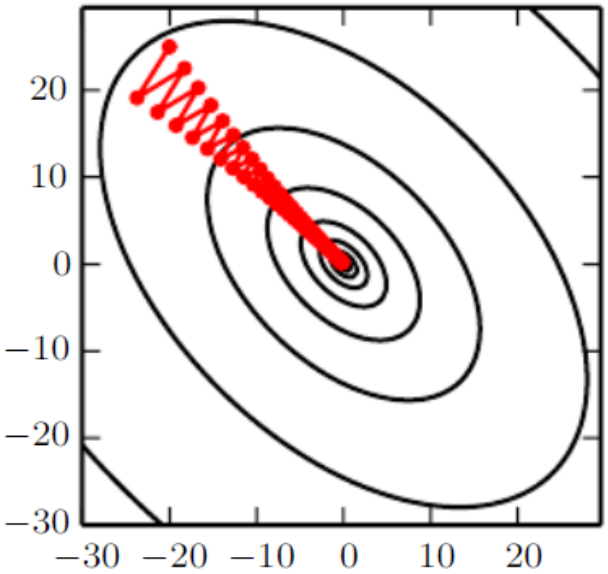
$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \hat{\boldsymbol{g}}$$

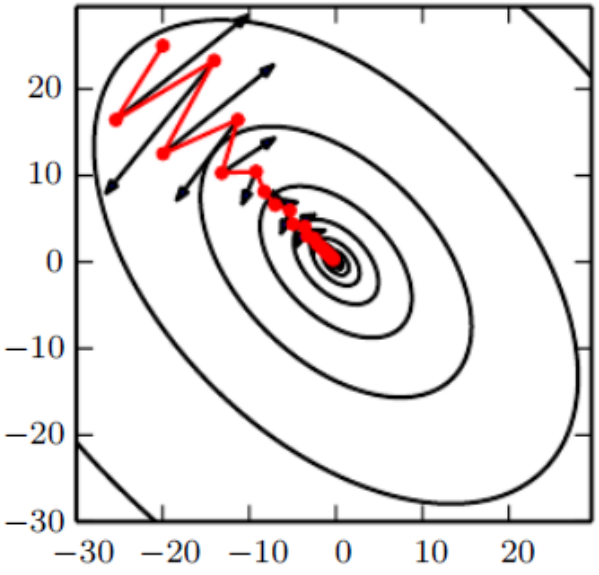
  - learning rate ϵ is a critical hyperparameter for SGD

Momentum

- **Momentum is designed to accelerate learning**
  - ➢ accumulates an exponentially decaying moving-average of past gradients
  - ➢ continues to move in their direction

- **Comparison of GD and GD with momentum:**





SGD with Momentum

- **Minibatch of the training set:**

  data $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ **with targets** $\{y^{(1)}, \ldots, y^{(m)}\}$

- **Gradient:**

$$\hat{\boldsymbol{g}} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_{i}^{m} L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), y^{(i)})$$

- **Accumulate velocity:**

$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \hat{\boldsymbol{g}}$$

  - $0 < \alpha < 1$, a hyperparameter

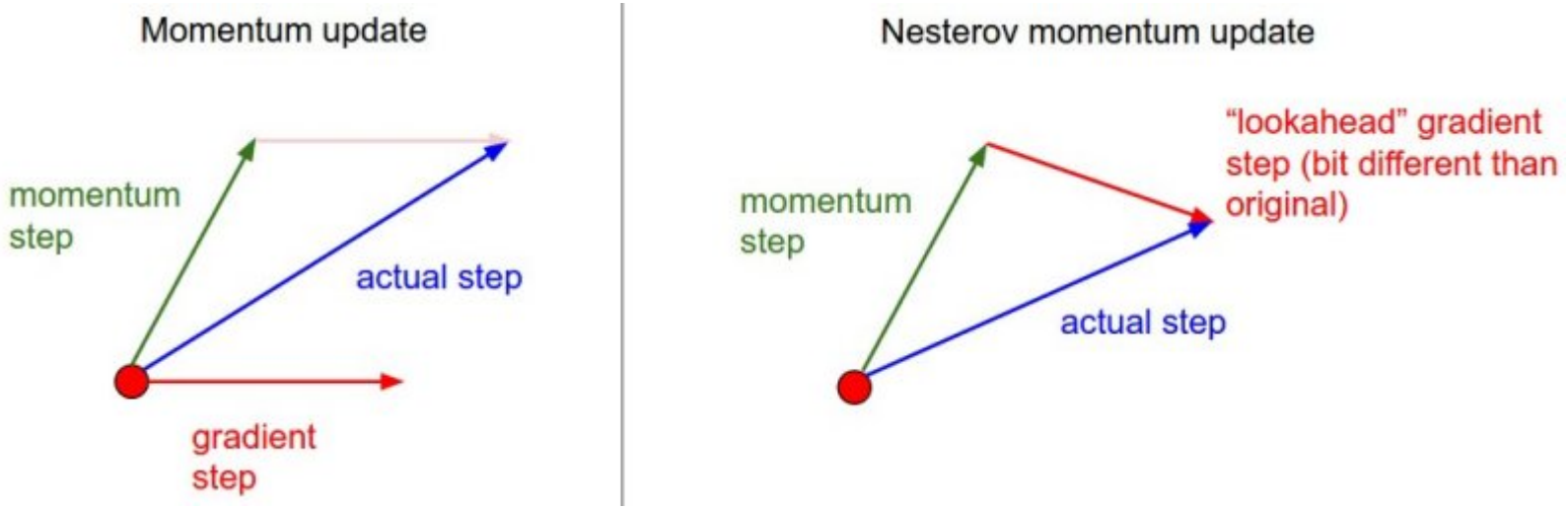
- **Apply update:**

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

Nesterov Momentum (Nesterov, 2004)

- With Nesterov momentum, the gradient is evaluated after current velocity is applied

- Comparison of momentum and Nesterov momentum:



From lecture material of CS231, Stanford Univ.

SGD with Nesterov Momentum

- **Minibatch of the training set:**

  data $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ **with targets** $\{y^{(1)}, \ldots, y^{(m)}\}$

- **Apply interim update:**

$$\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$$

- **Gradient:**

$$\hat{\boldsymbol{g}} \leftarrow \frac{1}{m} \nabla_{\tilde{\boldsymbol{\theta}}} \sum_{i}^{m} L\left(f\left(\boldsymbol{x}^{(i)}; \tilde{\boldsymbol{\theta}}\right), y^{(i)}\right)$$

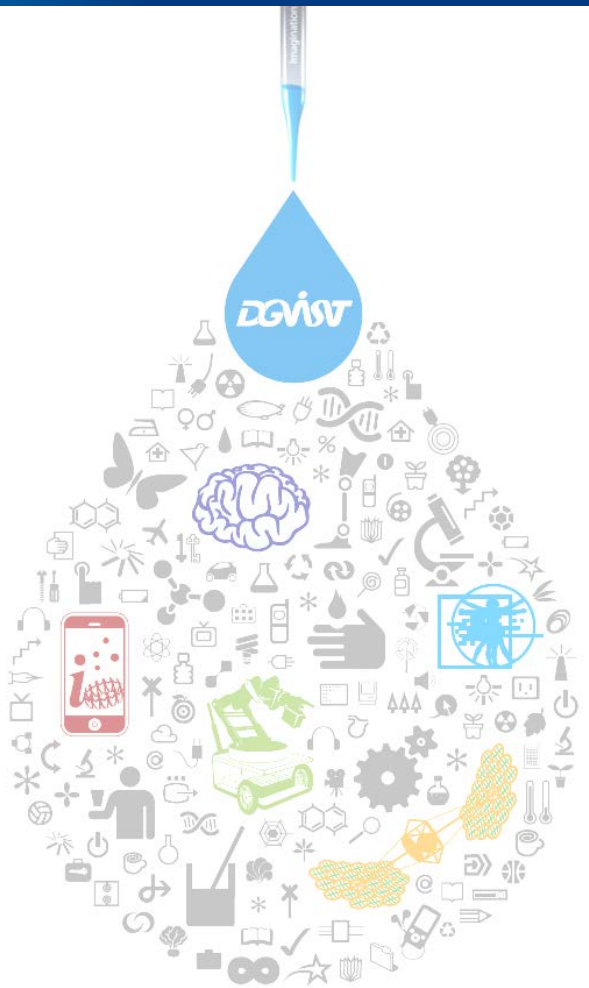
- **Velocity:**

$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \hat{\boldsymbol{g}}$$

- **Apply update:**

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

Algorithms with Adaptive Learning Rates



AdaGrad (Duchi et al., 2011)

- **Adaptive gradient (AdaGrad) is a modified SGD with per-parameter learning rate**
  - ➢ increases the learning rate for parameters having small gradient
  - ➢ decreases the learning rate for parameters having large gradient
- **Gradient:**

$$\hat{\boldsymbol{g}} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_{i}^{m} L\left(f\left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right), \boldsymbol{y}^{(i)}\right)$$

- **Accumulate squared gradients:**

$$\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$$

- **Apply update:**

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

element-wise operation

$\odot$: element-wise multiplication
r: initialized to 0
δ: small constant value (e.g., 10^{-7})

RMSProp (Hinton, 2012)

- **Root mean square propagation (RMSProp) is the modified version of AdaGrad**
  - ➢ by changing the gradient accumulation into an exponentially weighted moving average
- **Gradient:**

$$\hat{\boldsymbol{g}} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}}\sum_{i}^{m} L\left(f\left(\boldsymbol{x}^{(i)};\boldsymbol{\theta}\right), y^{(i)}\right)$$

- **Accumulate squared gradients:**

$$\boldsymbol{r} \leftarrow \rho\boldsymbol{r} + (1-\rho)\boldsymbol{g}\odot\boldsymbol{g} \qquad 0 < \rho < 1$$

- **Apply update:**

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

Adam (Kingma and Ba, 2014)

- **Adaptive moments (Adam) is the combination of RMSProp and momentum**

- **Accumulate first and second moment estimates:**

$$\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1)\boldsymbol{g}$$
$$\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2)\boldsymbol{g} \odot \boldsymbol{g}$$

- **Correct bias in first and second moments:**

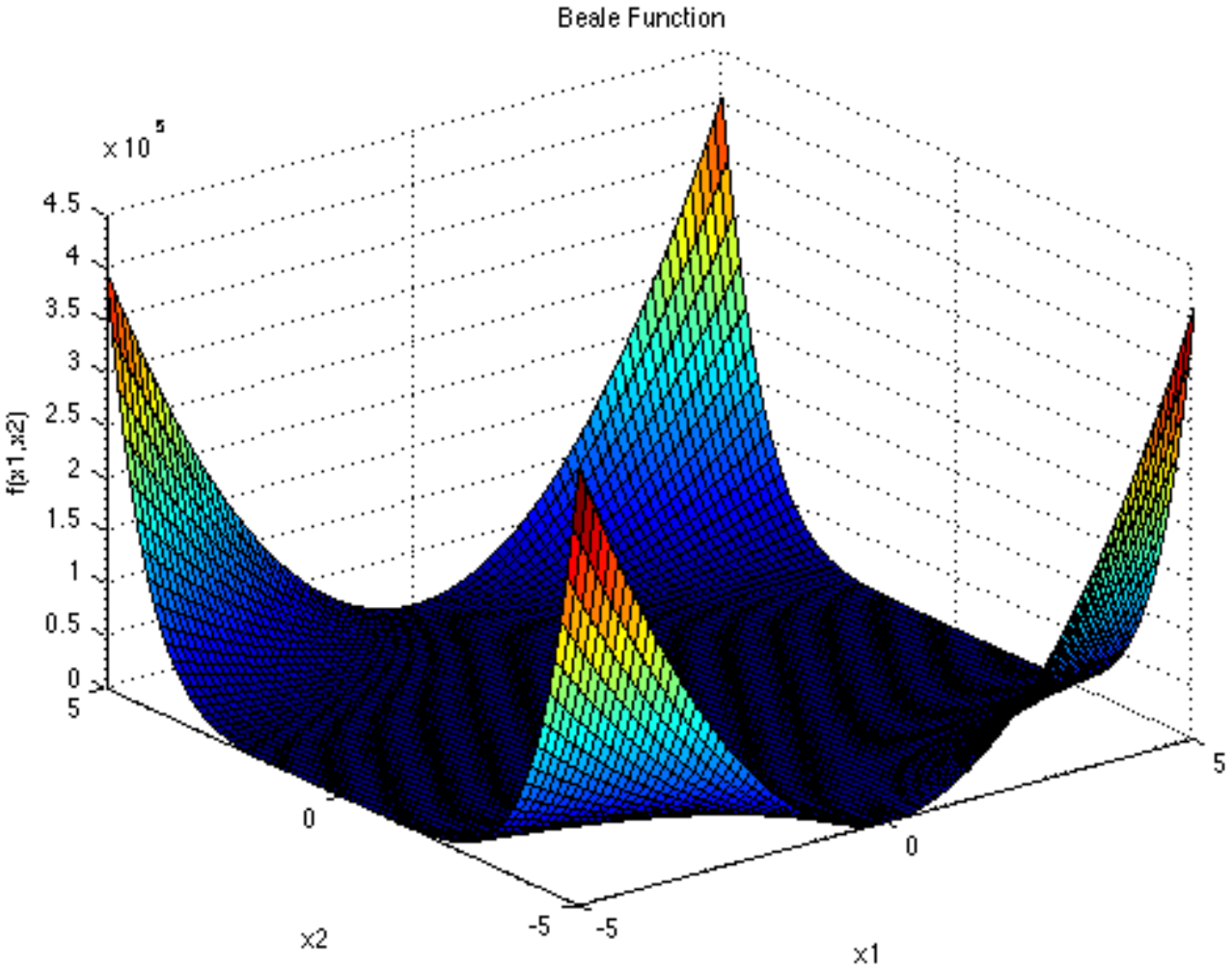
$$\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1 - \rho_1^t}, \qquad \hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1 - \rho_2^t}$$

- **Apply update:**

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}} + \delta}$$

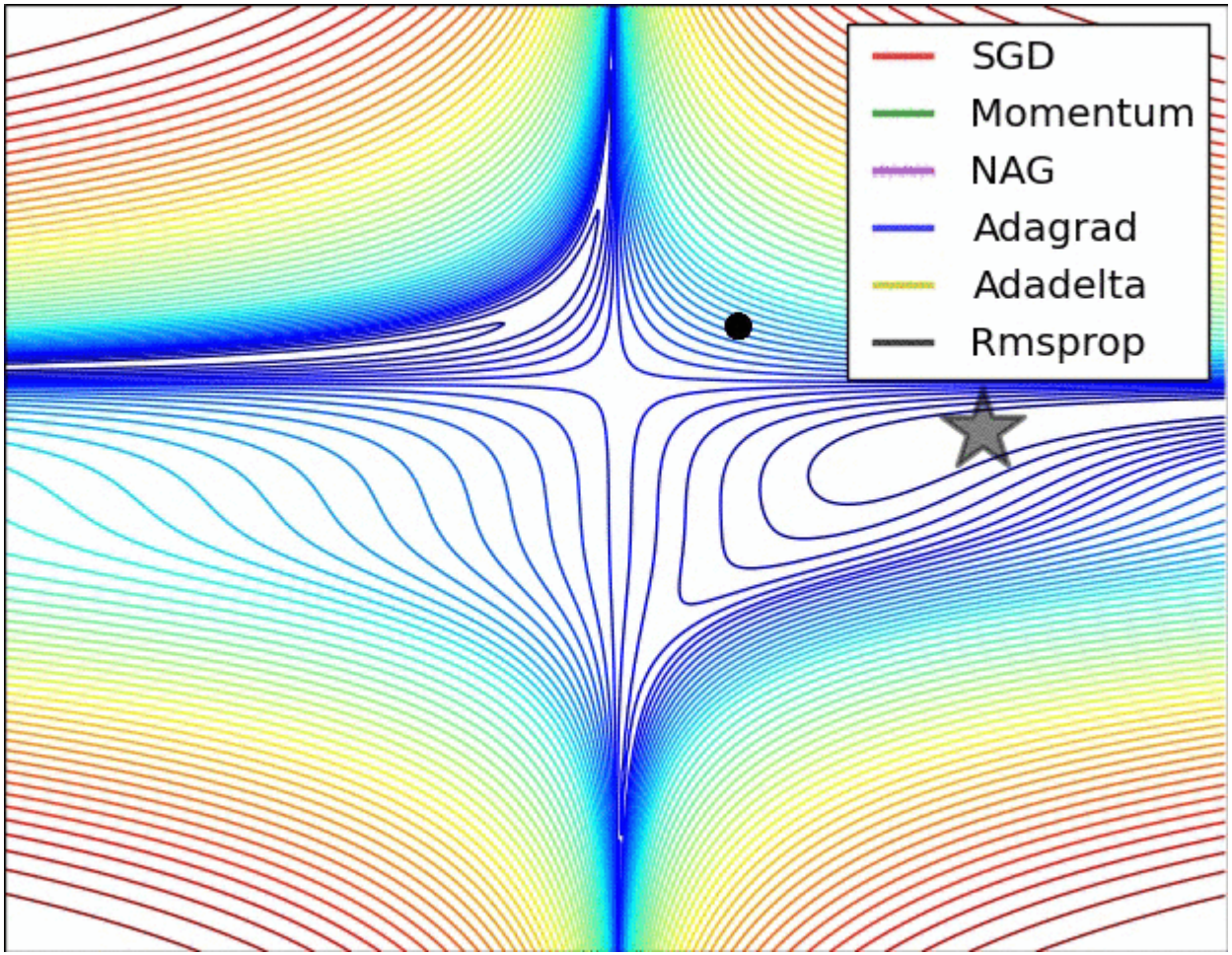
ρ_1: decay rate for 1st moment estimate (usually 0.9)
ρ_2: decay rate for 2nd moment estimate (usually 0.999)
t: current number of iteration

Example: Beale's Function



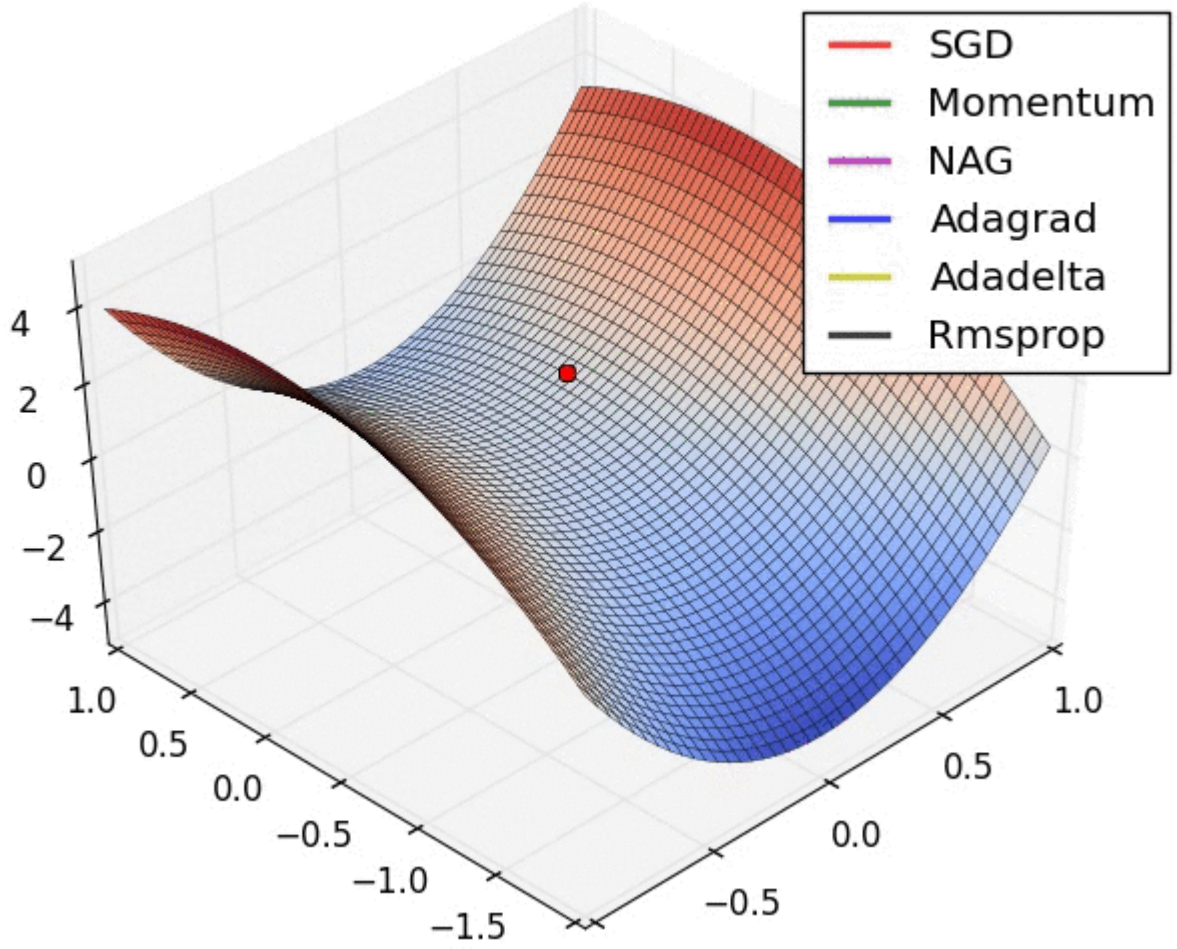
$$f(\mathbf{x}) = (1.5 - x_1 + x_1 x_2)^2 + (2.25 - x_1 + x_1 x_2^2)^2 + (2.625 - x_1 + x_1 x_2^3)^2$$

Algorithms at Beale's Function



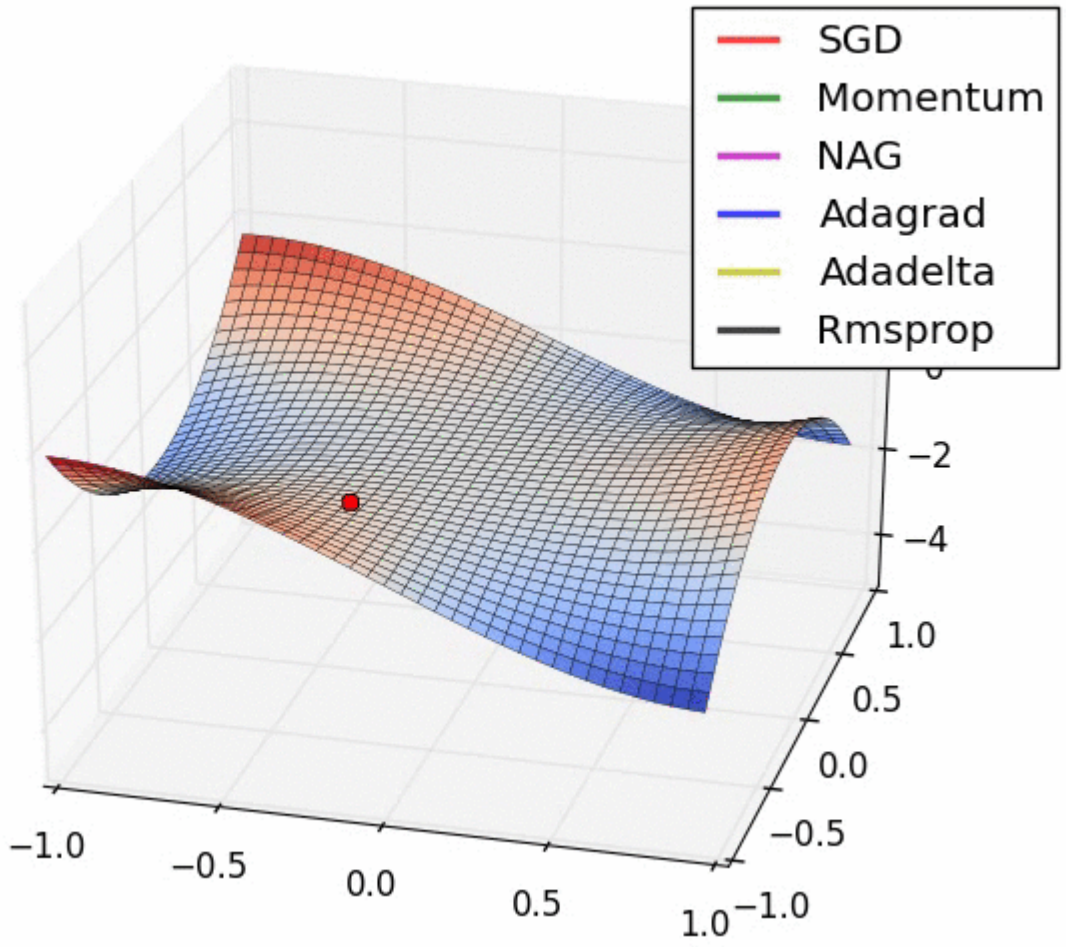
Lecture material in CS231n, “Convolutional Neural Networks for Visual Recognition”, Stanford Univ. (Image credit: Alec Radford)

Algorithms at Long Valley



Lecture material in CS231n, “Convolutional Neural Networks for Visual Recognition”, Stanford Univ. (Image credit: Alec Radford)

Algorithms at Saddle Point



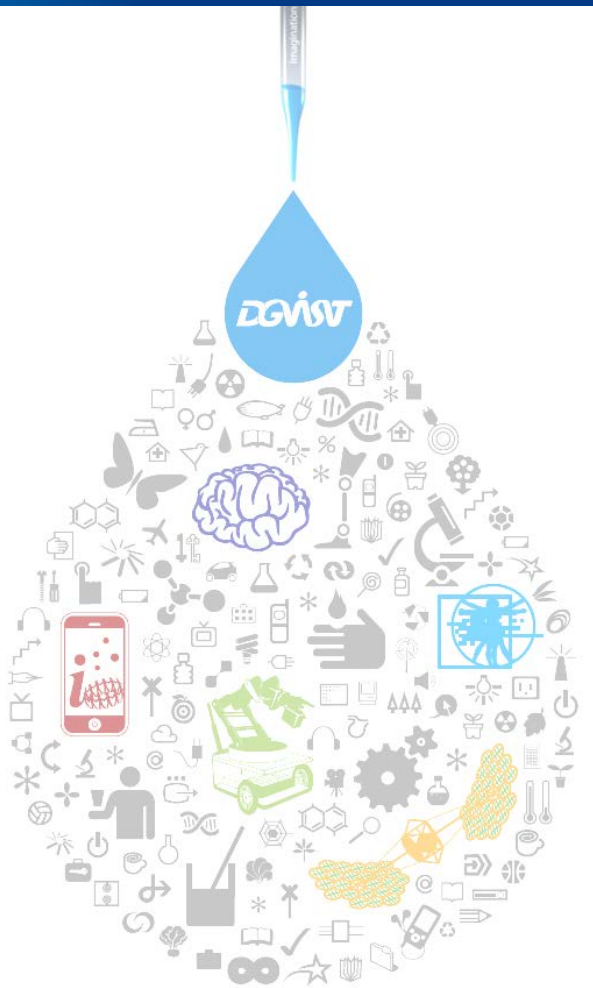
Lecture material in CS231n, “Convolutional Neural Networks for Visual Recognition”, Stanford Univ. (Image credit: Alec Radford)

Hyperparameter Setting in CAFFE

```
net: "svhn/fix2-svhn-3conv-model.prototxt"

test_iter: 100
test_interval: 1000
max_iter: 100000

lr_policy: "fixed"
base_lr: 0.001
momentum: 0.9
weight_decay: 0.004

type: "SGD"

display: 10000
snapshot: 80000
snapshot_prefix: "svhn/snapshot/fix2_svhn_3conv"
solver_mode: GPU
```

- Stochastic Gradient Descent (type: "SGD"),
- AdaDelta (type: "AdaDelta"),
- Adaptive Gradient (type: "AdaGrad"),
- Adam (type: "Adam"),
- Nesterov's Accelerated Gradient (type: "Nesterov") and
- RMSprop (type: "RMSProp")

Parameter Initialization



Initial Parameters in Deep Learning

- **Determining appropriate initial parameters for a deep model is one of the most important part of optimization**

- **Initial parameters can determine,**
  - whether the algorithm converges or not
  - whether the model converges to a point with a high or low cost
  - how quickly learning converges

- **However, the initialization strategies are still heuristic**
  - initial values are drawn randomly from a Gaussian or uniform distribution

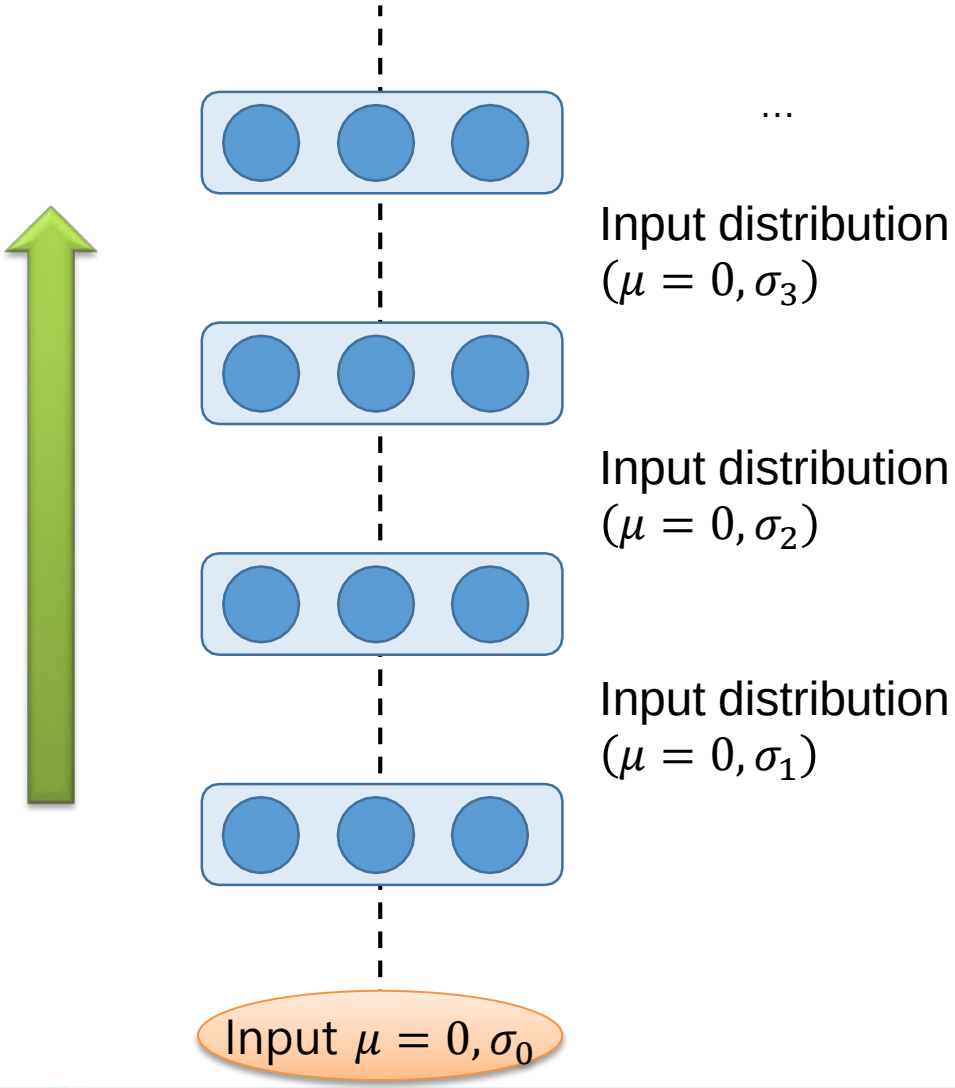
Initialization with Appropriate Scale

- **If the weights in a network start too small,**
  - ➢ the signal shrinks as it passes through each layer
- **If the weights in a network start too large,**
  - ➢ the signal explodes as it passes through each layer

- **Xavier initialization makes the signal keep in a reasonable range of values through many layers**

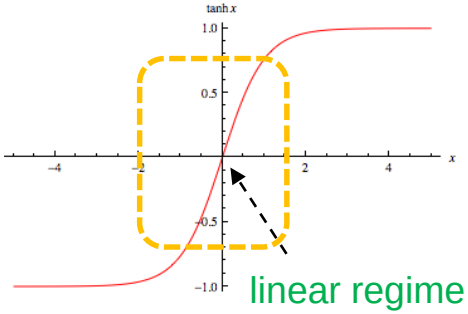
- **Xavier algorithm initializes W's with drawing them from a zero mean and $\mathbf{Var}(W_i) = \frac{1}{n_{\text{in}}}$**

Basic Idea of Xavier Initialization



Linear Activation at Each Layer

- $\boldsymbol{Y_i = W_1 X_1 + \cdots + W_n X_n}$
  - input $\boldsymbol{X}$ with $\boldsymbol{n}$ components
  - linear neuron with random weight W



linear regime at $x = 0$

- $\mathbf{Var}(\boldsymbol{W_i X_i}) =$
  $\boldsymbol{E[X_i]^2 Var(W_i) + E[W_i]^2 Var(X_i) + \mathbf{Var}(W_i)\mathbf{Var}(X_i)}$
  - assume, $\boldsymbol{E[X_i] = 0}$ and $\boldsymbol{E[W_i] = 0}$

  $\therefore \boldsymbol{Var(W_i X_i) = Var(W_i)Var(X_i)}$

- $\mathbf{Var}(\boldsymbol{Y}) = \mathbf{Var}(\boldsymbol{W_1 X_1 + \cdots + W_n X_n}) = n\mathbf{Var}(\boldsymbol{W_i})\mathbf{Var}(\boldsymbol{X_i})$
  - assume, X_i and W_i are i.i.d.

Xavier Initialization Algorithm

- $\mathbf{Var}(Y) = n_{\text{in}}\mathbf{Var}(W_i)\mathbf{Var}(X_i)$
  - the variance of the output Y is the variance of input X scaled by $n_{\text{in}}\boldsymbol{Var}(\boldsymbol{W_i})$

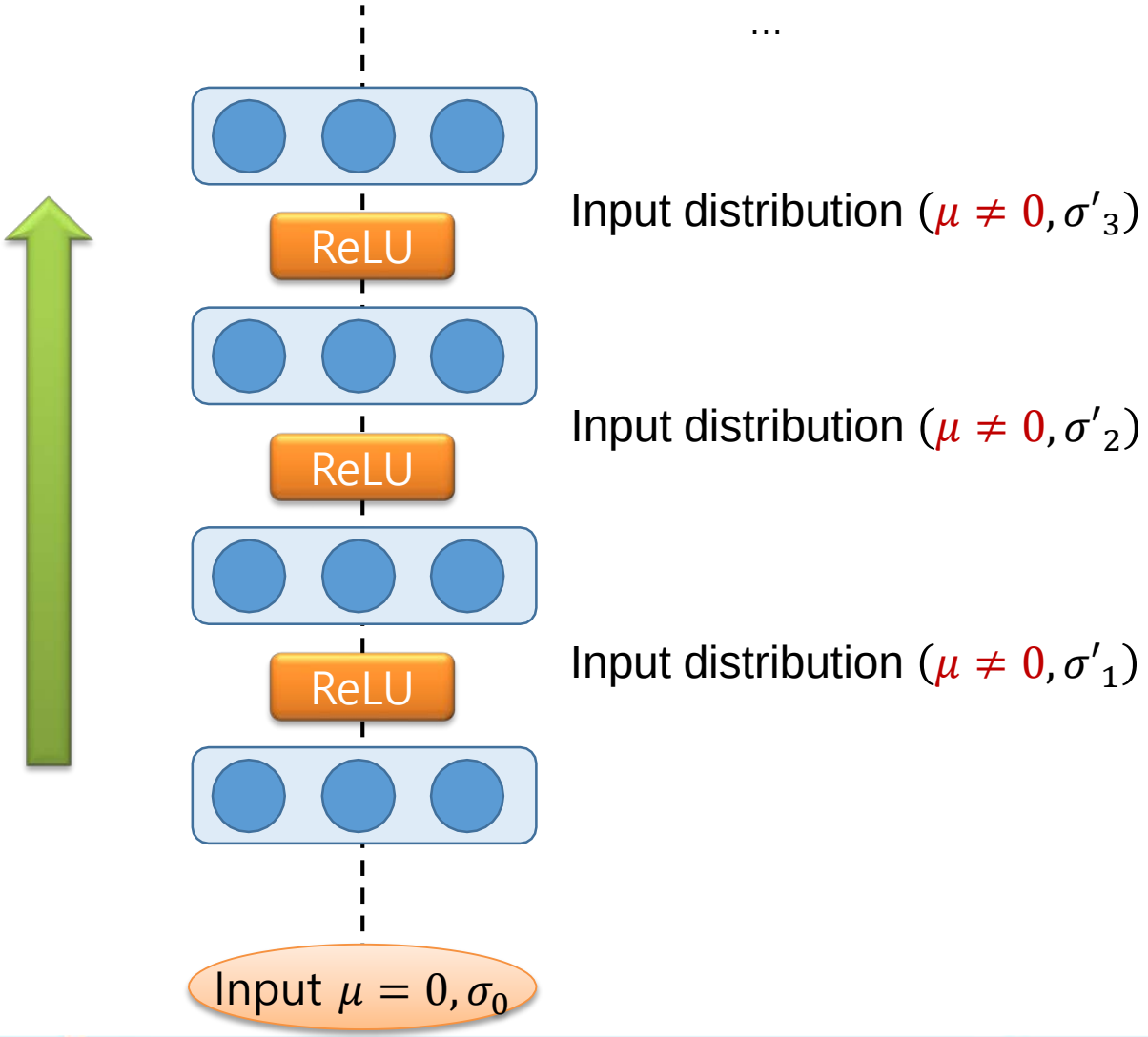
- **Ideal case: $\mathbf{Var}(Y) = \mathbf{Var}(X_i)$**
  - i.e. $n_{\text{in}}\mathbf{Var}(W_i) = 1$

- **So, for the weight initialization,**

$$\mathbf{Var}(W_i) = \frac{1}{n_{\text{in}}}$$

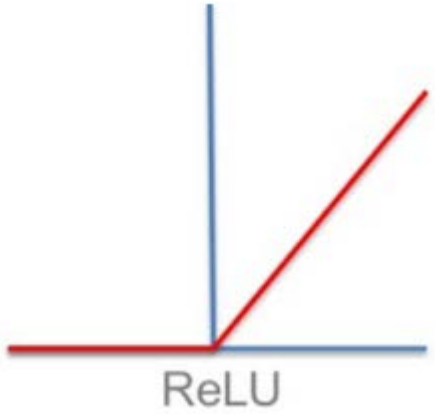
$$\left(\text{or } \mathbf{Var}(W_i) = \frac{2}{n_{\text{in}}+n_{\text{out}}}\right)$$

Basic Idea of MSRA Initialization



Initialization for Rectifiers (a.k.a. MSRA)

- $y_l = w_l x_l + b_l$
  - n: # inputs for current layer
  - l: an index of layer
  - b: a bias term



- **ReLU does not have zero mean**

$$\because x_l = max(0, y_{l-1})$$

- $\mathbf{Var}(y_l) = n_l \mathbf{Var}(w_l x_l) = n_l \mathbf{Var}(w_l) E(x_l^2)$
  - assume, $E[w] = 0$, but $E[x] \neq 0$
  - x and w are i.i.d.

Distribution in l-th Layer

$$E[x_l^2] = \int_{-\infty}^{\infty} \boldsymbol{max}(\mathbf{0}, y_{l-1})^2 \boldsymbol{p}(\boldsymbol{y}) \boldsymbol{dy}$$

$$= \int_{0}^{\infty} \boldsymbol{y_{l-1}^2 p}(\boldsymbol{y_{l-1}}) \boldsymbol{dy}$$

- $\because \boldsymbol{y_{l-1}}^2$ is symmetric around 0
- $\boldsymbol{p}(\boldsymbol{y_{l-1}})$ is assumed to symmetric around 0
  ($\because \boldsymbol{w}_{l-1}$ is assumed to symmetric around 0)

$$= \frac{1}{2} \int_{-\infty}^{\infty} \boldsymbol{y_{l-1}^2 p}(\boldsymbol{y_{l-1}}) \boldsymbol{dy}$$

$$= \frac{1}{2} \int_{-\infty}^{\infty} (\boldsymbol{y_{l-1}} - \boldsymbol{E}[\boldsymbol{y_{l-1}}])^2 \boldsymbol{p}(\boldsymbol{y_{l-1}}) \boldsymbol{dy}$$

$$= \frac{1}{2} Var[y_{l-1}]$$

Variance Propagation for All L Layers

- $E(x_l^2) = \frac{1}{2}\mathrm{Var}(y_{l-1})$, when f is ReLU

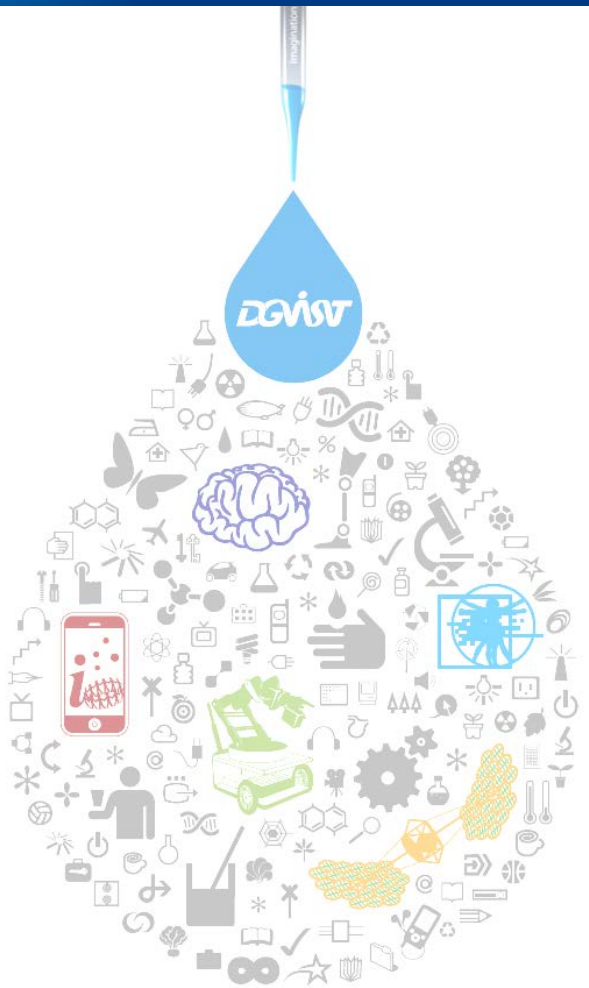
- $\mathrm{Var}(y_l) = \frac{1}{2} n_l \mathrm{Var}(w_l)\mathrm{Var}(y_{l-1})$
- With L layers put together,

should avoid exponential growth
$\rightarrow \frac{1}{2} n_l \mathrm{Var}(w_l) = 1, \forall\, l$

$$\mathrm{Var}(y_L) = \mathrm{Var}(y_1)\left(\prod_{l=2}^{L} \frac{1}{2} n_l \mathrm{Var}(w_l)\right)$$

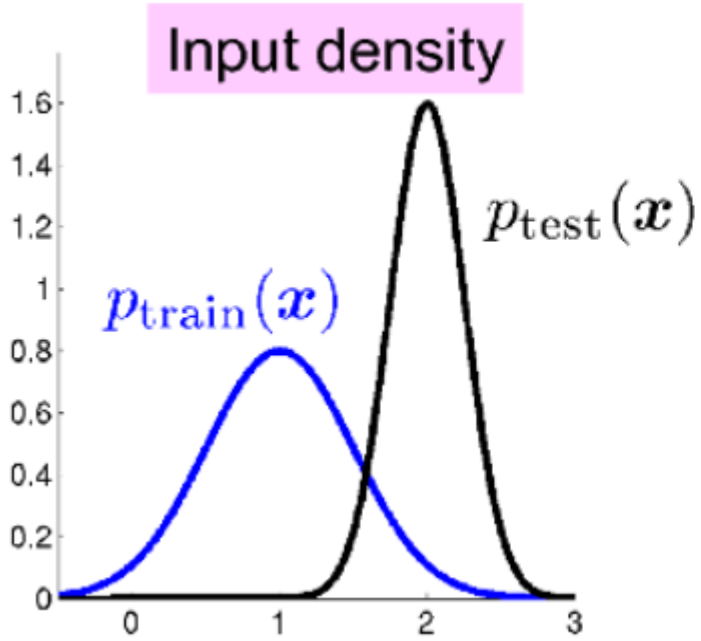
- Initialize W's with drawing them from a zero-mean Gaussian distribution whose $\mathrm{Var}(w_l) = \frac{2}{n_{in}}$

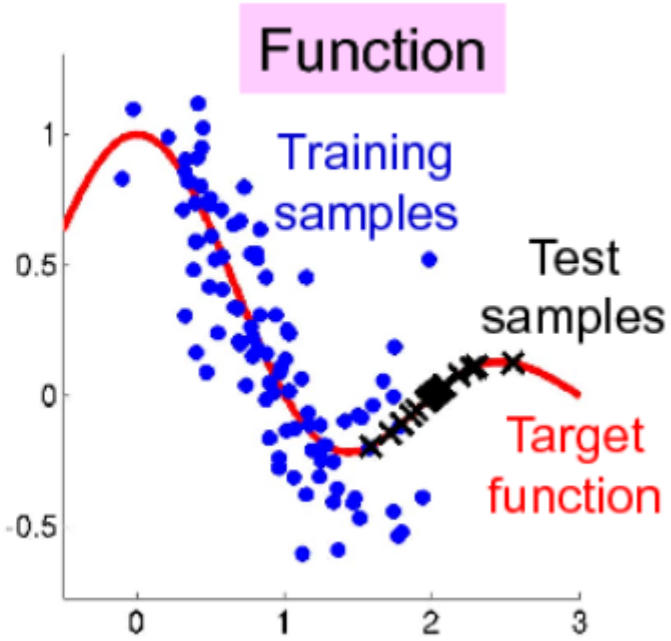
Batch Normalization



Covariate Shift (Shimodaira, 2000)

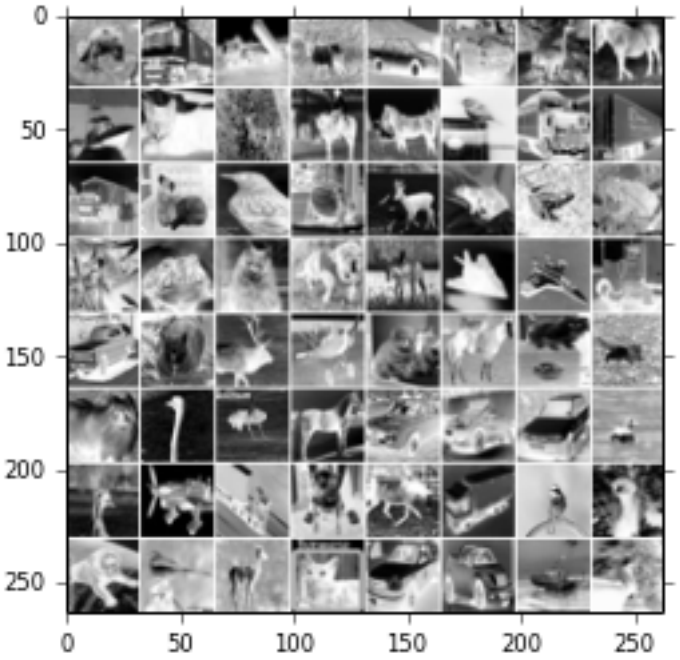
- **Training / testing input distributions are different, but target function remains unchanged**
  - causing weak extrapolation





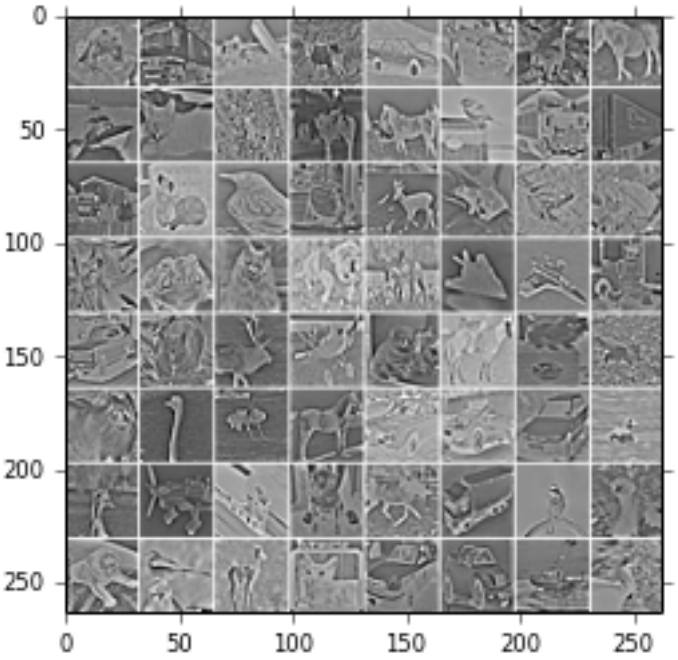
Masashi Sugiyama, “Density Ratio Estimation in Machine Learning”, 2012

Solution for Covariate Shift



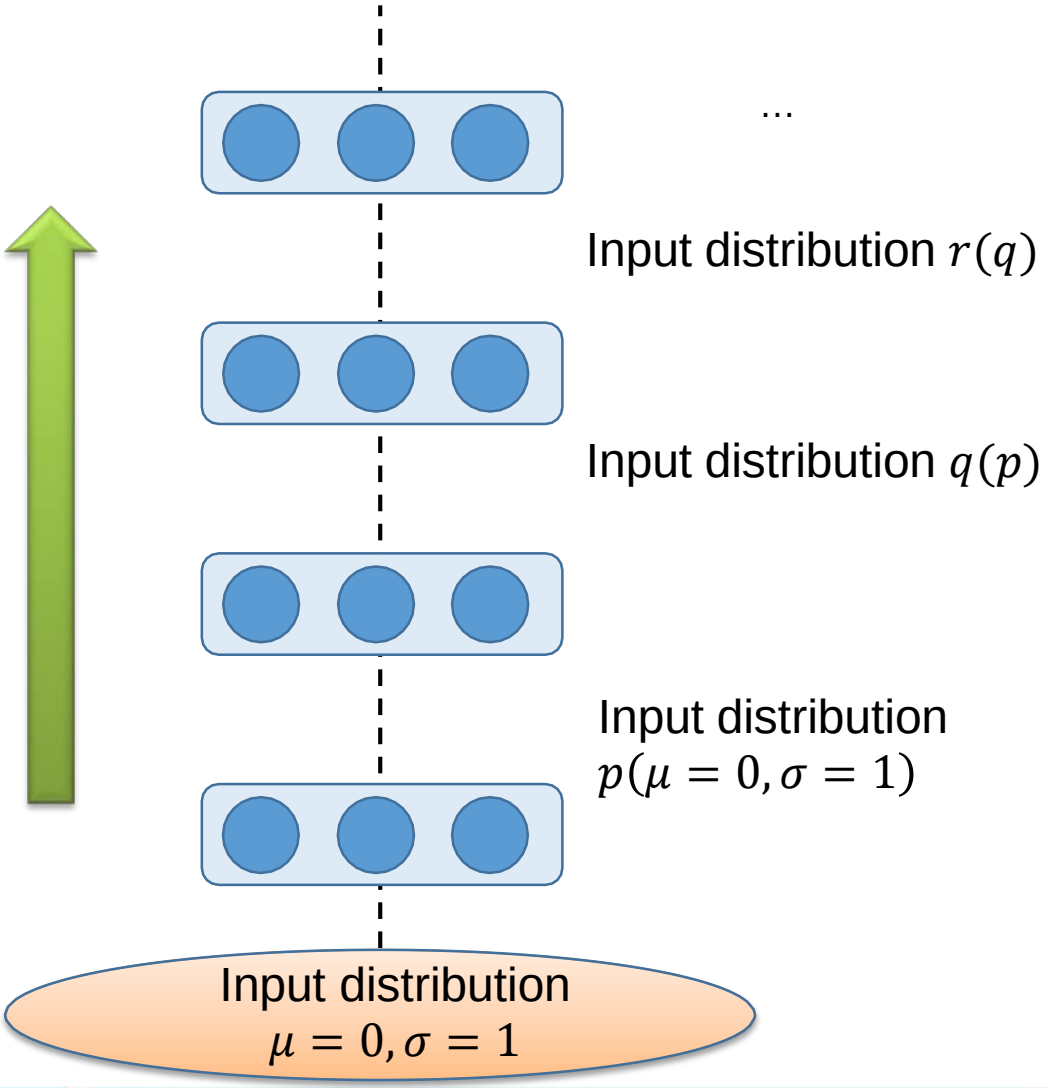
Transform:
mean = 0
variance = 1

whitening

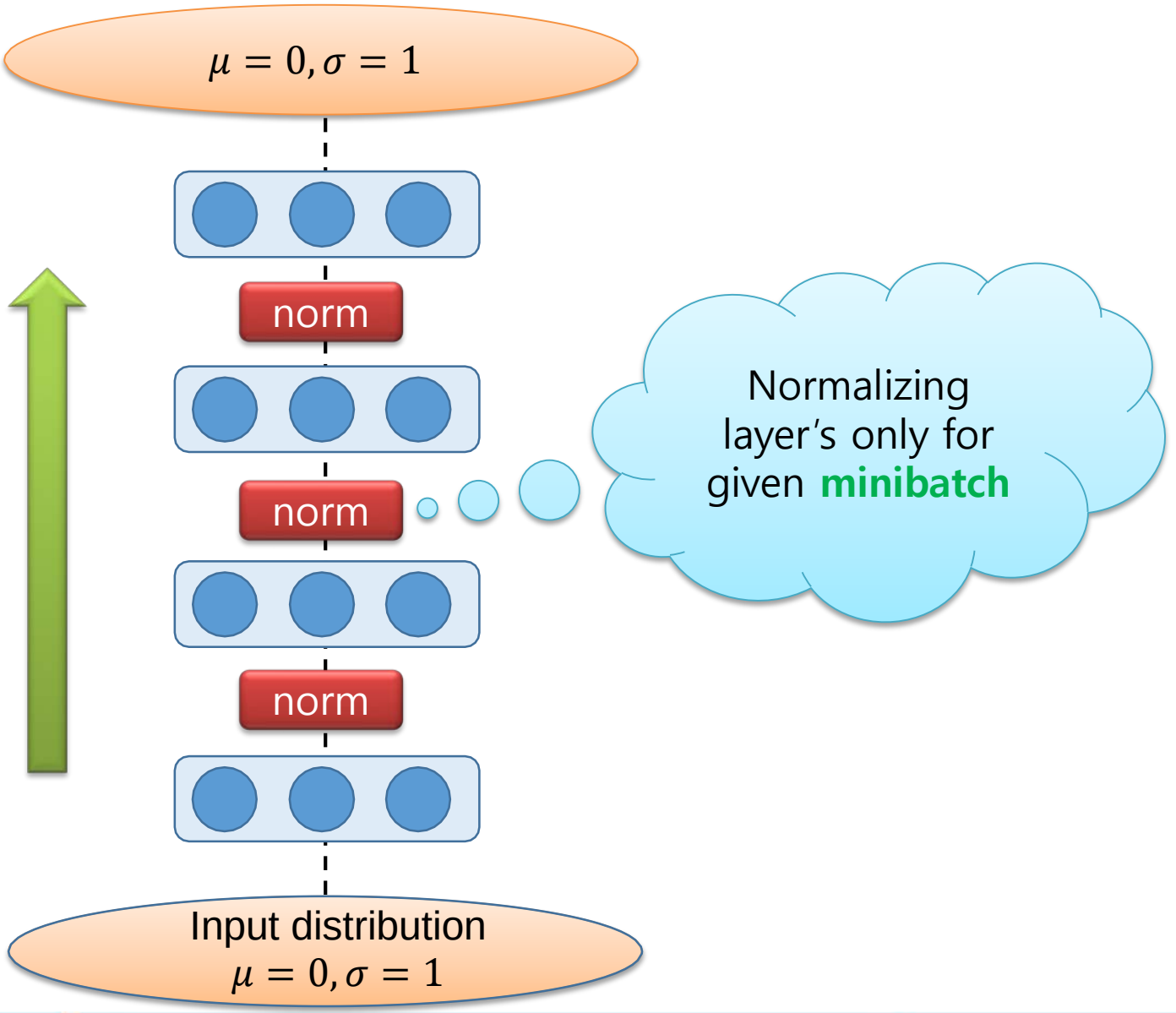


Answer by bayerj, “What is the difference between ZCA whitening and PCA whitening?”, Oct 1, 2014

Internal Covariate Shift in DNN



Solution for Internal Covariate Shift



Batch Normalization (BN)

- **Normalizing each layer's input of given minibatch**

m: size of mini-batch
$\mathcal{B} = \{x_{1\ldots m}\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$$ ← minibatch mean

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2$$ ← minibatch variance

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$$ ← normalization

- **Two necessary simplifications:**
  - normalize each feature independently, $N(0,1)$
  - calculate mean and variance only for a current training minibatch

BN Transformation

- **Simply normalizing each input of a layer may change what the layer can represent**

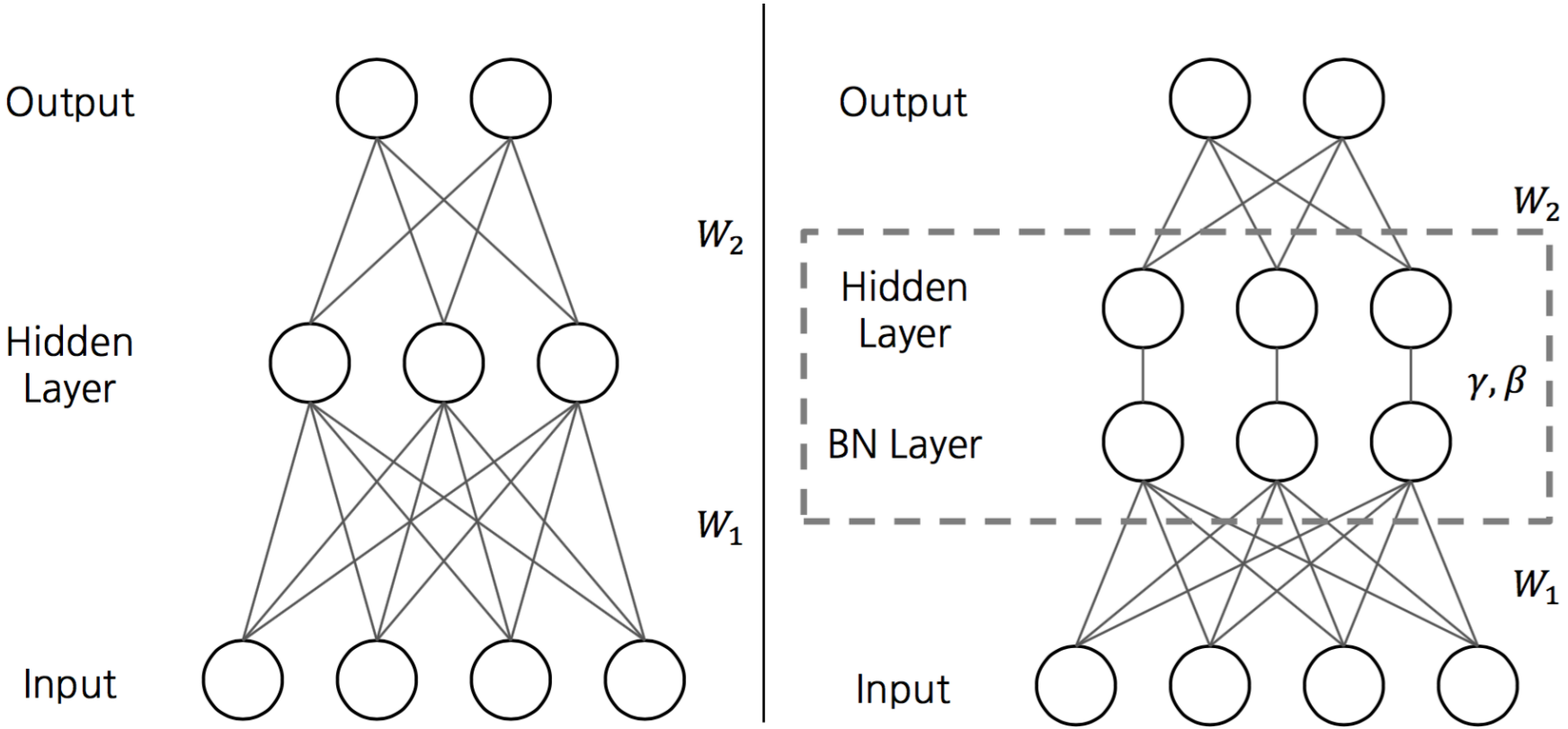
- **Introduce a linear transformation for each activation**

$$BN_{\gamma,\beta}\left(\boldsymbol{x}_{1\ldots m}\right) = \gamma\hat{\boldsymbol{x}}_{1\ldots m} + \beta$$

γ, β: parameters of BN transformation
$\hat{x}_i$: normalized input

  - γ and β are learned (updated) at BP steps
    - γ: scale factor
    - β: shift factor

NN with & without BN



Comparison between neural networks with and without BN.

http://sanghyukchun.github.io/88

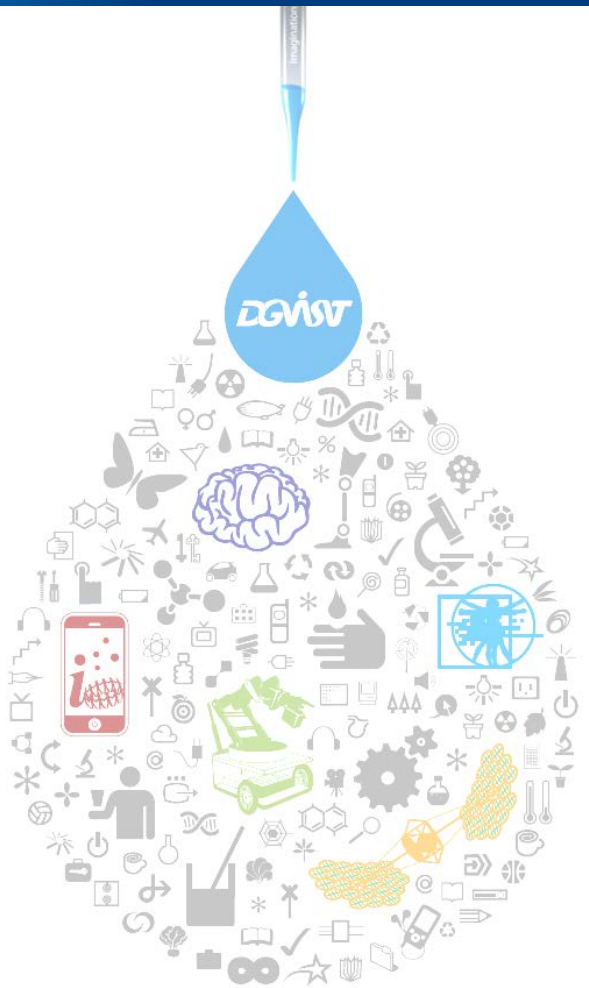
Next Chapter: Convolutional Networks

- The Convolution Operation
- Motivation
- Pooling
- Convolution and Pooling as an Infinitely Strong Prior
- Variants of the Basic Convolution Function
- Structured Outputs
- Data Types
- Efficient Convolution Algorithms
- Random or Unsupervised Features
- The Neuroscientific Basis for Convolutional Networks
- Convolutional Networks and the History of Deep Learning

InfoLab

Q&A

Thank you



DGIST 대구경북과학기술원
Daegu Gyeongbuk Institute of Science & Technology