Optimization Methods for Deep Learning

Gradient descent for non-convex optimization

$\|A\|_2 = \max_{\|v\|_2=1} v^T A v \quad , v \in \mathbb{R}^d$

Decsent Lemma: Let $f : \mathbb{R}^d \to \mathbb{R}$ be twice differentiable, and $\|\nabla^2 f\|_2 \le \beta$. Then setting the learning rate $\eta = 1/\beta$, and applying gradient descent, $x_{t+1} = x_t - \eta \nabla f(x_t)$, we have:

can $\eta \le \frac{1}{\beta}$

$$f(x_t) - f(x_{t+1}) \ge \frac{1}{2\beta} \|\nabla f(x_t)\|_2^2. \quad > 0$$

Pf: by Taylor expansion & mean-value theorem unless $\nabla f(x_t) = 0$

$f(x+\Delta) = f(x) + \Delta^T \nabla f(x) + \frac{1}{2} \Delta^T \nabla^2 f(y) \Delta$ for some y

$\Delta^T \nabla^2 f(y) \Delta \le \|\nabla^2 f(y)\|_2 \cdot \|\Delta\|_2^2 \le \beta \cdot \|\Delta\|_2^2$

choose $\Delta = -\eta \nabla f(x_t)$

$f(x_{t+1}) \le f(x_t) - \frac{1}{\beta} \|\nabla f(x_t)\|_2^2 + \frac{1}{2} \frac{1}{\beta} \|\nabla f(x_t)\|_2^2$

$= f(x_t) - \frac{1}{2\beta} \|\nabla f(x_t)\|_2^2$

Converging to stationary points

$\eta_t = \frac{1}{\beta}$

Theorem: In $T = O(\frac{\beta}{\epsilon^2})$ iterations, we have $\|\nabla f(x)\|_2 \le \epsilon$.

Pf: $f(x_{t+1}) \le f(x_t) - \frac{\eta_t}{2}\|\nabla f(x_t)\|_2^2$

sum over $t = 0, \dots, T-1$

$$\sum_{t=1}^{T} f(x_t) \le \sum_{t=0}^{T-1} f(x_t) - \frac{\eta_t}{2}\sum_{t=0}^{T-1}\|\nabla f(x_t)\|_2^2$$

$$\Rightarrow f(x_T) \le f(x_0) - \frac{\eta_t}{2}\sum_{t=0}^{T-1}\|\nabla f(x_t)\|_2^2$$

$$\Rightarrow \sum_{t=0}^{T-1}\|\nabla f(x_t)\|_2^2 \le \frac{2(f(x_0) - f(x_T))}{\eta_t}$$

want

$T \cdot \min_{0\le t\le T-1}\|\nabla f(x_t)\|_2^2 \le$

$$\min_{0\le t\le T-1}\|\nabla f(x_t)\|_2 \le \sqrt{\frac{2\cdot f(x_0) - f(x_T)}{\eta_t \cdot T}} \le \epsilon$$

$T = O(\frac{\beta}{\epsilon^2})$

Gradient Descent for Quadratic Functions

$x^* = 0$ $\nabla f(x) = Ax$ $\lambda_{min} > 0$

Problem: $\min_x \frac{1}{2} x^\top A x$ with $A \in \mathbb{R}^{d \times d}$ being positive-definite.

Theorem: Let $\lambda_{\max}$ and $\lambda_{\min}$ be the largest and the smallest eigenvalues of A. If we set $\eta \leq \frac{1}{\lambda_{\max}}$, we have

$$\|x_t\|_2 \leq \left(1 - \eta\lambda_{\min}\right)^t \|x_0\|_2$$

$$\|x_{t+1}\|_2 = \|x_t - \eta_t A x_t\|_2$$
$$= \|(I - \eta_t A) x_t\|_2$$
$$\leq \|I - \eta_t A\|_2 \cdot \|x_t\|_2 \quad (\lambda_{max}(\eta_t A) \leq 1)$$
$$\leq (1 - \eta_t \lambda_{min}) \cdot \|x_t\|_2$$
$$\leq (1 - \eta_t \lambda_{min})^{t+1} \|x_0\|_2$$

linear convergence

Implication:

If $\|x_T\|_2 \leq \varepsilon$

if $\eta_t = \frac{1}{\lambda_{max}}$

$\Rightarrow T = O\left(\frac{\lambda_{max}}{\lambda_{min}} \cdot \log\left(\frac{1}{\varepsilon}\right)\right)$

$\kappa = \frac{\lambda_{max}}{\lambda_{min}}$ condition number

Momentum: Heavy-Ball Method (Polyak '64)

$v_0 = 0$

$v_1 = -\alpha \nabla f(x_0)$

Problem: $\min_x f(x)$

Method: $v_{t+1} = -\nabla f(x_t) + \beta v_t$

$$x_{t+1} = x_t + \eta v_{t+1}$$

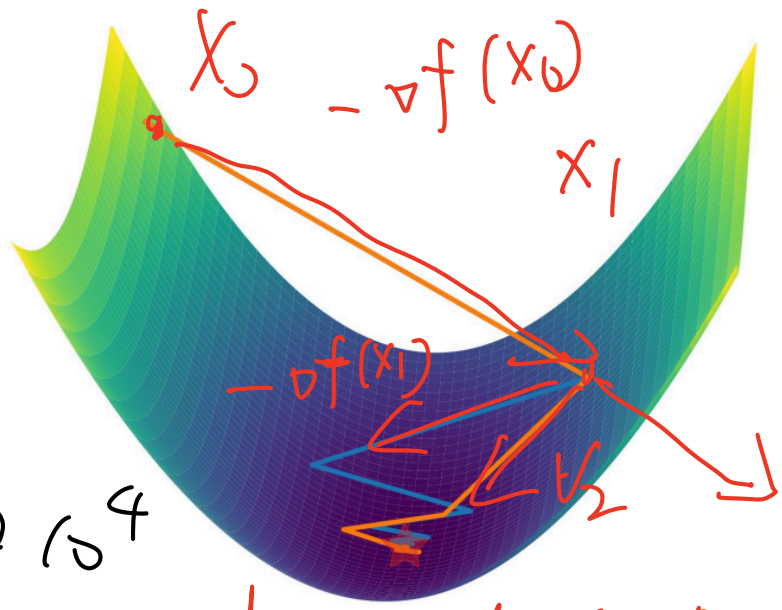
For quadratic optimization

$O\left(\sqrt{\kappa} \log\left(\frac{1}{\epsilon}\right)\right)$

v.s. gradient descent

$O\left(\kappa \log\left(\frac{1}{\epsilon}\right)\right)$

$\kappa = 10^8 \Rightarrow$ improve $10^8 \to 10^4$



However, does not hold for general strongly convex function

Momentum: Nesterov Acceleration (Nesterov '89)

Problem: $\min_x f(x)$

Method: $v_{t+1} = -\nabla f(x_t + \beta v_t) + \beta v_t$

$x_{t+1} = x_t + \eta v_{t+1}$

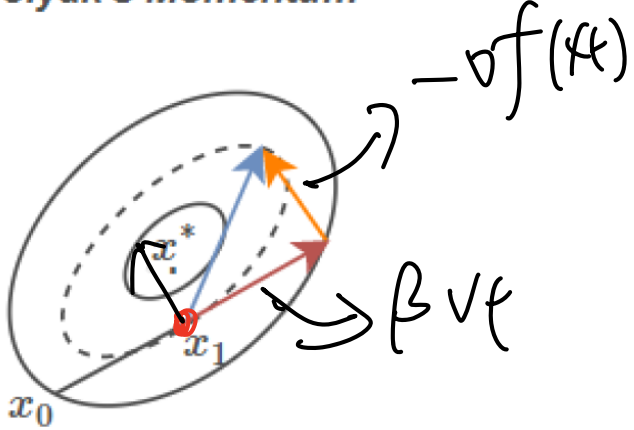
vs. $-\nabla f(x_t)$

For general strongly convex function

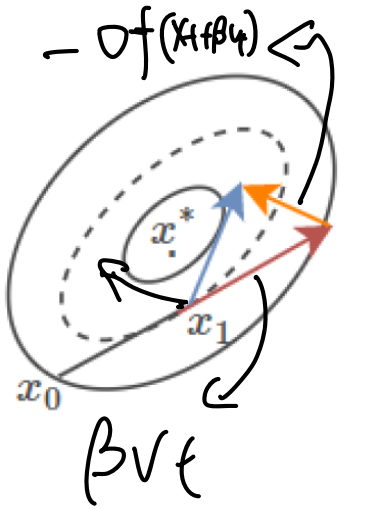
$\sqrt{\kappa} \cdot \log\left(\frac{1}{\epsilon}\right)$

tight

Polyak's Momentum



Nesterov Momentum



$x \in \mathbb{R}^d$ $O(d^2)$ inverse $O(d^3)$

Newton's Method

$d \times d$

Newton's Method: $x_{t+1} = x_t - \eta(\nabla^2 f(x_t))^{-1} \nabla f(x_t)$

- GD: $x_{t+1} = x_t - \eta \nabla f(x_t)$

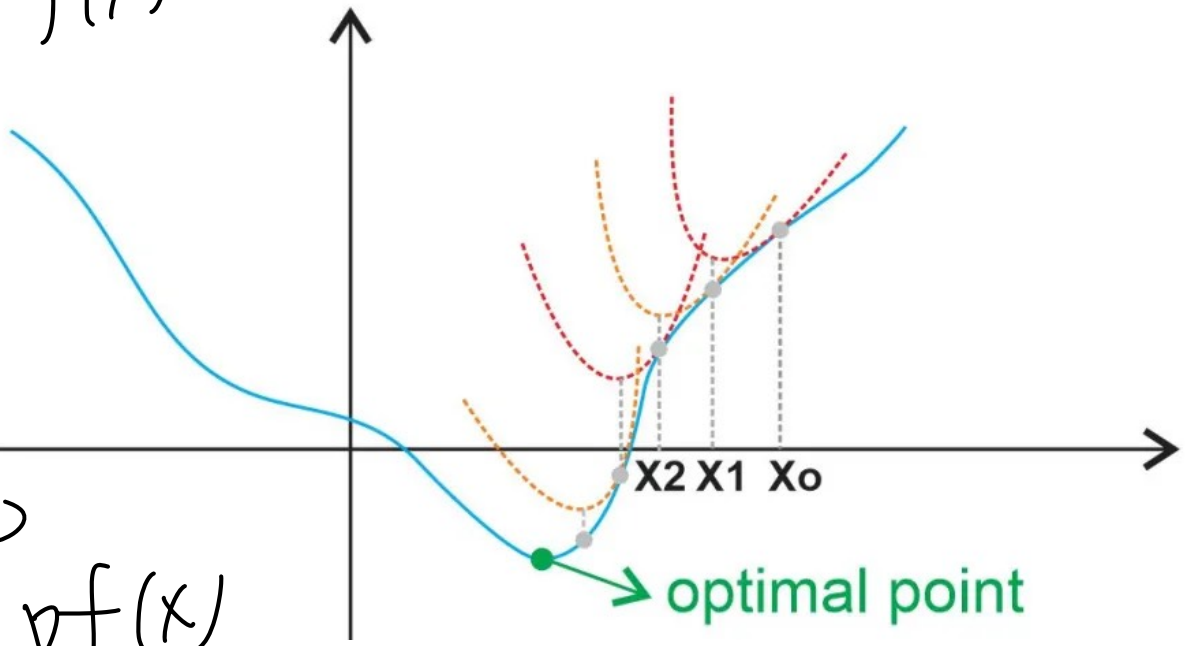
$f(x+\Delta) \approx f(x) + \Delta^T \nabla f(x) + \frac{1}{2}\|\Delta\|_2^2$

$\Rightarrow \Delta^* = -\nabla f(x_t)$

- Newton

$f(x+\Delta) \approx f(x) + \Delta^T \nabla f(x) + \frac{1}{2}\Delta^T \nabla^2 f(x)\Delta$

$\Rightarrow \Delta^* = -(\nabla^2 f(x))^{-1} \nabla f(x)$



Theorem: $O(\log \log(\frac{1}{\epsilon}))$ quadratic convergence

vs. $\log(\frac{1}{\epsilon})$

AdaGrad (Duchi et al. '11)

suppose diagonal
$O(d)$

Newton Method: $x_{t+1} = x_t - \eta(\nabla^2 f(x_t))^{-1} \nabla f(x_t)$

AdaGrad: separate learning rate for every parameter

diagonal

$$x_{t+1} = x_t - \eta(G_{t+1} + \epsilon I)^{-1} \nabla f(x_t), \; (G_t)_{ii} = \sqrt{\sum_{j=1}^{t-1} \left(\nabla f(x_t)_i\right)^2}$$

make invertible

$x_t \in \mathbb{R}^d$

preconditioning

RMSProp (Hinton et al. '12)

AdaGrad: separate learning rate for every parameter

$$x_{t+1} = x_t - \eta(G_{t+1} + \epsilon I)^{-1} \nabla f(x_t), \; (G_t)_{ii} = \sqrt{\sum_{j=1}^{t-1} \left(\nabla f(x_t)_i \right)^2}$$

Strictly increasing

RMSProp: exponential weighting of gradient norms

$$x_{t+1} = x_t - \eta(G_{t+1} + \epsilon I)^{-1/2} \nabla f(x_t),$$
$$(G_{t+1})_{ii} = \beta(G_t)_{ii} + (1-\beta)(\nabla f(x_t)_i)^2$$

$0 < \beta < 1$, close to 1

exponential weighting

EMA

AdaDelta (Zeiler '12)

$\frac{\partial f}{\partial x}$: $\frac{1}{\text{unit of } x}$

RMSProp:

$$x_{t+1} = x_t - \eta (G_{t+1} + \epsilon I)^{-1/2} \nabla f(x_t),$$
$$(G_{t+1})_{ii} = \beta (G_t)_{ii} + (1-\beta)(\nabla f(x_t)_i)^2$$

AdaDelta:

$$x_{t+1} = x_t - \eta \Delta x_t,$$
$$\Delta x_t = \sqrt{u_t + \epsilon} \cdot (G_{t+1} + \epsilon I)^{-1/2} \nabla f(x_t)$$
$$(G_{t+1})_{ii} = \rho (G_t)_{ii} + (1-\rho)(\nabla f(x_t)_i)^2,$$
$$u_{t+1} = \rho u_t + (1-\rho)\|\Delta x_t\|_2^2 \quad 0 < \rho < 1$$

GD : $-\alpha f(x_t)$: $\frac{1}{\text{unit of } x}$

Newton : $\Delta x_t \propto \left(\frac{\partial^2 f}{\partial x^2}\right)^{-1} \frac{\partial f}{\partial x} \propto \left(\left(\frac{1}{\text{unit of } x}\right)^2\right)^{-1} \cdot \frac{1}{\text{unit of } x} \propto$ unit of x

Adam (Kingma & Ba '14)

Momentum: heavy ball

$$v_{t+1} = -\nabla f(x_t) + \beta v_t,\ x_{t+1} = x_t + \eta v_{t+1}$$

RMSProp: exponential weighting of gradient norms

$$x_{t+1} = x_t - \eta(G_{t+1} + \epsilon I)^{-1} \nabla f(x_t),$$
$$(G_t)_{ii} = \beta(G_t)_{ii} + (1-\beta)(\nabla f(x_t)_i)^2$$

Adam

$$v_{t+1} = \beta_1 v_t + (1-\beta_1)\nabla f(x_t)$$
$$(G_{t+1})_{ii} = \beta_2(G_t)_{ii} + (1-\beta_2)(\nabla f(x_t)_i)^2$$
$$x_{t+1} = x_t - \eta(G_{t+1} + \epsilon I)^{-1/2} v_{t+1}$$

Default choice nowadays.

Important Techniques in Neural Network Training

Gradient Explosion / Vanishing

- Deeper networks are harder to train:
  - Intuition: gradients are products over layers
  - Hard to control the learning rate

$$f(x, W_1, \cdots, W_{H+1}) = W_{H+1}\,\sigma(W_H \cdots \sigma(W_1 x)\cdots)$$

$$\frac{\partial f}{\partial W_h} = \left(W_{H+1} A_{H-1} \cdots W_{h+1} A_h\right)^T \left(A_{h-1} W_{h-1} \cdots W_1 x\right)$$

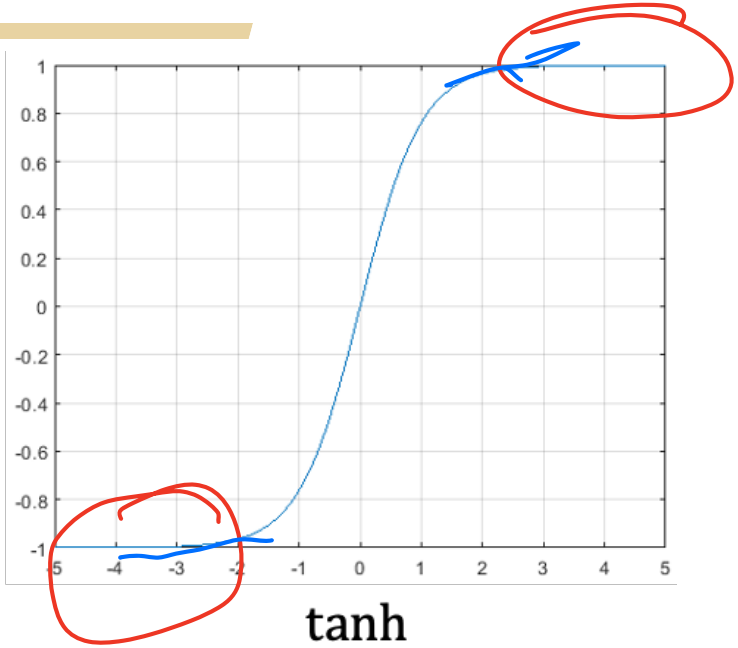
$$A_h = \mathrm{diag}\left(\sigma'(W_h\,\sigma(\cdots \sigma(W_1 x)\cdots)\right)$$

If $W_1, \ldots, W_{H+1}$ magnitude large → gradient explosion

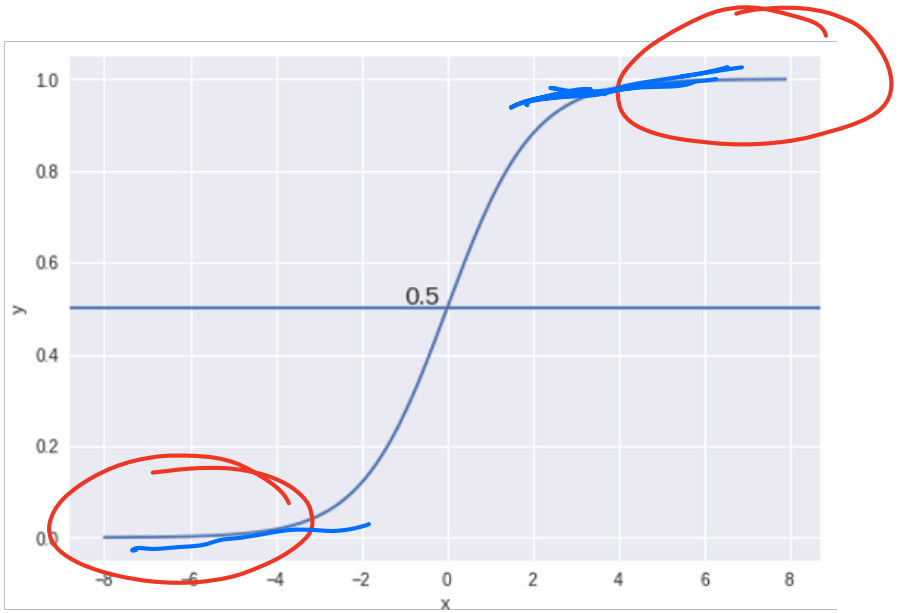
$W_1, \ldots, W_{H+1}$ magnitude small → gradient vanishing

Activation Functions

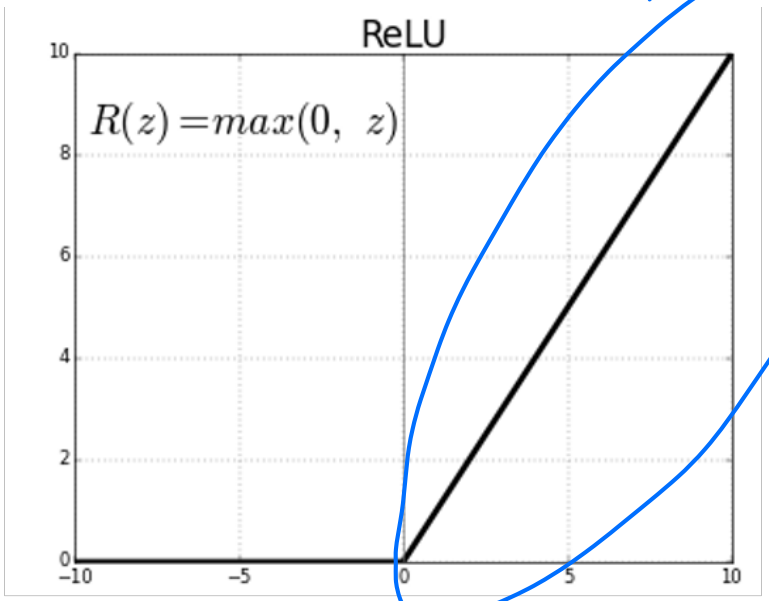
gradient vanishing



tanh



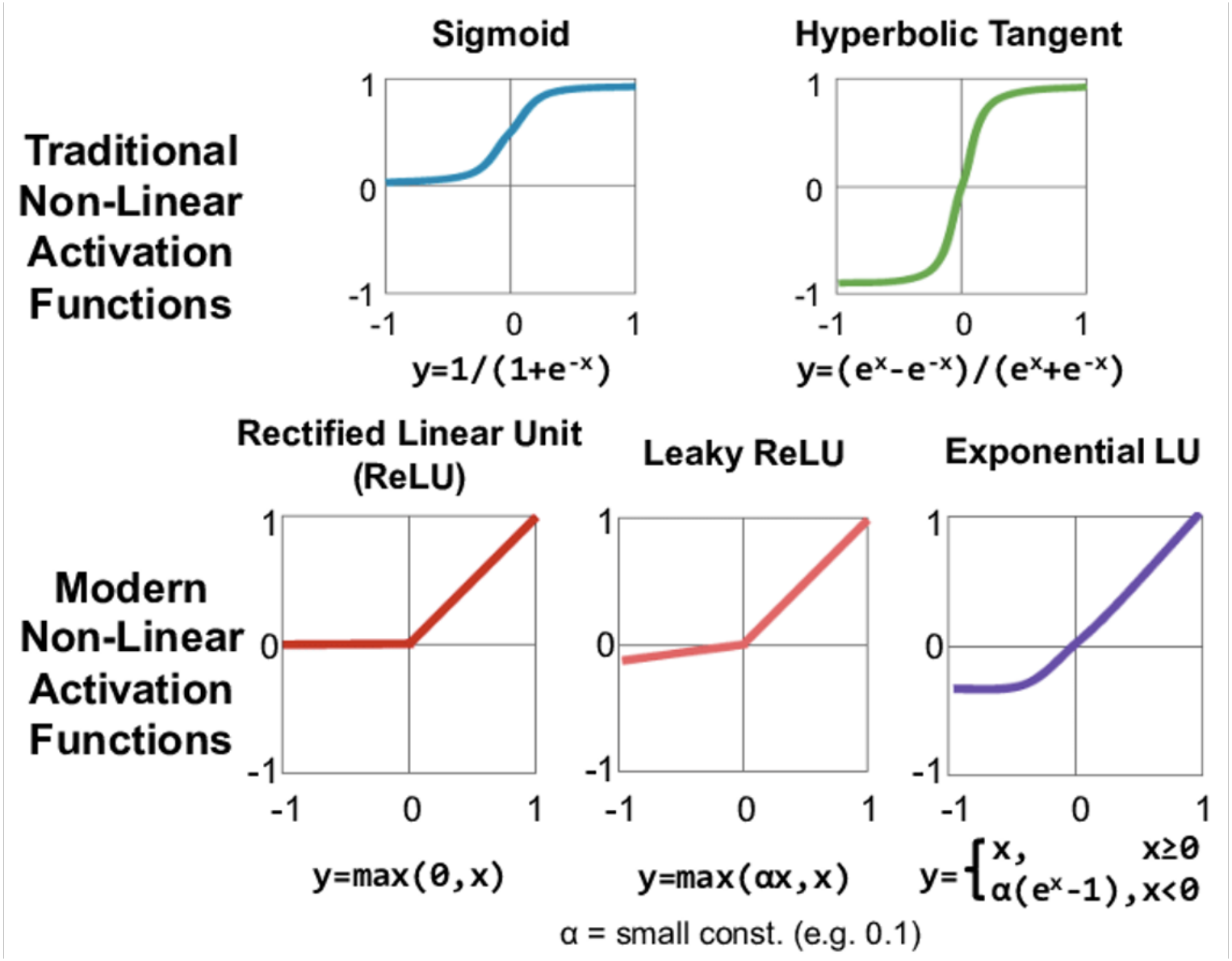
sigmoid



Rectified Linear United

$g'(\cdot) = 1$

Activation Function



Initialization

- Zero-initialization
- Large initialization
- Small initialization

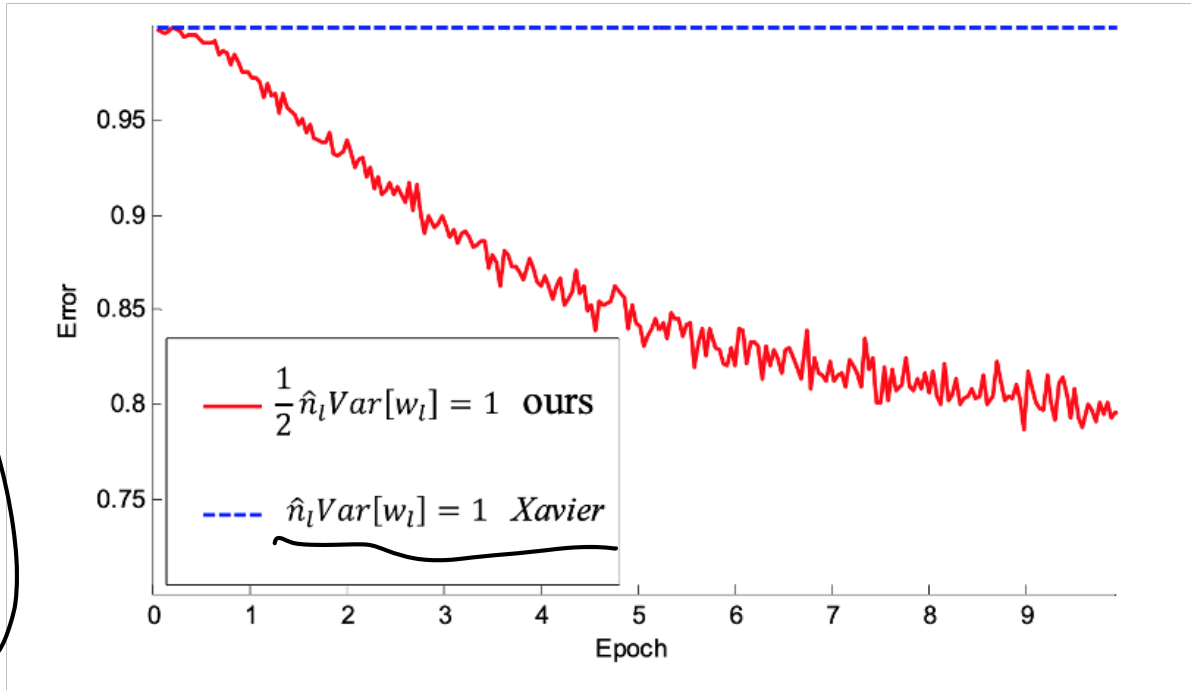
- Design principles:
  - Zero activation mean
  - Activation variance remains same across layers

Kaiming Initialization (He et al. '15)

- $W_{ij}^{(h)} \sim \mathcal{N}\left(0, \frac{2}{d_h}\right)$.
- $b^{(h)} = 0$
- Designed for ReLU activation
- 30-layer neural network

$W^{(h)} \in \mathbb{R}^{d_{h+1} \times d_h}$

$\mathcal{N}\left(0, \frac{1}{d_h}\right)$



Kaiming Initialization (He et al. '15)

For layer h:

$$z^h = W^h x^h$$

pre-activation

$$x^{h+1} = \sigma(z^h)$$

$$z_i^h = \sum_{j=1}^{d_h} W_{ij}^h \cdot x_j^h$$

Goal: z^h: mean zero, same variance for all layers

if $\mathbb{E}[W_{ij}^h] = 0$ $\Rightarrow$ $\mathbb{E}[z_i^h] = \mathbb{E}\left[\sum_{j=1}^{d_h} W_{ij}^h x_j^h\right]$

$$= \sum_{j=1}^{d_h} \mathbb{E}[W_{ij}^h] \cdot x_j^h$$

$$= 0$$

Kaiming Initialization (He et al. '15)

$$\mathrm{Var}(z_i^h) = d_h \,\mathrm{Var}\left(w_{ij}^h \cdot x_j^h\right)$$

$$= d_h \Big[\mathrm{Var}(w_{ij}^h)\cdot\mathrm{Var}(x_j^h) + \left(\mathbb{E}[w_{ij}]\right)^2\cdot\mathrm{Var}(x_j^h) + \mathrm{Var}(w_{ij}^h)\cdot\left(\mathbb{E}[x_j^h]\right)^2\Big]$$

($\mathbb{E}[w_{ij}] = 0$)

$$= d_h \,\mathrm{Var}(w_{ij}^h)\cdot\mathbb{E}\left[(x_j^h)^2\right]$$

$$\mathbb{E}\left[(x_j^h)^2\right] = \int_{-\infty}^{\infty} (x_j^h)^2 \, p(x_j^h)\, dx_j^h$$

(ReLU) $$= \int_{-\infty}^{\infty} \max(0, z_j^{h-1})^2 \, p(z_j^{h-1})\, dz_j^{h-1}$$

$$= \int_{0}^{\infty} (z_j^{h-1})^2 \, p(z_j^{h-1})\, dz_j^{h-1}$$

Kaiming Initialization (He et al. '15)

(symmetry of init around 0) $= \frac{1}{2}\int_{-\infty}^{\infty} (z_j^{l-1})^2 \, p(z_j^{l-1}) \, dz_j^{l-1}$

$= \frac{1}{2} \mathrm{Var}(z_j^{l-1})$

$\mathrm{Var}(z_i^l) = d_l \, \mathrm{Var}(w_{ij}^l) \cdot \frac{1}{2} \mathrm{Var}(z_j^{l-1})$

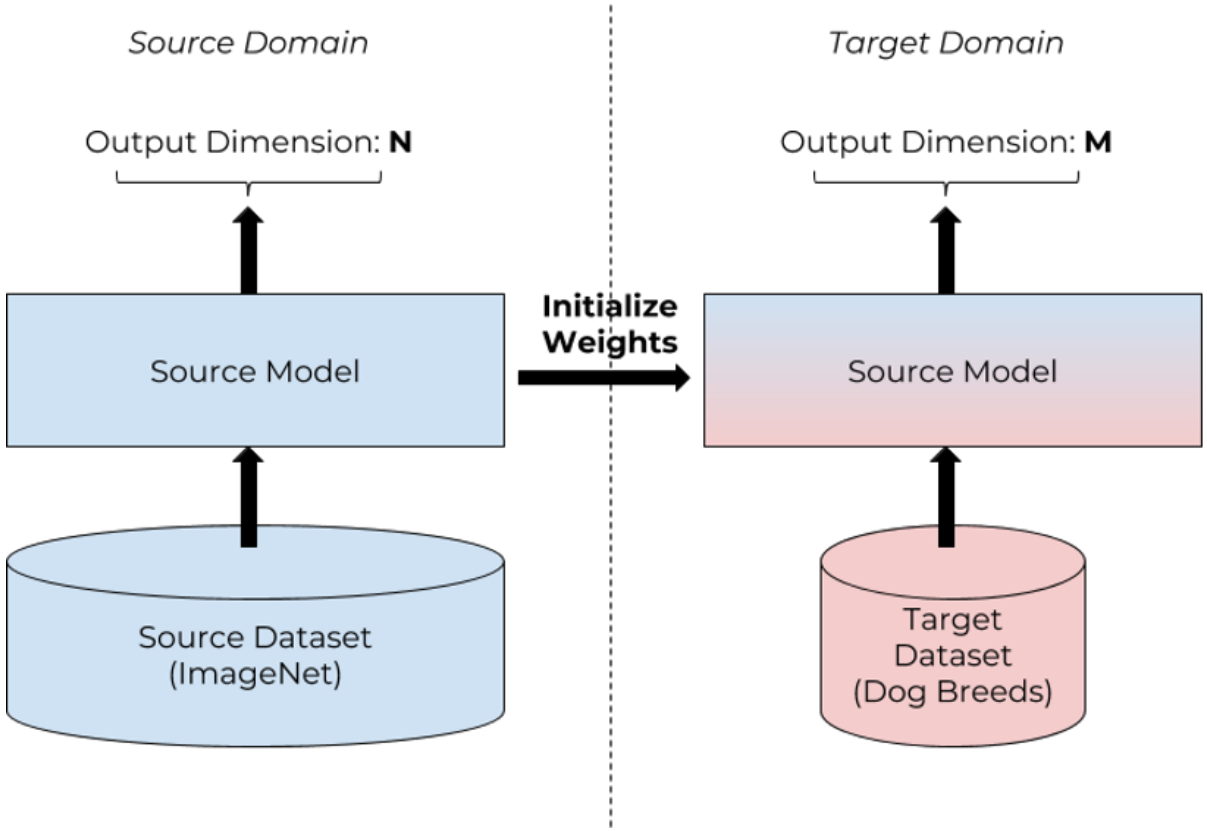
Want $\mathrm{Var}(z_i^l) = \mathrm{Var}(z_j^{l-1})$

need: $d_l \cdot \mathrm{Var}(w_{ij}^l) \cdot \frac{1}{2} = 1$

$\mathrm{Var}(w_{ij}^l) = \frac{2}{d_l}$

Initialization by Pre-training

- Use a pre-trained network as initialization
- And then fine-tuning



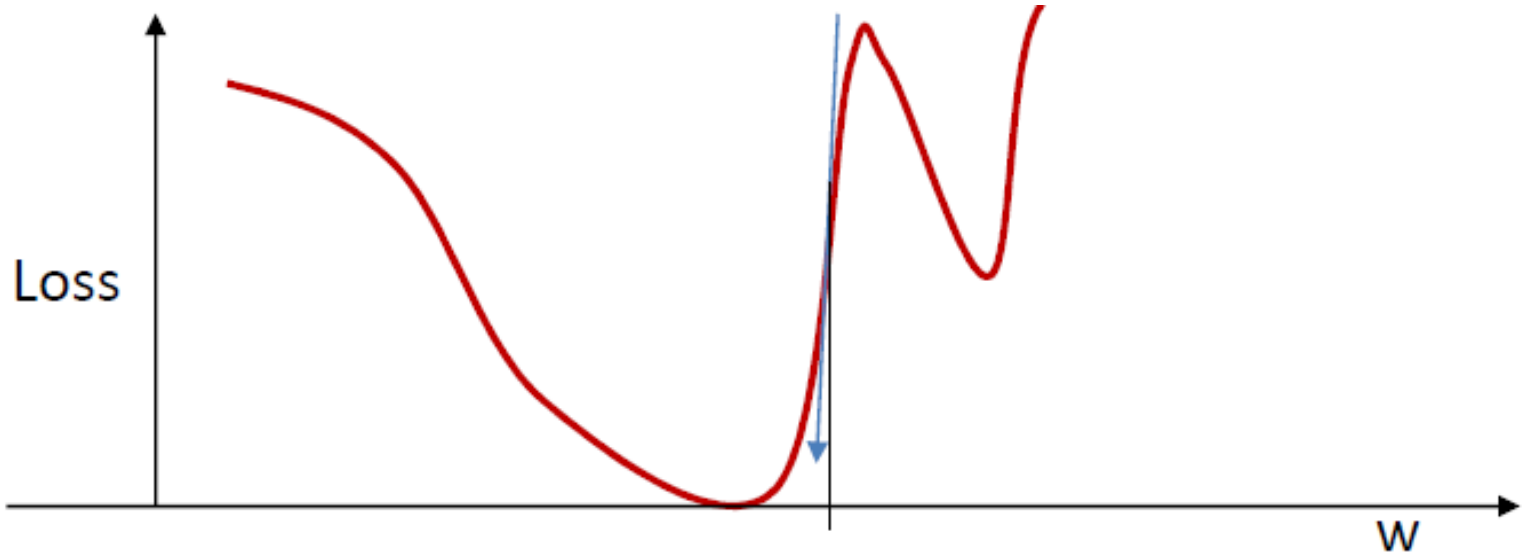
Gradient Clipping

if $\|\nabla f(x_t)\|_2 \geq$ threshold

$g_t \leftarrow \frac{\nabla f(x_t)}{\|\nabla f(x_t)\|_2} \cdot$ threshold

$x_{t+1} \leftarrow x_t - \eta g_t$

- The loss can occasionally lead to a steep descent
- This result in immediate instability
- If gradient norm bigger than a threshold, set the gradient to the threshold.

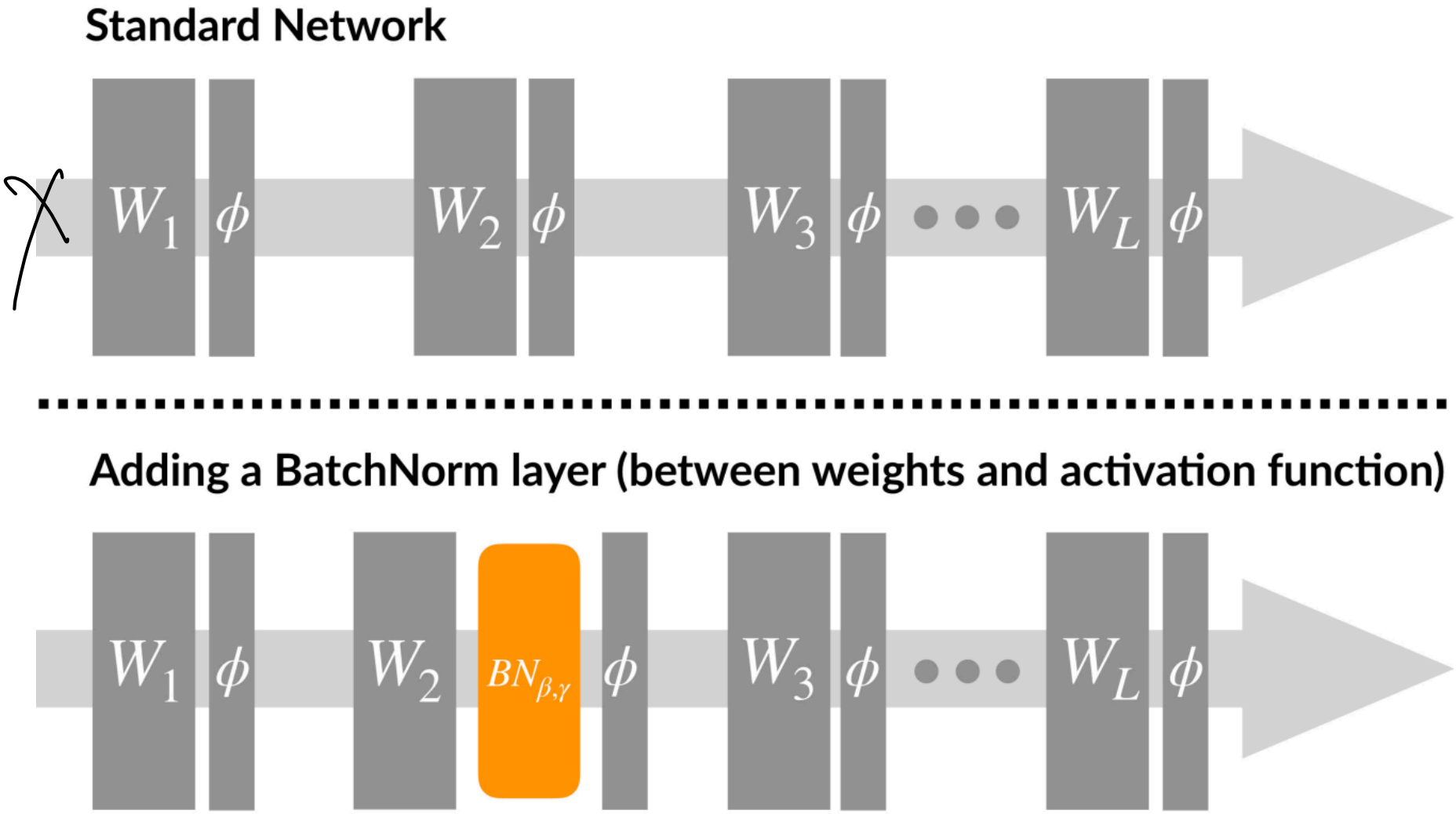


Batch Normalization (Ioffe & Szegedy, '14)

- Normalizing/whitening (mean = 0, variance = 1) the inputs is generally useful in machine learning.
  - Could normalization be useful at the level of hidden layers?
  - **Internal covariate shift**: the calculations of the neural networks change the distribution in hidden layers even if the inputs are normalized

- **Batch normalization** is an attempt to do that:
  - Each unit's **pre-activation** is normalized (mean subtraction, std division)
  - During training, mean and std is computed for each minibatch (can be backproped!

Batch Normalization (Ioffe & Szegedy, '14)

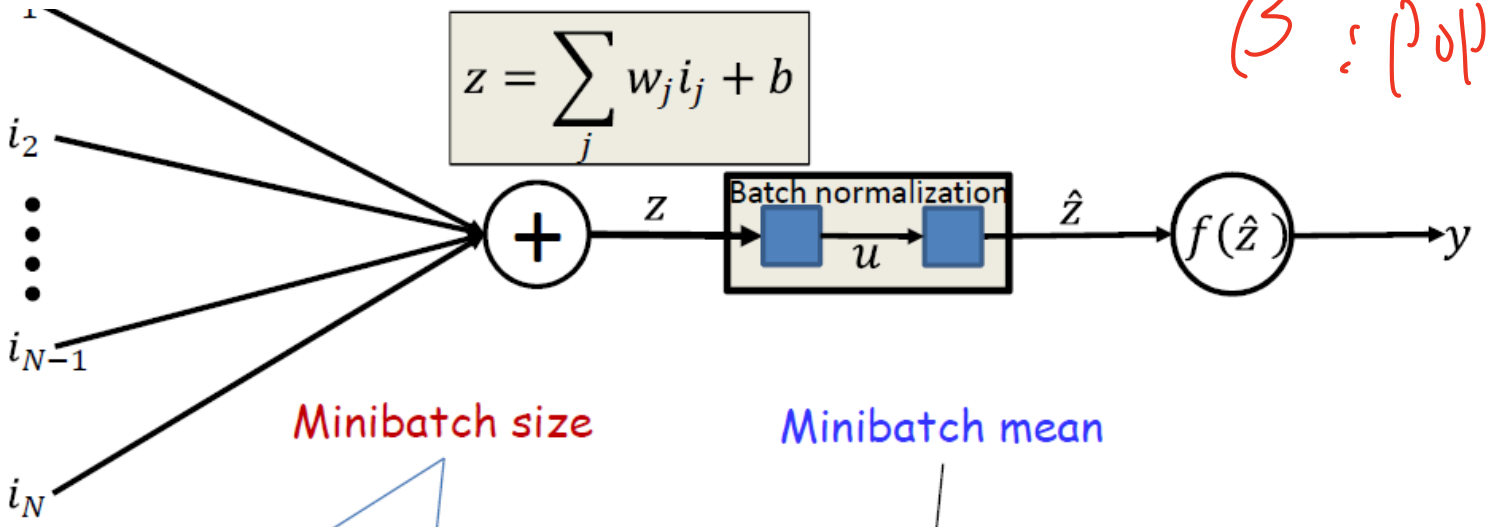


Batch Normalization (Ioffe & Szegedy, '14)

input $z_1, \ldots, z_B$

γ : population std.

β : population mean



Minibatch size

Minibatch mean

Minibatch standard deviation

$$\mu_B = \frac{1}{B}\sum_{i=1}^{B} z_i \qquad \sigma_B^2 = \frac{1}{B}\sum_{i=1}^{B}(z_i - \mu_B)^2$$

$$u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \qquad \hat{z}_i = \gamma u_i + \beta$$

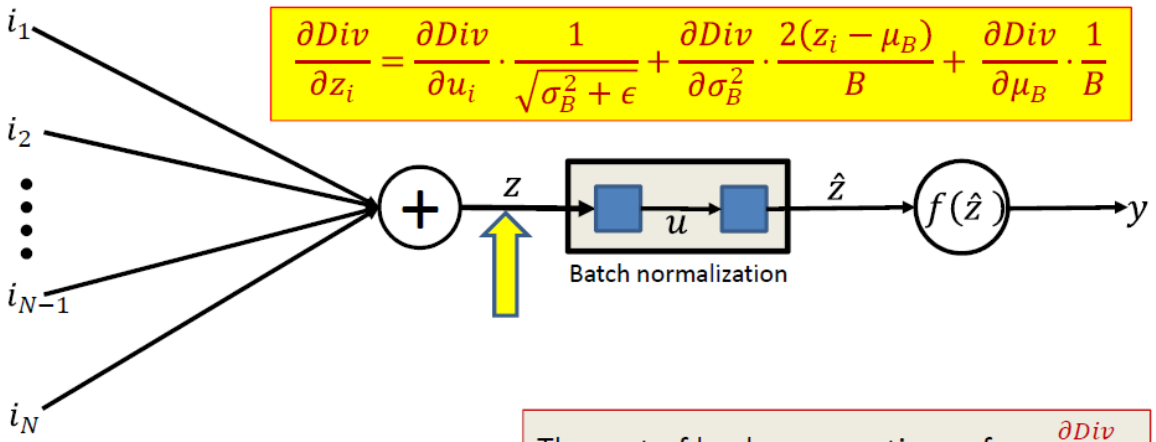
Batch Normalization (Ioffe & Szegedy, '14)

- BatchNorm at training time
  - Standard backprop performed for each single training data
  - Now backprop is performed over entire batch.

$$\frac{\partial Div}{\partial \sigma_B^2} = \frac{-1}{2}(\sigma_B^2 + \epsilon)^{-3/2} \sum_{i=1}^{B} \frac{\partial Div}{\partial u_i}$$

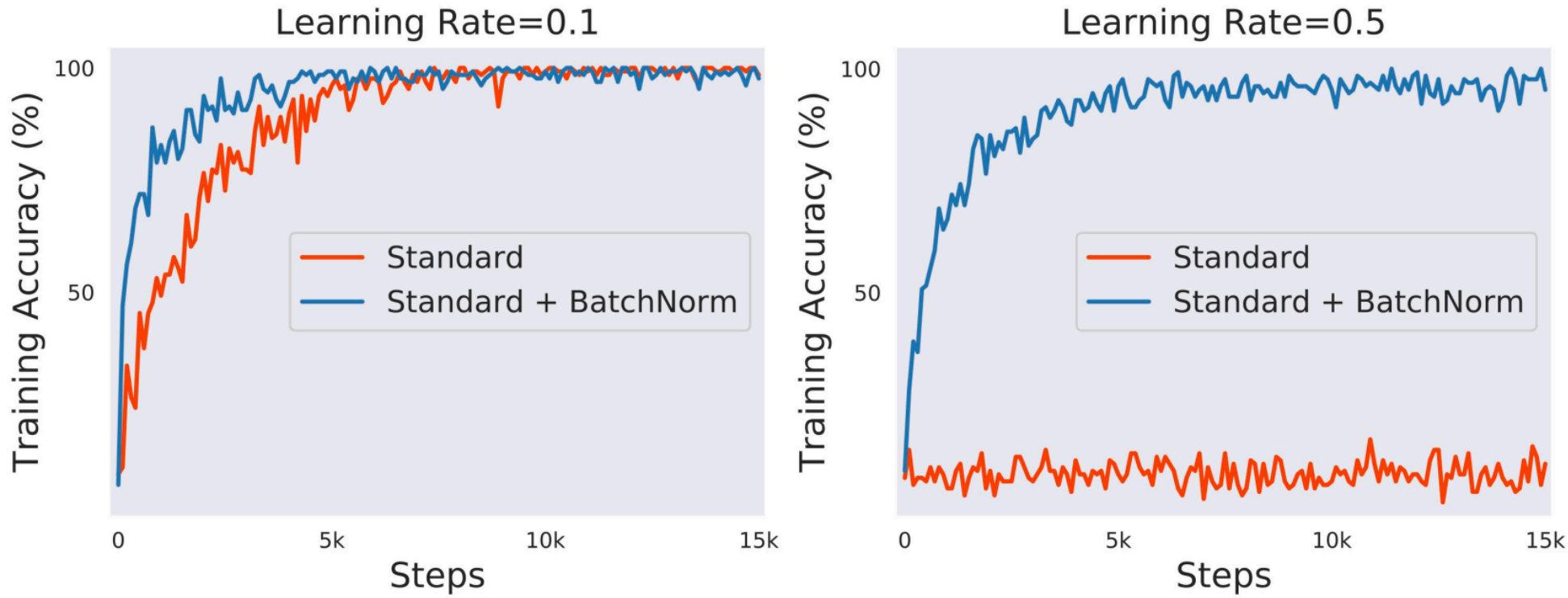
$$\frac{\partial Div}{\partial \mu_B} = \frac{-1}{\sqrt{\sigma_B^2 + \epsilon}} \sum_{i=1}^{B} \frac{\partial Div}{\partial u_i}$$

$$\frac{\partial Div}{\partial z_i} = \frac{\partial Div}{\partial u_i} \cdot \frac{1}{\sqrt{\sigma_B^2 + \epsilon}} + \frac{\partial Div}{\partial \sigma_B^2} \cdot \frac{2(z_i - \mu_B)}{B} + \frac{\partial Div}{\partial \mu_B} \cdot \frac{1}{B}$$



The rest of backprop continues from $\frac{\partial Div}{\partial z_i}$

Batch Normalization (Ioffe & Szegedy, ’14)



What is BatchNorm actually doing?

- May not due to covariate shift (Santurkar et al. ‘18):
  - Inject non-zero mean, non-standard covariance Gaussian noise after BN layer: removes the whitening effect
  - Still performs well.

- Only training β, γ with random convolution kernels gives non-trivial performance (Frankle et al. ’20)

- BN can use exponentially increasing learning rate! (Li & Arora ’19)

More normalizations

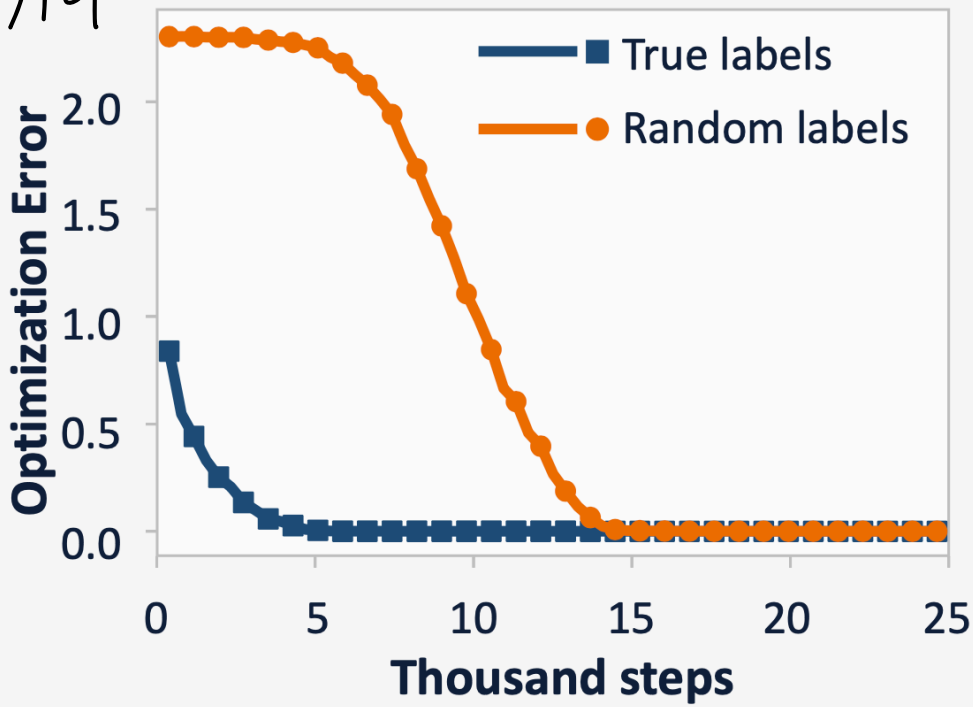
- Layer normalization (Ba, Kiros, Hinton, ’16)
  - Batch-independent
  - Suitable for RNN, MLP
- Weight normalization (Salimans, Kingma, ’16)
  - Suitable for meta-learning (higher order gradients are needed)
- Instance normalization (Ulyanov, Vedaldi, Lempitsky, ’16)
  - Batch-independent, suitable for generation tasks
- Group normalization (Wu & He, ‘18)
  - Batch-independent, improve BatchNorm for small batch size

Non-convex Optimization Landscape

Gradient descent finds global minima

Practice: gradient descent

$$\theta(t+1) \leftarrow \theta(t) - \eta \frac{\partial L(\theta(t))}{\partial \theta(t)}$$



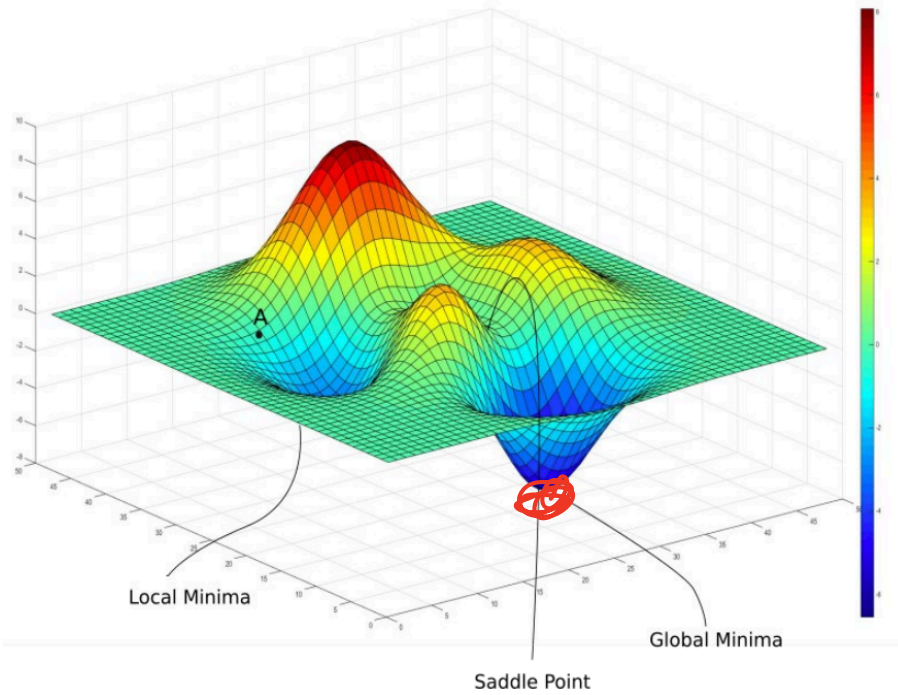
Optimization error -> **0** for both ***true labels*** and ***random labels*** !

Zhang Bengio Hardt Recht Vinyals 2017

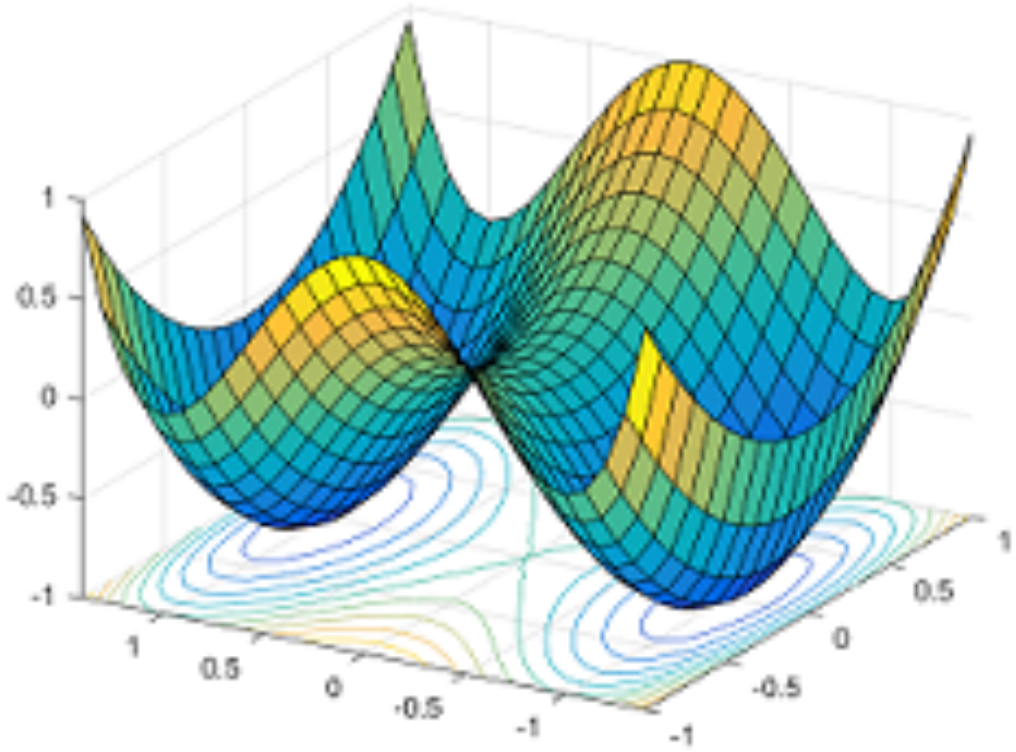
Understanding DL Requires Rethinking Generalization

Types of stationary points

- Stationary points: $x : \nabla f(x) = 0$
- Global minimum:
  $x : f(x) \leq f(x') \forall x' \in \mathbb{R}^d$
- Local minimum:
  $x : f(x) \leq f(x') \forall x' : \|x - x'\| \leq \epsilon$
- Local maximum:
  $x : f(x) \geq f(x') \forall x' : \|x - x'\| \leq \epsilon$
- Saddle points: stationary points that are not a local min/max

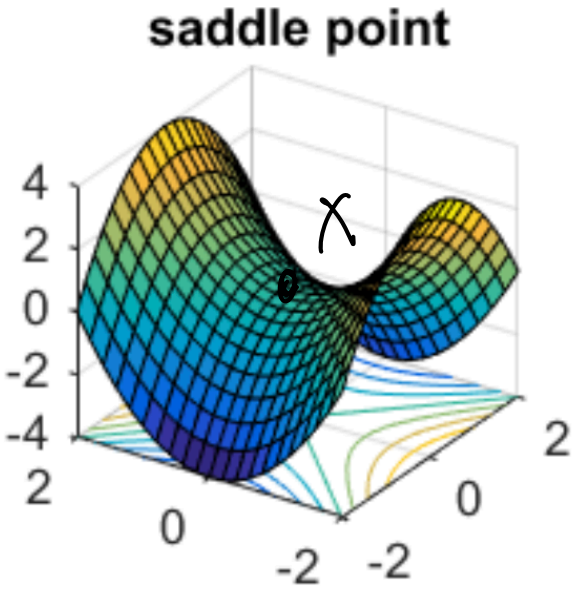


Landscape Analysis



- All local minima are global!
- Gradient descent can escape saddle points.

Strict Saddle Points (Ge et al. '15, Sun et al. '15)



local min: x

$\nabla^2 f(x)$ is positive def

- Strict saddle point: a saddle point and $\lambda_{\min}(\underbrace{\nabla^2 f(x)}_{A}) < 0$

$\min -\frac{1}{2} x A^T x$, $\nabla f(x) = Ax$

$|v^T x_{t+1}| = v^T(x_t - \eta A x_t)$

$= v^T x_t - \eta \lambda_{\min} v^T x_t$

$= (1 - \eta \lambda_{\min}) |v^T x_t|$

> 1

$x = 0$ stationary point

v: eigen vector (corresponds to smallest eigenvalue)

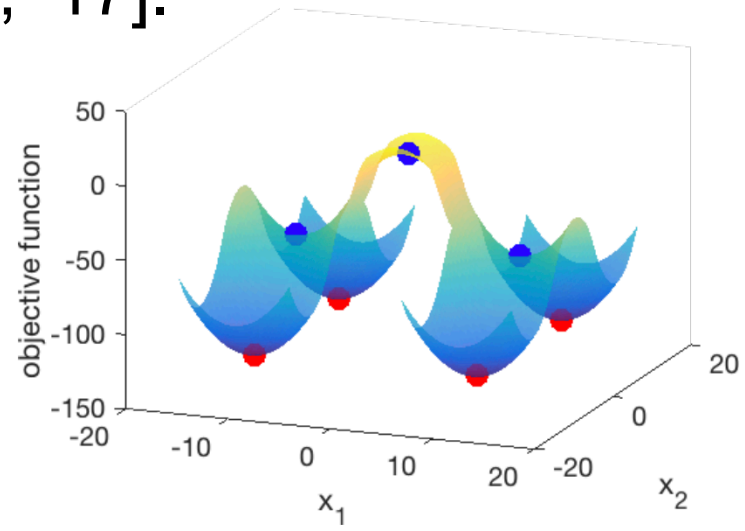
Escaping Strict Saddle Points

- Noise-injected gradient descent can escape strict saddle points in polynomial time [Ge et al., ’15, Jin et al., ’17].

- Randomly initialized gradient descent can escape all strict saddle points asymptotically [Lee et al., ’15].
  - Stable manifold theorem.

- Randomly initialized gradient descent can take exponential time to escape strict saddle points [Du et al., ’17].

If 1) all local minima are global, and 2) are saddle points are strict, then noise-injected (stochastic) gradient descent finds a global minimum in polynomial time



What problems satisfy these two conditions

- Matrix factorization
- Matrix sensing
- Matrix completion
- Tensor factorization
- Two-layer neural network with quadratic activation

What about neural networks?

- Linear networks (neural networks with linear activations functions): all local minima are global, but there exists saddle points that are not strict [Kawaguchi ’16].

- Non-linear neural networks with:
  - Virtually any non-linearity,
  - Even with Gaussian inputs,
  - Labels are generated by a neural network of the same architecture,

There are many bad local minima [Safran-Shamir ’18, Yun-Sra-Jadbaie ’19].