Optimization Methods for Deep Learning

Gradient descent for non-convex optimization

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

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

Converging to stationary points

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

Gradient Descent for Quadratic Functions

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

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

$$\|x_{t+1}\|_2 = \|x_t - \eta A x_t\|_2 = \|(I - \eta A) x_t\|_2$$

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

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

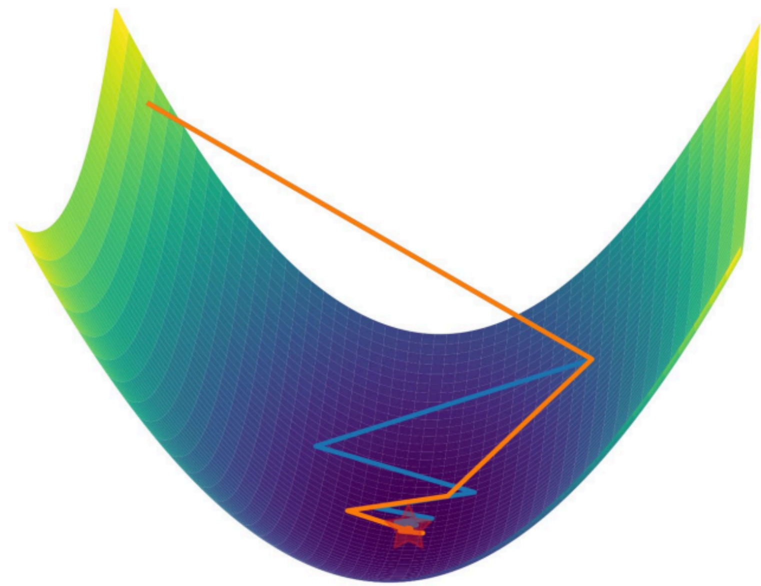
Momentum: Heavy-Ball Method (Polyak '64)

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

Problem: $\min_x f(x)$

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

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



Momentum: Nesterov Acceleration (Nesterov '89)

Problem: $\min_x f(x)$

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

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

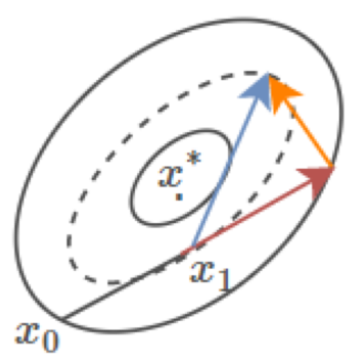
For general convex

$\sqrt{K} \log\left(\frac{1}{\varepsilon}\right)$

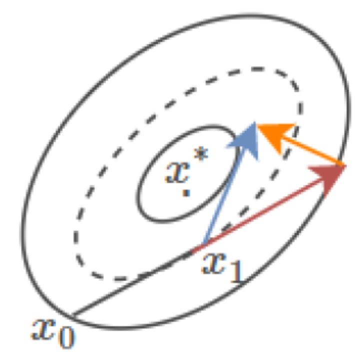
First order: only uses gradient

$\Omega\left(\sqrt{K} \log\left(\frac{1}{\varepsilon}\right)\right)$

Polyak's Momentum



Nesterov Momentum



Newton's Method

$x_t \in \mathbb{R}^d$ $\quad \nabla^2 f(x_t) \to O(d^2)$

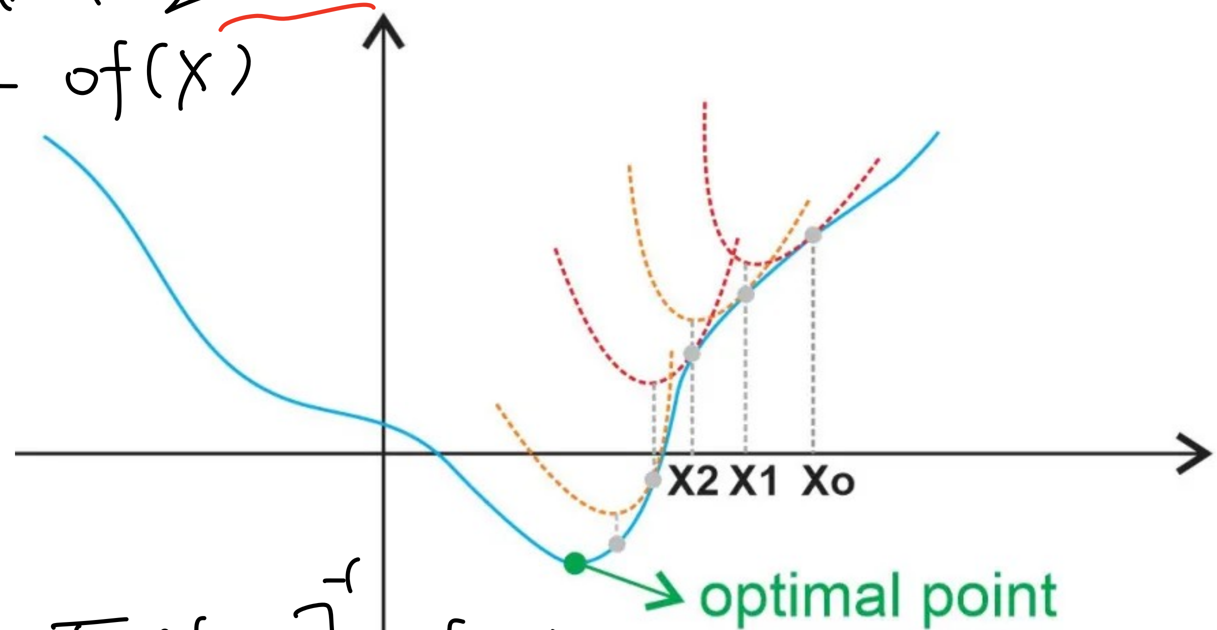
$\Rightarrow O(d^3)$

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

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

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

$\Rightarrow$ minimizer: $\Delta = -\nabla f(x)$



- Newton:

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

$\Rightarrow$ minimizer: $\Delta = -[\nabla^2 f(x)]^{-1}\nabla f(x)$

$O(\log\log(\frac{1}{\epsilon}))$ for convex function

• no κ

AdaGrad (Duchi et al. '11)

per-coordinate learning rate

BFGS

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

pre-conditioning

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

usually diagonal: $O(d)$

ϵI: regularization, $I = \begin{pmatrix} 1 & & 0 \\ & 1 & \\ 0 & & 1 \end{pmatrix}$

- Default hyper-parameter: η: 0.01, $\epsilon = 10^{-8}$
- Learning rate can be very small

RMSProp (Hinton et al. ’12)

AdaGrad: separate learning rate for every parameter

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

RMSProp: exponential weighting of gradient norms

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

exponential weighting

AdaDelta (Zeiler '12)

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:

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

$\frac{\partial f}{\partial x} \Big/ \left(\frac{\partial^2 f}{\partial x^2}\right)^{1/2}$: unitless term

(unit of x)^2

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

Newton : $\Delta x \propto \frac{\partial f}{\partial x} \Big/ \frac{\partial^2 f}{\partial x^2} \propto$ unit of x

Adam (Kingma & Ba '14)

Momentum:

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

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

momentum

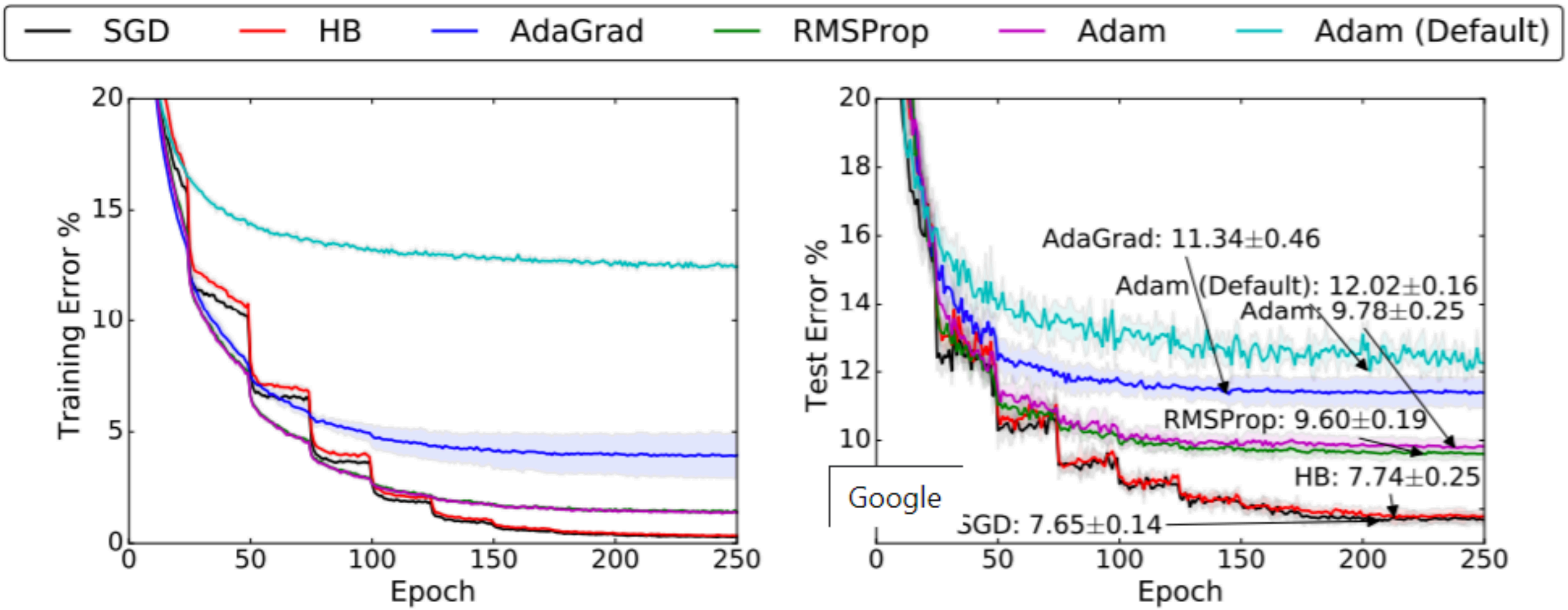
$v_{t+1} = \beta_1 v_t + (1-\beta_1) \nabla f(x_t)$

$(G_{t+1})_{ii} = \beta_2(G_t)_{ii} + (1-\beta_2)(\nabla f(x_t)_i)^2$

$x_{t+1} = x_t - \eta(G_{t+1} + \epsilon I)^{-1/2} v_{t+1}$

Default choice nowadays.

Are these actually useful



(a) CIFAR-10 (Train) **(b)** CIFAR-10 (Test)

Figure 1: Training (left) and top-1 test error (right) on CIFAR-10. The annotations indicate where the best performance is attained for each method. The shading represents ± one standard deviation computed across five runs from random initial starting points. In all cases, adaptive methods are performing worse on both train and test than non-adaptive methods.

Wilson, Roelofs, Stern, Srebro, Recht '18

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

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

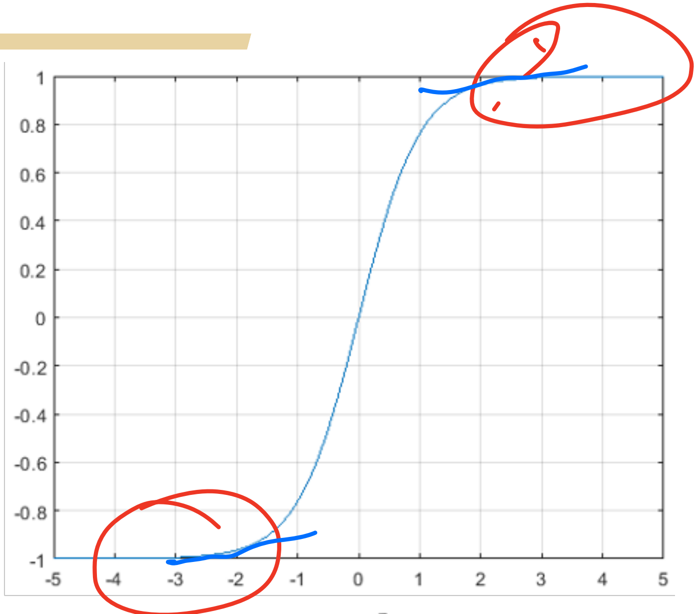
$$\frac{\partial f}{\partial W_h} = (W_{H+1} A_H \cdots W_{h+1} A_h)^T (A_{h-1} W_{h-1} \cdots W_1 x)$$

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

① magnitude: small → exp small; large → exp large

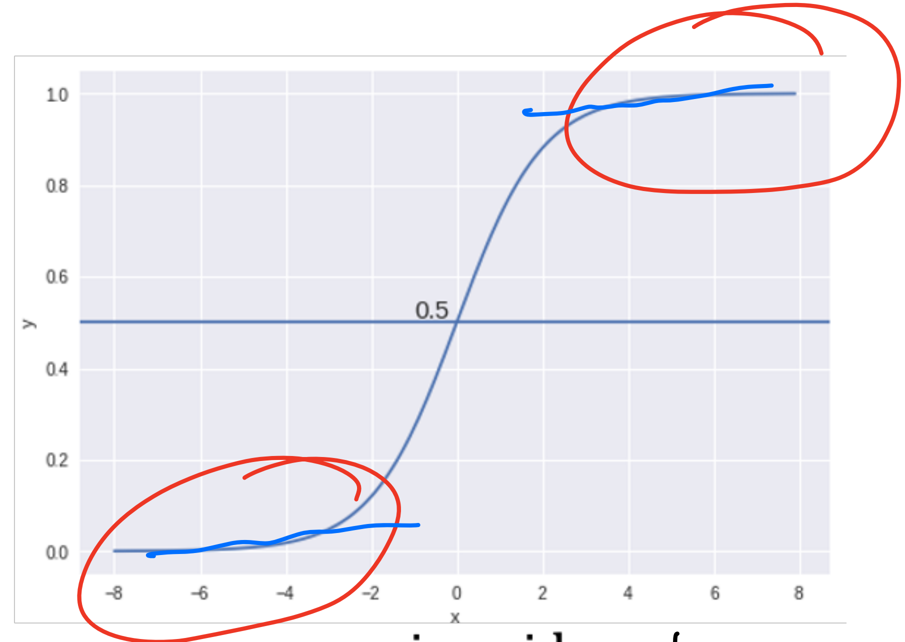
② space of matrices matter: A B

Activation Functions



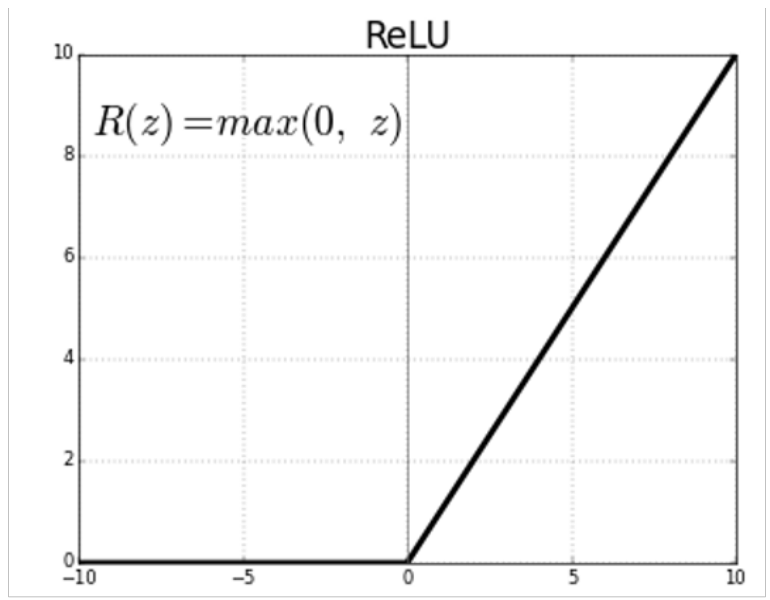
tanh

$$\tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$



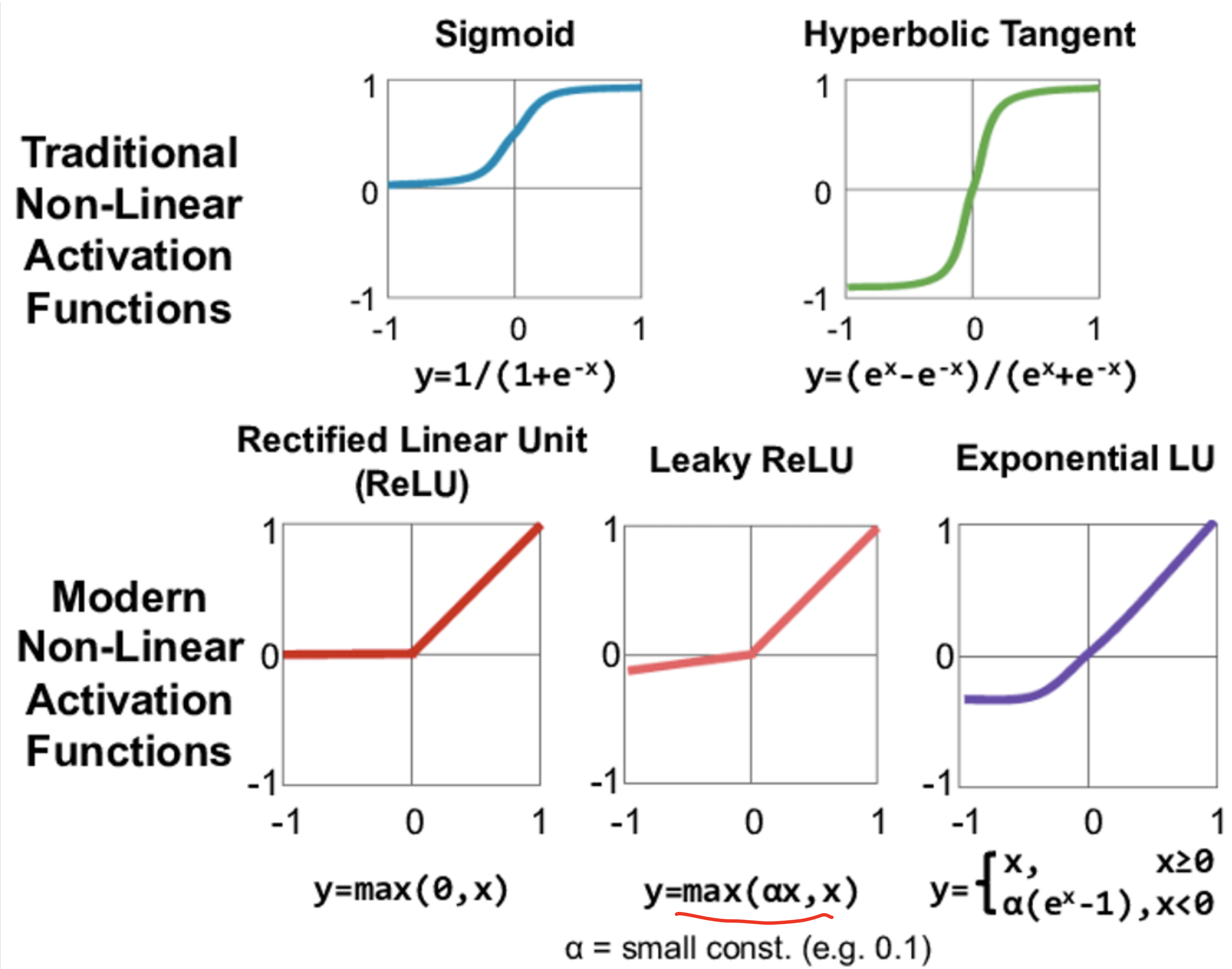
sigmoid

$$\frac{1}{1+e^{-z}}$$



Rectified Linear United

Activation Function



Initialization

- Zero-initialization → all gradient 0 → random init
- Large initialization
- Small initialization

→ scaling ⇄ gradient explosion / gradient vanishing

$N(0, \sigma^2 I)$

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

Xavier Initialization (Glorot & Bengio, '10)

- $W_{ij}^{(h)} \sim \text{Unif}\left[-\frac{\sqrt{6}}{\sqrt{d_h + d_{h+1}}}, \frac{\sqrt{6}}{\sqrt{d_h + d_{h+1}}}\right]$
- $b^{(h)} = 0$
- Experiments (tanh activation)

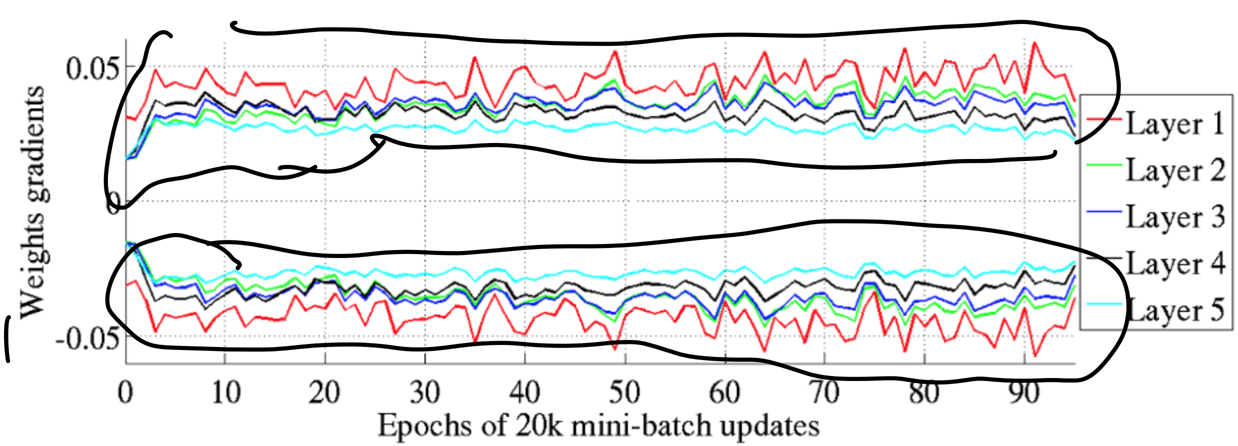
$W^{(h)} \in \mathbb{R}^{d_{h+1} \times d_h}$

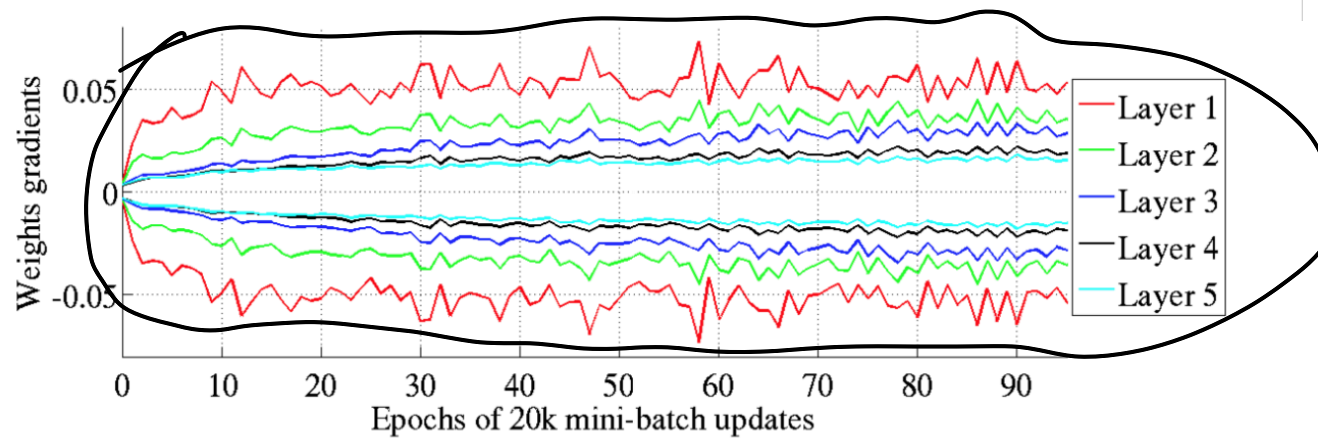
d_h : fan-in

d_{h+1} : fan-out

$\mathcal{N}\left(0, \frac{2}{d_h + d_{h+1}}\right)$

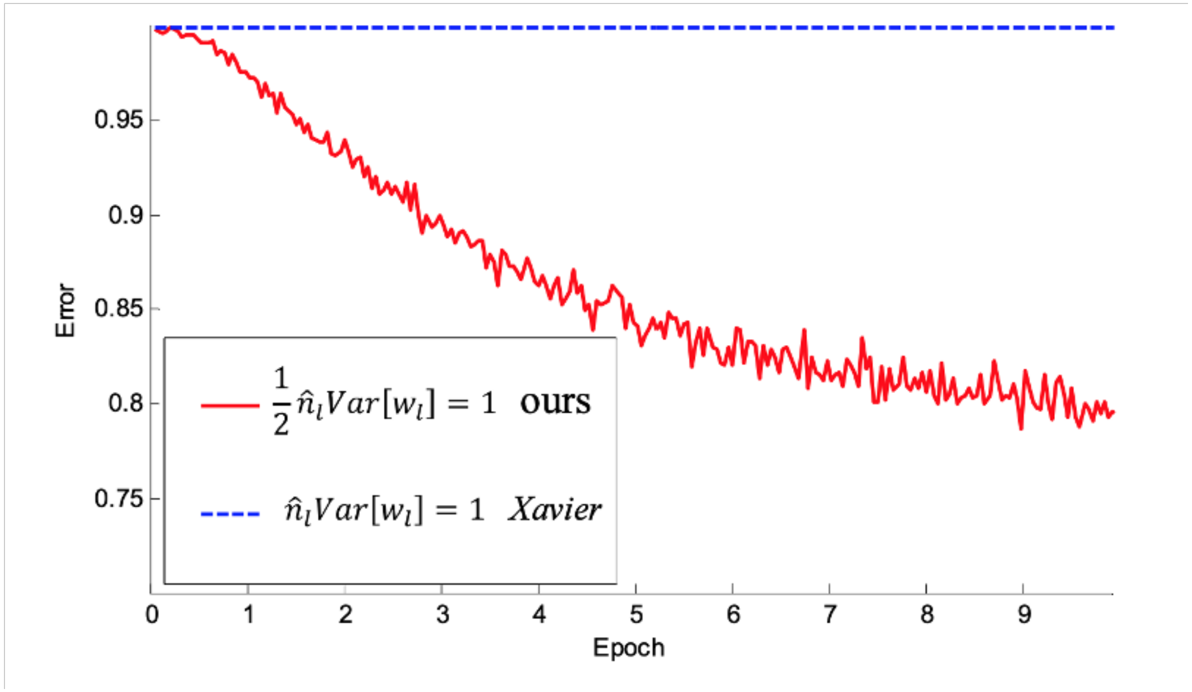
$Var(w_{ij}^{(h)}) = \frac{1}{12} \frac{(2\sqrt{6})^2}{(\sqrt{d_h + d_{h+1}})^2} = \frac{2}{d_h + d_{h+1}}$





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



Kaiming Initialization (He et al. '15)

Each layer $\quad z^{h} = W^{h} \cdot x^{h}$

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

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

W_{ij}: 0-mean $\longrightarrow$ $\mathbb{E}[z_i^{h}] = 0$

$$\mathrm{Var}(z_i^{h}) = d_h \mathrm{Var}(W_{ij}^{h} \cdot x_j^{h})$$

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

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

Kaiming Initialization (He et al. '15)

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

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

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

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

(Symmetry of init)

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

Kaiming Initialization (He et al. '15)

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

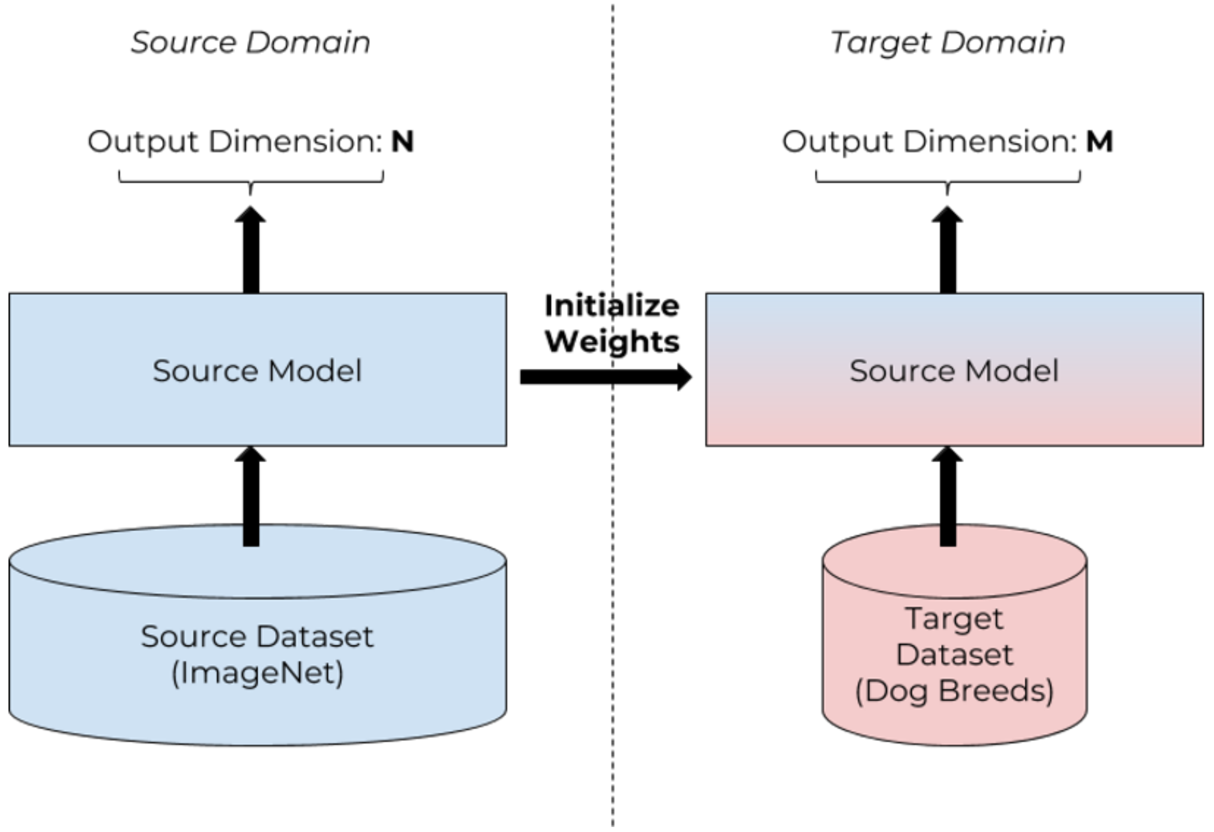
$$d_h \, \mathrm{Var}(w_{ij}^h) \cdot \frac{1}{2} \mathrm{Var}(z_j^{h-1}) = \mathrm{Var}(z_j^{h-1})$$

$$\mathrm{Var}(w_{ij}^h) = \frac{2}{d_h}$$

$$\mathrm{Var}(z^{H+1}) = \mathrm{Var}(z^1) \underbrace{\left(\prod_{h=1}^{H} \frac{d_h}{2} \mathrm{Var}(w_{ij}^h) \right)}_{1}$$

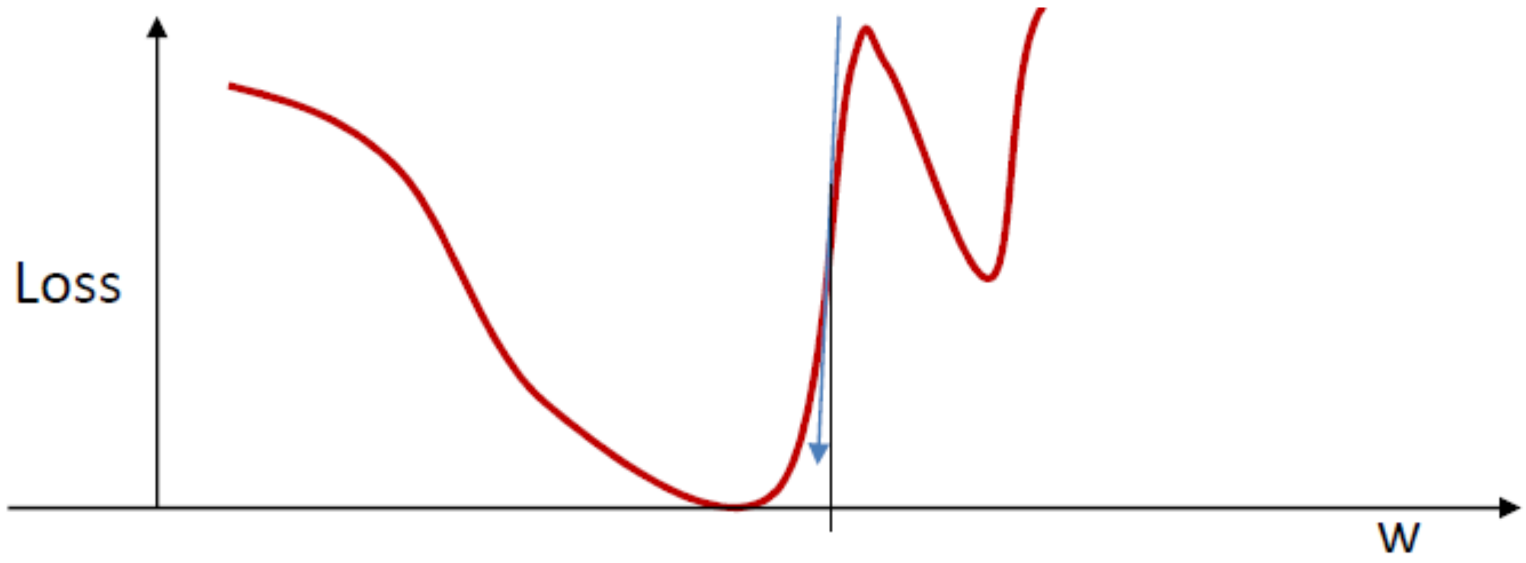
Initialization by Pre-training

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



Gradient Clipping

- 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.

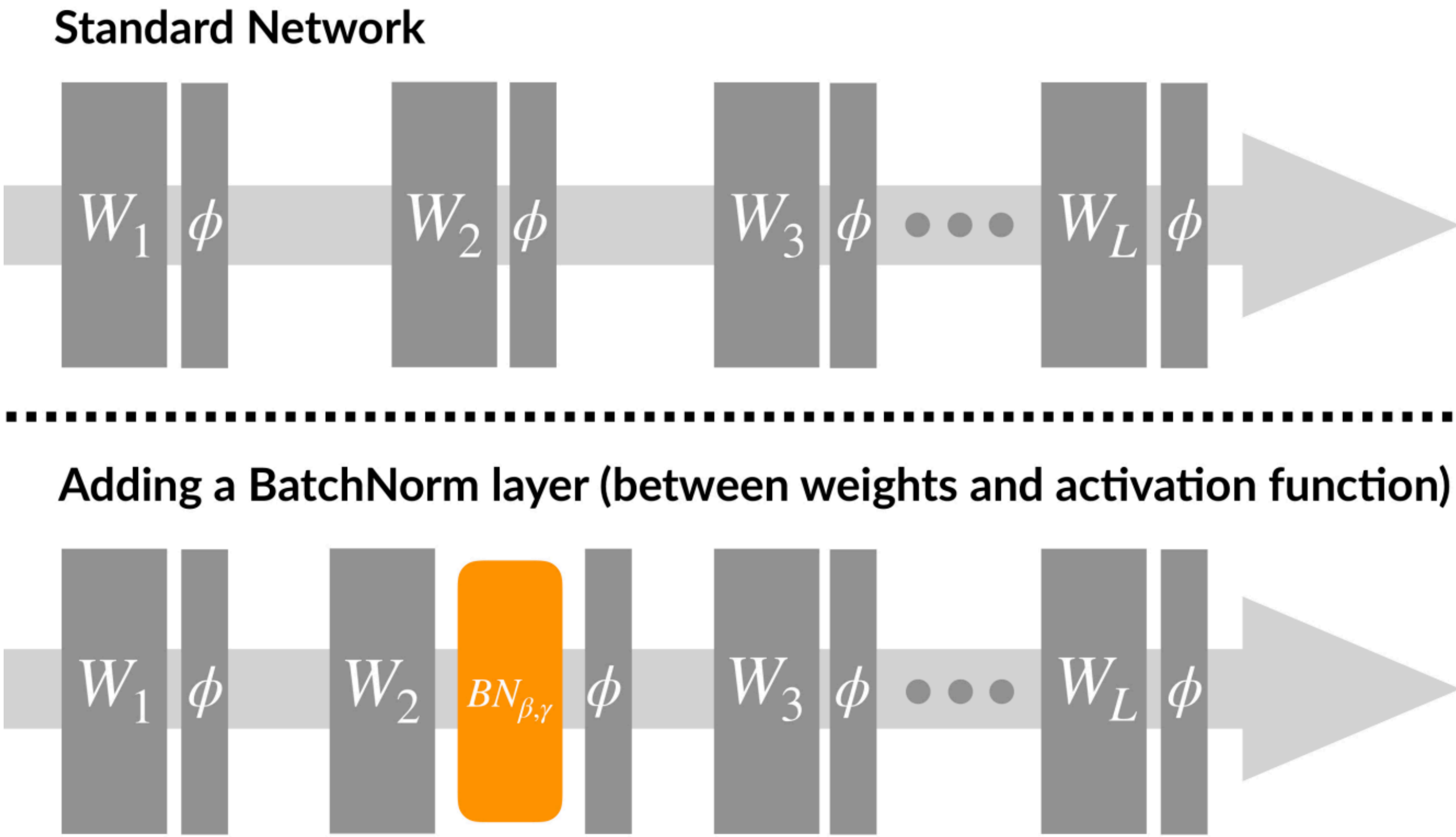


Batch Normalization (Ioffe & Szegedy, ’14)

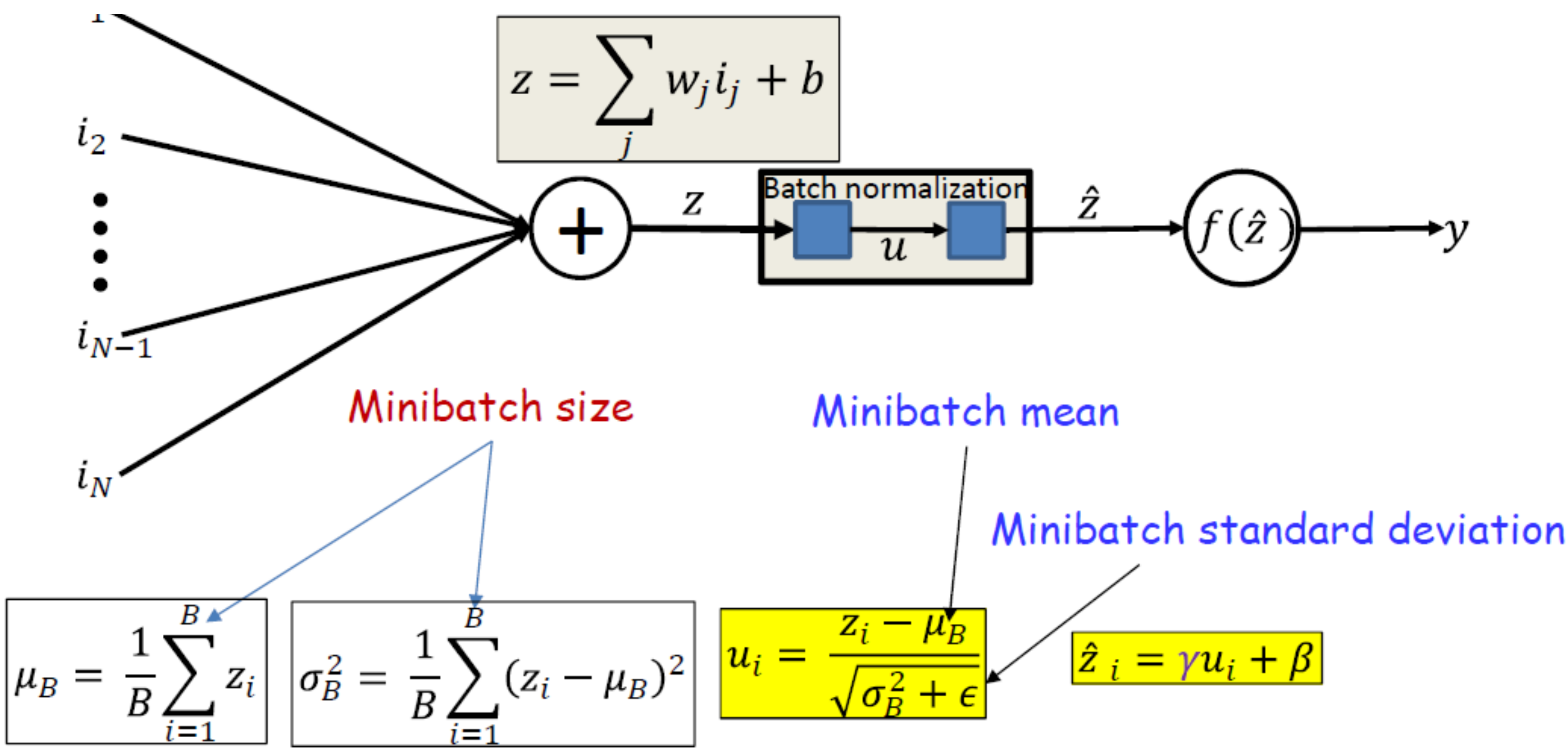
- Normalizing/whitening (mean = 0, variance = 1) the inputs is generally useful in machine learning.
  - Could normalization be useful at the level of hidden layers?
  - **Internal covariate shift**: the calculations of the neural networks change the distribution in hidden layers even if the inputs are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



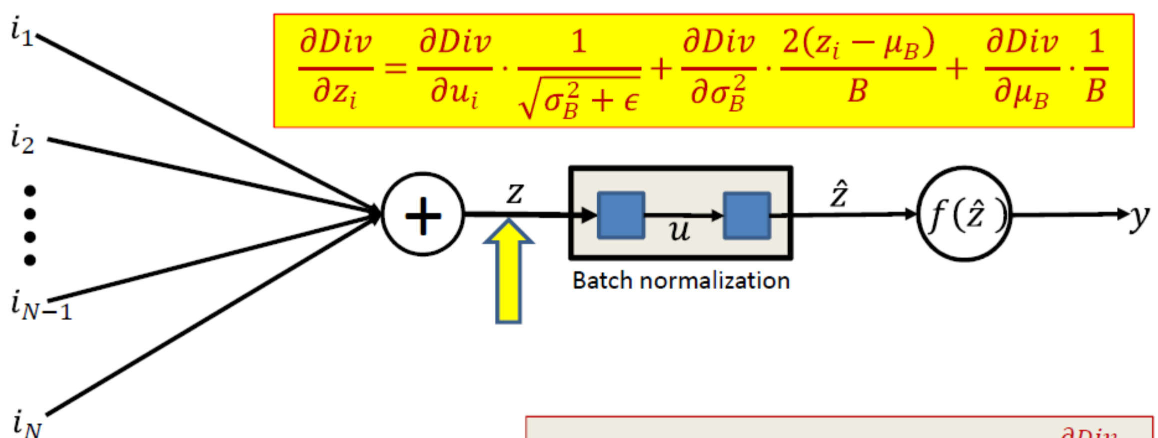
Batch Normalization (Ioffe & Szegedy, '14)

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

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

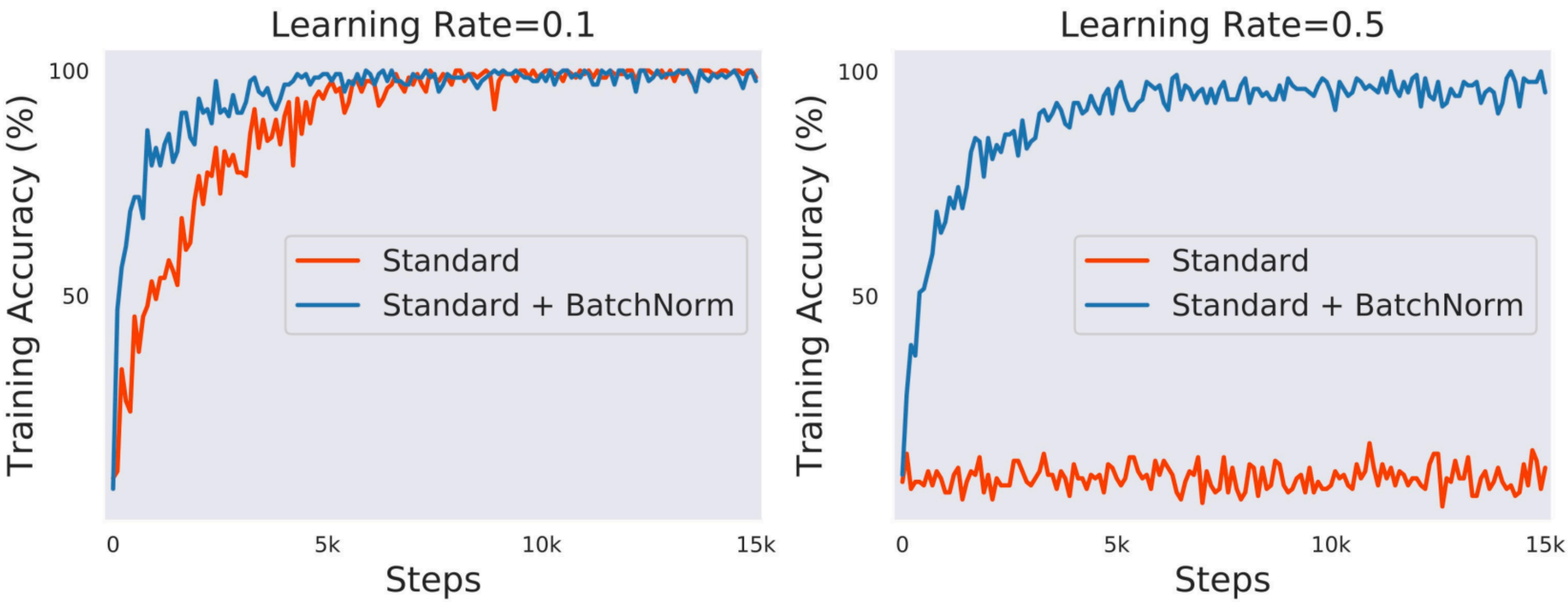
$$\frac{\partial Div}{\partial \mu_B} = \frac{-1}{\sqrt{\sigma_B^2 + \epsilon}} \sum_{i=1}^{B} \frac{\partial Div}{\partial u_i}$$

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



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

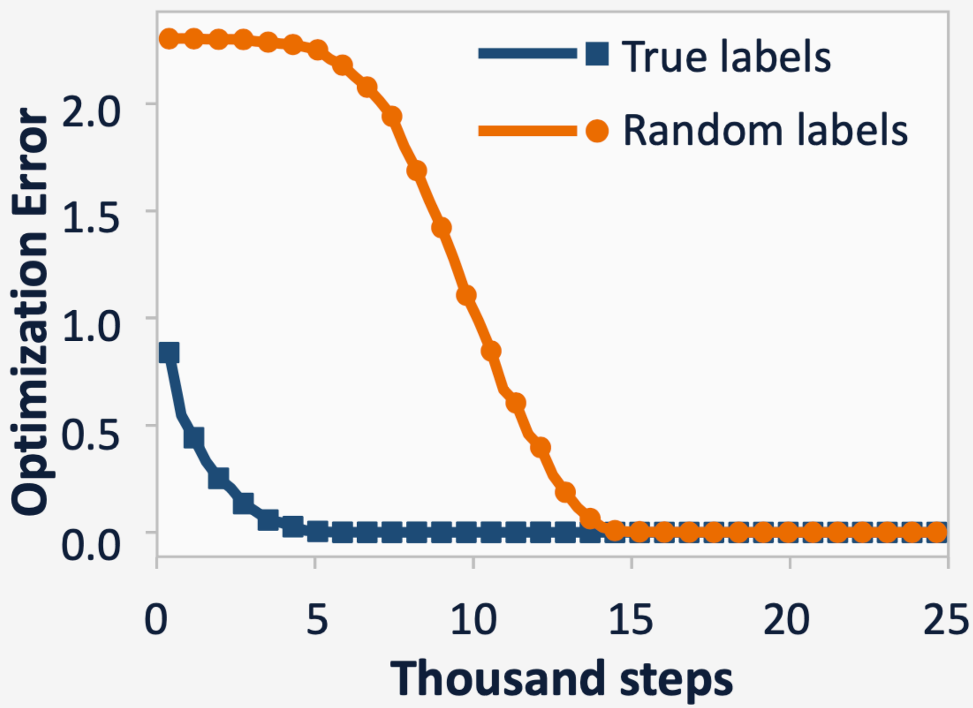
- Layer normalization (Ba, Kiros, Hinton, ’16)
  - Batch-independent
  - Suitable for RNN, MLP
- Weight normalization (Salimans, Kingma, ’16)
  - Suitable for meta-learning (higher order gradients are needed)
- Instance normalization (Ulyanov, Vedaldi, Lempitsky, ’16)
  - Batch-independent, suitable 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

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



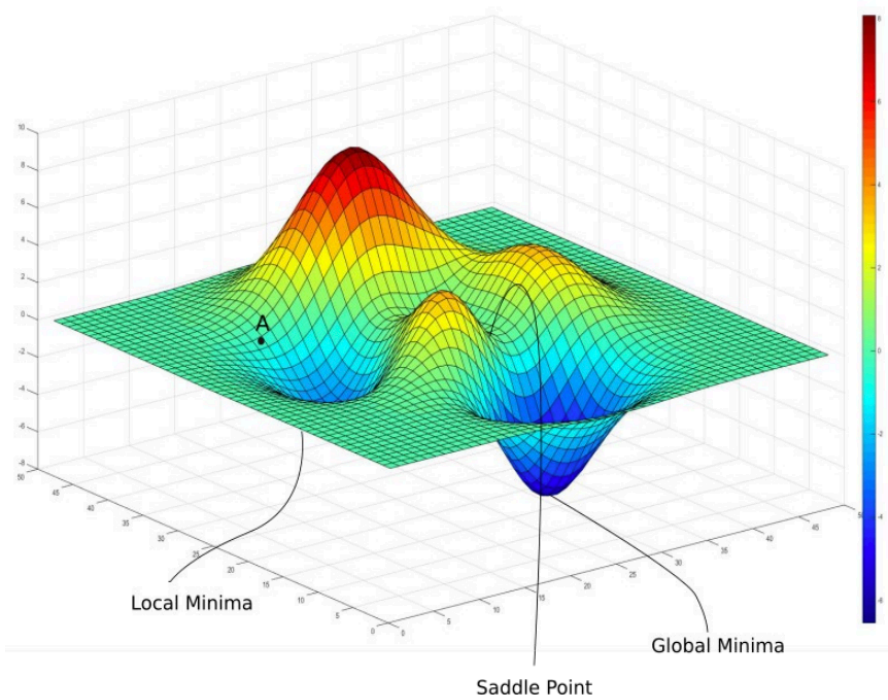
Optimization error **-> 0** for both ***true labels*** and ***random labels*** !

Zhang Bengio Hardt Recht Vinyals 2017

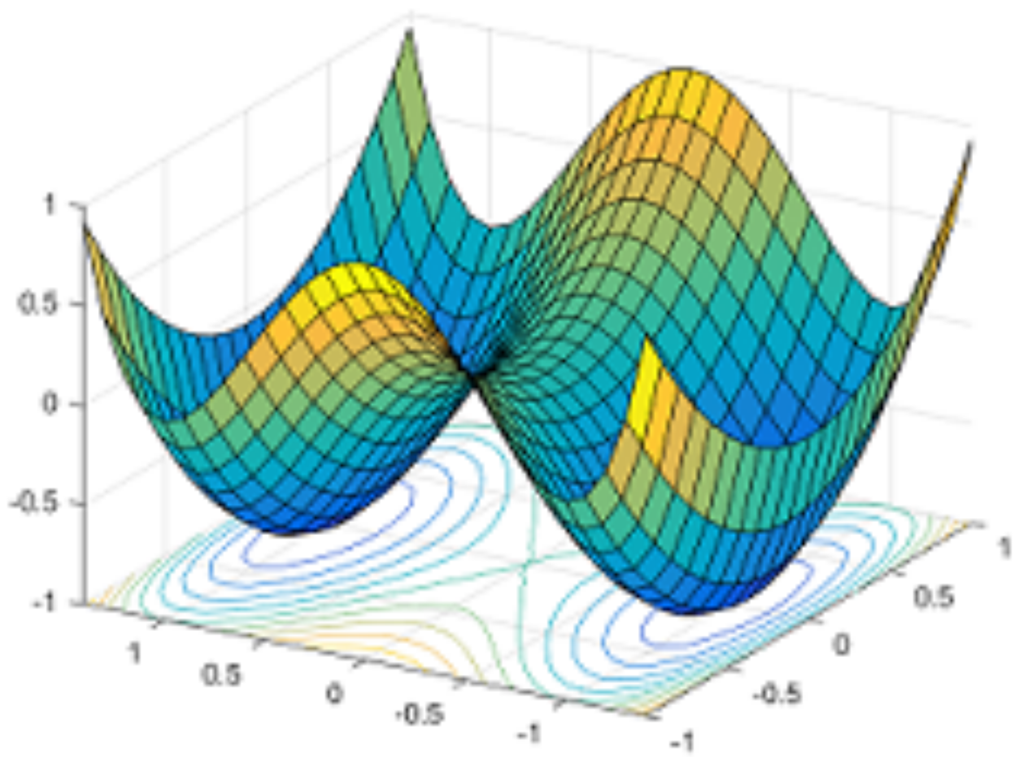
Understanding DL Requires Rethinking Generalization

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

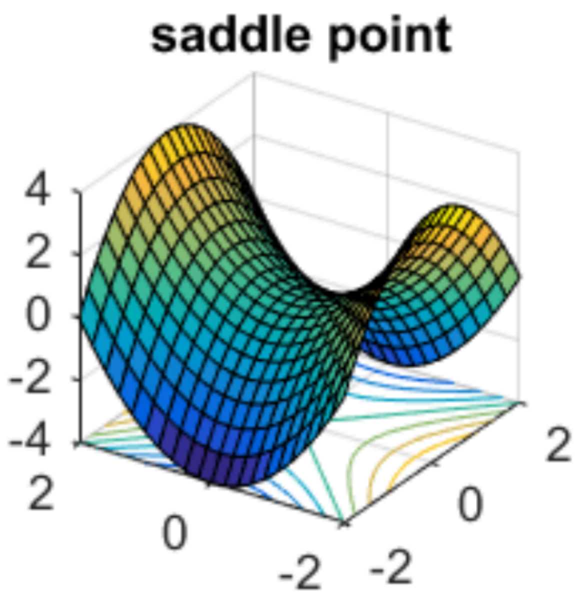


Landscape Analysis



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

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

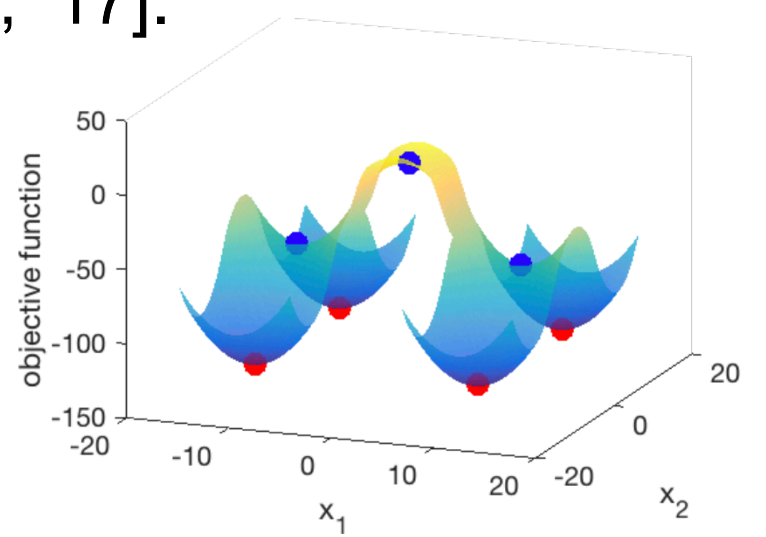


- Strict saddle point: a saddle point and $\lambda_{\min}(\nabla^2 f(x)) < 0$

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].