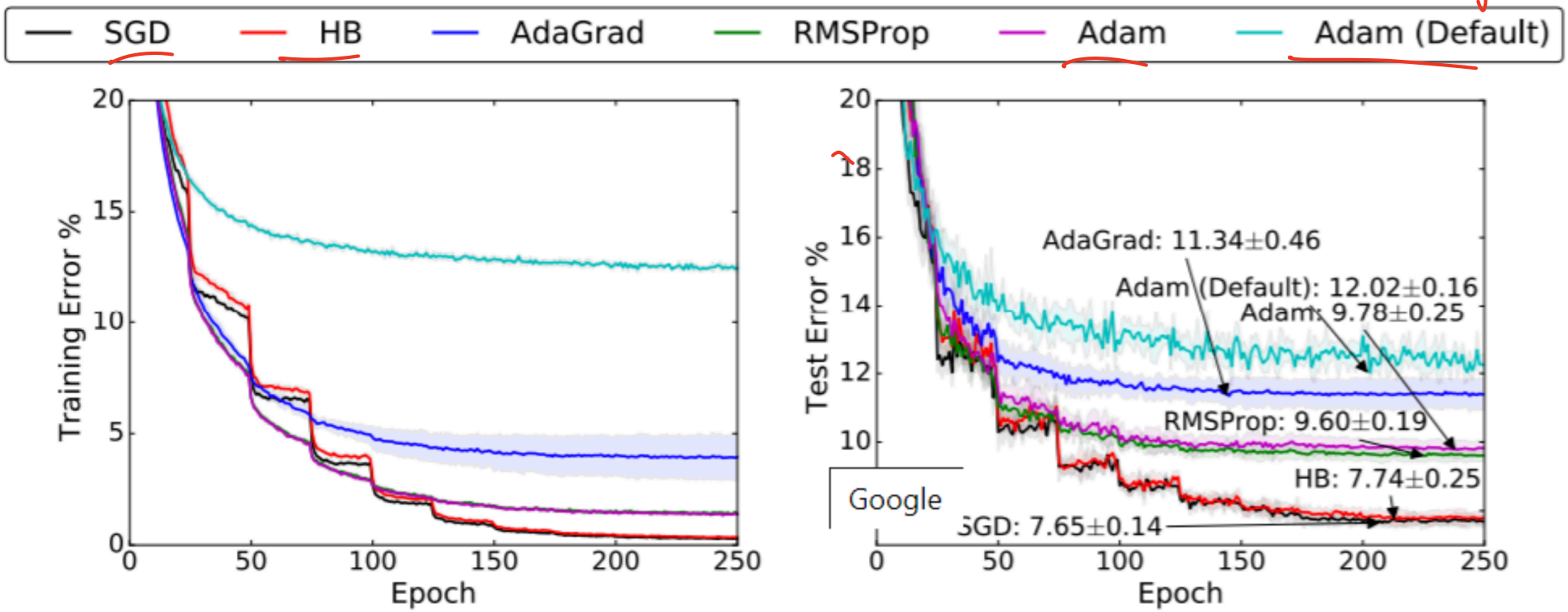
Are these actually useful



(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

$$f(x, w_1, \dots, w_{l+1}) = w_{l+1}\,\sigma(w_l \cdots \sigma(w_1 x) \cdots)$$

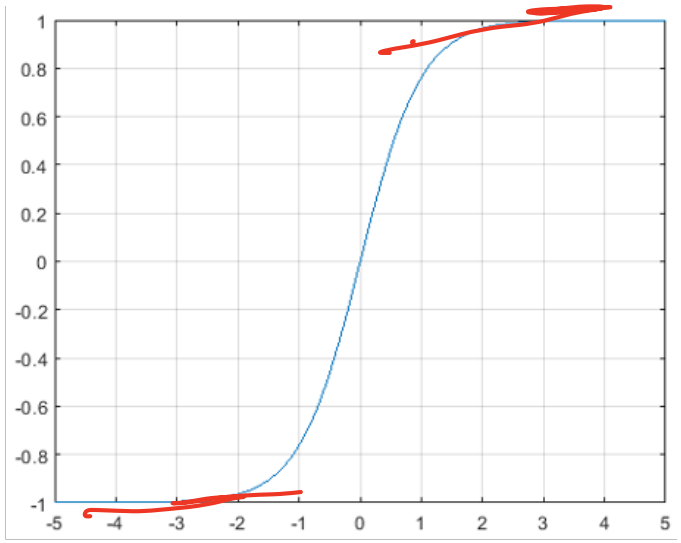
$$\frac{\partial f}{\partial w_n} = (w_{l+1} A_{l-1} \cdots w_{n+1} A_n)^T (A_{n-1} w_{n-1} \cdots w_1 x)$$

$$A_n = \mathrm{diag}(\sigma'(w_n\, \sigma(w_{n-1} \cdots \sigma(w_1 x) \cdots)))$$

(w_n, A_n) ① magnitude small → exp small
large → exp large

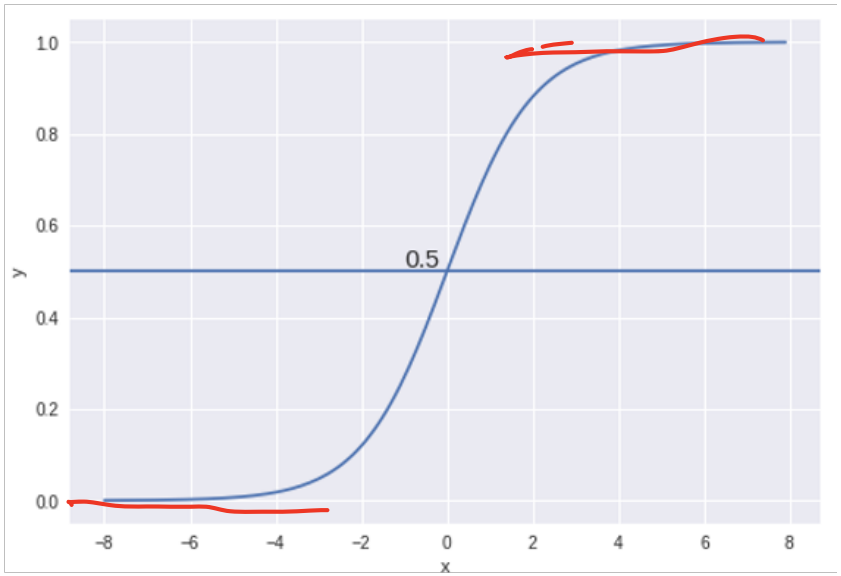
② space not align → product exp small

Activation Functions



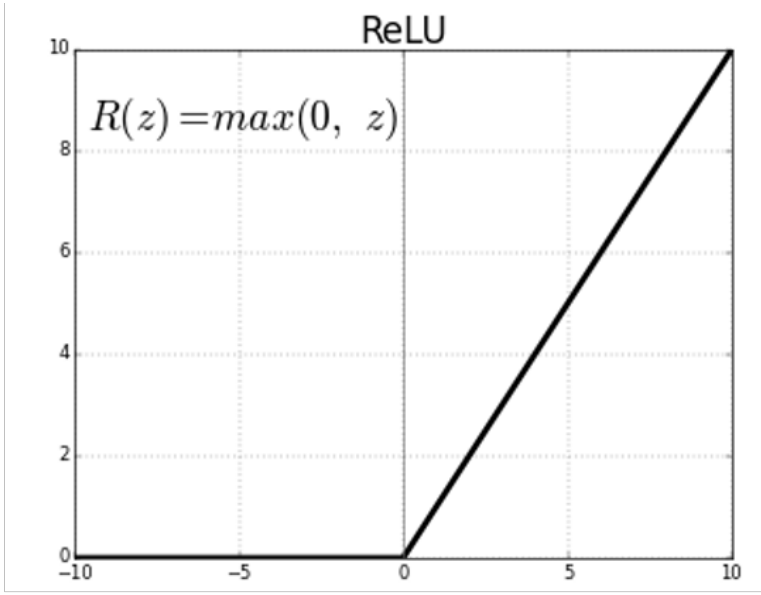
tanh

$g'(z)$
$\to 0$



sigmoid

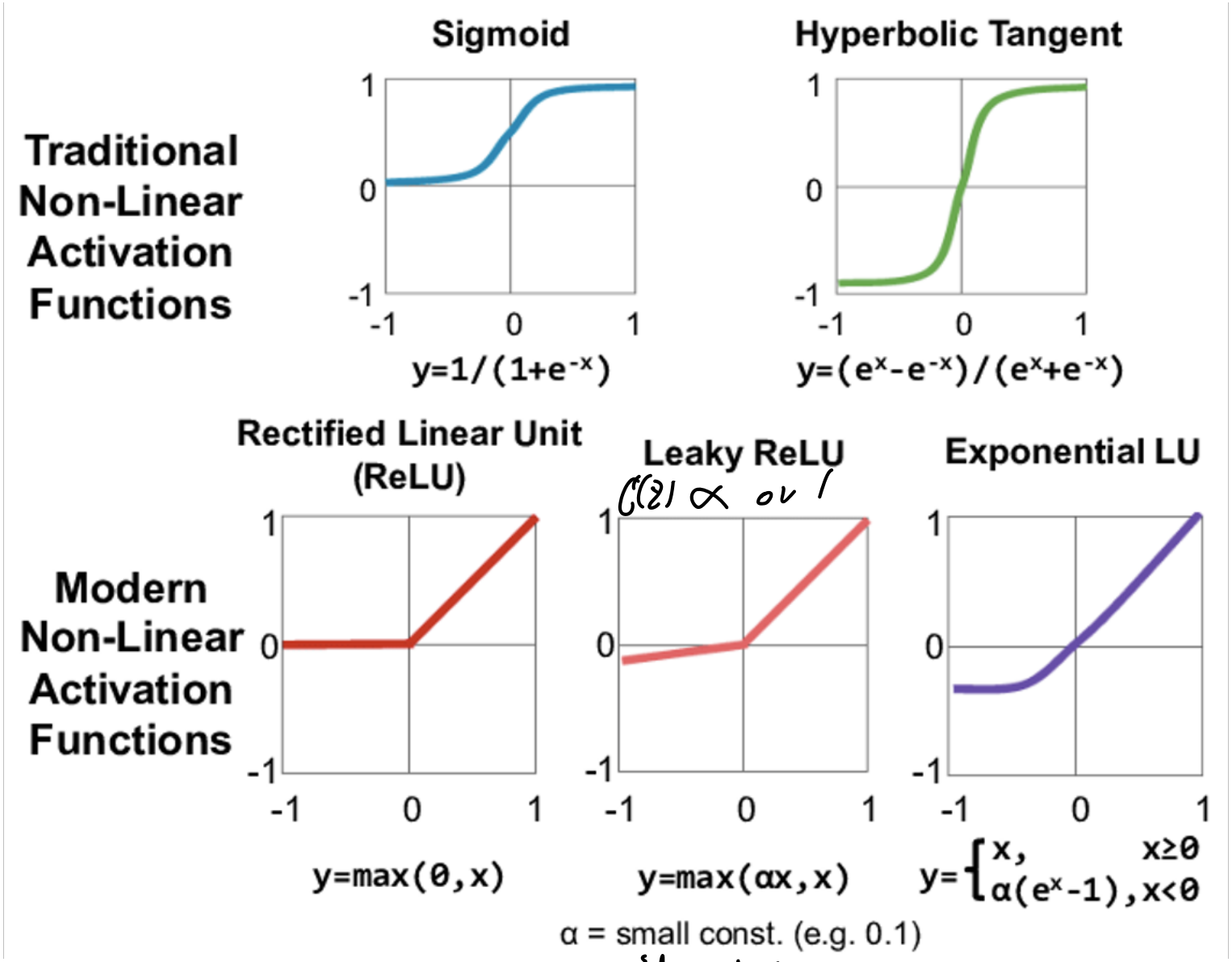
$$\tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$



Rectified Linear United

$g'(z) = 0$ or 1

Activation Function



convex: $\|x_T - x^*\|_2 \le \epsilon^{-T} \|x_0 - x^*\|_2$

Initialization

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

scaling → exp large / exp small

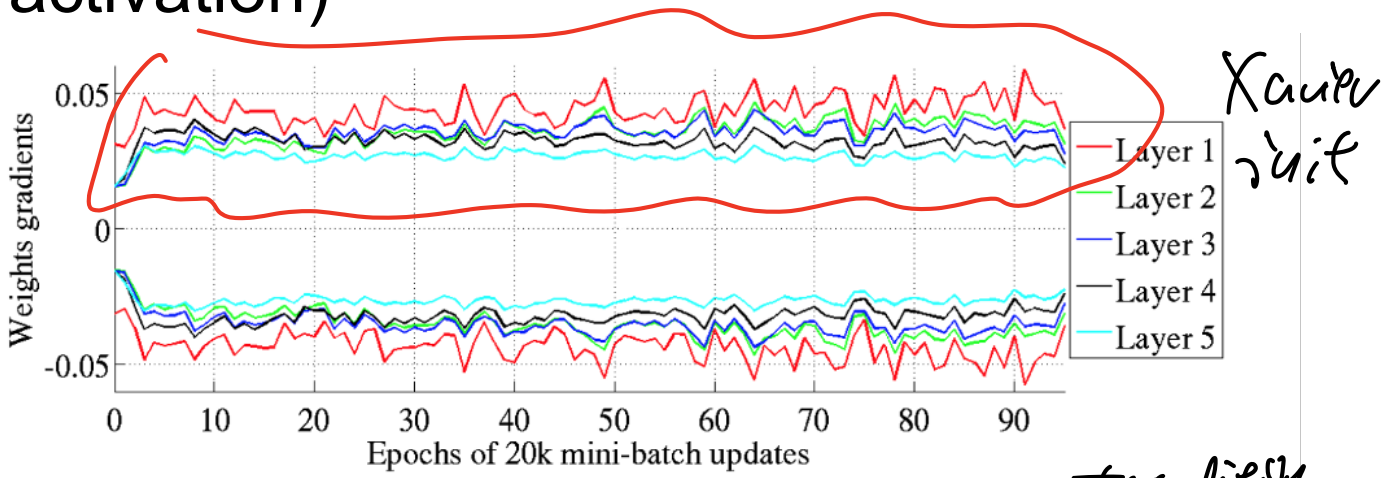
- Design principles:
  - Zero activation mean

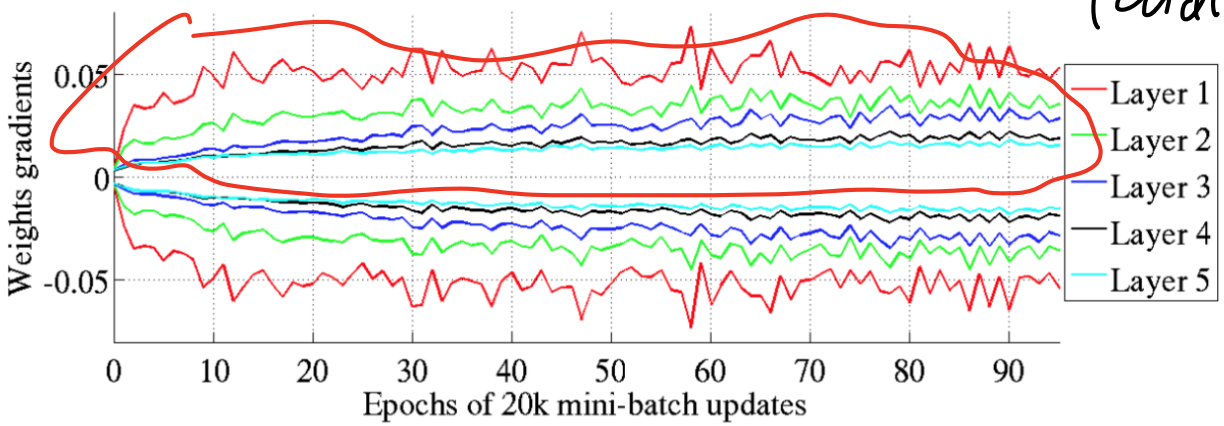
random / distribution D_{in} for W_{in} (no prior knowledge)

  - Activation variance remains same across layers

Xavier Initialization (Glorot & Bengio, '10)

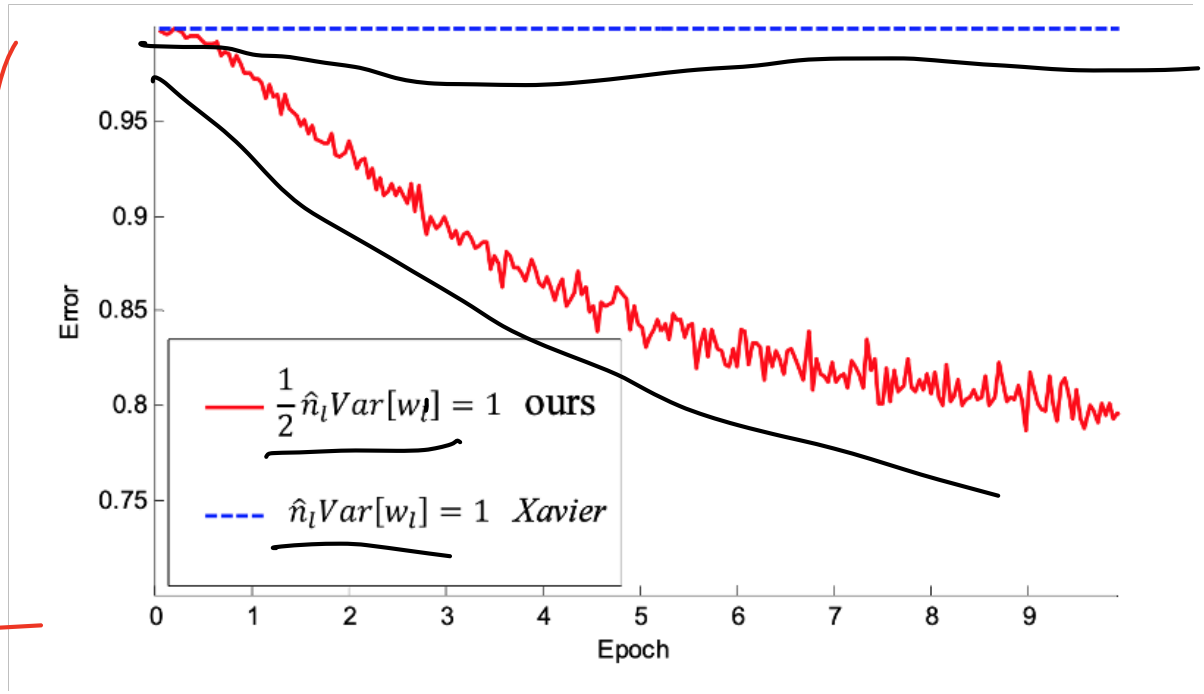
- $W_{ij}^{(h)} \sim \text{Unif}\left[-\frac{\sqrt{6}}{\sqrt{d_h + d_{h+1}}}, \frac{\sqrt{6}}{\sqrt{d_h + d_{h+1}}}\right]$
- $b^{(h)} = 0$
- Experiments (tanh activation)





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 $z^h = W^h x^h \in \mathbb{R}^{d_{h+1}}$

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

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

$\mathbb{E}[W_{ij}] = 0 \Rightarrow \mathbb{E}[z_i^h] = 0$

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

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

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

Kaiming Initialization (He et al. '15)

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

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

(ReLU)

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

(Symmetric)

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

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

Kaiming Initialization (He et al. '15)

We want $Var(z_i^h) = Var(z_j^{h+1})$

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

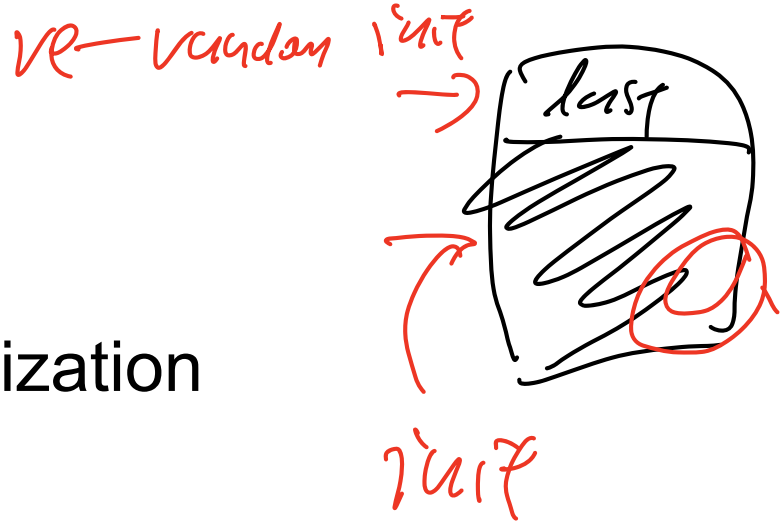
$$Var(w_{ij}^h) = \frac{2}{d_h}$$

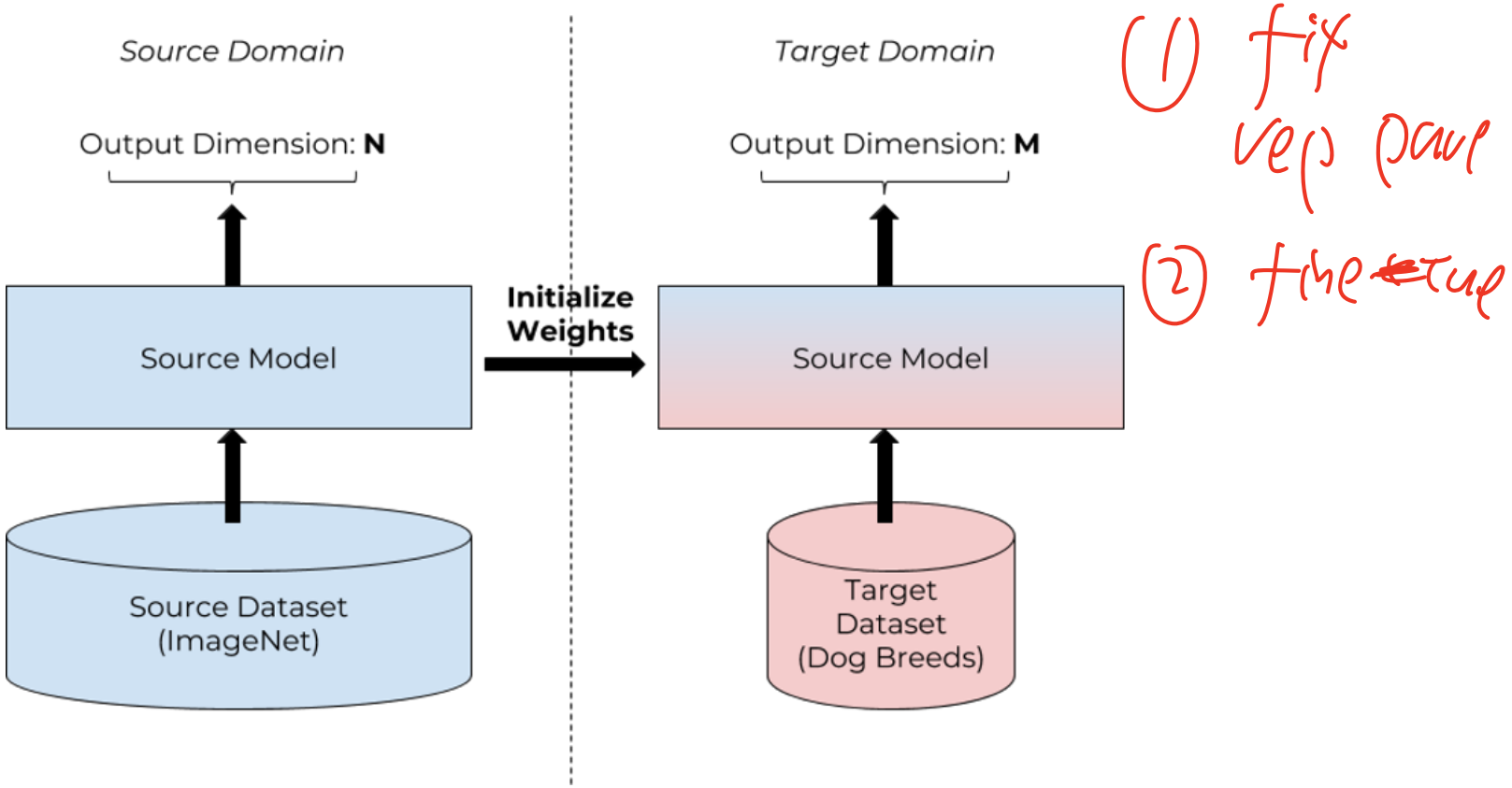
$$Var(z^{H+1}) = Var(z^1) \cdot \left(\prod_{h=1}^{H} \frac{d_h}{2} Var(w_{ij}^h) \right)$$

1

Initialization by Pre-training

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



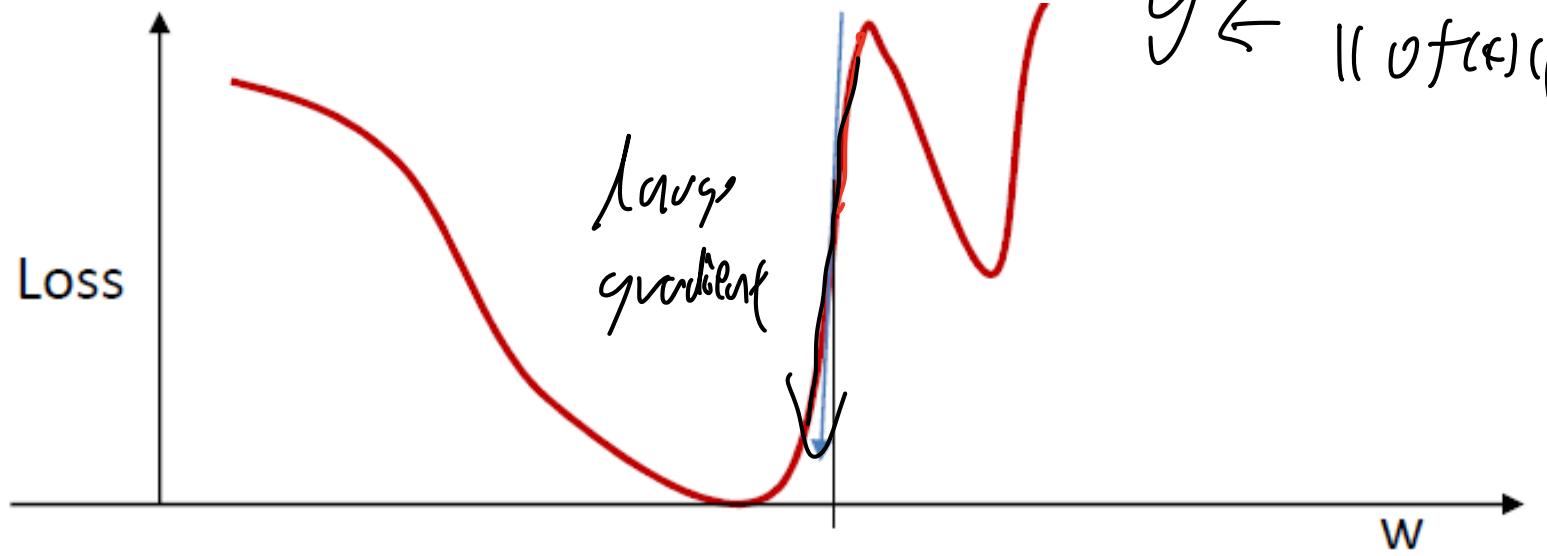


Gradient Clipping

- 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.

$\nabla f(x)$, if $\|\nabla f(x)\| > \text{threshold}$

$$g \leftarrow \frac{\nabla f(x)}{\|\nabla f(x)\|} \cdot \text{threshold}$$



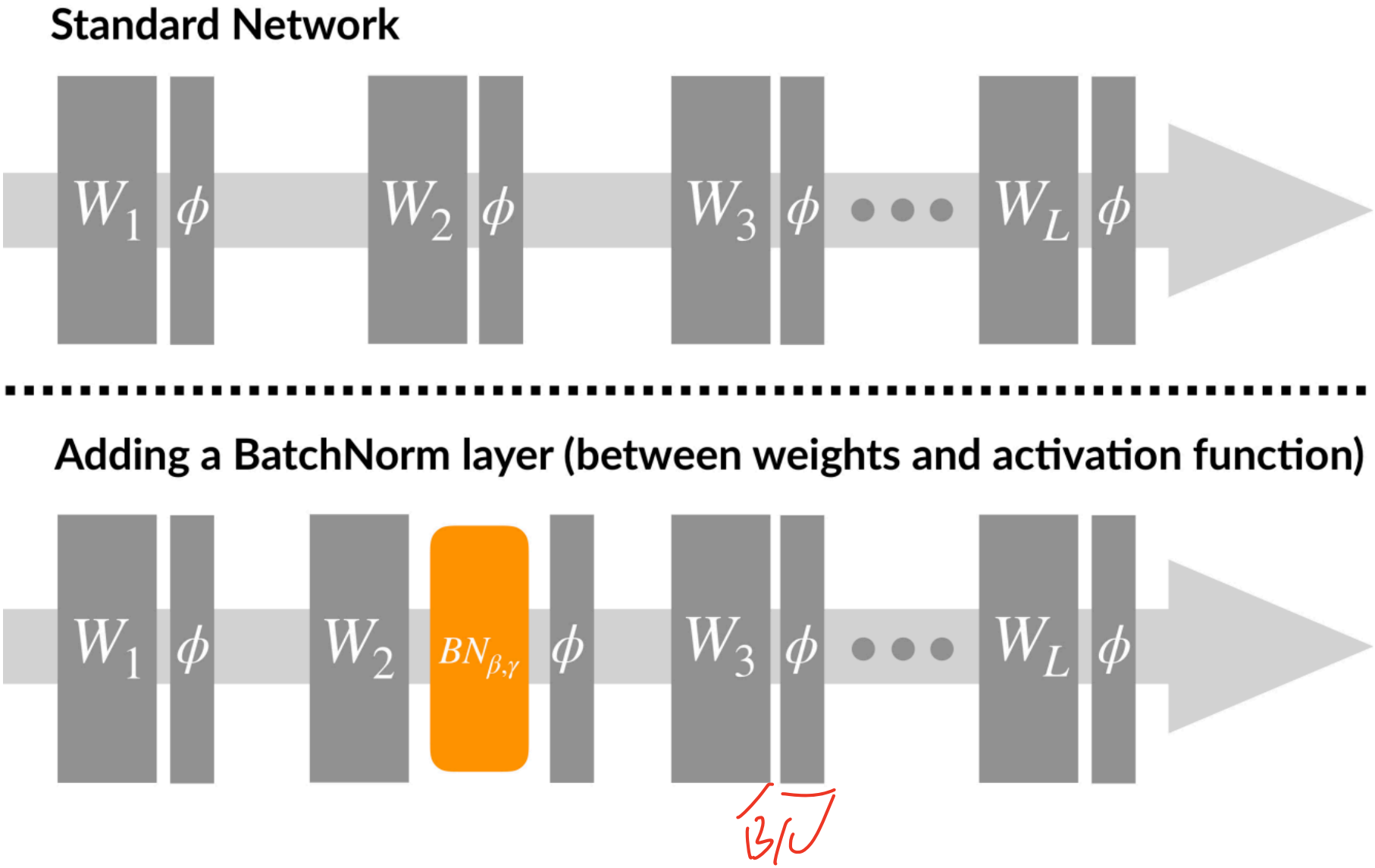
Batch Normalization (Ioffe & Szegedy, '14)

- 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)

ϕ: activation

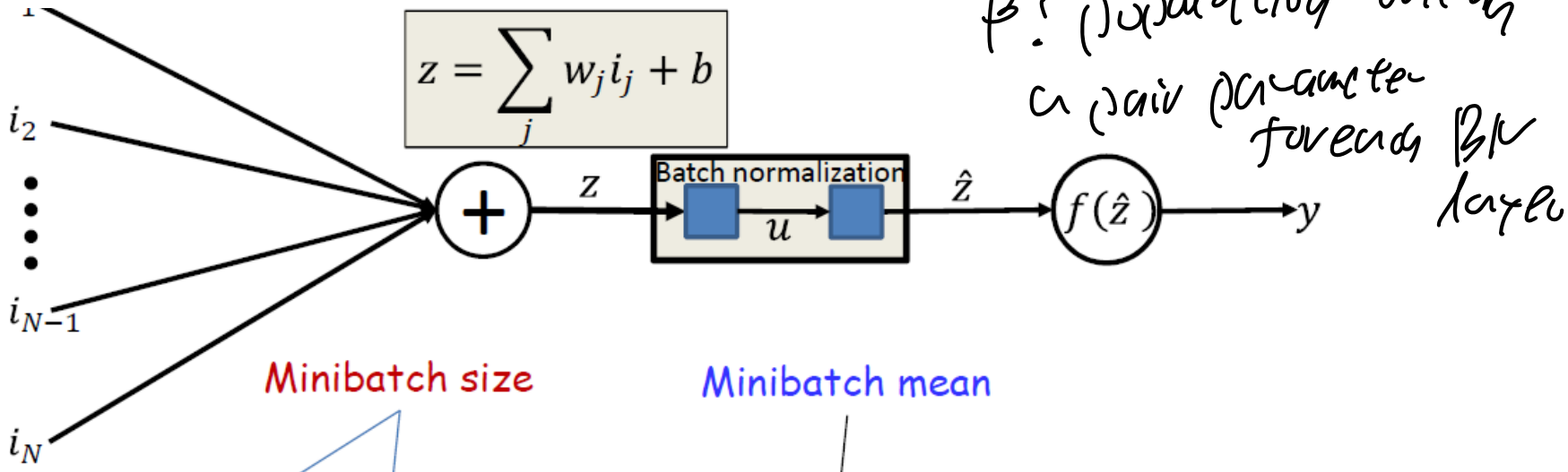


Batch Normalization (Ioffe & Szegedy, '14)

V: population std

β: population mean

a pair parameter for each BN layer



Minibatch size

Minibatch mean

Minibatch standard deviation

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

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

rescale

per dim

$z_1, \ldots z_B$

B: batch size

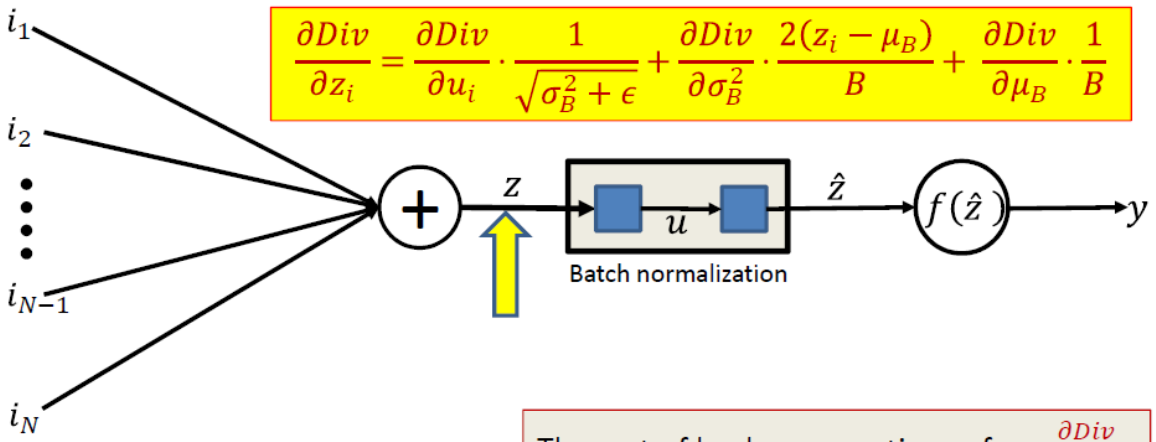
Batch Normalization (Ioffe & Szegedy, '14)

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

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

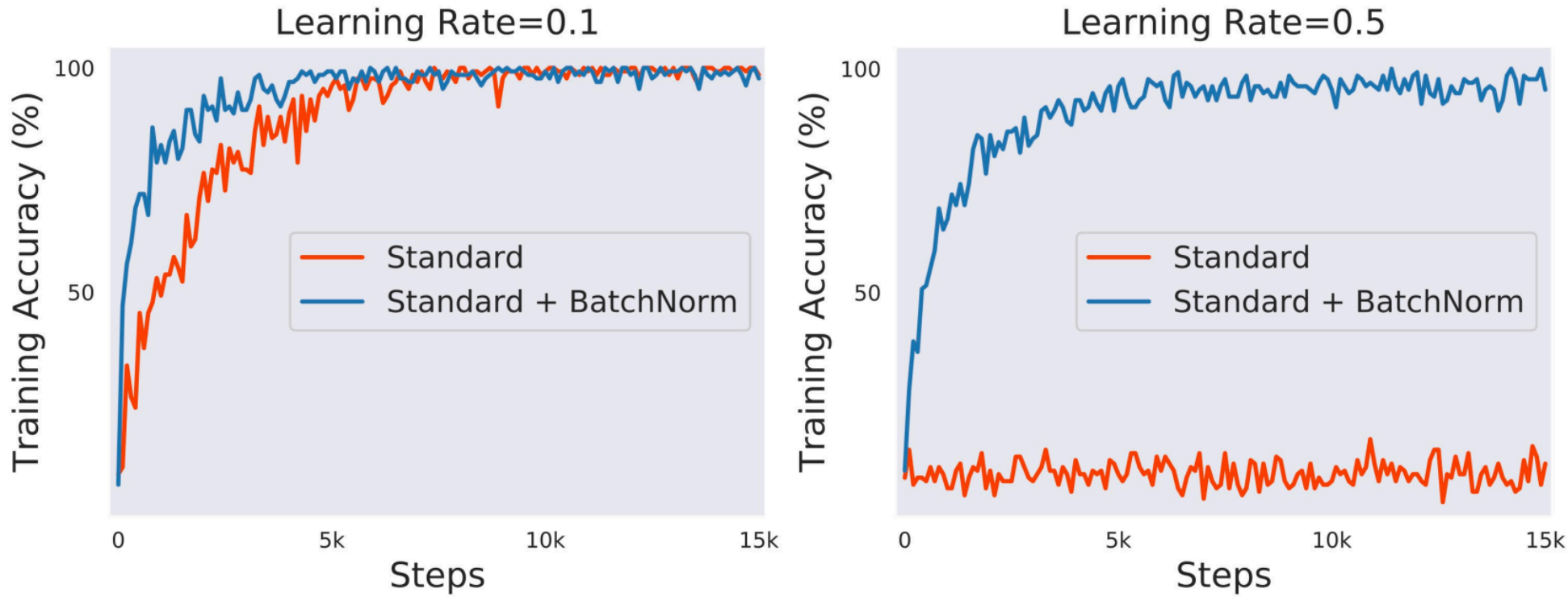
$$\frac{\partial Div}{\partial \mu_B} = \frac{-1}{\sqrt{\sigma_B^2 + \epsilon}} \sum_{i=1}^{B} \frac{\partial Div}{\partial u_i}$$

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



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

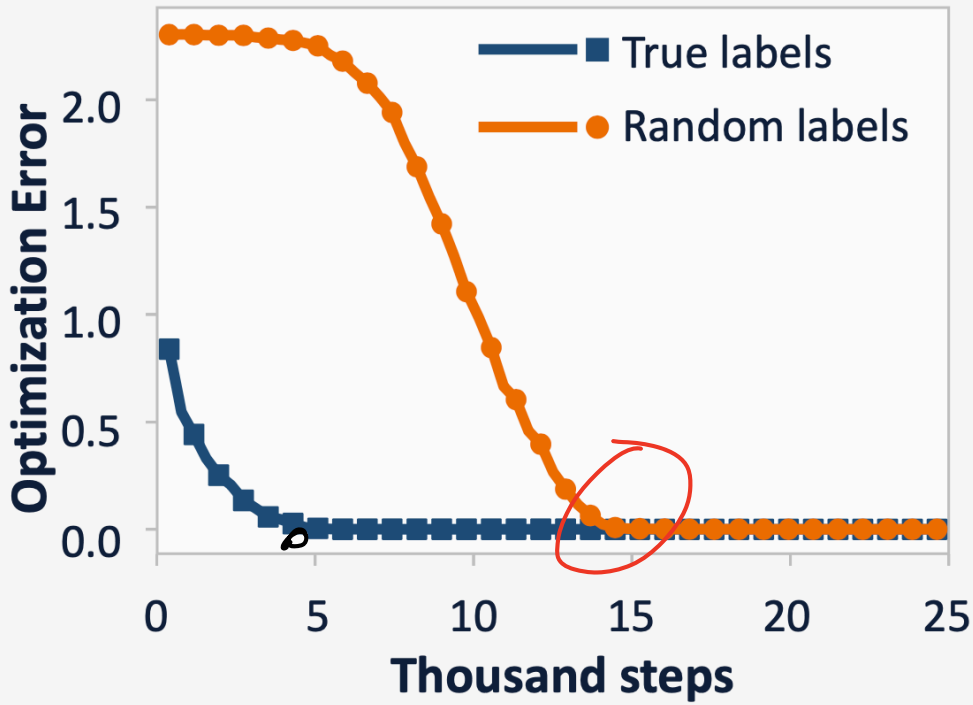
- Layer normalization (Ba, Kiros, Hinton, ’16)
  - Batch-independent
  - Suitable for RNN, MLP
- Weight normalization (Salimans, Kingma, ’16)
  - Suitable for meta-learning (higher order gradients are needed)
- Instance normalization (Ulyanov, Vedaldi, Lempitsky, ’16)
  - Batch-independent, suitable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

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



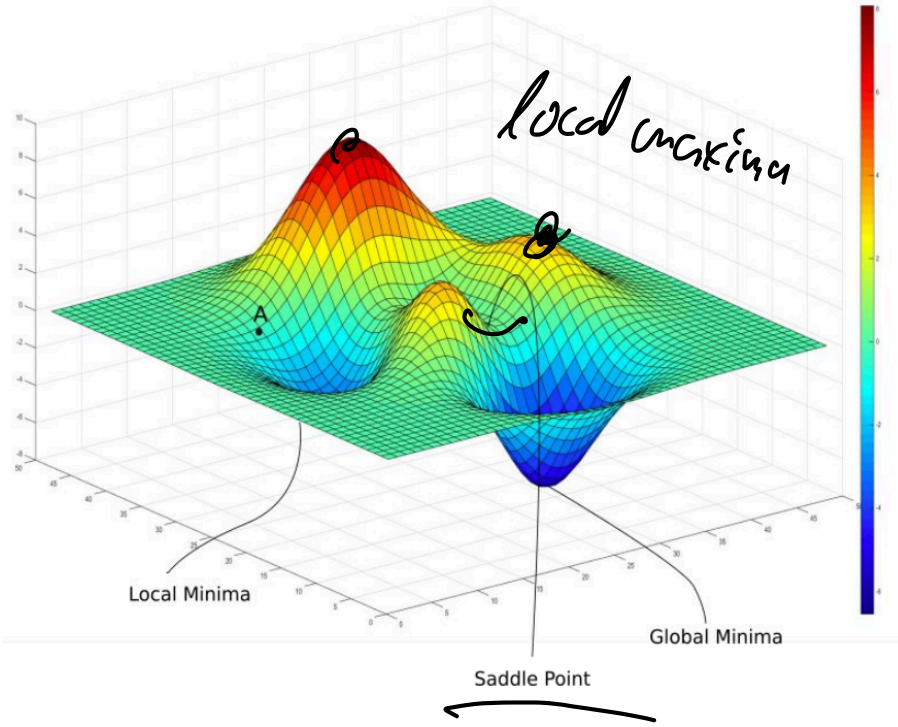
Optimization error -> **0** for both ***true labels*** and ***random labels*** !

Zhang Bengio Hardt Recht Vinyals 2017

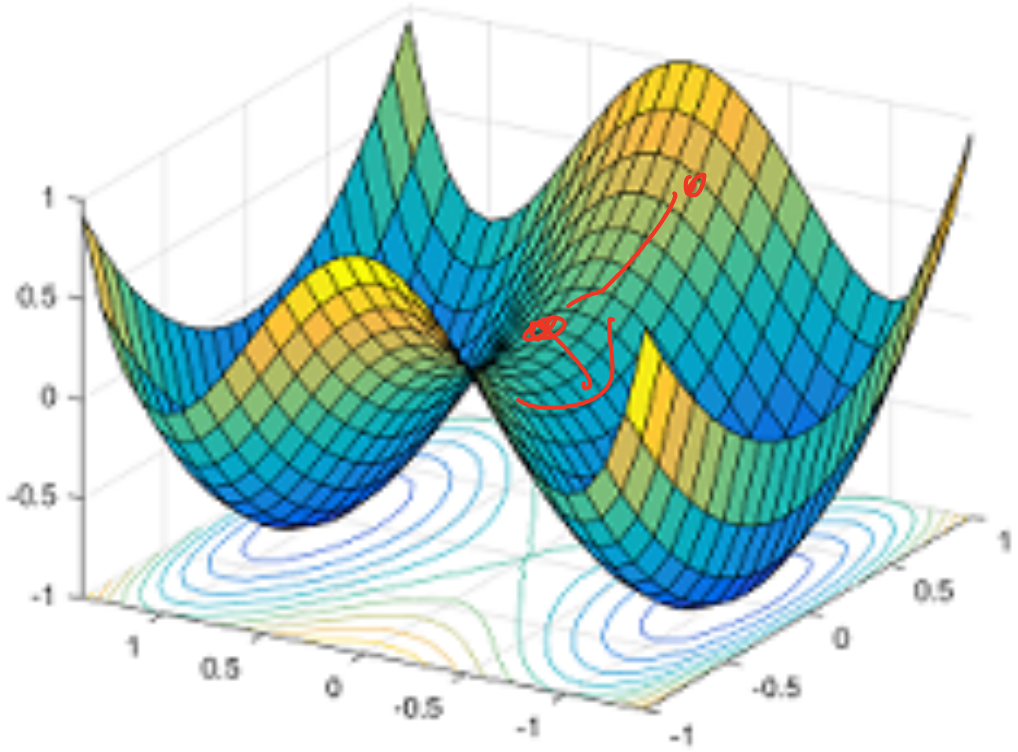
Understanding DL Requires Rethinking Generalization

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

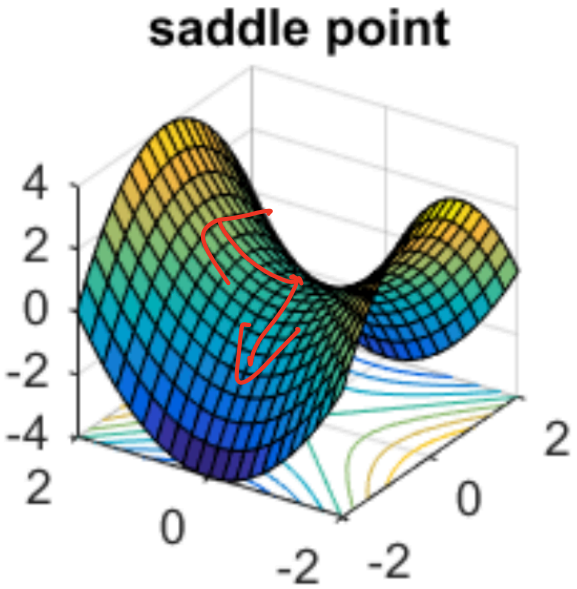


Landscape Analysis



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

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



$\nabla^2 f(x) = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$

$= \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$

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

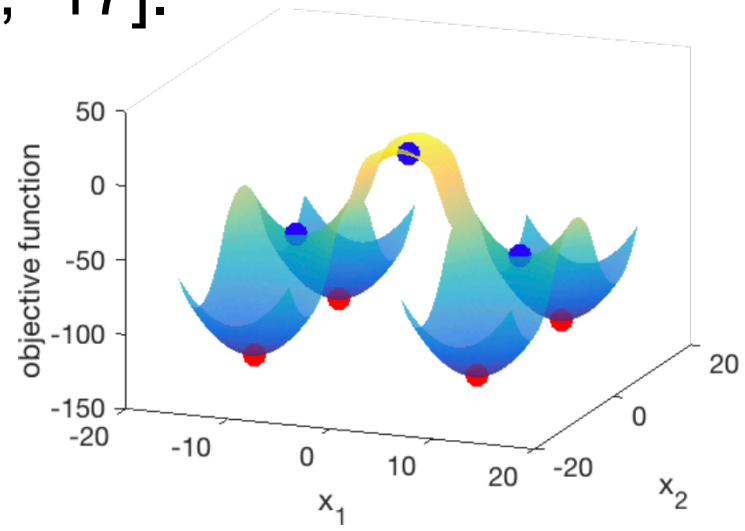
Escaping Strict Saddle Points

- 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].