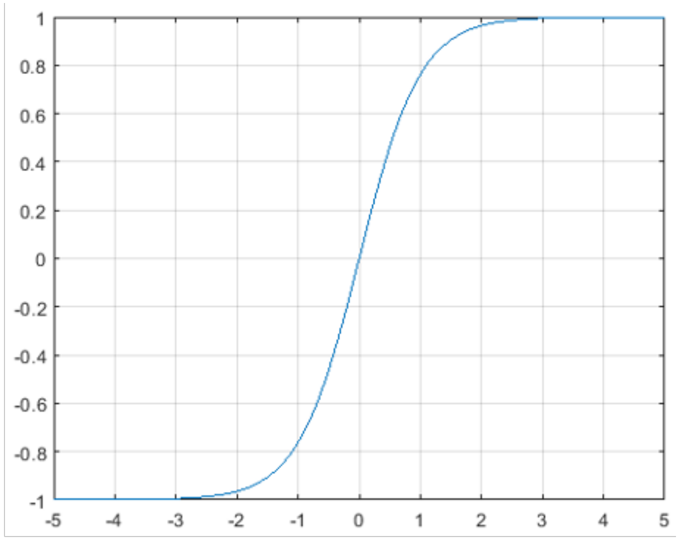
Important Techniques in Neural Network Training

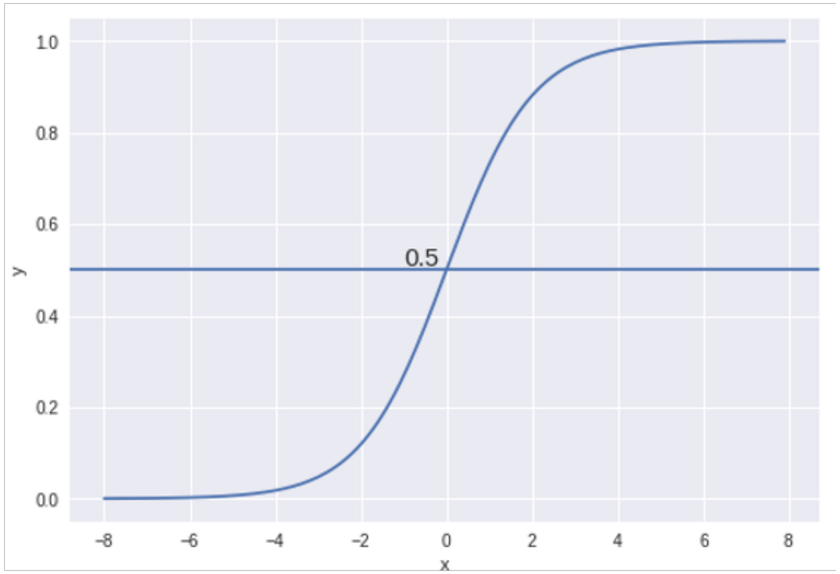
Gradient Explosion / Vanishing

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

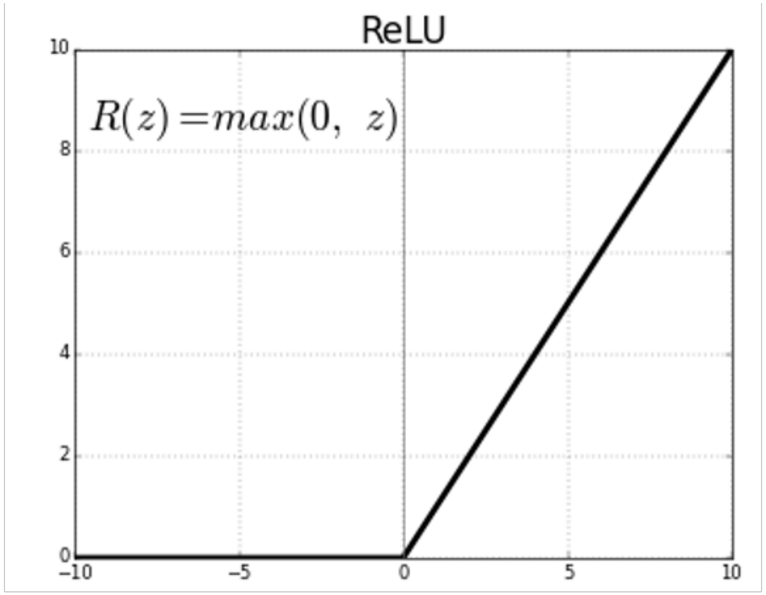
Activation Functions



tanh

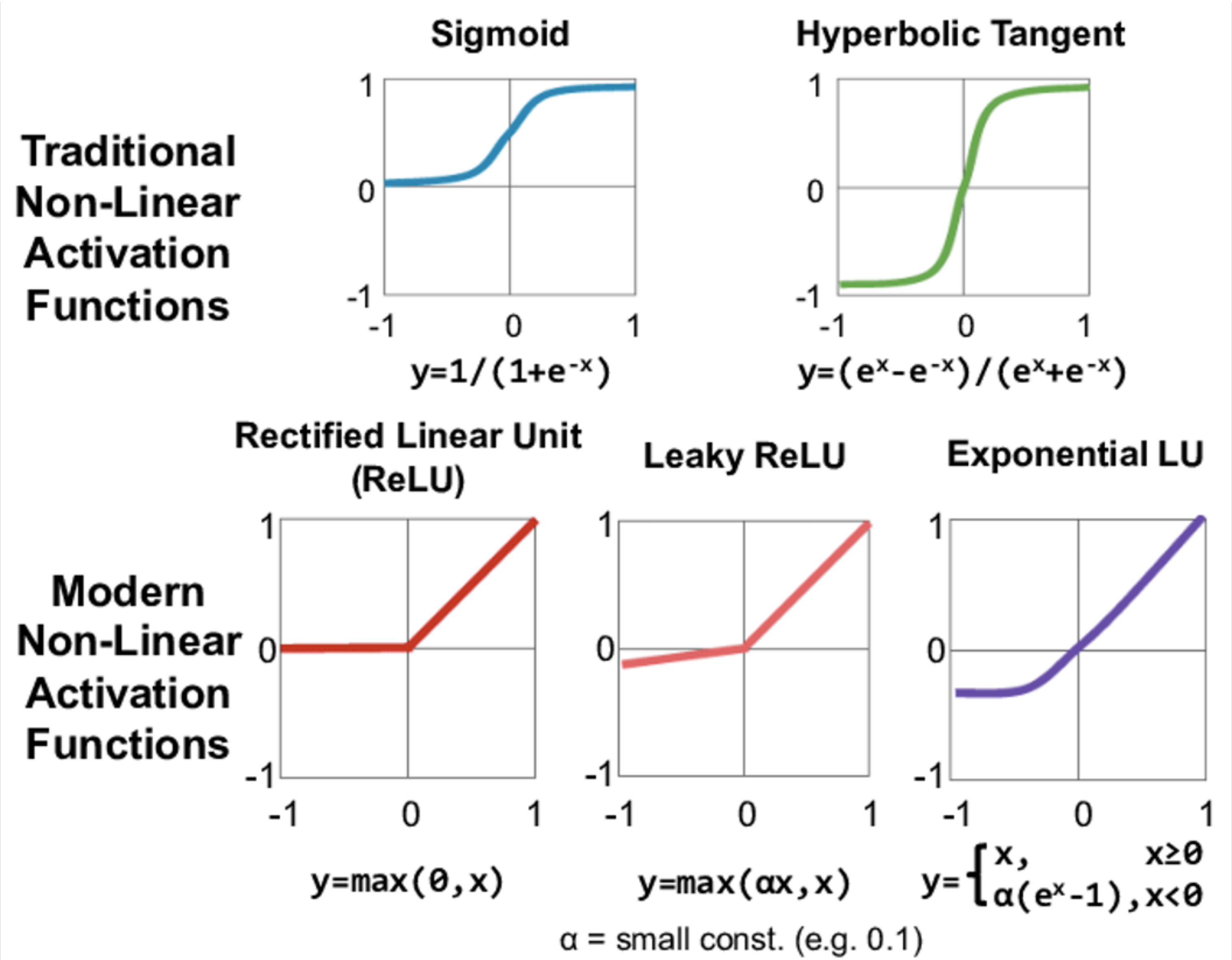


sigmoid



Rectified Linear United

Activation Function



Initialization

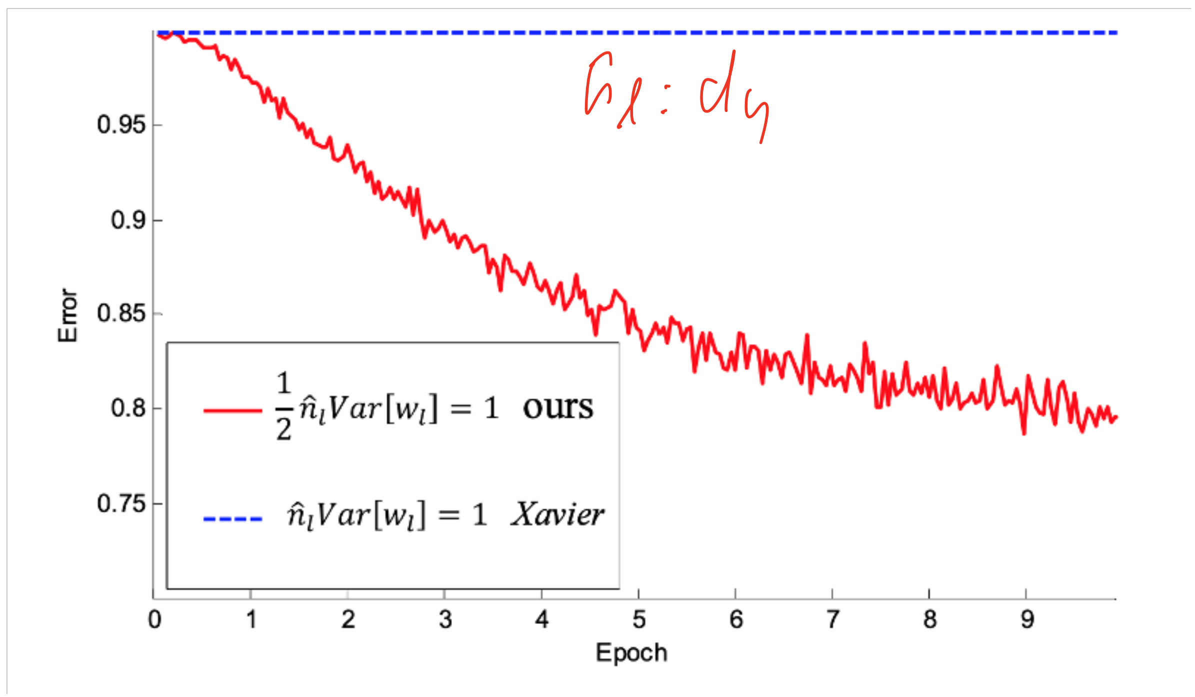
- Zero-initialization
- Large initialization $\Rightarrow$ random
- Small initialization

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

Kaiming Initialization (He et al. '15)

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

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



Kaiming Initialization (He et al. '15)

Each layer: $z^h = W^h x^h$ 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

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

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

$$= d_h \Big(\mathrm{Var}(W_{ij}^h) \cdot \mathrm{Var}(x_j^h) + \big(\underbrace{\mathbb{E}[W_{ij}^h]}_{=0}\big)^2 \cdot \mathrm{Var}(x_j^h) + \mathrm{Var}(W_{ij}^h) \cdot \big(\mathbb{E}[x_j^h]\big)^2 \Big)$$

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

Kaiming Initialization (He et al. '15)

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

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

(ReLU)

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

(Symmetry of init around 0)

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

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

Kaiming Initialization (He et al. '15)

$$\underline{\mathrm{Var}(z_i^h)} = d_h \cdot \mathrm{Var}(W_{i,j}^h) \cdot \frac{1}{2} \underline{\mathrm{Var}(z_j^{h-1})}$$

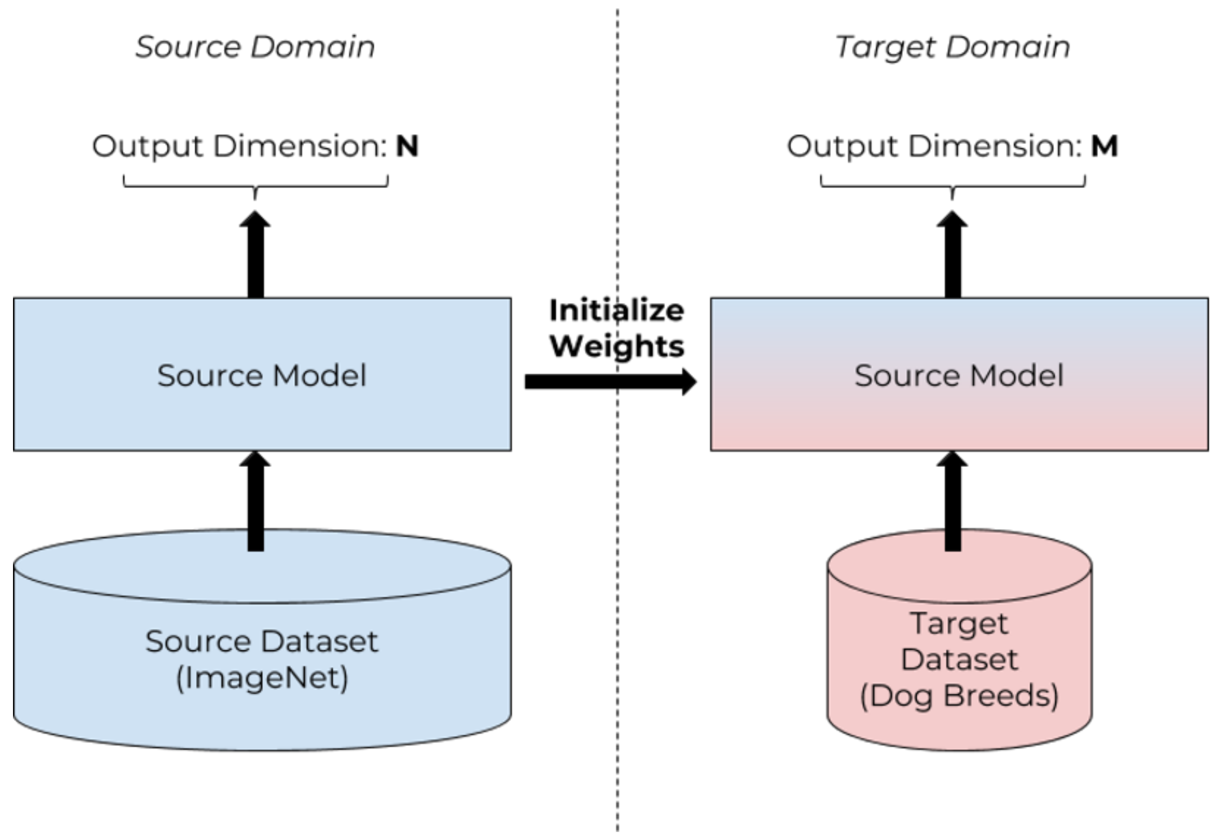
$$\Rightarrow \quad \frac{1}{2} d_h \cdot \mathrm{Var}(W_{i,j}^h) = 1$$

$$\mathrm{Var}(W_{i,j}^h) = \frac{2}{d_h}$$

$$\mathrm{Var}(z_i^h) = \mathrm{Var}(z^0) \cdot \left(\prod_{h'=1}^{h} \underbrace{\frac{d_{h'}}{2} \mathrm{Var}(W_{i,j}^{h'})}_{=1} \right)$$

Initialization by Pre-training

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

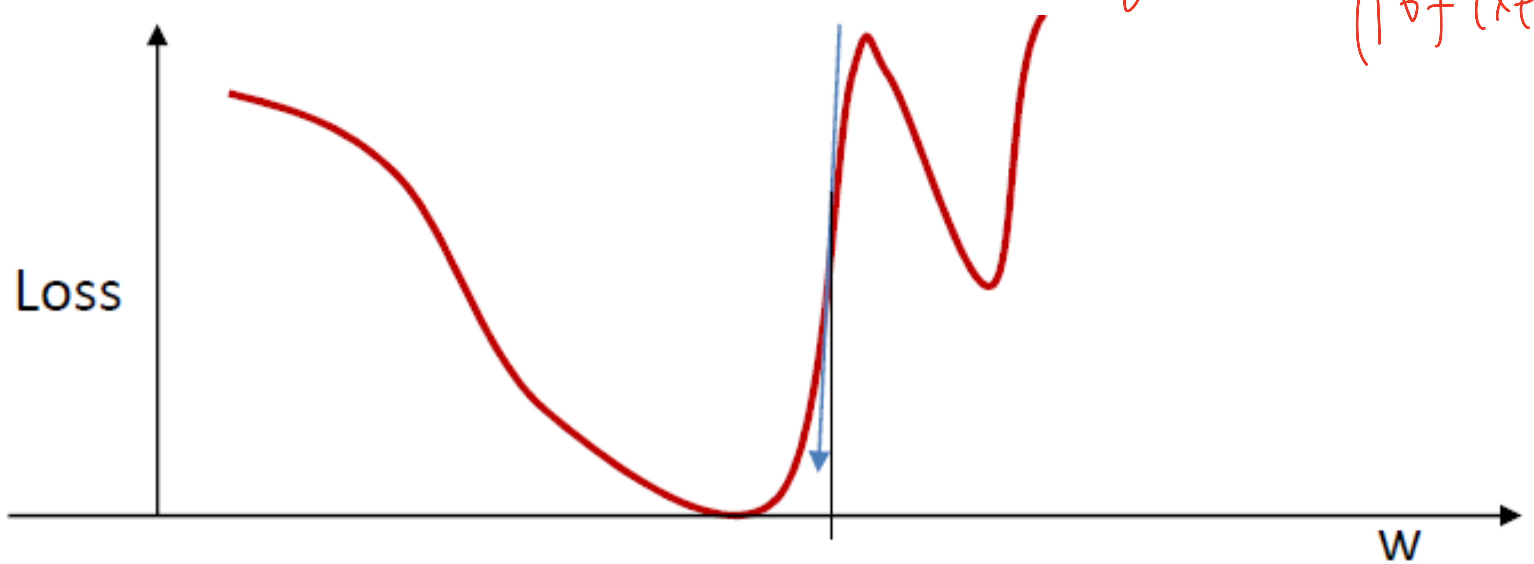


Gradient Clipping

- 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.

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

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



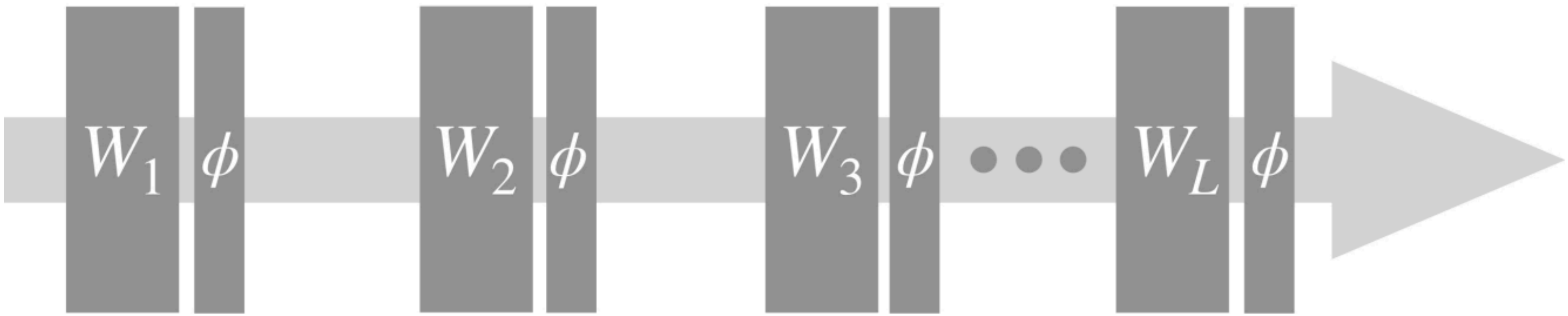
Batch Normalization (Ioffe & Szegedy, ’14)

- 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

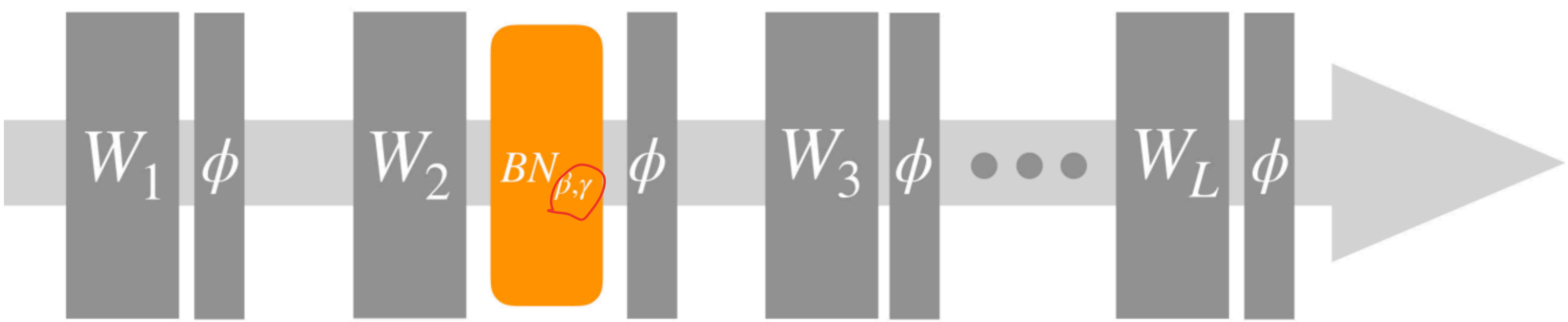
- **Batch normalization** is an attempt to do that：
  - Each unit’s **pre-activation** is normalized (mean subtraction, std division)
  - During training, mean and std is computed for each minibatch (can be backproped!

Batch Normalization (Ioffe & Szegedy, ’14)

Standard Network



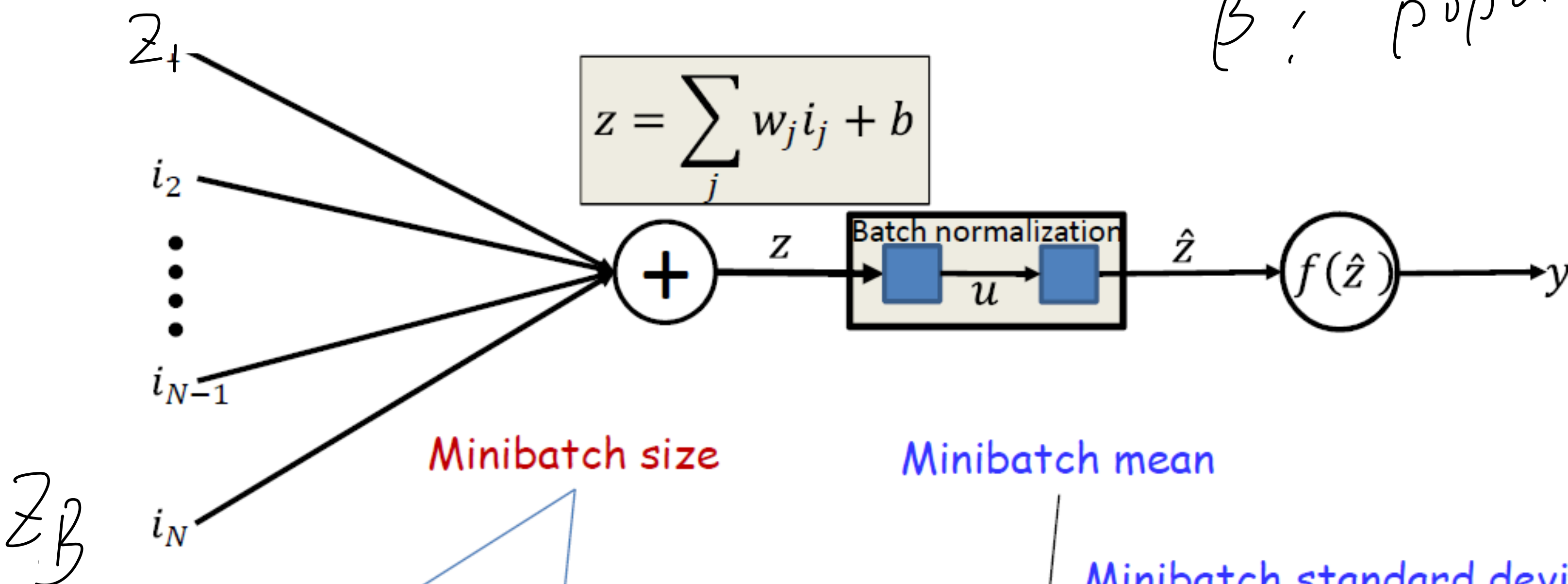
Adding a BatchNorm layer (between weights and activation function)



Batch Normalization (Ioffe & Szegedy, '14)

γ: population std

β: population mean



Minibatch size

Minibatch mean

Minibatch standard deviation

$$\mu_B = \frac{1}{B}\sum_{i=1}^{B} z_i \qquad \sigma_B^2 = \frac{1}{B}\sum_{i=1}^{B}(z_i - \mu_B)^2 \qquad \boxed{u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}} \qquad \boxed{\hat{z}_i = \gamma u_i + \beta}$$

$u_1, \ldots, u_B$

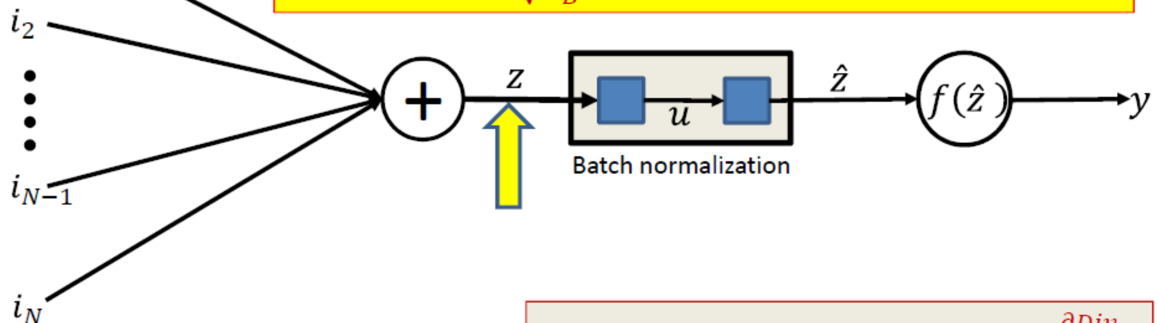
Batch Normalization (Ioffe & Szegedy, '14)

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

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

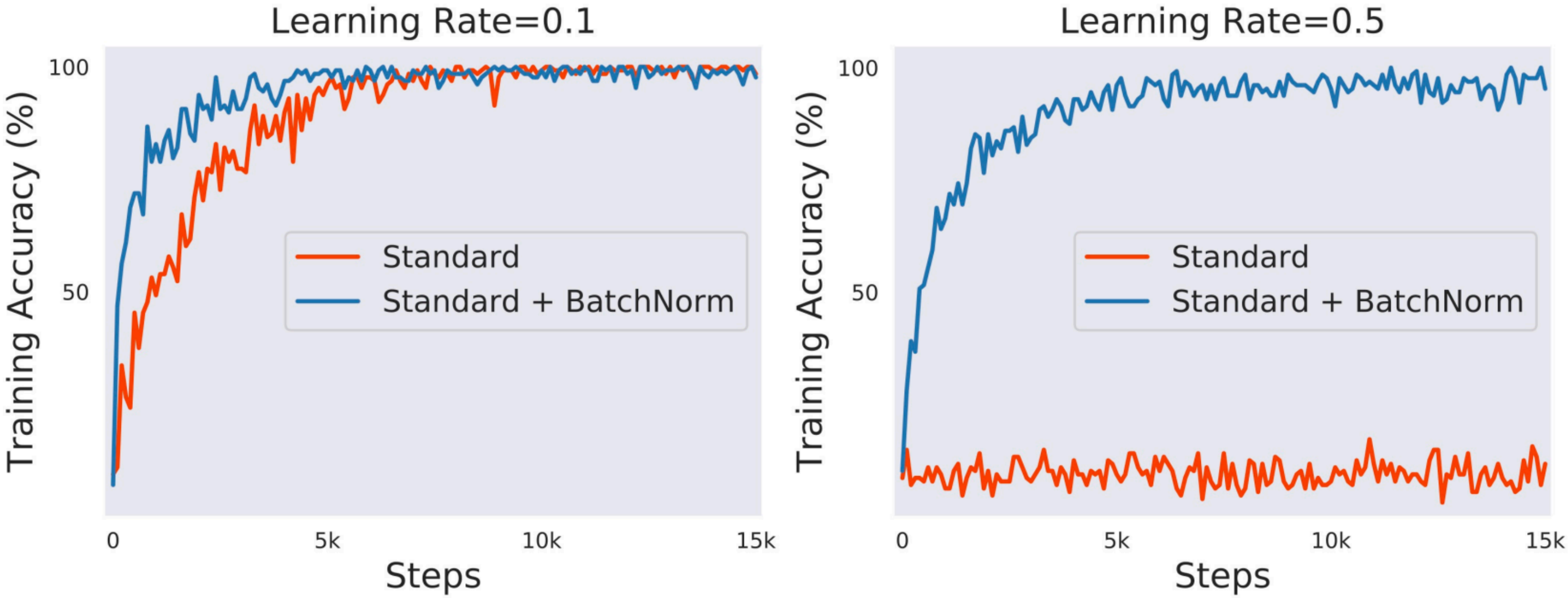
$$\frac{\partial Div}{\partial \mu_B} = \frac{-1}{\sqrt{\sigma_B^2 + \epsilon}} \sum_{i=1}^{B} \frac{\partial Div}{\partial u_i}$$

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



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)
- ….

Non-convex
Optimization Landscape

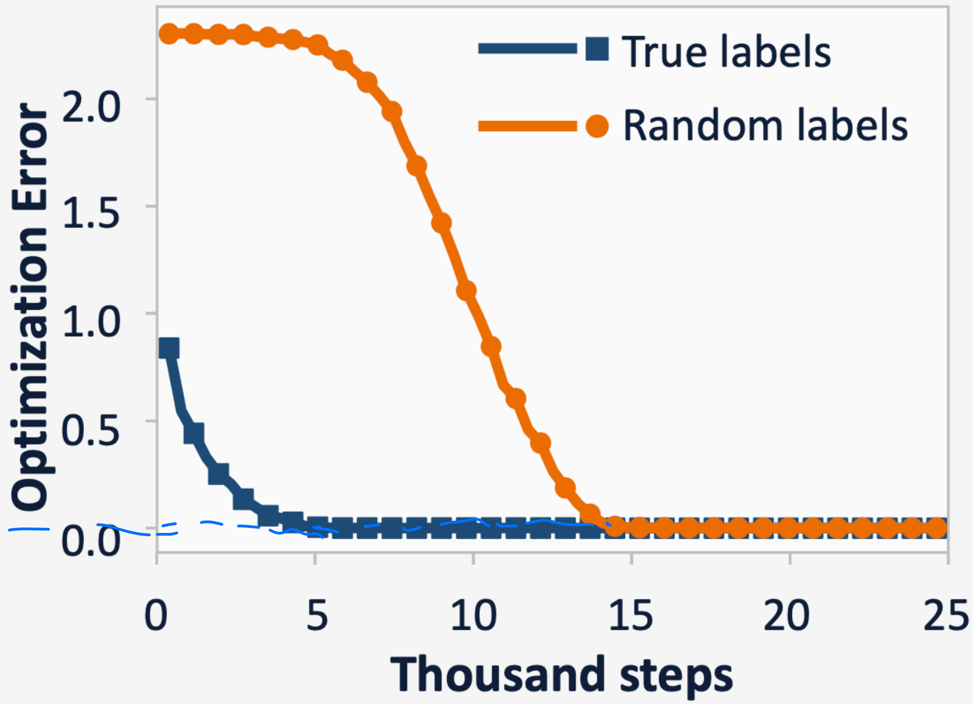
Gradient descent finds global minima

as long as $x_1 = x_2$

no $y_1 \neq y_2$

Practice: gradient descent

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



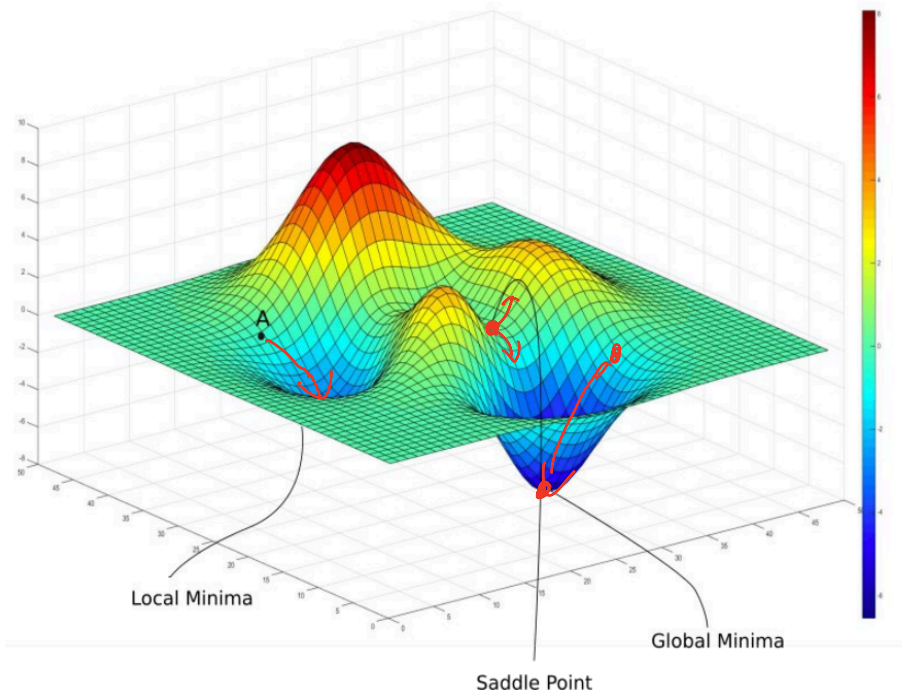
Optimization error **-> 0** for both ***true labels*** and ***random labels*** !

Zhang Bengio Hardt Recht Vinyals 2017

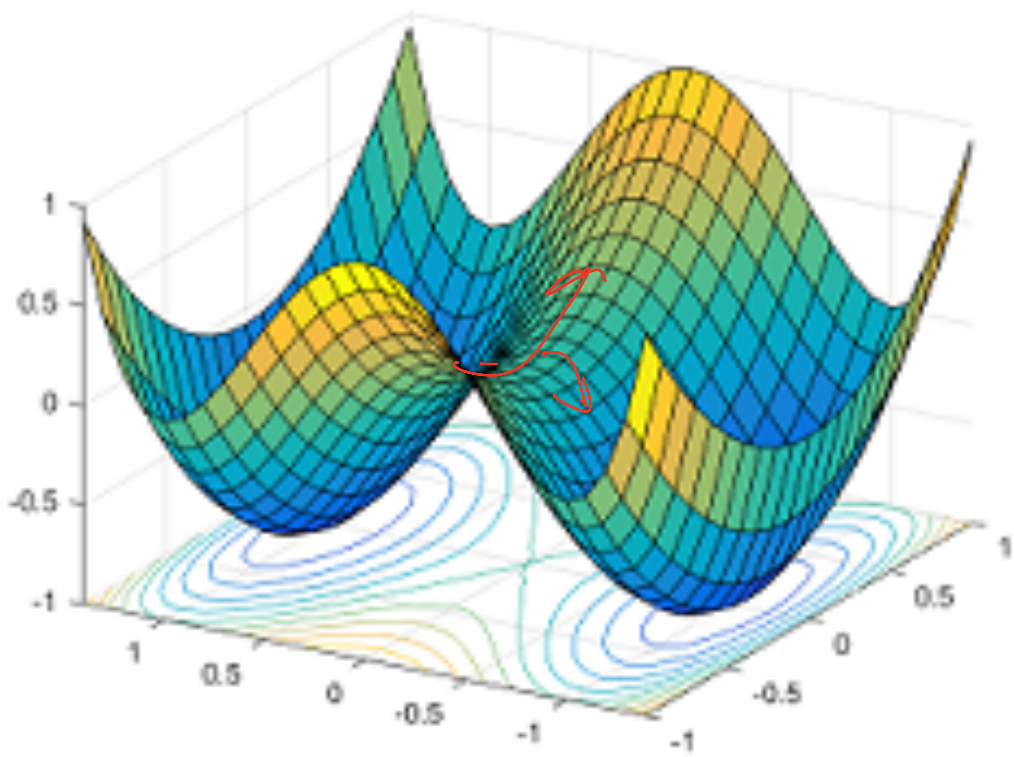
Understanding DL Requires Rethinking Generalization

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

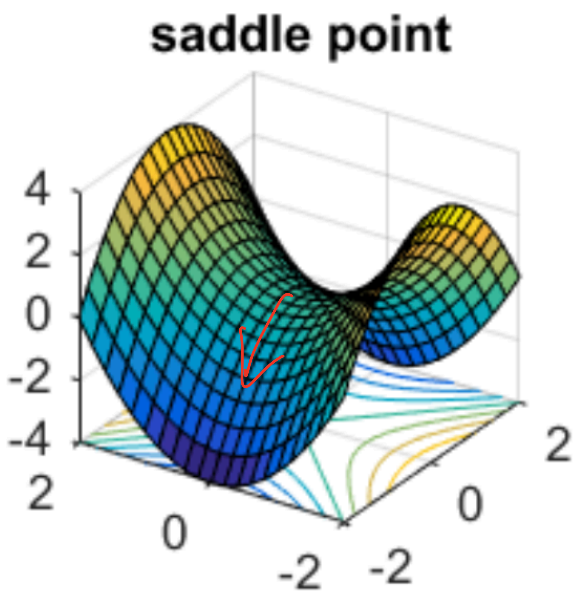


Landscape Analysis



- 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

x: saddle

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

D

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

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

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

$|v^T x_{t+1}| = (\underbrace{1 - \eta \lambda_{\min}}_{>1}) \cdot |v^T x_t|$

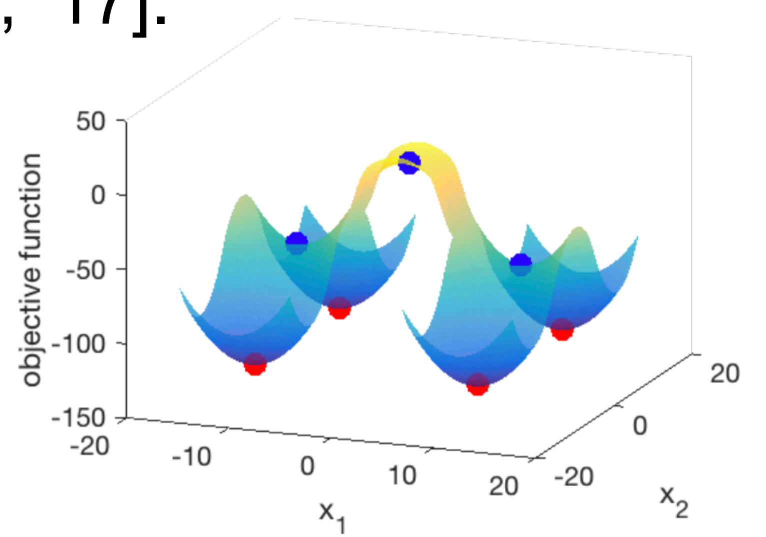
v: eigen vector corresponds to smallest eigen value

Escaping Strict Saddle Points

$x_{t+1} \leftarrow x_t - \eta \nabla f(x_t)$
$+ \varepsilon \cdot N(0,1)$

- 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

$\langle A_i, BC \rangle = y_i$, $\min_{B,C} (\langle A_i, BC \rangle - y_i)^2$

- 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].

$$W_H W_{H-1} \cdots W_1 X$$

- 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].