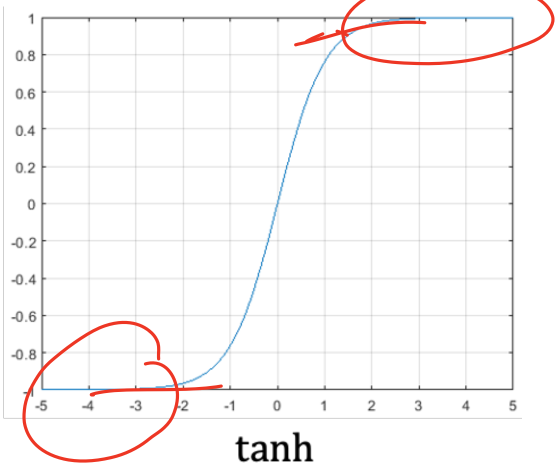
Important Techniques in Neural Network Training

Gradient Explosion / Vanishing

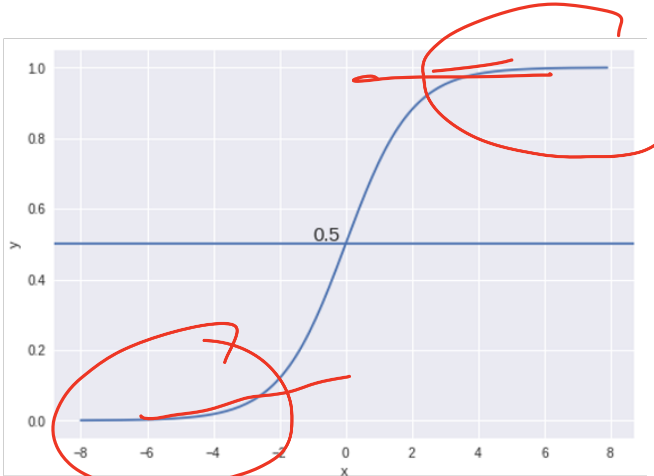
- Deeper networks are harder to train:
  - Intuition: gradients are products over layers
  - Hard to control the learning rate

$$\frac{\partial f}{\partial W_n} = \left(W_{H+1} A_H \cdots W_n A_n\right)^T \left(A_{n-1} W_{n-1} \cdots W_1 X\right)$$

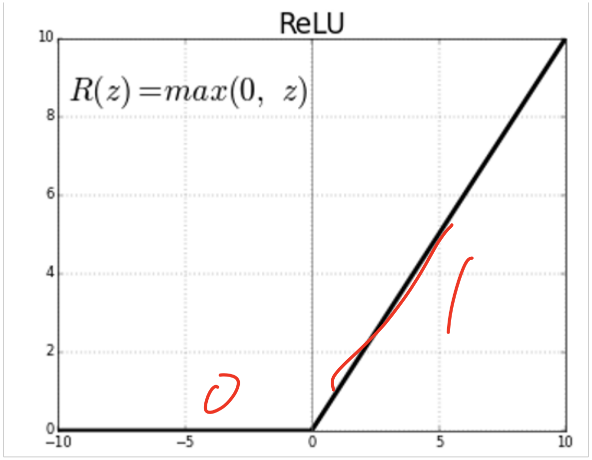
Activation Functions



tanh

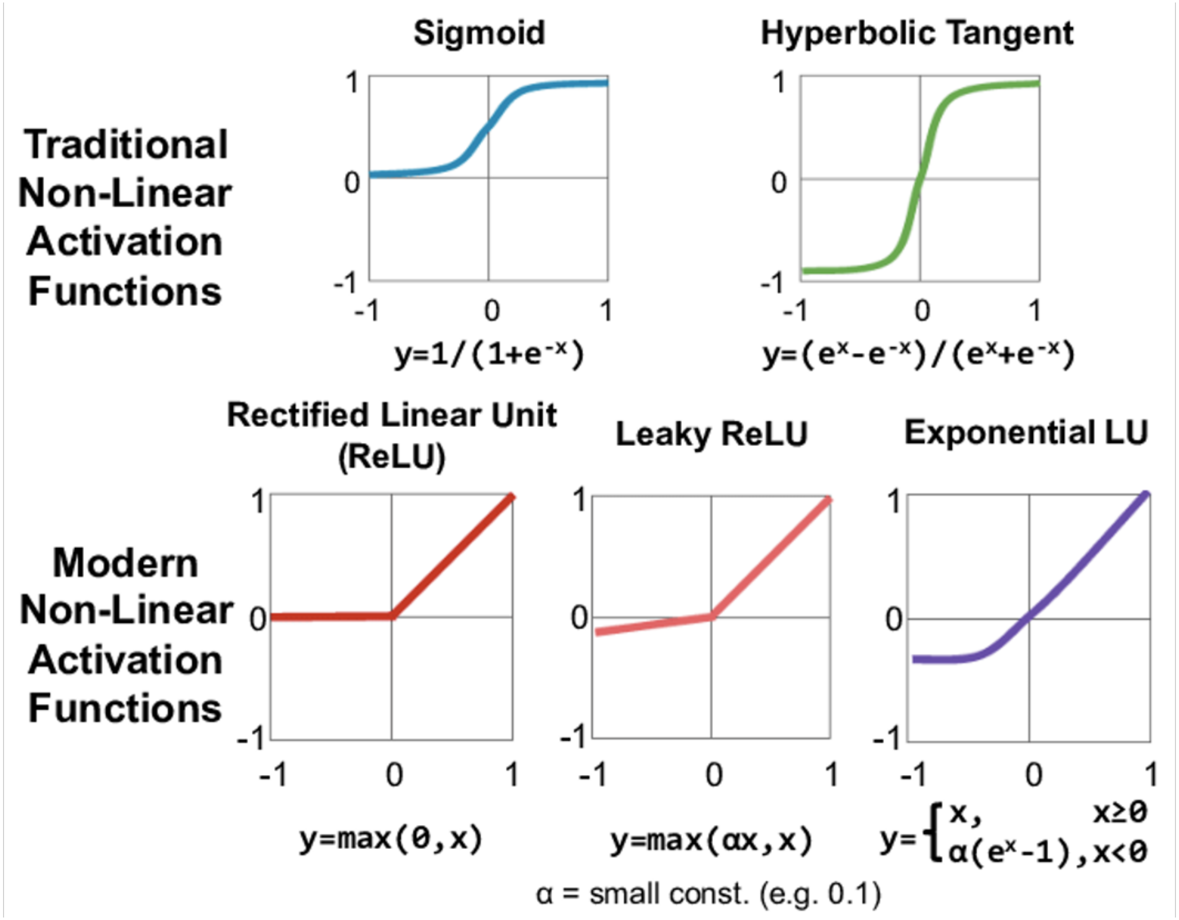


sigmoid



Rectified Linear United

Activation Function



Initialization

- Zero-initialization
- Large initialization
- Small initialization

gradient 0

scaling

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

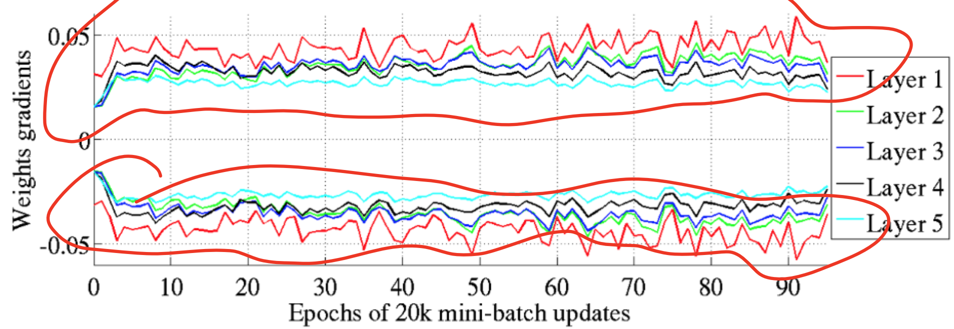
balancing

Xavier Initialization (Glorot & Bengio, '10)

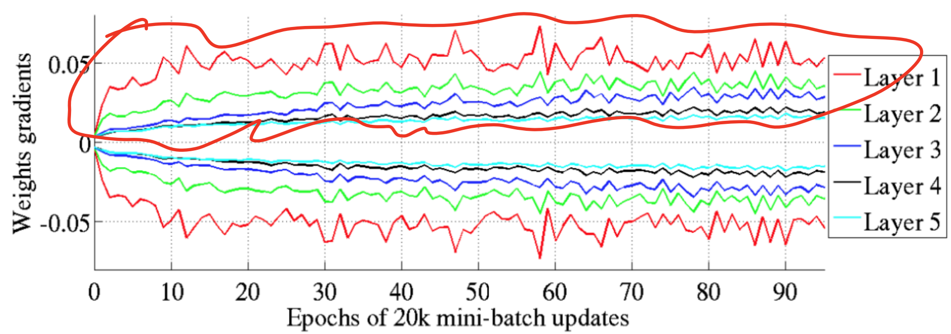
- $W_{ij}^{(h)} \sim \text{Unif}\left[-\frac{\sqrt{6}}{\sqrt{d_h + d_{h+1}}}, \frac{\sqrt{6}}{\sqrt{d_h + d_{h+1}}}\right]$
- $b^{(h)} = 0$
- Experiments (tanh activation)

$\mathbb{E}[W_{ij}^{(h)}] = 0$

$\text{Var}(W_{ij}^{(h)}) = \frac{2}{d_h + d_{h+1}}$



unif $\left[-\frac{1}{\sqrt{d_h}}, \frac{1}{\sqrt{d_h}}\right]$

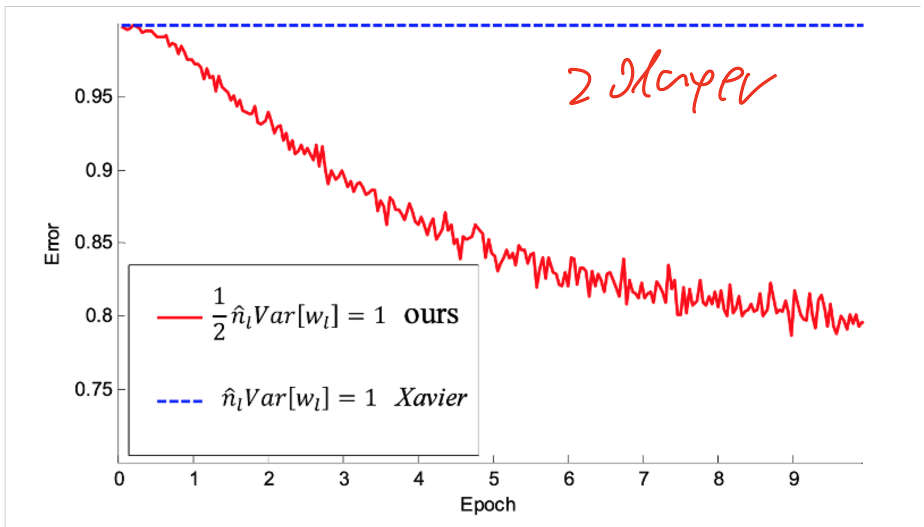


Kaiming Initialization (He et al. ’15)

if $d_h = d_{h+1}$

v.s. $var = \frac{1}{d_h}$

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



Kaiming Initialization (He et al. '15)

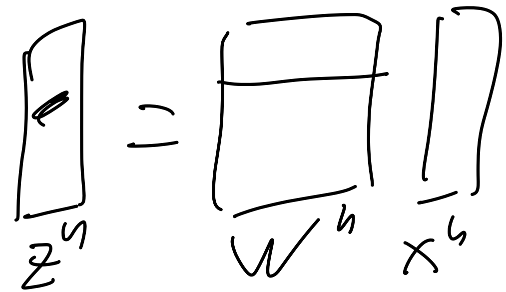
$$z^n = W^n \cdot x^n$$

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

$$z_i^n = \sum_{j=1}^{d_n} W_{ij}^n \cdot x_j^n$$

$x^1, \; W^1 x^1, \; \sigma(W^1 x^1) \ldots$

$z^1, \; x^2,$



$$\mathbb{E}[z_i^n] = 0$$

$$\mathrm{Var}(z_i^n) = d_n \cdot \mathrm{Var}(W_{ij}^n \cdot x_j^n)$$

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

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

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 initialization)

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

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

Want

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

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

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

$$\mathrm{Var}(z^{L+1}) = \mathrm{Var}(z^1) \underbrace{\left(\prod_{h=1}^{L} \frac{d_h}{2} \mathrm{Var}(W_{ij}^h) \right)}_{O(C)}$$

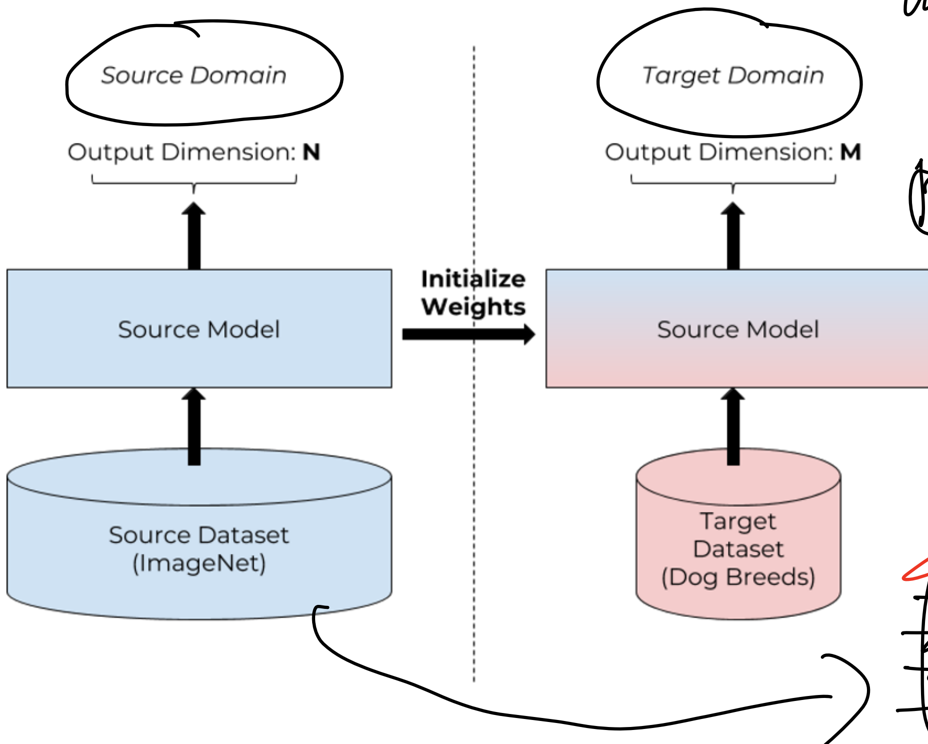
Initialization by Pre-training

BERT, GPT-3

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

train on wiki

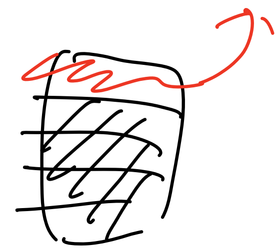
predict missing word



machine translation

only the last layer

re-randomized initialization

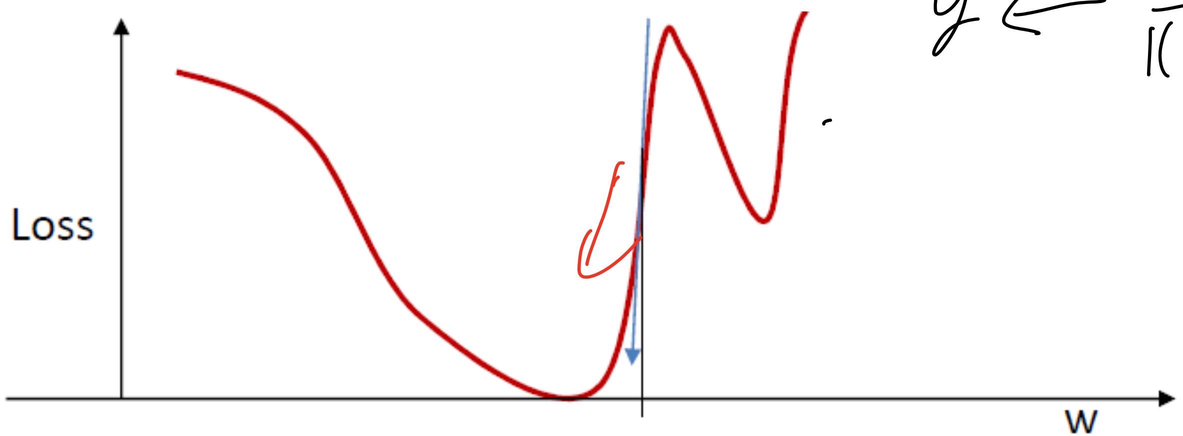


Gradient Clipping

- 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.

$g = \nabla f(x_t)$, $\|g\|_2 > \text{threshold}$

$g \leftarrow \frac{g}{\|g\|_2} \cdot \text{threshold}$



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

population mean variance

iteration → calculation

calculation

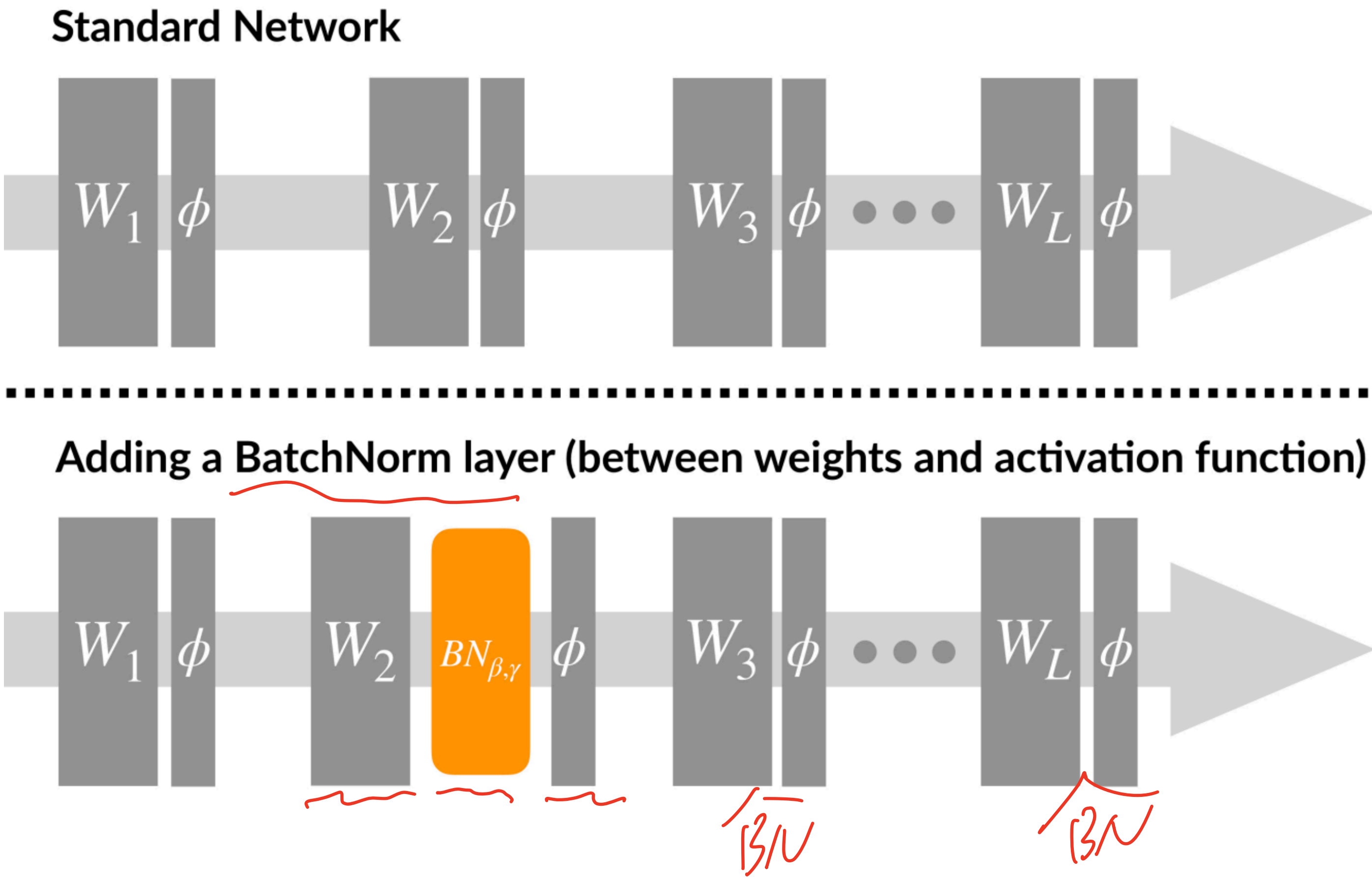
same mean

var

- **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)

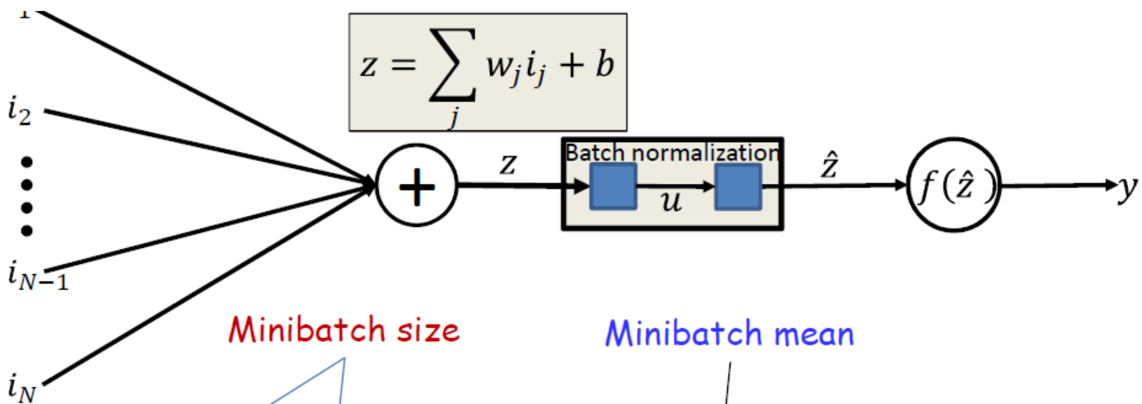
new layer



Batch Normalization (Ioffe & Szegedy, '14)

$z_j = W_j \cdot X$

ϵ: hyper-parameter



Minibatch size

Minibatch mean

Minibatch standard deviation

(1)

$$\mu_B = \frac{1}{B}\sum_{i=1}^{B} z_i$$

(2)

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

$$u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

$$\hat{z}_i = \gamma u_i + \beta$$

$\to \sigma(z_i)$

β: mean

γ: std

$z_1, \ldots z_B$

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.

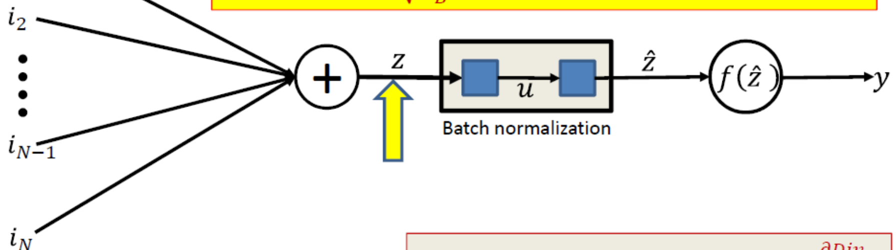
32, 128

1024

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

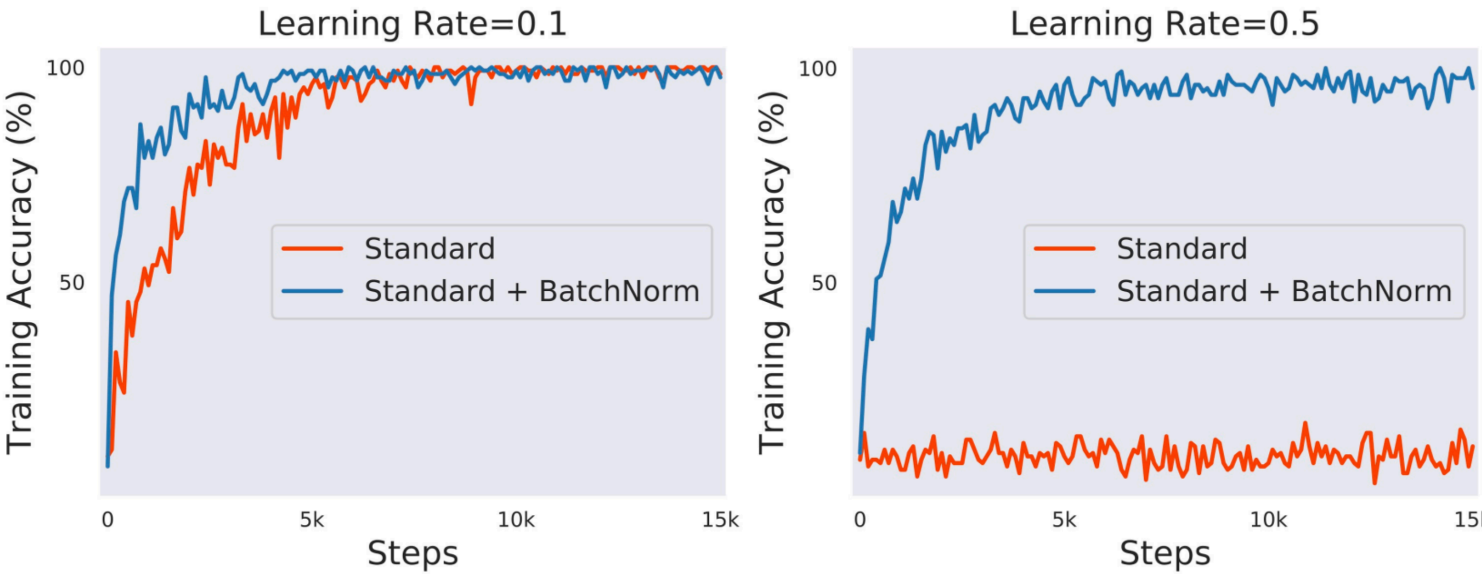
$$\frac{\partial Div}{\partial \mu_B} = \frac{-1}{\sqrt{\sigma_B^2 + \epsilon}}\sum_{i=1}^{B}\frac{\partial Div}{\partial u_i}$$

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



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

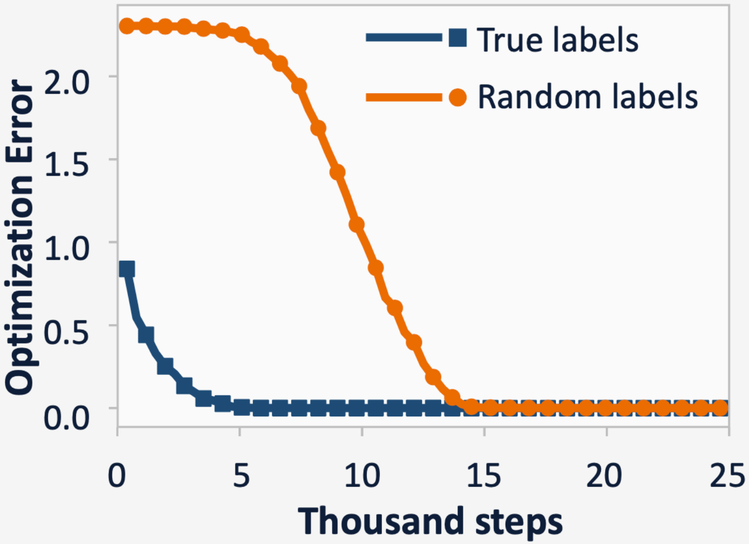
- Layer normalization (Ba, Kiros, Hinton, ’16)
  - Batch-independent
  - Suitable for RNN, MLP
- Weight normalization (Salimans, Kingma, ’16)
  - Suitable for meta-learning (higher order gradients are needed)
- Instant normalization (Ulyanov, Vedaldi, Lempitsky, ’16)
  - Batch-independent, suitable for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

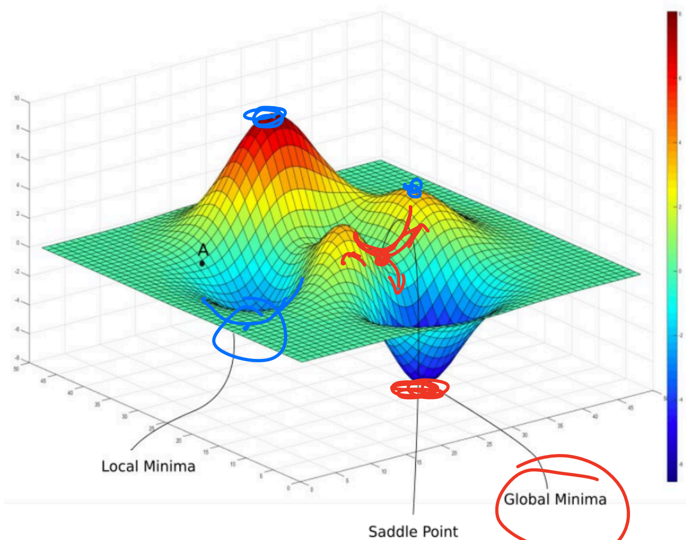
Understanding DL Requires Rethinking Generalization

Types of stationary points

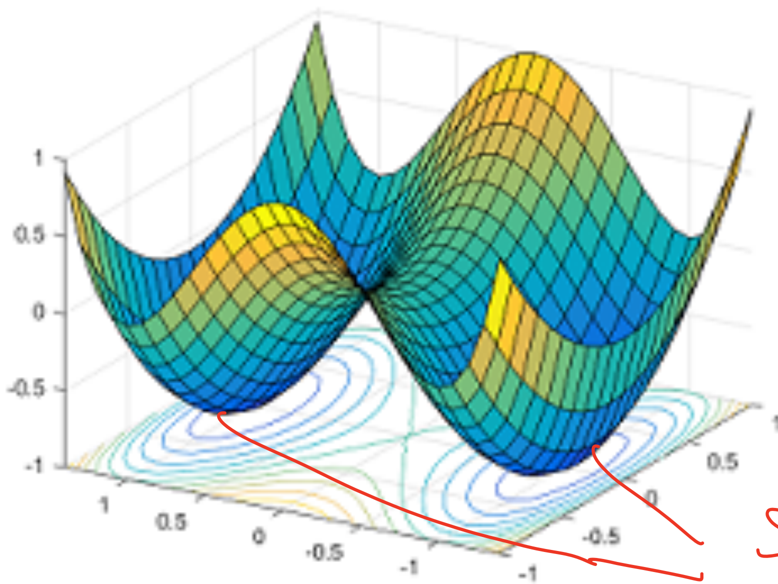
$\min_x f(x)$

found by GD

- Stationary points: $x : \nabla f(x) = 0$
- Global minimum:
  $x : f(x) \leq f(x') \forall x' \in \mathbb{R}^d$
- Local minimum:
  neighborhood
  $x : f(x) \leq f(x') \forall x' : \|x - x'\| \leq \epsilon$
- 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

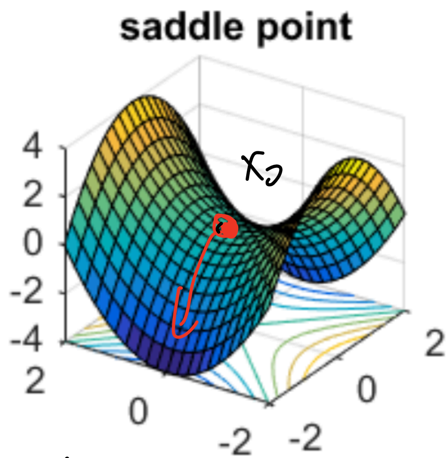


same function value

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

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

$f(x) \simeq (x-x_0)^T \nabla^2 f(x_0) \cdot (x-x_0)$



$|v^T x_{t+1}| \geq (1-\eta\lambda_{min}(A))^t \cdot |v^T x_0|$

$x_0 \neq 0$

$\lambda_{min}(A) < 0$

denote v the eigenvector for $\lambda_{min}(A)$, $\|v\|_2 = 1$

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

Quadratic

$f(x) = \frac{1}{2} x^T A x$

$x = 0$ is a stationary point

$\|x_{t+1}\|_2 \geq |v^T x_{t+1}|$

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

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

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

$\underbrace{1-\eta\lambda_{min}(A)}_{>1}$

Escaping Strict Saddle Points

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

- Noise-injected gradient descent can escape strict saddle points in polynomial time [Ge et al., '15, Jin et al., '17]. $\varepsilon \sim N(0, I)$

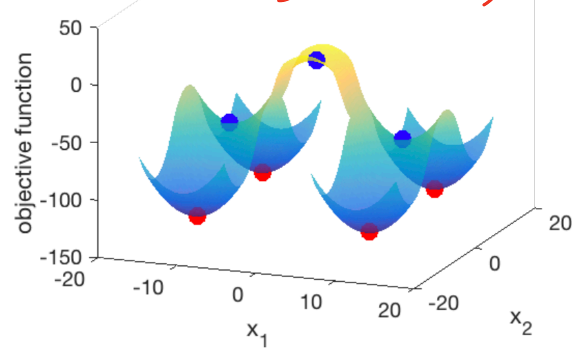
- Randomly initialized gradient descent can escape all strict saddle points asymptotically [Lee et al., '15].
  - Stable manifold theorem.

all points → strict saddle

$\mu(\quad) = 0$

- Randomly initialized gradient descent can take exponential time to escape strict saddle points [Du et al., '17]. $\Omega(2^d)$

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].