Simon OH Today 10:30A

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

$X: \nabla f(x) = 0$

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

ϵ – approximate stationary points

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

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

Sum over $t = 0, \dots, T$

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

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

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

$$\Rightarrow T \cdot \frac{\eta}{2} \min_{0\le t\le T-1}\|\nabla f(x_t)\|_2^2 \le f(x_0) - \min_x f(x)$$

$$\Rightarrow \min_{0\le t\le T-1}\|\nabla f(x_t)\|_2 = \sqrt{\frac{2\beta(f(x_0) - \min_x f(x))}{T}} \le \epsilon$$

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

Gradient Descent for Quadratic Functions

opt: $x = 0$

symmetric, full-rank, $\lambda_{\min}(A) > 0$

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

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

to make $\|x_t\| \leq \varepsilon$

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

$= \|(I - \eta A) x_t\|_2$

$\leq \|I - \eta A\|_2 \cdot \|x_t\|_2$

$= (1 - \eta\lambda_{\min}) \cdot \|x_t\|_2$

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

$\eta = \frac{1}{\lambda_{\max}}$

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

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

- can be generalized strongly convex

- e.g. $A = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$

Momentum: Heavy-Ball Method (Polyak '64)

$x_{t+1} = x_t - \eta_t \cdot \nabla f(x_t)$

Problem: $\min_x f(x)$

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

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

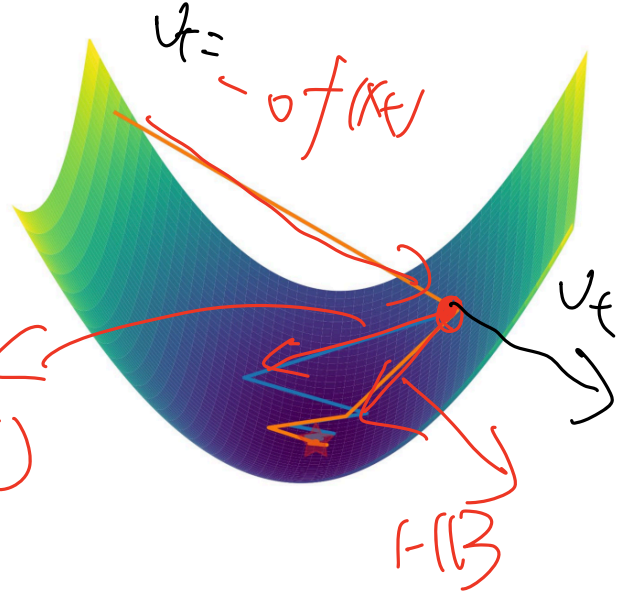
- does not work for strongly convex

For quadratic function:

provable improvement

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

vs. $O\left(K \cdot \log\left(\frac{1}{\varepsilon}\right)\right)$



Momentum: Nesterov Acceleration (Nesterov '89)

Problem: $\min_x f(x)$

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

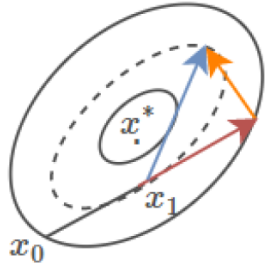
$-\nabla f(x_t)$ for β

lookhead

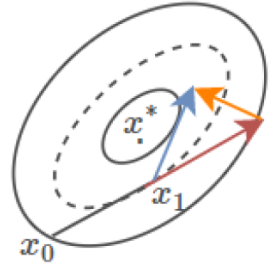
For general strongly convex functions: $O(|x_0 - x^*|^{1/\epsilon})$

- use ODE

Polyak's Momentum



Nesterov Momentum



Newton's Method

2nd order method

odd 3 !

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

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

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

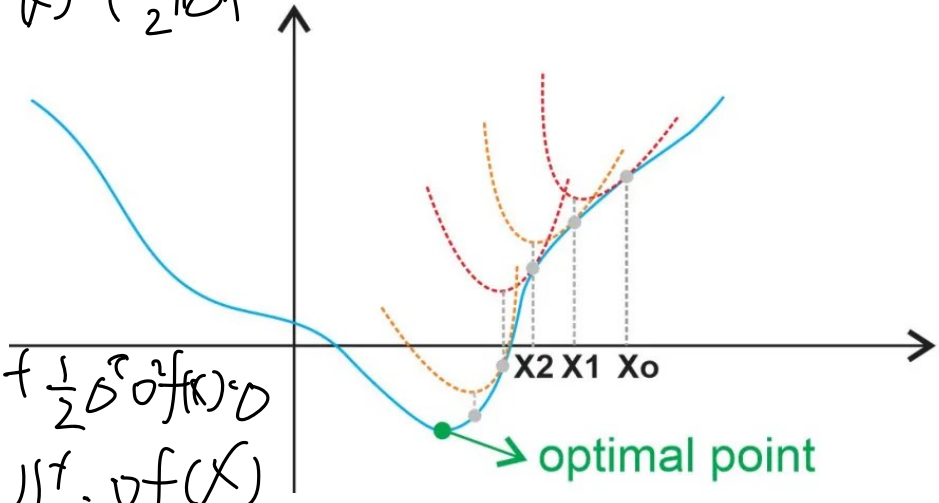
$\Delta \propto -\nabla f(x)$

- Newton:

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

$\Rightarrow \Delta \propto -(\nabla^2 f(x))^{-1} \cdot \nabla f(x)$

$T = O(\log\log(\frac{1}{\varepsilon}))$



AdaGrad (Duchi et al. '11)

(1) diagonal
(2) build inverse Hessian in an online

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

regularization

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

G_t is always diagonal

↑

- effective learning rate: $\eta \cdot (G_t + \epsilon I)^{-1}$ small
- Default hyperparameter work well

RMSProp (Hinton et al. '12)

root mean Squared propagation

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

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

$1 \le t$

$\alpha_1 : \beta^{t-1} \to 0$

$0 < \beta < 1$

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:

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

unit of x

unit issue

$\frac{\partial f}{\partial x} \Big/ \left(\frac{\partial f}{\partial x}\right) \propto$ unitless

$\propto$ unit^2 of x

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

Newton: $\Delta x \propto \frac{\partial f}{\partial x} \Big/ \left(\frac{\partial^2 f}{\partial x^2}\right) \propto$ 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

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

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.

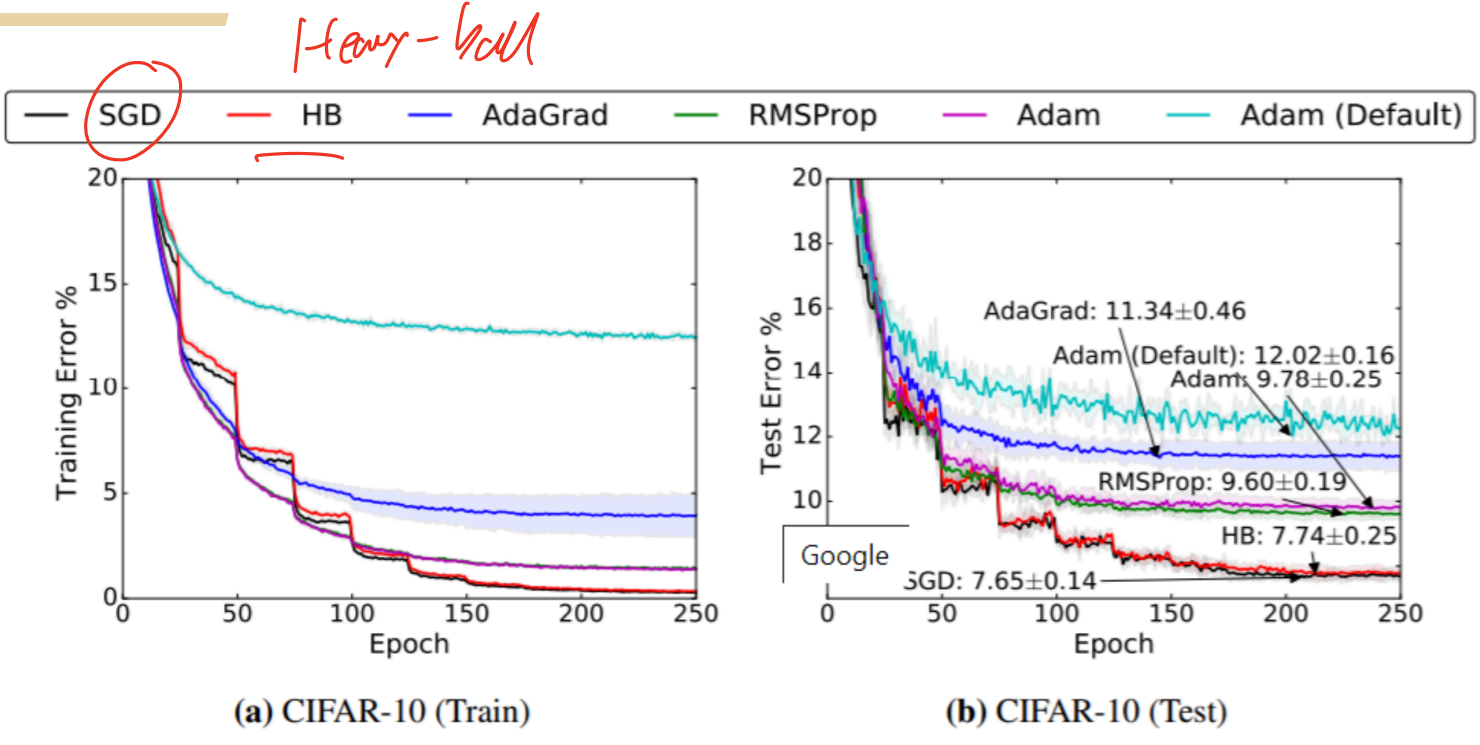
∃ convex problem s.t. Adam does not converge

Other Optimizers

- AdamW
- NAdam
- RAdam
- GGT
- K-FAC
- …

adaptive

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

σ: ReLU

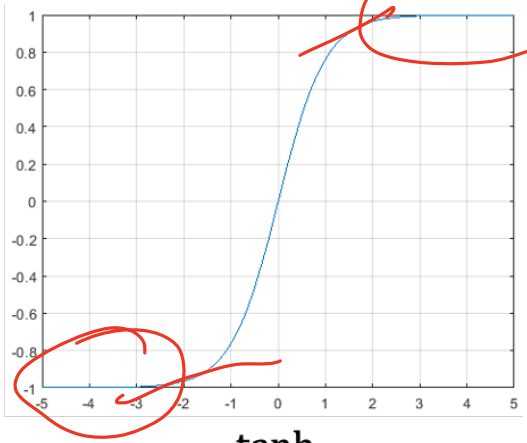
- Deeper networks are harder to train:
  - Intuition: gradients are products over layers
  - Hard to control the learning rate

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

$\frac{\partial f}{\partial w_n} = (W_{H+1} A_H \cdots W_{n+1} A_n)^T (A_{n-1} W_{n-1} \cdots W_1 x)$

- magnitude ↑ gradient exp large
- magnitude ↓ / subspace does align gradient exp small

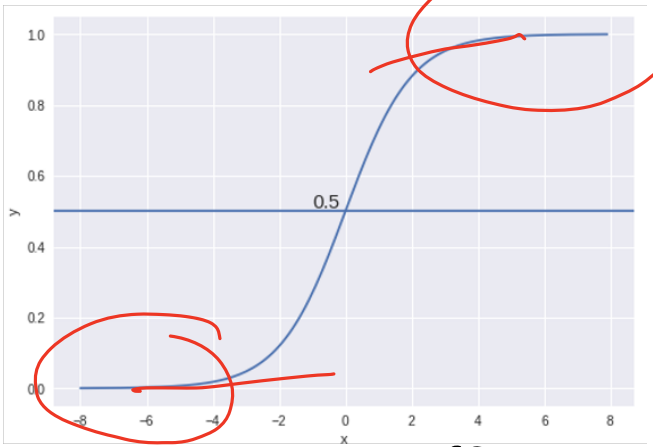
Activation Functions



tanh

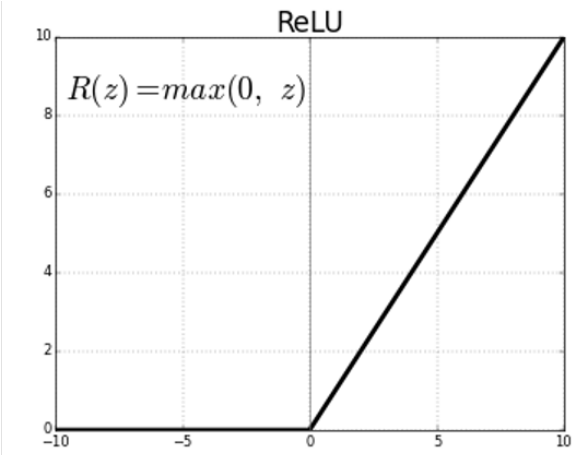
gradient vanishing

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



sigmoid (z)

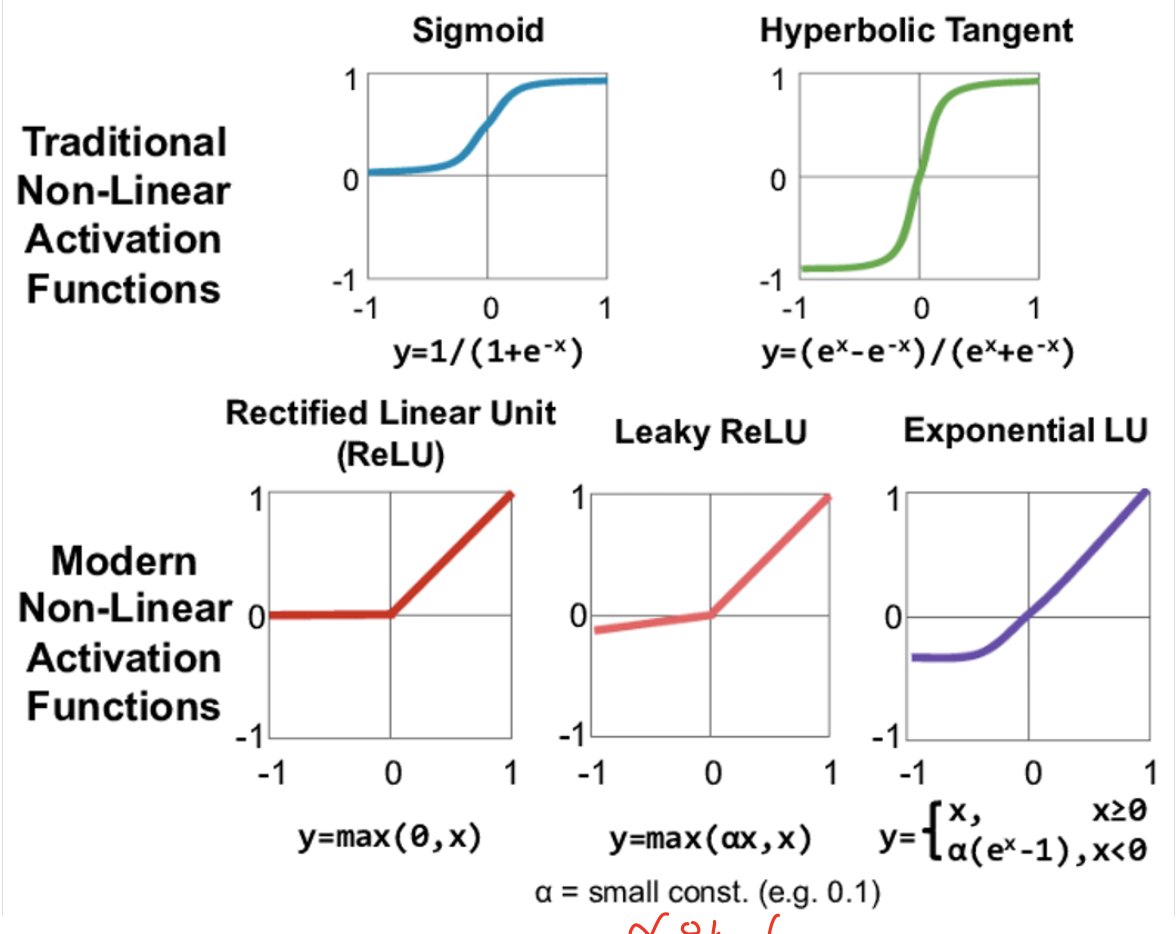
$= \frac{1}{1 + \exp(-z)}$



Rectified Linear United

$g'(z) = 0$ or 1

Activation Function



Initialization

- Zero-initialization
- Large initialization
- Small initialization

- 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)

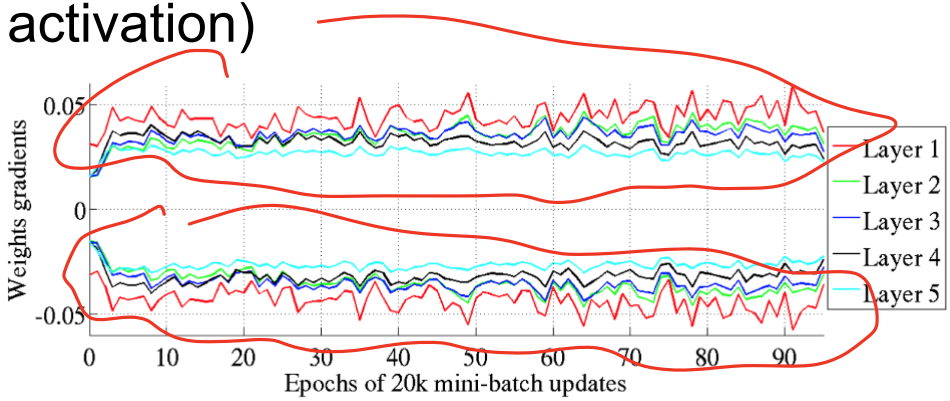
$W^{(h)} \in \mathbb{R}^{d_{h+1} \times d_h}$

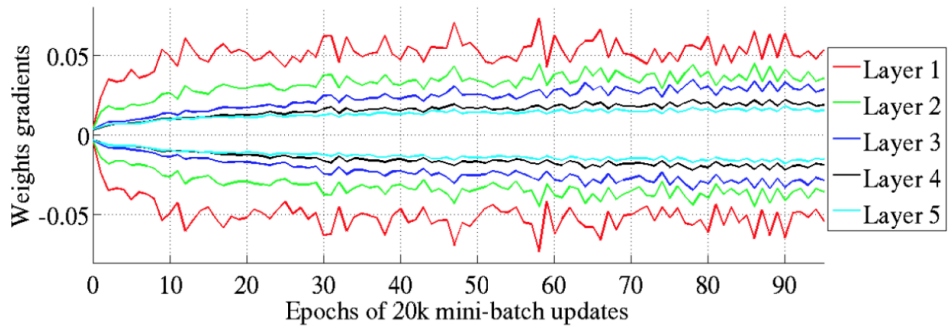
d_h : fan-in

d_{h+1} : fan-out

$$\text{Var}(W_{ij}^{(h)}) = \frac{1}{12} \cdot \left(\frac{2\sqrt{6}}{\sqrt{d_h + d_{h+1}}}\right)^2 = \frac{2}{d_h + d_{h+1}}$$

$$\text{Unif}\left[-\frac{1}{\sqrt{d_{h+1}}}, \frac{1}{\sqrt{d_h}}\right]$$





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