

CS 190I

Deep Learning

Regularization

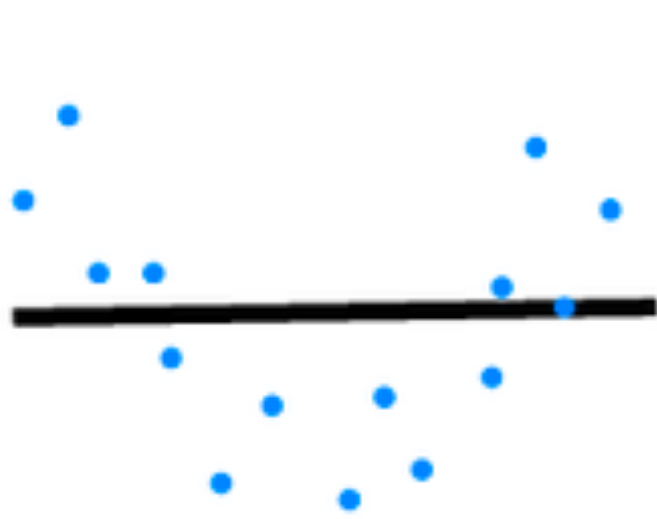
Lei Li (leili@cs)
UCSB

Acknowledgement: Slides borrowed from Bhiksha Raj's 11485 and Mu Li & Alex Smola's 157 courses on Deep Learning, with modification

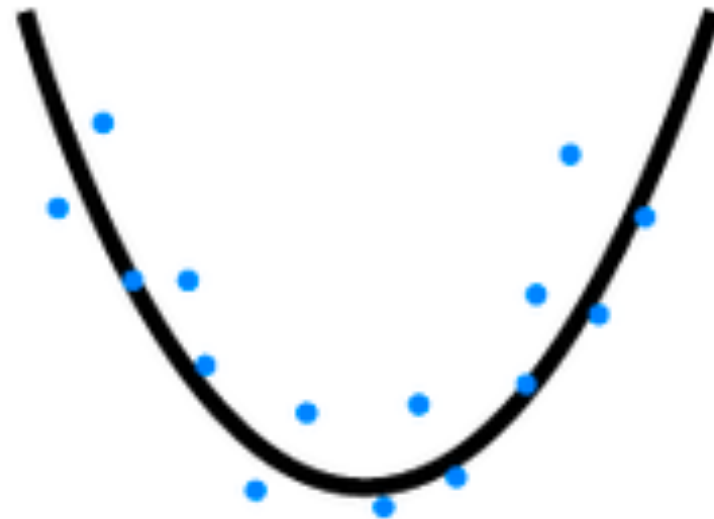
Recap

- Back propagation for Feed-forward neural network
- Model evaluation
- Cross validation
- Overfitting and underfitting

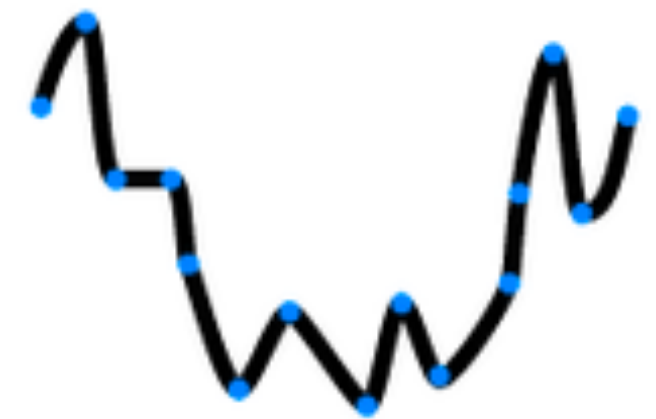
Underfitting and Overfitting



Underfitting



Desired

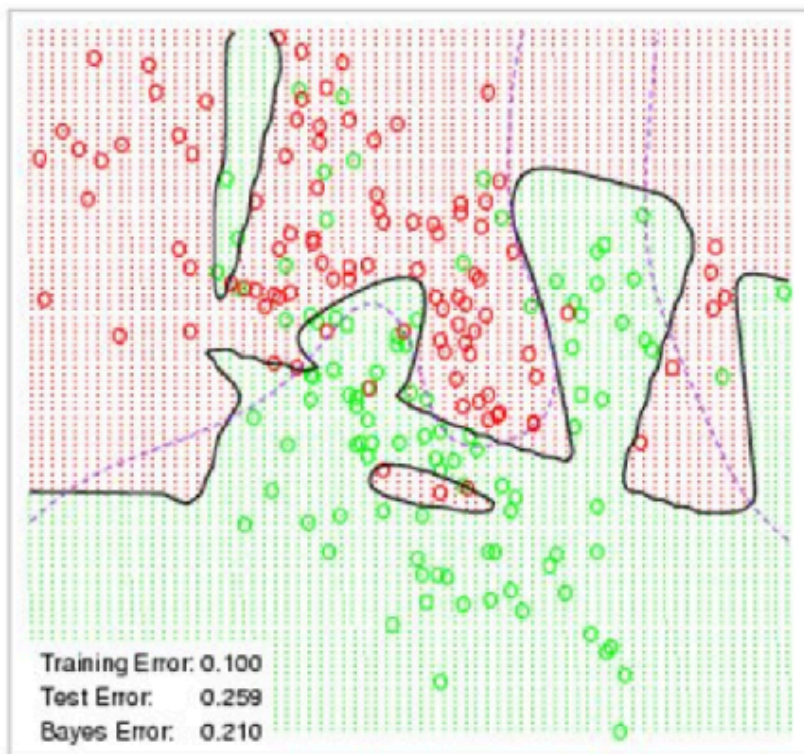


Overfitting

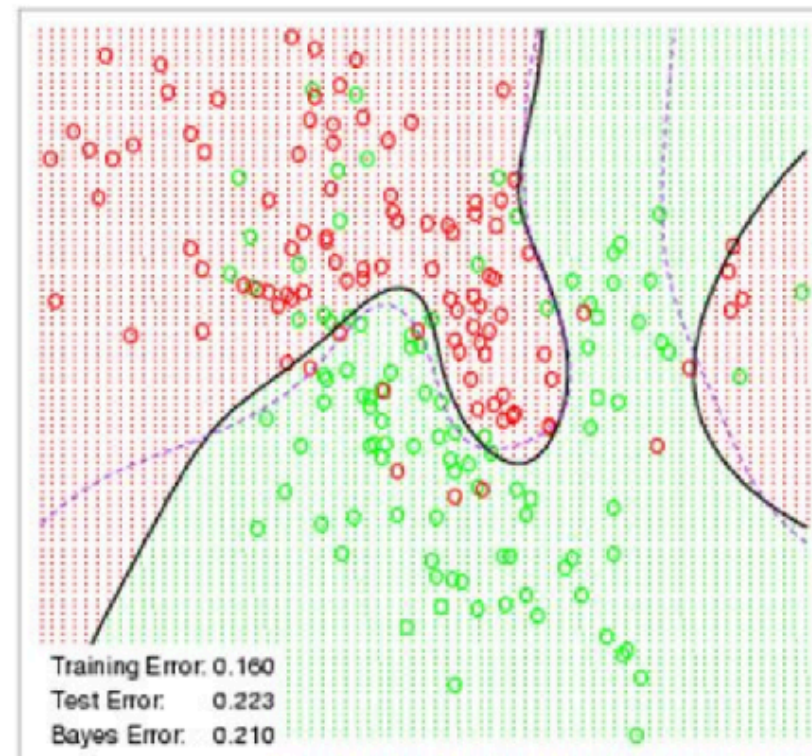
Image credit: hackernoon.com

Regularization

Neural Network - 10 Units, No Weight Decay



Neural Network - 10 Units, Weight Decay=0.02

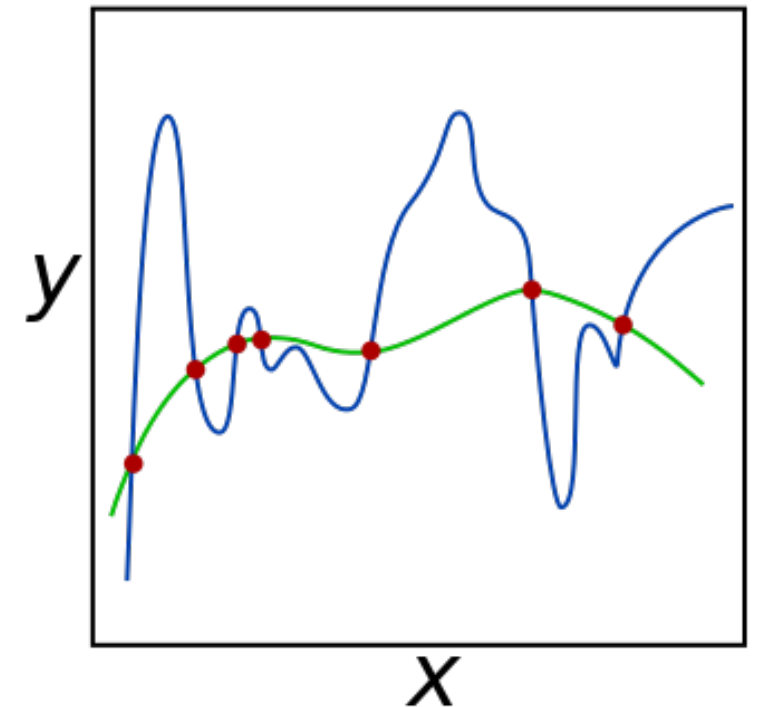


L₂ Regularization as Hard Constraint

- Reduce model complexity by limiting value range

$$\min \ell(\theta) \quad \text{subject to} \quad \|\theta\|^2 \leq \lambda$$

- Often do not regularize bias b
 - Doing or not doing has little difference in practice
- A small λ means more regularization



L₂ Regularization as Soft Constraint

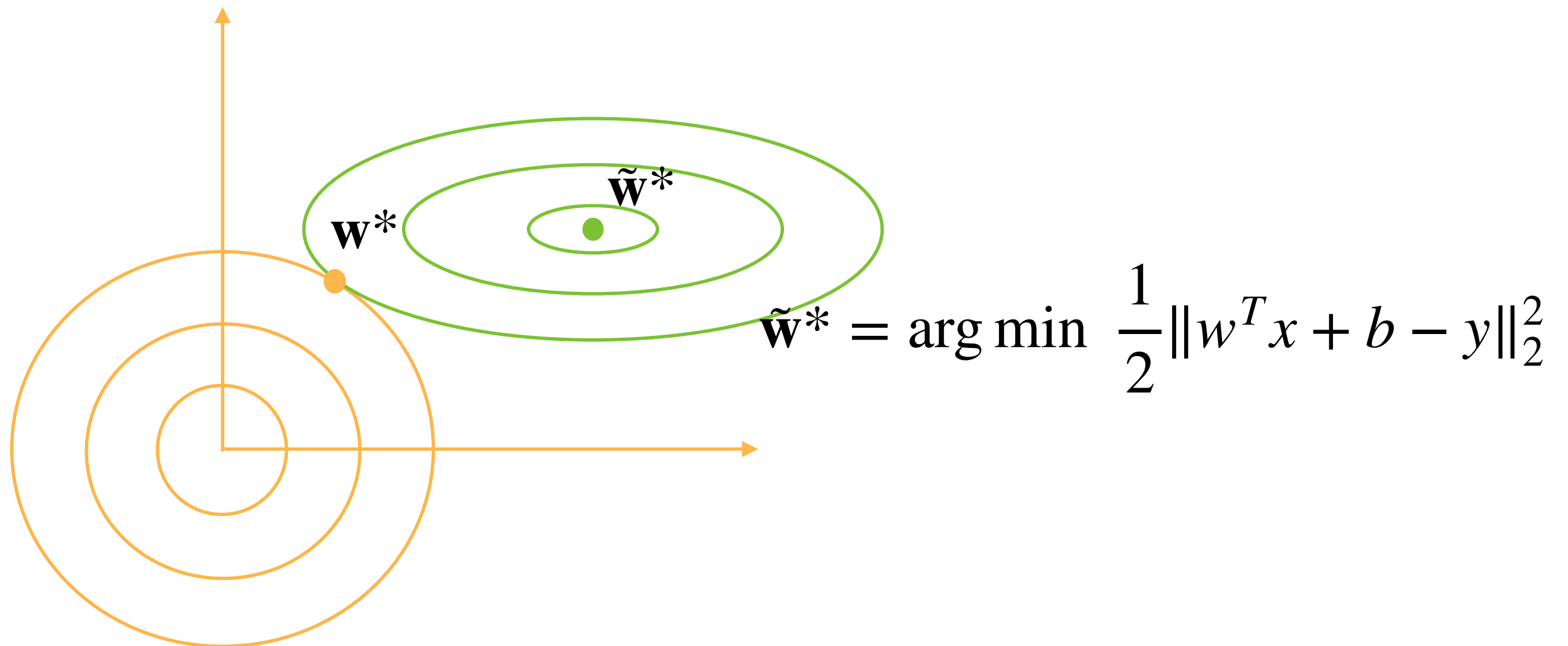
- Using Lagrangian multiplier method
- Minimizing the loss plus additional penalty

$$\min \ell(\theta) + \frac{\lambda}{2} \|\theta\|^2$$

- Hyper-parameter λ controls regularization importance
- $\lambda = 0$: no effect
- $\lambda \rightarrow \infty, \theta^* \rightarrow \mathbf{0}$

Illustrate the Effect on Optimal Solutions

$$\mathbf{w}^* = \arg \min \frac{1}{2} \|w^T x + b - y\|_2^2 + \frac{\lambda}{2} \|\mathbf{w}\|^2$$



Update Rule - Weight Decay

- Compute the gradient

$$\frac{\partial}{\partial \theta} \left(\ell(\theta) + \frac{\lambda}{2} \|\theta\|^2 \right) = \frac{\partial \ell(\theta)}{\partial \theta} + \lambda \theta$$

- Update weight at step t

$$\theta_{t+1} = (1 - \eta\lambda)\theta_t - \eta \frac{\partial \ell(\theta_t)}{\partial \theta_t}$$

backprop



- Often $\eta\lambda < 1$, so also called weight decay in deep learning

Weight Decay in Pytorch

```
import torch
```

```
learning_rate = 1e-3
```

```
weight_decay = 1.0
```

```
optimizer =
```

```
torch.optim.SGD(model.parameters())
```

```
, lr=learning_rate,
```

```
weight_decay=weight_decay)
```

General Penalty

- Minimizing the loss plus additional penalty

$$\min \ell(\theta) + R(\theta)$$

- $\ell(\theta)$ is the original loss
- $R(\theta)$ is penalty (or regularization term), not necessary smooth

L1 Regularization

- Minimizing the loss plus additional penalty

$$\min \ell(\theta) + \lambda |\theta|$$

- $\ell(\theta)$ is the original loss
- using L1 norm as penalty

L1 Update Rule - Soft Thresholding

- $\ell(\theta) + \lambda |\theta|$ is not always differentiable!

- Soft-threshold (Proximal operator):

$$S_\lambda(x) = \text{sign}(x) \max(0, |x| - \lambda) = \text{sign}(x) \text{Relu}(|x| - \lambda)$$

- Update weight at step t

$$\tilde{\theta}_t = \theta_t - \eta \frac{\partial \ell(\theta_t)}{\partial \theta_t}$$

$$\theta_{t+1} = S_\lambda(\tilde{\theta})$$

- Also known as Proximal Gradient Descent

Effects of L1 and L2 Regularization

- L1 Regularization
 - will make parameters sparse (many parameters will be zeros)
 - could be useful for model pruning
- L2 Regularization
 - will make the parameter shrink towards 0, but not necessary 0.

Dropout



Motivation

- A good model should be robust under modest changes in the input
 - Dropout: inject noises into internal layers (simulating the noise)



Add Noise without Bias

- Add noise into \mathbf{x} to get \mathbf{x}' , we hope

$$\mathbf{E}[\mathbf{x}'] = \mathbf{x}$$

- Dropout perturbs each element by

$$x'_i = \begin{cases} 0 & \text{with probability } p \\ \frac{x_i}{1-p} & \text{otherwise} \end{cases}$$

Apply Dropout

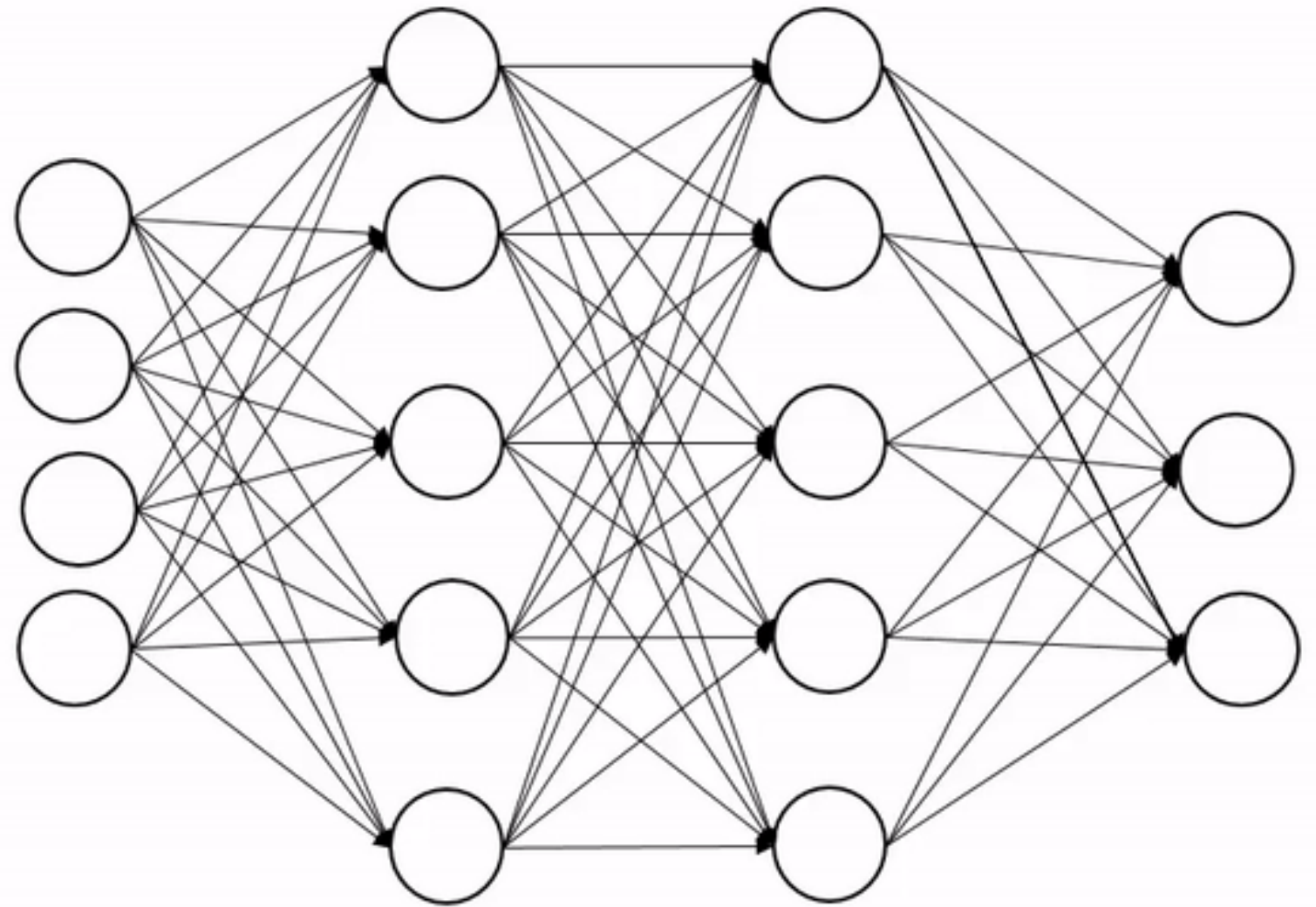
- Often apply dropout on the output of hidden fully-connected layers

$$\mathbf{h} = \sigma(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1)$$

$$\mathbf{h}' = \text{dropout}(\mathbf{h})$$

$$\mathbf{o} = \mathbf{W}_2 \mathbf{h}' + \mathbf{b}_2$$

$$\mathbf{y} = \text{softmax}(\mathbf{o})$$



Dropout in Training and Inference

- Dropout is only used in training

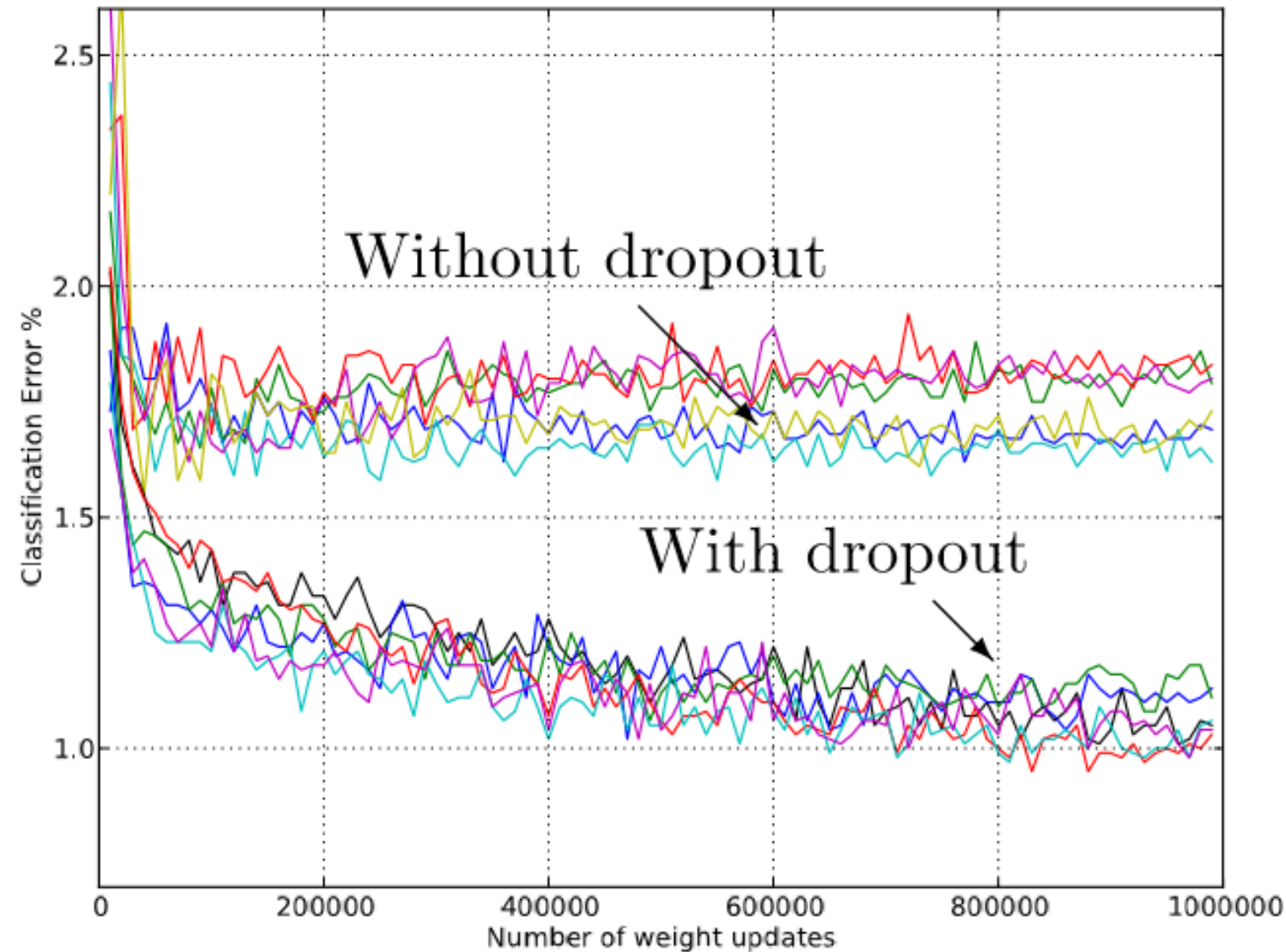
$$\mathbf{h}' = \text{dropout}(\mathbf{h})$$

- No dropout is applied during inference!
- Pytorch Layer:

```
torch.nn.Dropout(p=0.5)
```

Dropout: Typical results

- From Sr
architect
– 2-4 hidc



r different
ropout

Recap

- Regularization
 - to avoid model overfitting
 - L1 \implies more sparse parameters
 - L2/Weight decay \implies shrink parameters
 - Dropout, equivalent to L2, but as a network Layer

Numerical Stability

Gradients for Neural Networks

- Consider a network with d layers

$$\mathbf{h}^t = f_t(\mathbf{h}^{t-1}) \quad \text{and} \quad y = \ell \circ f_d \circ \dots \circ f_1(\mathbf{x})$$

- Compute the gradient of the loss ℓ w.r.t. \mathbf{W}_t

$$\frac{\partial \ell}{\partial \mathbf{W}^t} = \frac{\partial \ell}{\partial \mathbf{h}^d} \frac{\partial \mathbf{h}^d}{\partial \mathbf{h}^{d-1}} \cdots \frac{\partial \mathbf{h}^{t+1}}{\partial \mathbf{h}^t} \frac{\partial \mathbf{h}^t}{\partial \mathbf{W}^t}$$

Multiplication of $d-t$ matrices

Two Issues for Deep Neural Networks

- Two common issues with $\prod_{i=t}^{d-1} \frac{\partial \mathbf{h}^{i+1}}{\partial \mathbf{h}^i}$

Gradient Exploding



$$1.5^{100} \approx 4 \times 10^{17}$$

Gradient Vanishing



$$0.8^{100} \approx 2 \times 10^{-10}$$

Example: FFN

- Assume FFN (without bias for simplicity)

$$f_t(\mathbf{h}^{t-1}) = \sigma(\mathbf{W}^t \mathbf{h}^{t-1}) \quad \sigma \text{ is the activation function}$$

$$\frac{\partial \mathbf{h}^t}{\partial \mathbf{h}^{t-1}} = \text{diag}(\sigma'(\mathbf{W}^t \mathbf{h}^{t-1})) (\mathbf{W}^t)^T \quad \sigma' \text{ is the gradient function of } \sigma$$

$$\prod_{i=t}^{d-1} \frac{\partial \mathbf{h}^{i+1}}{\partial \mathbf{h}^i} = \prod_{i=t}^{d-1} \text{diag}(\sigma'(\mathbf{W}^i \mathbf{h}^{i-1})) (\mathbf{W}^i)^T$$

Gradient Exploding

- Use ReLU as the activation function

$$\sigma(x) = \max(0, x) \quad \text{and} \quad \sigma'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

- Elements of $\prod_{i=t}^{d-1} \frac{\partial \mathbf{h}^{i+1}}{\partial \mathbf{h}^i} = \prod_{i=t}^{d-1} \text{diag}(\sigma'(\mathbf{W}^i \mathbf{h}^{i-1})) (\mathbf{W}^i)^T$ may
from $\prod_{i=t}^{d-1} (\mathbf{W}^i)^T$

- Leads to large values when $d-t$ is large

$$1.5^{100} \approx 4 \times 10^{17}$$

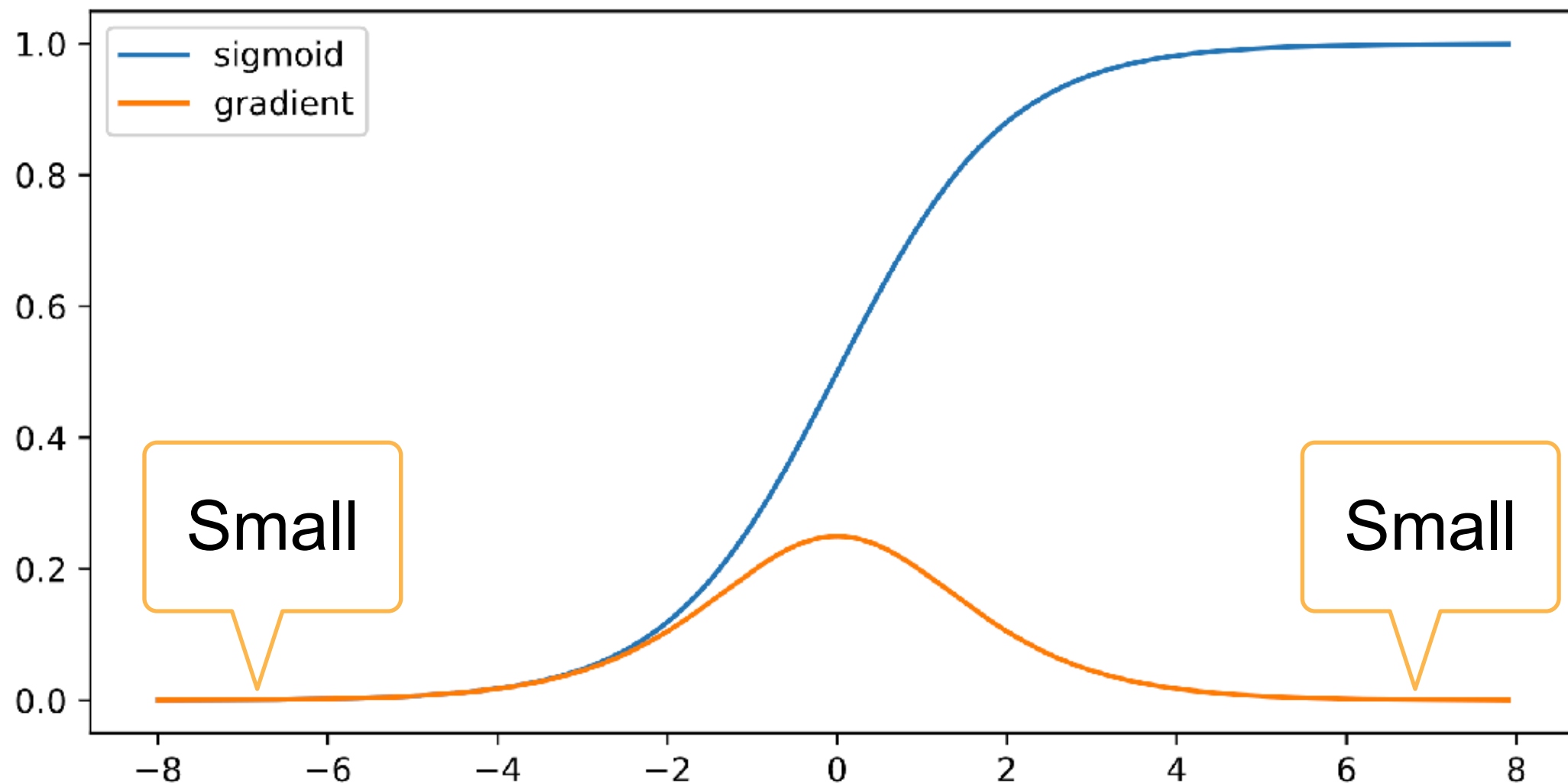
Issues with Gradient Exploding

- Value out of range: infinity value
 - Severe for using 16-bit floating points
 - Range: $6E-5 \sim 6E4$
- Sensitive to learning rate (LR)
 - Not small enough LR \rightarrow large weights \rightarrow larger gradients
 - Too small LR \rightarrow No progress
 - May need to change LR dramatically during training

Gradient Vanishing

- Use sigmoid as the activation function

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad \sigma'(x) = \sigma(x)(1 - \sigma(x))$$



Gradient Exploding

- Use sigmoid as the activation function

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad \sigma'(x) = \sigma(x)(1 - \sigma(x))$$

- Elements $\prod_{i=t}^{d-1} \frac{\partial \mathbf{h}^{i+1}}{\partial \mathbf{h}^i} = \prod_{i=t}^{d-1} \text{diag}(\sigma'(\mathbf{W}^i \mathbf{h}^{i-1})) (\mathbf{W}^i)^T$ are products of $d-t$ small values

$$0.8^{100} \approx 2 \times 10^{-10}$$

Issues with Gradient Vanishing

- Gradients with value 0
 - Severe with 16-bit floating points
- No progress in training
 - No matter how to choose learning rate
- Severe with bottom layers
 - Only top layers are well trained
 - No benefit to make networks deeper

Survey

- <https://edstem.org/us/courses/31035/lessons/55024/slides/311554>

Stabilize Training

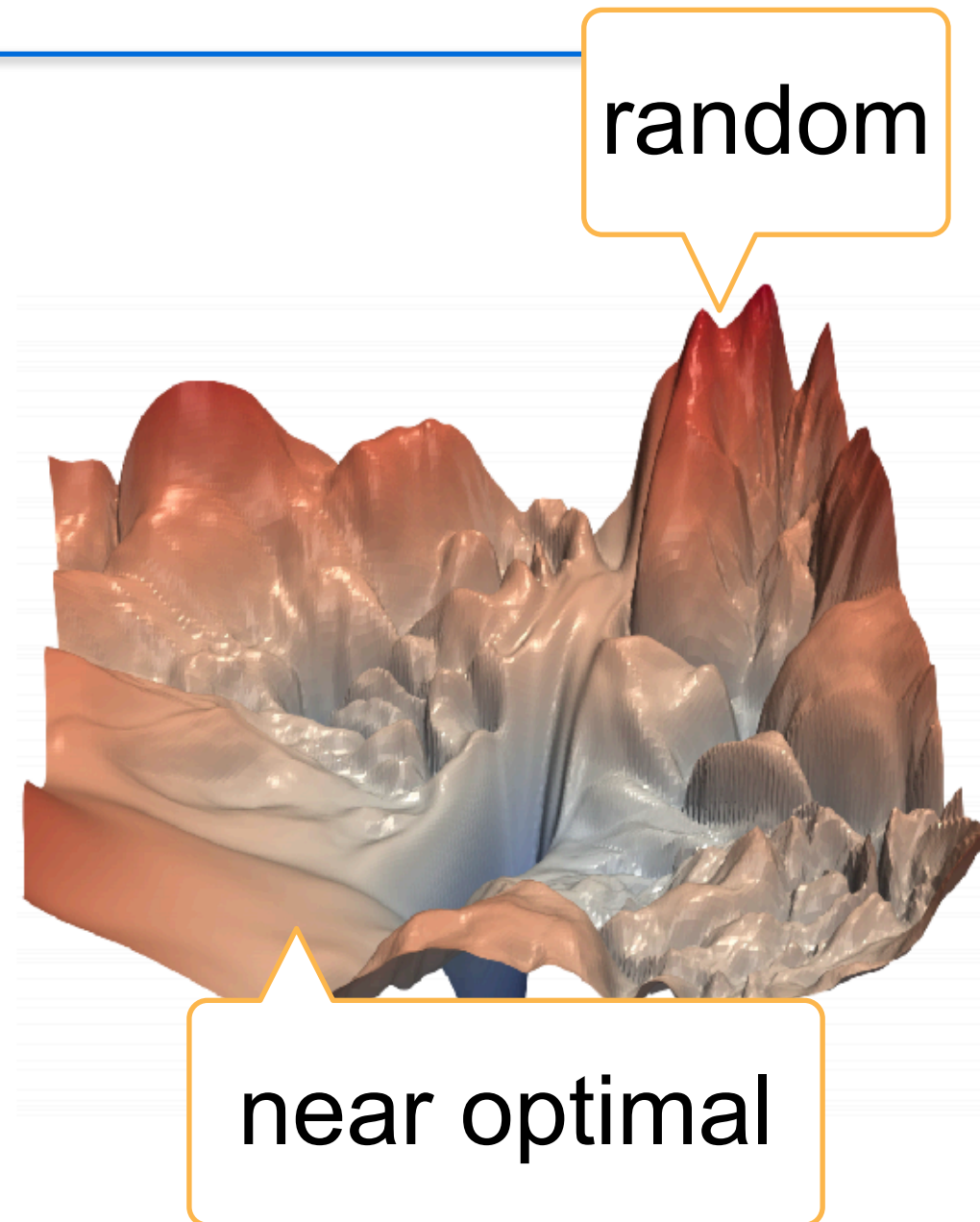


Stabilize Training

- Goal: make sure gradient values are in a proper range
 - E.g. in $[1e-6, 1e3]$
- Multiplication \rightarrow plus
 - ResNet, LSTM (later lecture)
- Normalize
 - Gradient clipping
 - Batch Normalization / Layer Normalization (later)
- Proper weight initialization and activation functions

Weight Initialization

- Initialize weights with random values in a proper range
- The beginning of training easily suffers to numerical instability
 - The surface far away from an optimal can be complex
 - Near optimal may be flatter
- Initializing according to $\mathcal{N}(0, 0.01)$ works well for small networks, but not guarantee for deep neural networks



Constant Variance for each Layer

- Treat both layer outputs and gradients are random variables
- Make the mean and variance for each layer's output are same, similar for gradients

Forward

$$\begin{aligned}\mathbb{E}[h_i^t] &= 0 \\ \text{Var}[h_i^t] &= a\end{aligned}$$

Backward

$$\mathbb{E}\left[\frac{\partial \ell}{\partial h_i^t}\right] = 0 \quad \text{Var}\left[\frac{\partial \ell}{\partial h_i^t}\right] = b \quad \forall i, t$$

a and b are constants

Example: FFN

- Assumptions $\mathbb{E}[w_{i,j}^t] = 0$, $\text{Var}[w_{i,j}^t] = \gamma_t$
 - i.i.d $w_{i,j}^t$,
 - h_i^{t-1} is independent to $w_{i,j}^t$
 - identity activation: $\mathbf{h}^t = \mathbf{W}^t \mathbf{h}^{t-1}$ with $\mathbf{W}^t \in \mathbb{R}^{n_t \times n_{t-1}}$

$$\mathbb{E}[h_i^t] = \mathbb{E} \left[\sum_j w_{i,j}^t h_j^{t-1} \right] = \sum_j \mathbb{E}[w_{i,j}^t] \mathbb{E}[h_j^{t-1}] = 0$$

Forward Variance

$$\begin{aligned}\text{Var}[h_i^t] &= \mathbb{E}[(h_i^t)^2] - \mathbb{E}[h_i^t]^2 = \mathbb{E} \left[\left(\sum_j w_{i,j}^t h_j^{t-1} \right)^2 \right] \\ &= \mathbb{E} \left[\sum_j (w_{i,j}^t)^2 (h_j^{t-1})^2 + \sum_{j \neq k} w_{i,j}^t w_{i,k}^t h_j^{t-1} h_k^{t-1} \right] \\ &= \sum_j \mathbb{E} \left[(w_{i,j}^t)^2 \right] \mathbb{E} \left[(h_j^{t-1})^2 \right] \\ &= \sum_j \text{Var}[w_{i,j}^t] \text{Var}[h_j^{t-1}] = n_{t-1} \gamma_t \text{Var}[h_j^{t-1}] \quad \Rightarrow \quad n_{t-1} \gamma_t = 1\end{aligned}$$

n_{t-1} is the number of units in t-1 layer

Backward Mean and Variance

- Apply forward analysis as well

$$\frac{\partial \ell}{\partial \mathbf{h}^{t-1}} = \frac{\partial \ell}{\partial \mathbf{h}^t} \mathbf{W}^t \quad \text{leads to} \quad \left(\frac{\partial \ell}{\partial \mathbf{h}^{t-1}} \right)^T = (\mathbf{W}^t)^T \left(\frac{\partial \ell}{\partial \mathbf{h}^t} \right)^T$$

$$\mathbb{E} \left[\frac{\partial \ell}{\partial h_i^{t-1}} \right] = 0$$

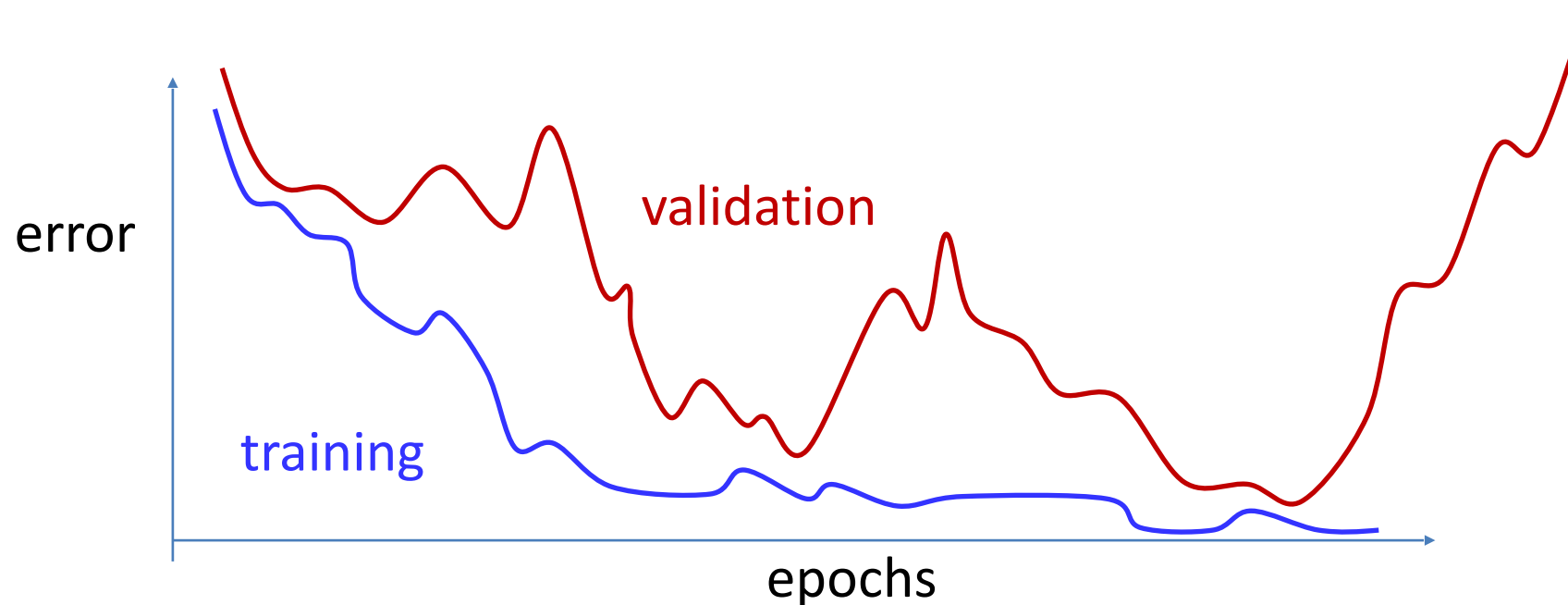
$$\text{Var} \left[\frac{\partial \ell}{\partial h_i^{t-1}} \right] = n_t \gamma_t \text{Var} \left[\frac{\partial \ell}{\partial h_j^t} \right] \quad \Rightarrow \quad n_t \gamma_t = 1$$

Xavier Initialization

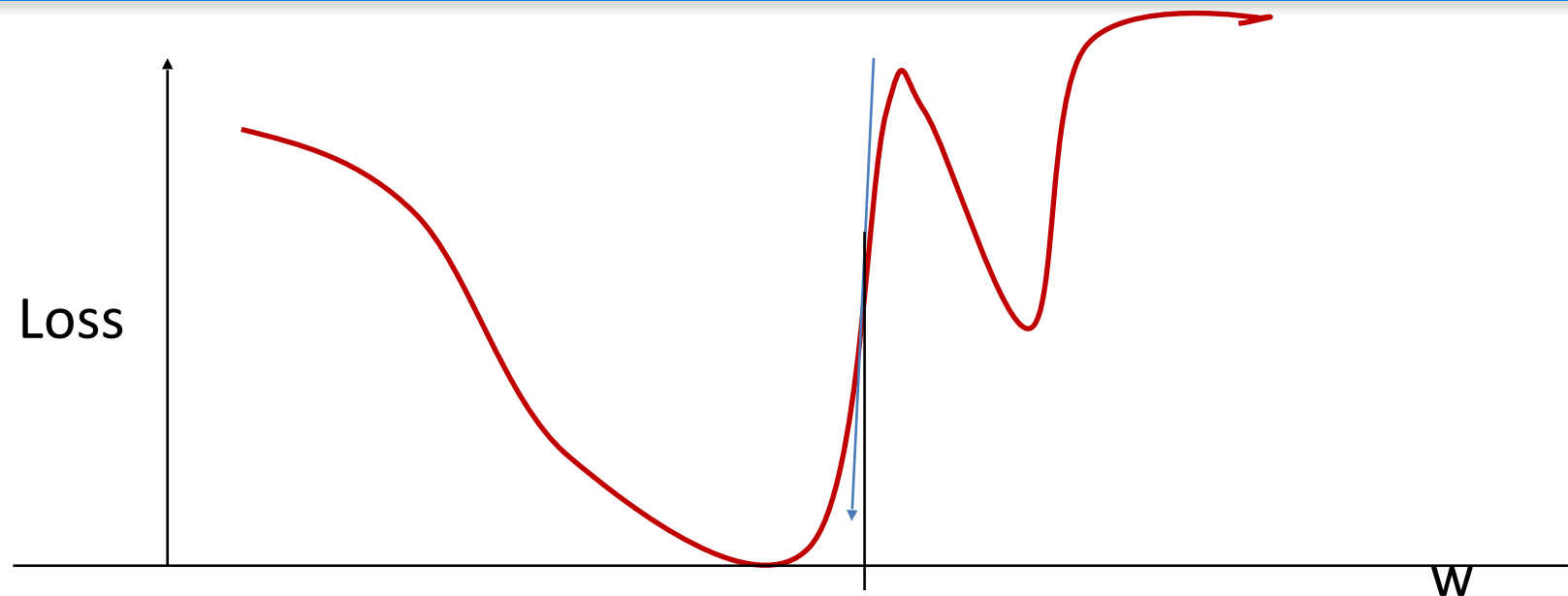
- Conflict goal to satisfies both $n_{t-1}\gamma_t = 1$ and $n_t\gamma_t = 1$
- **Xavier** $\gamma_t(n_{t-1} + n_t)/2 = 1 \rightarrow \gamma_t = 2/(n_{t-1} + n_t)$
 - Normal distribution $\mathcal{N}\left(0, \sqrt{2/(n_{t-1} + n_t)}\right)$
 - Uniform distribution $\mathcal{U}\left(-\sqrt{6/(n_{t-1} + n_t)}, \sqrt{6/(n_{t-1} + n_t)}\right)$
 - Variance of $\mathcal{U}[-a, a]$ is $a^2/3$
- Adaptive to weight shape, especially when n_t varies

Other heuristics: Early stopping

- Continued training can result in over fitting to training data
 - Track performance on a held-out validation set
 - Apply one of several early-stopping criterion to terminate training when performance on validation set degrades significantly



Additional heuristics: Gradient clipping



- Often the derivative will be too high
 - When the divergence has a steep slope
 - This can result in instability
- **Gradient clipping**: set a ceiling on derivative value

$$\textit{if } \partial_w D > \theta \textit{ then } \partial_w D = \theta$$

- Typical θ value is 5
- Can be easily set in pytorch/tensorflow

Recap

- Numerical issues in training
 - gradient explosion
 - gradient vanishing
- Proper initialization of parameters

Next Up

- Reading Material: Chap 7 in D2L
- Convolutional Neural Networks
- Visual perception:
 - Image classification
 - Object recognition
 - Face detection