Computational Intelligence

Samaneh Hosseini

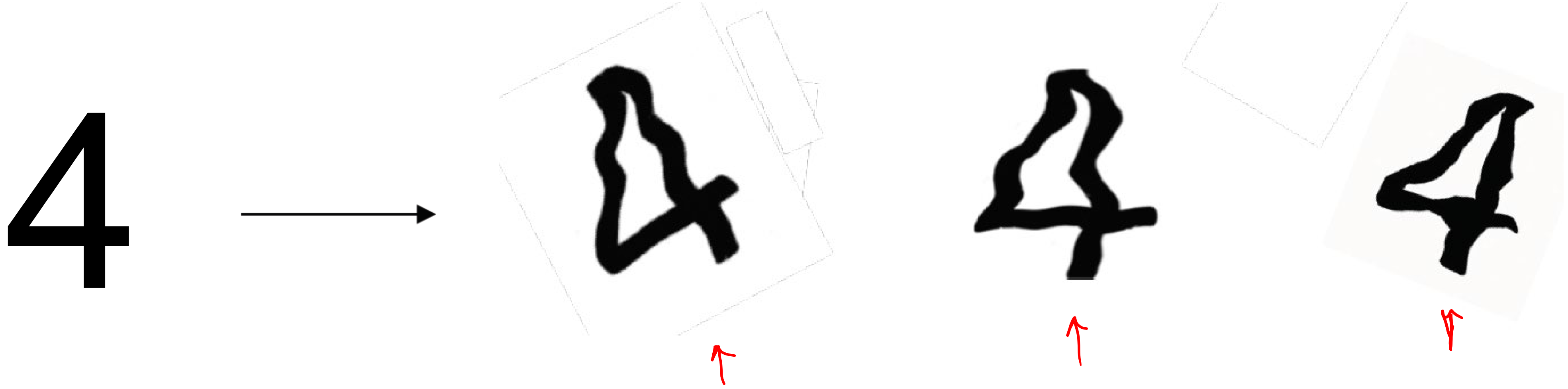
Isfahan University of Technology

Outline

- Other Regularization Techniques
  - Data Augmentation
  - Early Stopping
- Setting up Your Optimization Problem
  - Normalizing Inputs
  - Vanishing/Exploding Gradients
  - Numerical approximation of gradients
  - Gradient Checking
  - Gradient Checking Implementation Notes

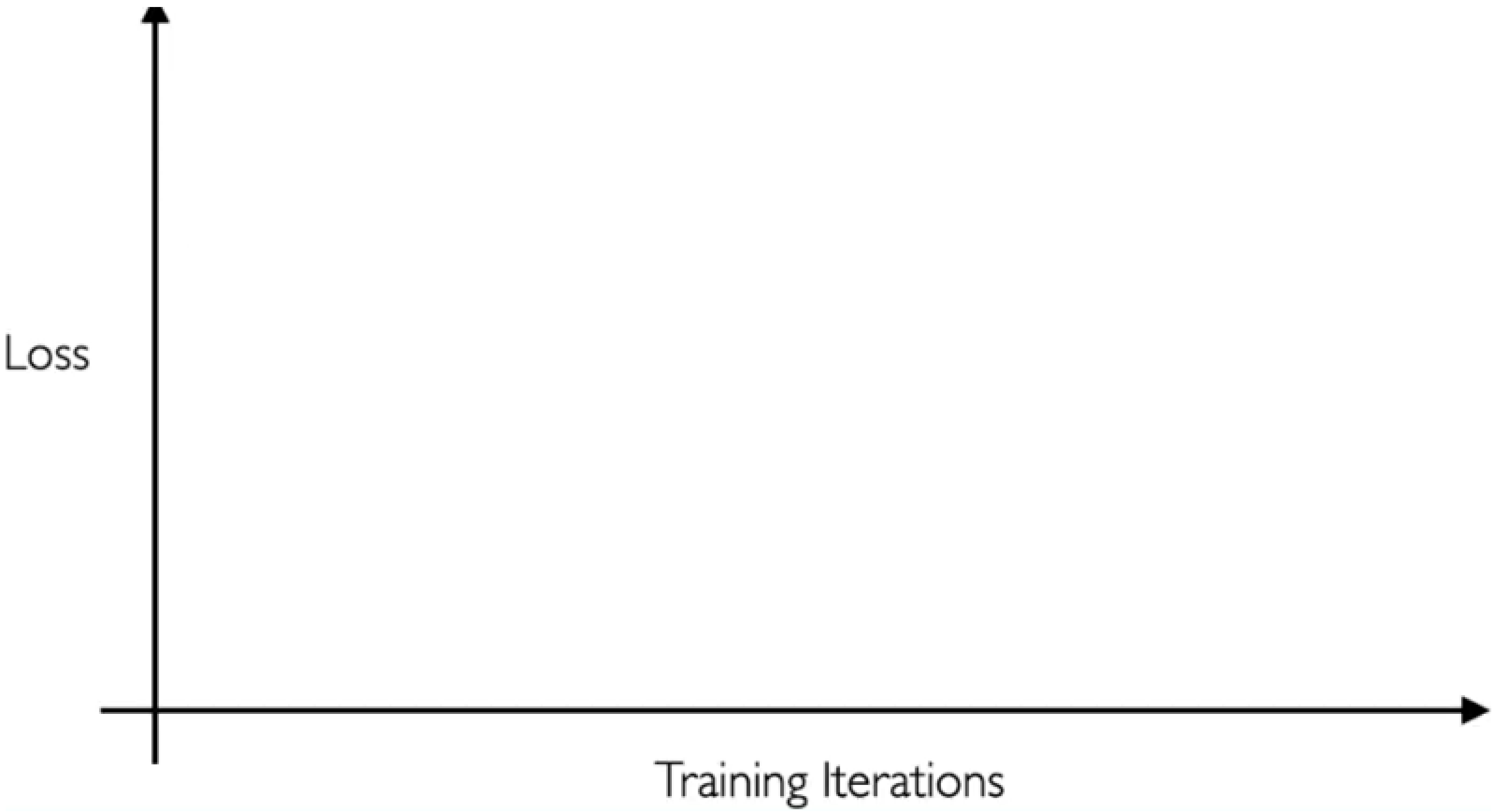
Other Regularization Techniques: Data Augmentation

Data Augmentation



Early Stopping

- Stop training before we have a chance to overfit



dev set

train set

Early Stopping

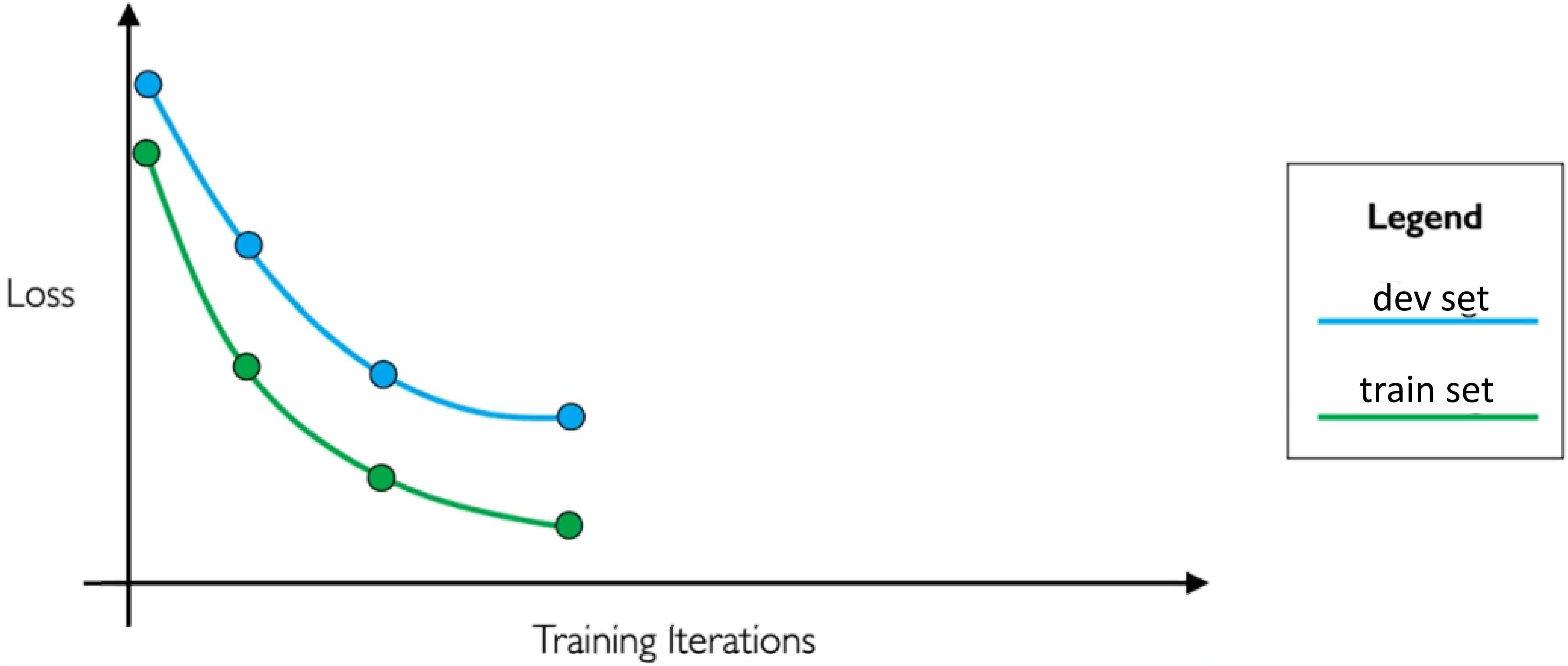
- Stop training before we have a chance to overfit

Early Stopping

- Stop training before we have a chance to overfit

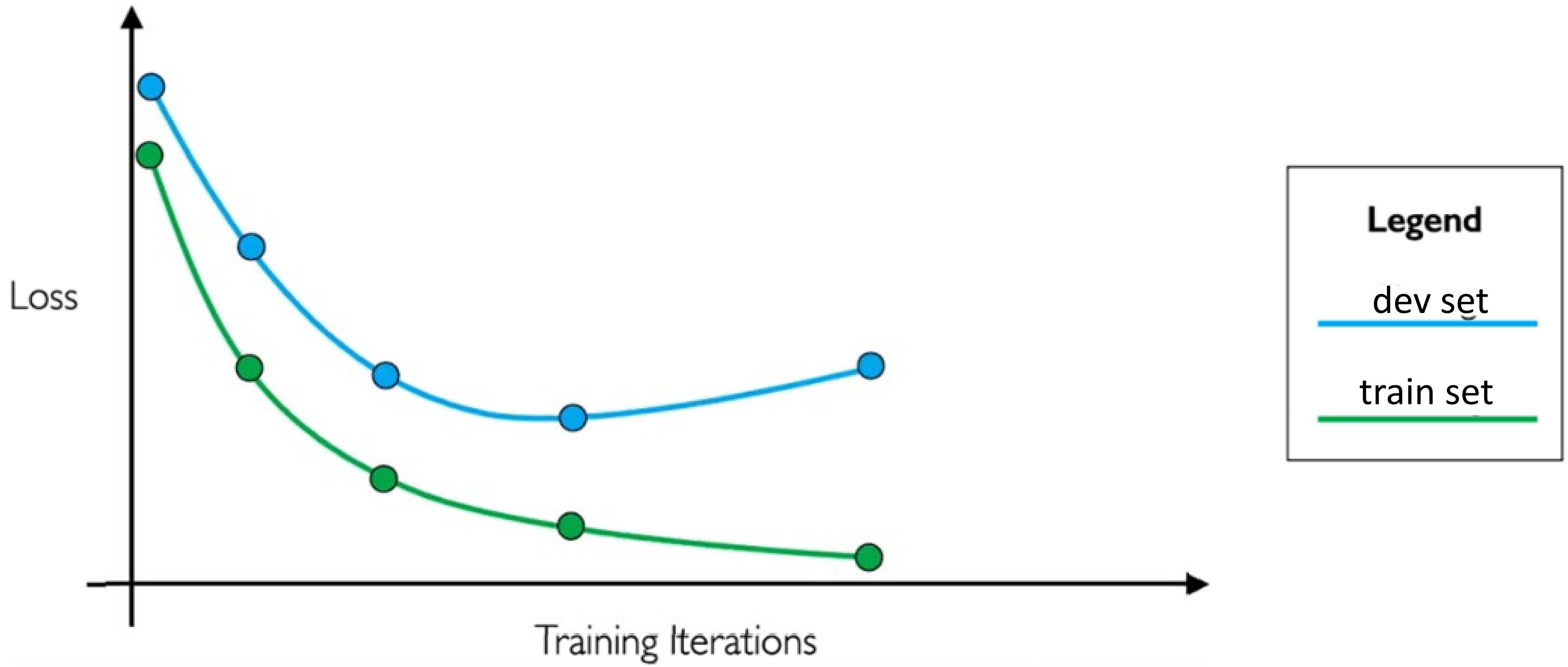
Early Stopping

- Stop training before we have a chance to overfit



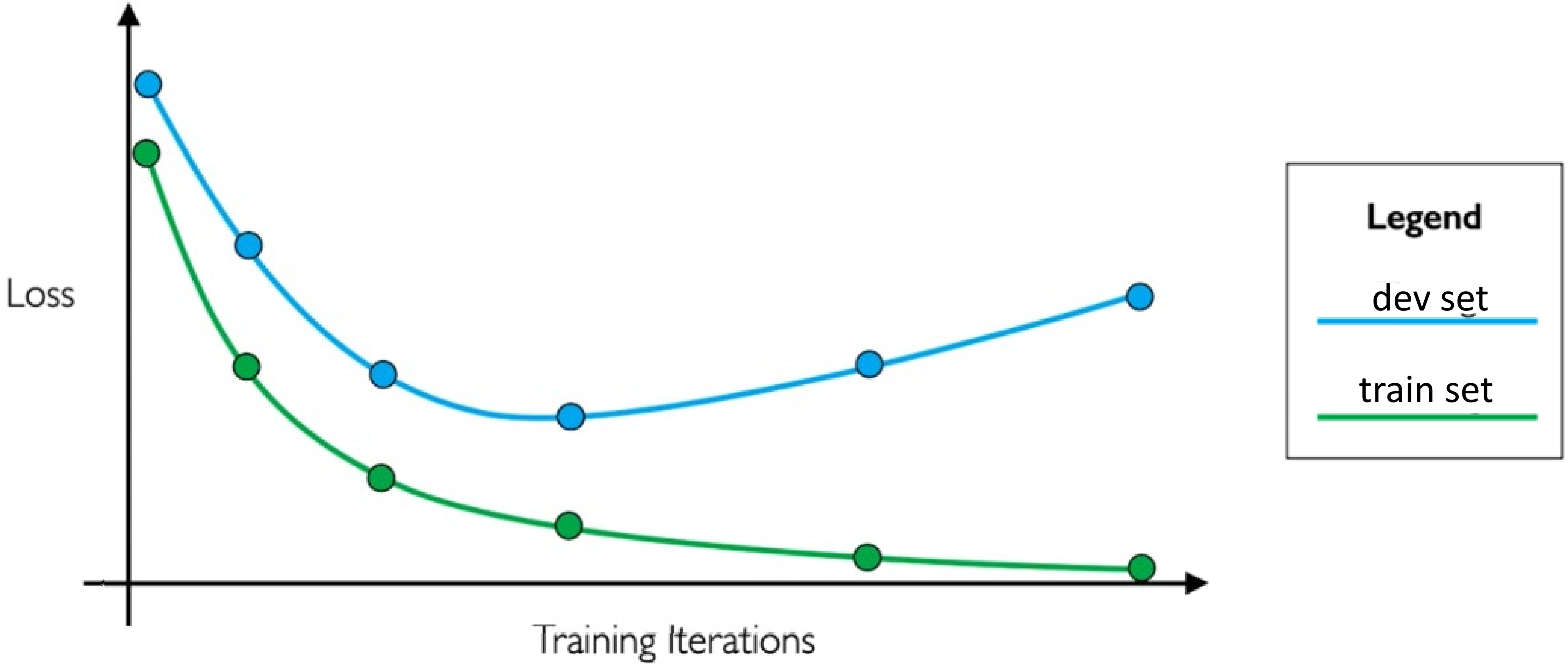
Early Stopping

- Stop training before we have a chance to overfit



Early Stopping

- Stop training before we have a chance to overfit



Early Stopping

- Stop training before we have a chance to overfit

Orthogonalization

- optimize cost function J
  - gradient

$J(w,b)$

- Not overfit
  - Regularization

$L_2 \to \lambda$

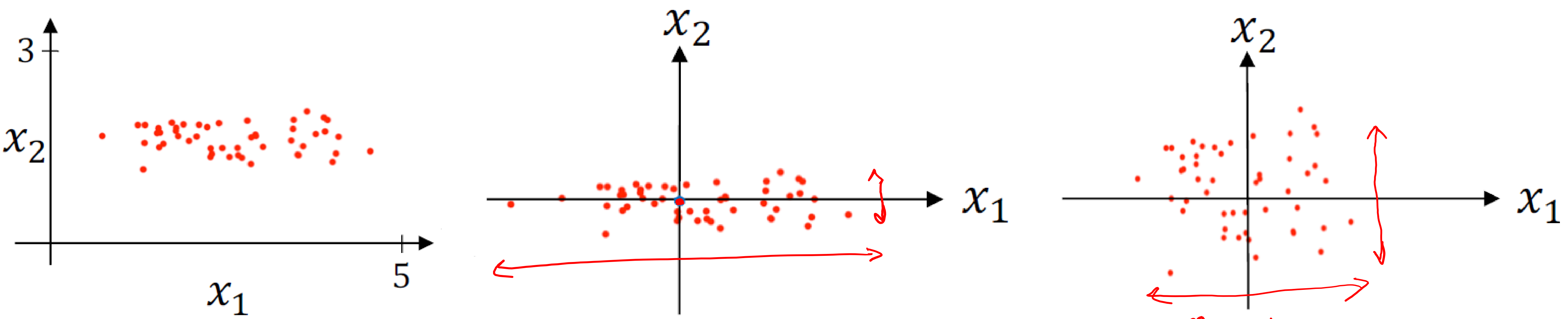
Early Stopping

- Stop training before we have a chance to overfit

Setting up Your Optimization Problem: Normalizing inputs

Normalizing inputs

$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$



subtract mean:

$$\mu = \frac{1}{m} \sum_{i=1}^{m} x^{(i)}$$

$$x := x - \mu$$

$$\sigma^2 = \frac{1}{m} \sum_{i=1}^{m} x^{(i)} ** 2$$

↑ element-wise

$$x \mathrel{/}= \sigma^2$$

↑ normalize variance

use μ σ^2 to normalize test data.

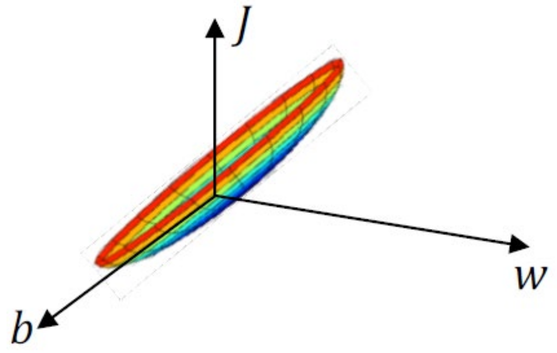
Why normalize inputs?

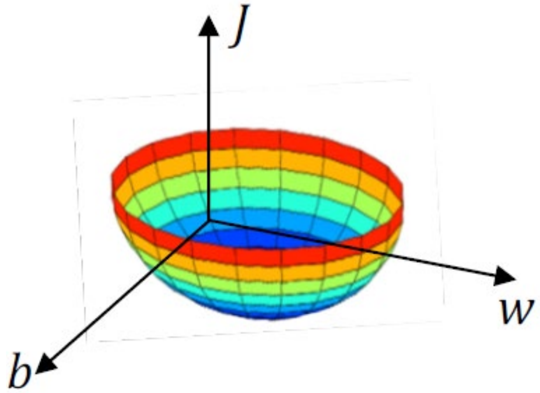
$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}\left(\hat{y}^{(i)}, y^{(i)}\right)$$

$w \quad x_1 : 1 \ldots\ldots 1000$

Unnormalized: $b \quad x_2 : 0 \ldots 1$

Normalized:

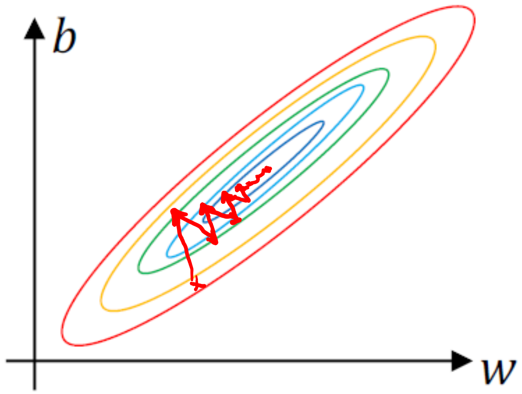


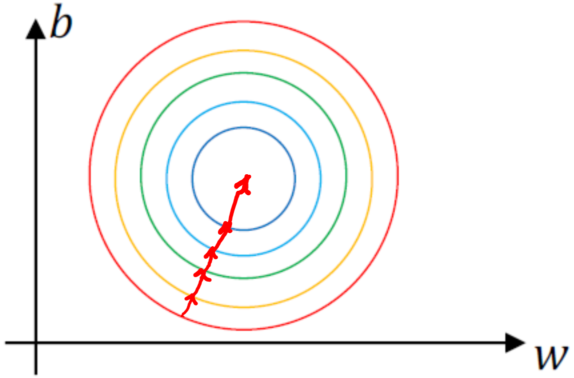


$x_1 : 0 \ldots 1$

$x_2 : -1 \ldots 1$

$x_3 : 1 \ldots 2$

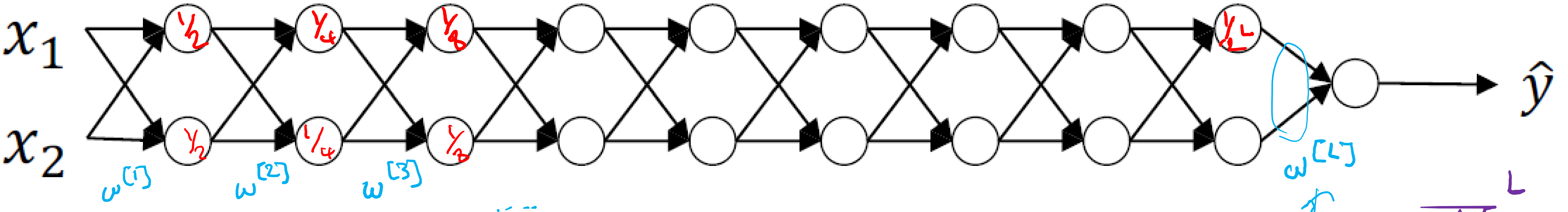




Setting up Your Optimization Problem: Vanishing/Exploding Gradients

Vanishing/Exploding Gradients

$L=150$



1.5^L

0.5^L

$g(z) = z \qquad b^{[l]} = 0$

$$\hat{y} = w^{[L]} \, w^{[L-1]} \, w^{[L-2]} \cdots w^{[3]} \, w^{[2]} \, w^{[1]} X$$

$$z^{[1]} = w^{[1]} X$$

$$a^{[1]} = g(z^{[1]}) = z^{[1]}$$

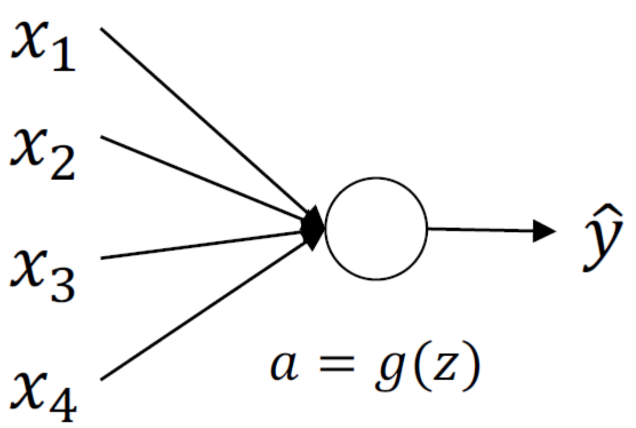
$$a^{[2]} = g(z^{[1]}) = g(w^{[2]} a^{[1]})$$

$$w^{[l]} = \begin{bmatrix} \cancel{1.5} \; 0.5 & 0 \\ 0 & \cancel{1.5} \; 0.5 \end{bmatrix}$$

$1 \leq l < L$

$$\hat{y} = w^{[L]} \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}^{L-1} X$$

Single neuron example



$z = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + \cancel{b}$

Large $n \longrightarrow$ smaller w_i

$var(w_i) \approx \frac{1}{n}$

$w^{[l]} = np.random.rand(shape) * np.sqrt\left(\frac{2}{n^{[l-1]}}\right)$

Relu

tanh $\sqrt{\frac{1}{n^{[l-1]}}}$

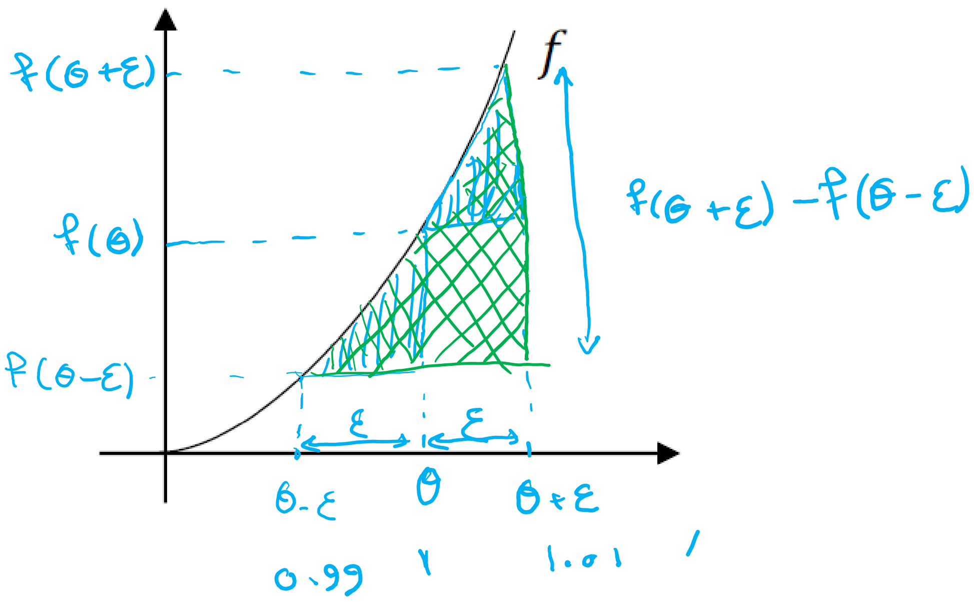
Xavier initialization

$\sqrt{\frac{2}{n^{[l-1]} + n^{[l]}}}$

Setting up Your Optimization Problem: Numerical approximation of gradients

Checking your derivative computation

$f(\theta) = \theta^3$



$\varepsilon = 0.01$

$\longleftrightarrow$ 2ε

$$\frac{f(\theta+\varepsilon) - f(\theta-\varepsilon)}{2\varepsilon} \approx g(\theta)$$

$$\frac{(1.01)^3 - (0.99)^3}{2(0.01)} = \underline{3.0001} \approx 3$$

$$g(\theta) = 3\theta^2 = \underline{3}$$

appox error : $\underline{0.0001}$

$0.1 \rightarrow 0.0001$ error : $\underline{0.03}$ $\leftarrow$ $g(\theta) : 3.0301$

$$\frac{f(\theta+\varepsilon) - f(\theta-\varepsilon)}{2\varepsilon} \rightarrow \text{error} : O(\varepsilon^2)$$

$$\frac{f(\theta+\varepsilon) - f(\theta)}{\varepsilon}$$ error : $O(\varepsilon)$ 0.01

Setting up Your Optimization Problem: Gradient Checking

Gradient check for a neural network

- Take $w^{[1]}, b^{[1]} \ldots, w^{[L]}, b^{[L]}$ and reshape into a big vector θ.

concatinate

$J(w^{[1]}, b^{[1]}, \ldots, w^{[L]}, b^{[L]}) = J(\theta)$

- Take $dw^{[1]}, db^{[1]} \ldots, dw^{[L]}, db^{[L]}$ and reshape into a big vector $d\theta$.

concatinate

Is $d\theta$ graidient of $J(\theta)$?

Gradient checking (Grad check)

$J(\theta) = J(\theta_1, \theta_2, \theta_3 \dots \theta_i)$

for each i :

$$d\theta_{approximate}[i] = \frac{J(\theta_1, \theta_2 \dots \theta_i + \varepsilon \dots) - J(\theta_1, \theta_2 \dots \theta_i - \varepsilon \dots)}{2\varepsilon}$$

$$\approx d\theta[i] = \frac{\partial J}{\partial \theta_i}$$

$d\theta_{approximate} \overset{?}{\approx} d\theta$

check : $\frac{\| d\theta_{app..} - d\theta \|_2}{\| d\theta_{app} \|_2 + \| d\theta \|_2}$ $\approx$ $\boxed{10^{-7} - \text{great!}}$

10^{-5}

10^{-3} – worry

Setting up Your Optimization Problem: Gradient Checking Implementation Notes

Gradient Checking Implementation Notes

- Don't use in training – only to debug

$\underbrace{d\theta_{approx}^{[i]}}$ $d\theta^{[i]}$

- If algorithm fails grad check, look at components to try to identify bug.

$db^{[l]}$

$J(\theta) = \frac{1}{m}\sum_i \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \underbrace{\frac{\lambda}{2m}\sum_l \|w^{[l]}\|_F^2}$

- Remember regularization.

- Doesn't work with dropout.

keep_prob=1.0 $d\theta$ J

- Run at random initialization; perhaps again after some training.

w, b ≈ 0

Core Foundation Review

- Other Regularization Techniques
  - Data Augmentation
  - Early Stopping
- Setting up Your Optimization Problem
  - Normalizing Inputs
  - Vanishing/Exploding Gradients
  - Numerical approximation of gradients
  - Gradient Checking
  - Gradient Checking Implementation Notes