



Deep Learning

Dr. Mehran Safayani

safayani@iut.ac.ir

safayani.iut.ac.ir



<https://www.aparat.com/mehran.safayani>



https://github.com/safayani/deep_learning_course

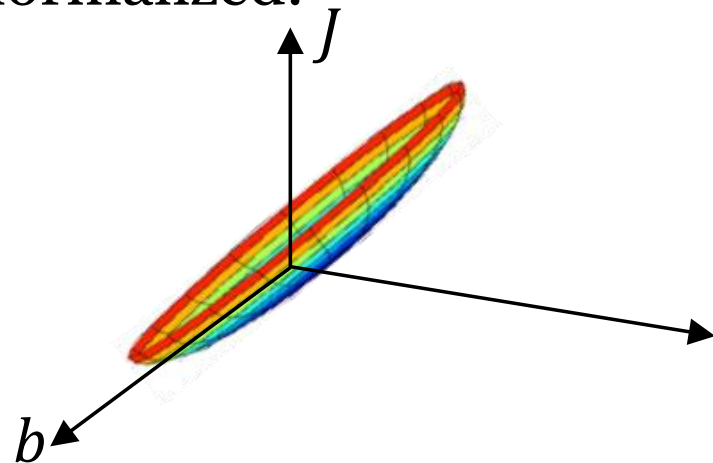


Improving deep neural networks

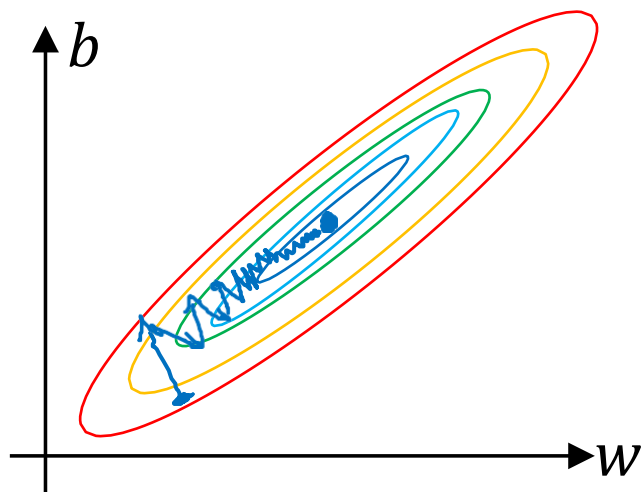
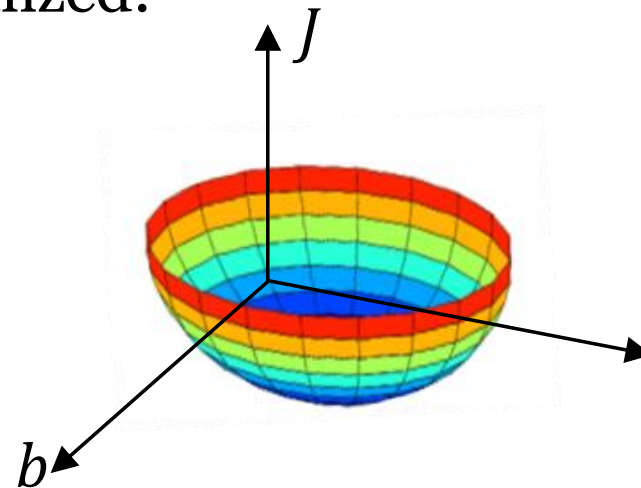
Why normalize inputs?

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

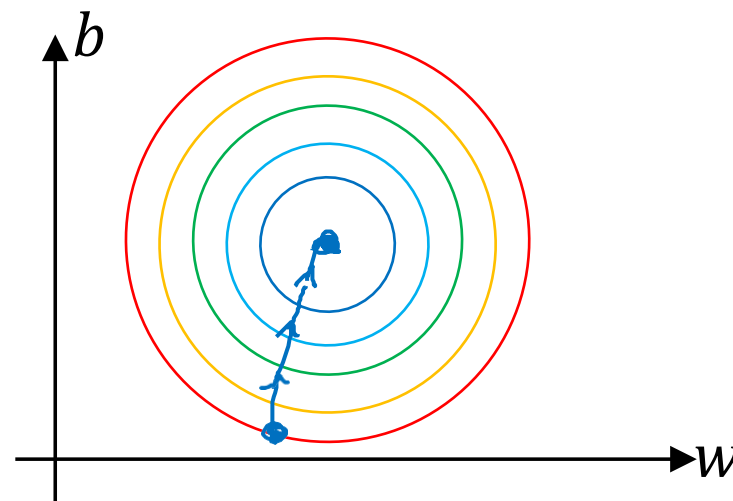
Unnormalized:



Normalized:



x_1 : 0_100 0...1
 x_2 : -1,1 -1...1
 x_3 : 0.001_0.005 1...2

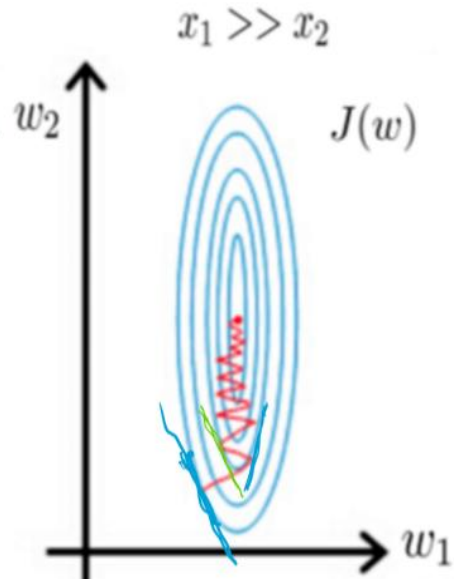


Contour Plot

$$\frac{w_1^2}{b^2} + \frac{w_2^2}{a^2} = 1$$

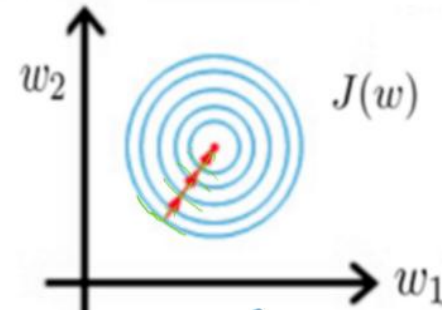
2a: قطر بزرگ
2b: قطر کوچک

Gradient descent
without scaling



Gradient descent
after scaling variables

$$0 \leq x_1 \leq 1$$
$$0 \leq x_2 \leq 1$$



$$\frac{w_1^2}{a^2} + \frac{w_2^2}{a^2} = 1$$

Feature Scaling

Scaled features:

- $0 \leq x_1 \leq 3$ ✓
- $-3 \leq x_1 \leq 3$ ✓
- $-2 \leq x_2 \leq 0.5$ ✓
- $-\frac{1}{3} \leq x_2 \leq \frac{1}{3}$ ✓

Need scaling:

$$-100 \leq x_3 \leq 100 \quad \times$$

$$-0.001 \leq x_4 \leq 0.001 \quad \times$$

Feature Scaling

$$x_1^* = \frac{x_1 - \mu_1}{\text{standard_deviation}}$$

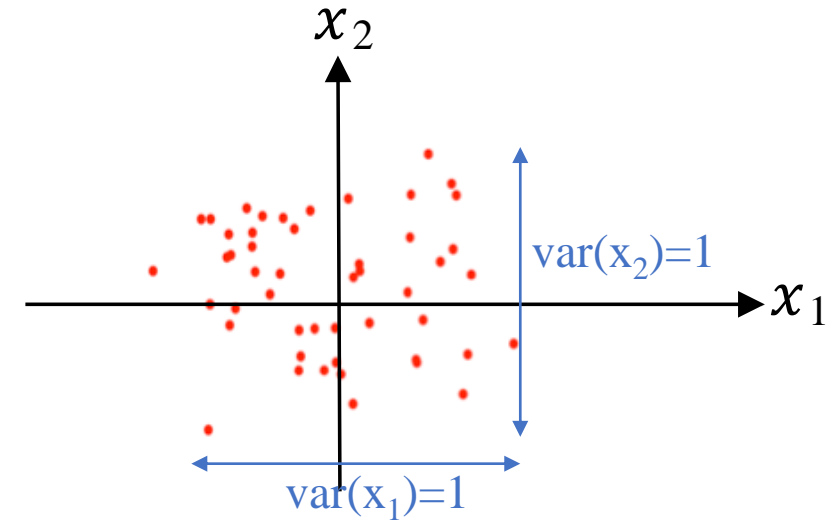
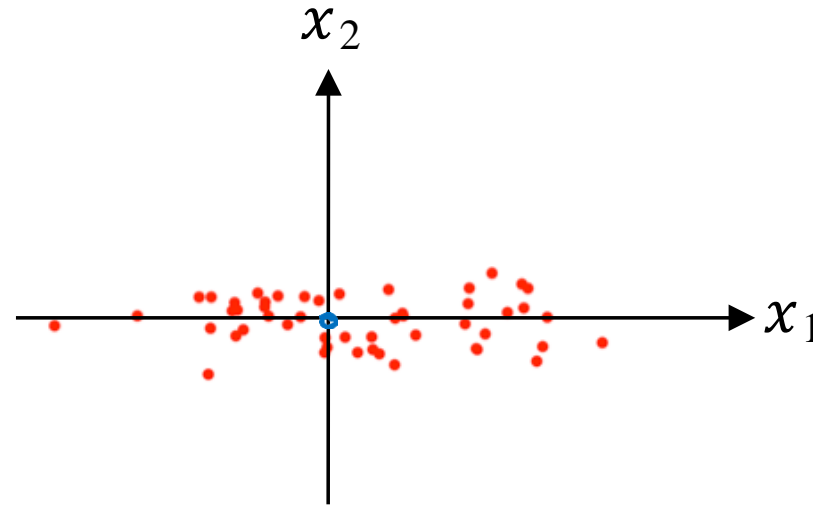
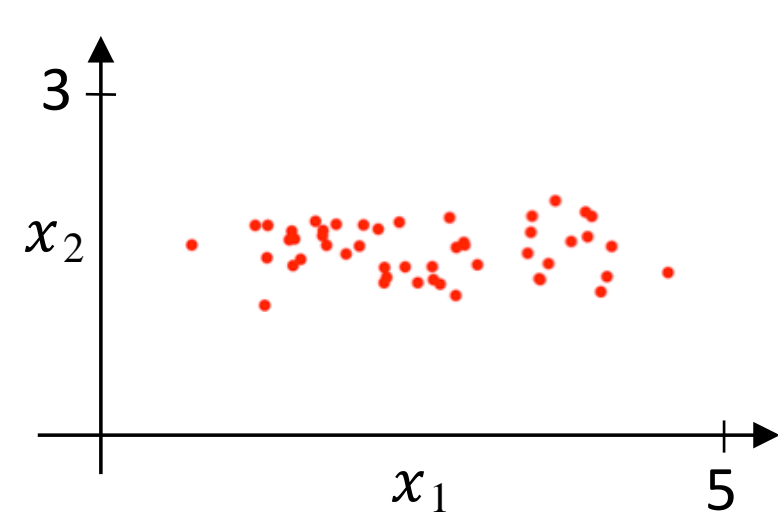
$$x_{\text{scaled}} = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

$$\mu_j = \frac{1}{m} \sum_{i=1}^m x_j^i$$

$$\text{bedroom}^* = \frac{\text{bedroom} - 2.5}{5}$$

$$\text{size}^* = \frac{\text{size} - 300}{2000}$$

Normalizing training set




Subtract mean:

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

Normalize variance:

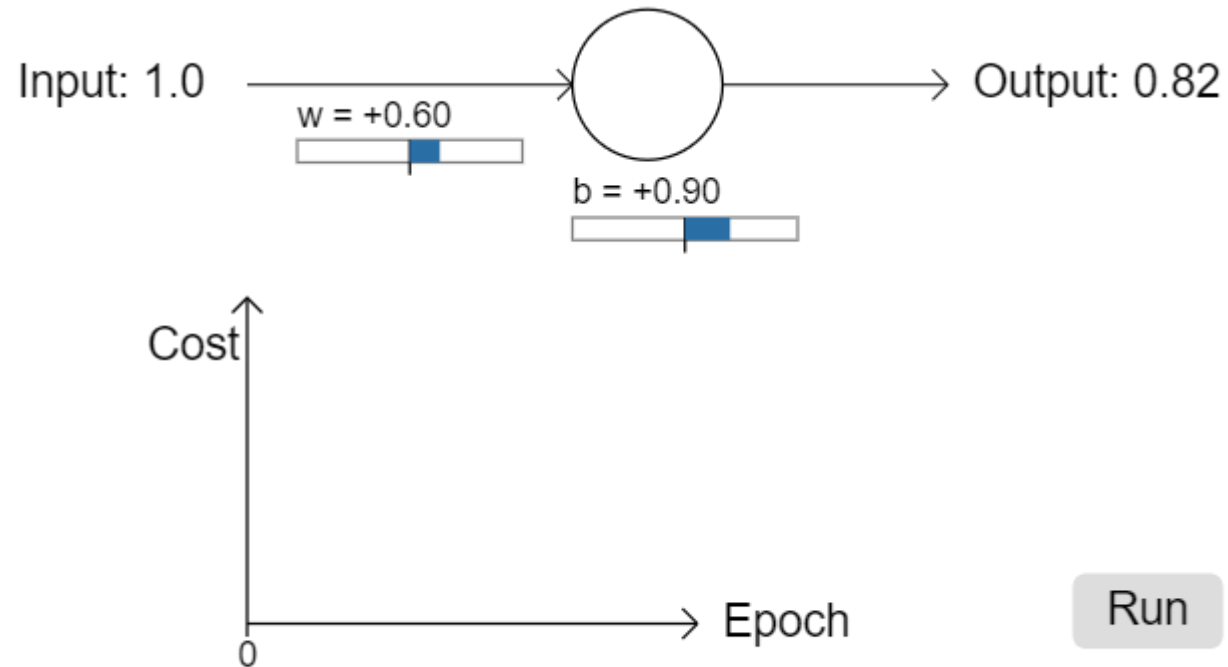
$$\sigma^2 = \frac{1}{m} \sum_{i=1}^m x^{(i)} ** 2$$

$x /= \sigma^2$  Element-wise

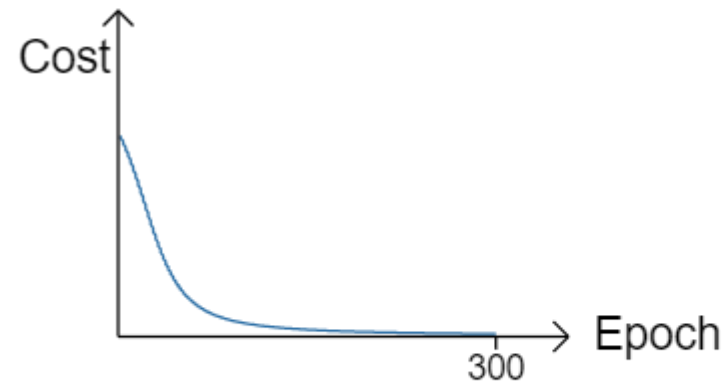
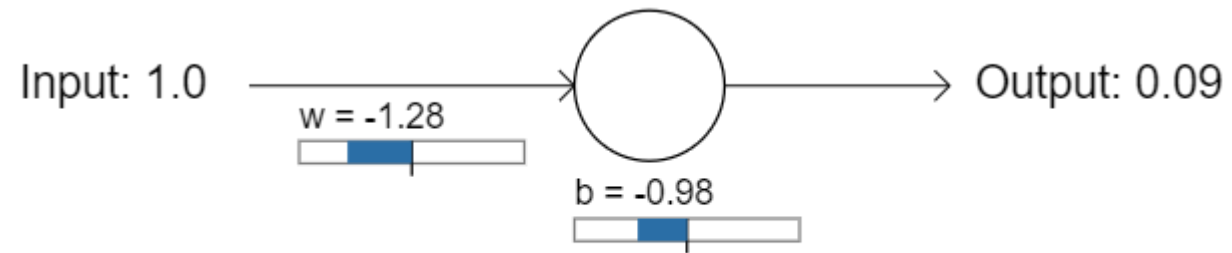
Use same μ, σ^2 to normalize test set.

Vanishing gradient problem

Loss: MSE

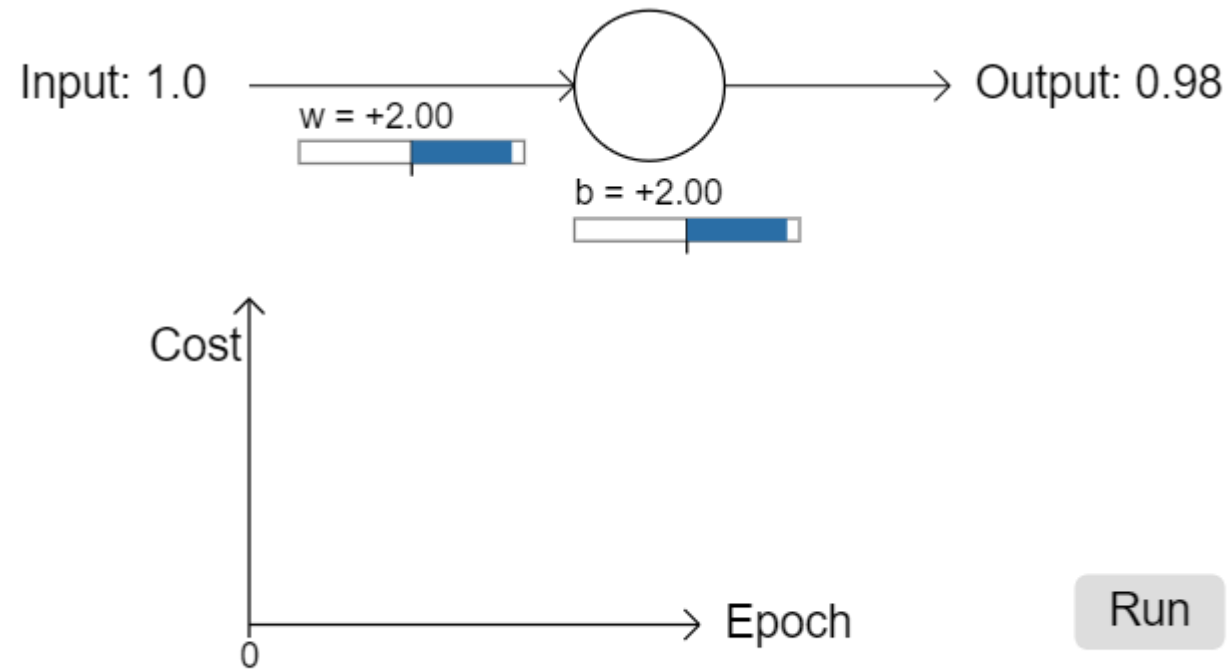


Loss: MSE

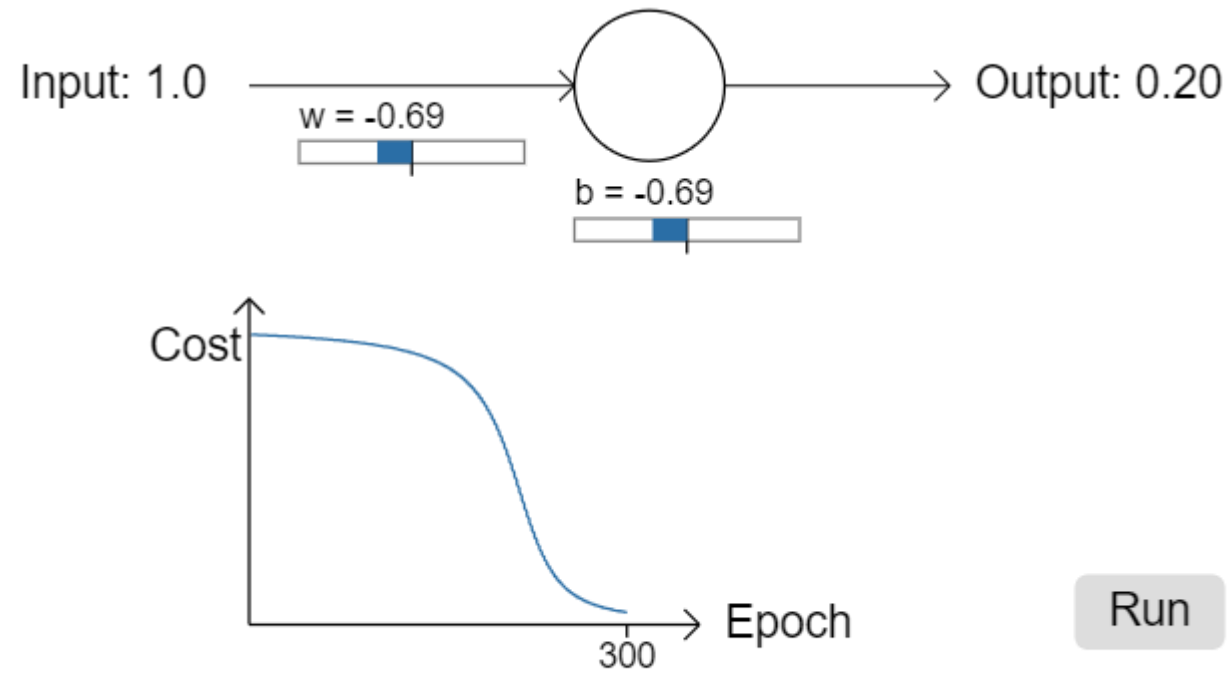


Run

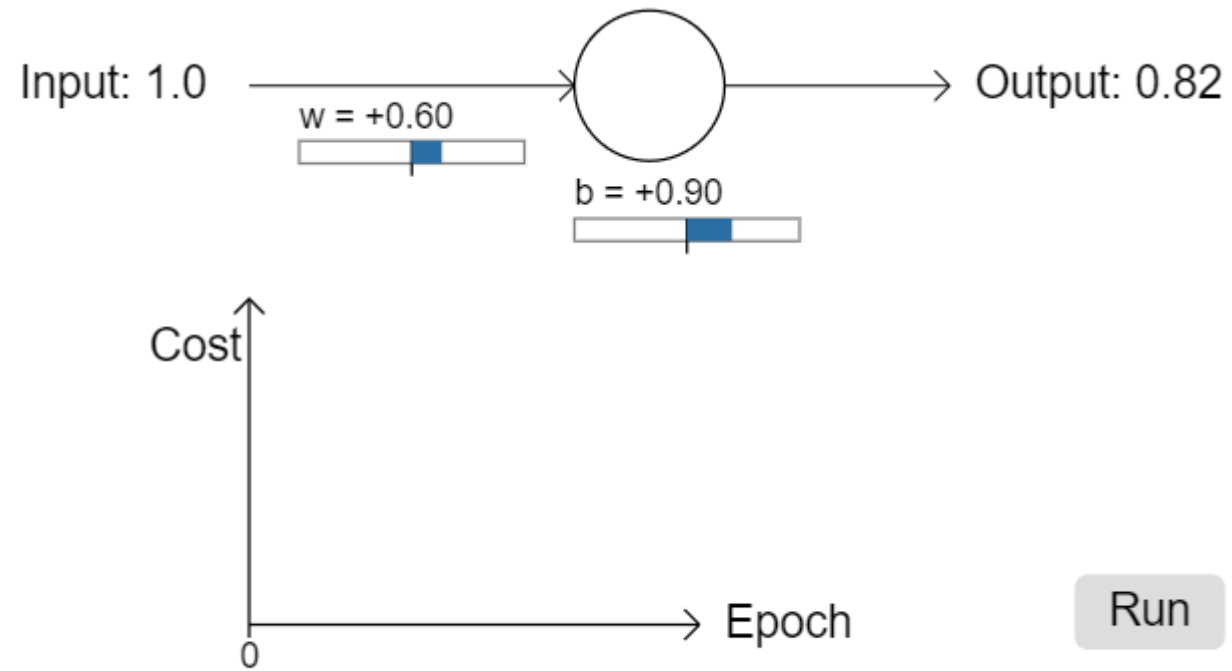
Loss:MSE



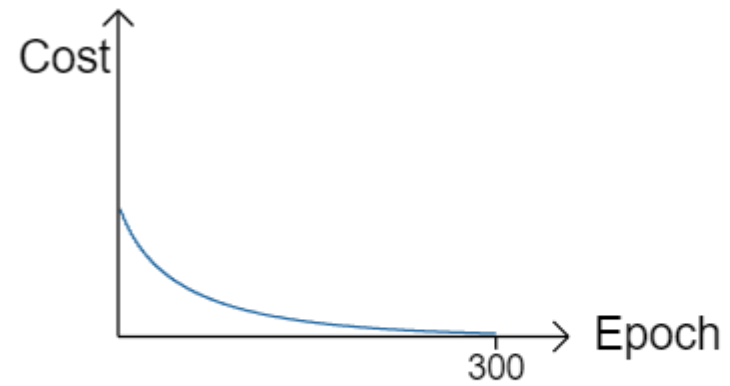
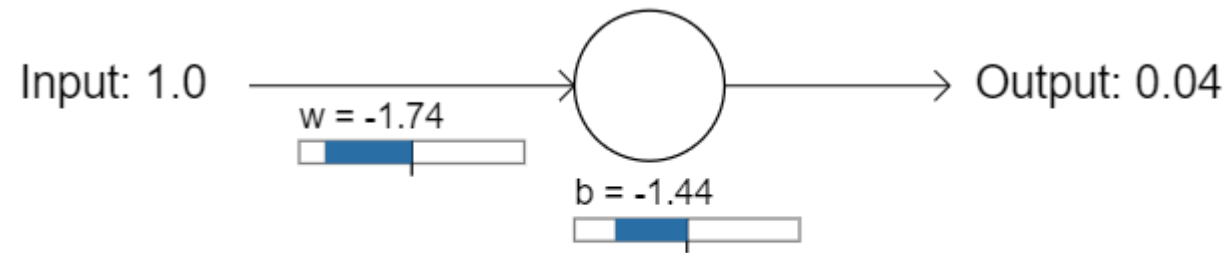
Loss: MSE



Loss:CE

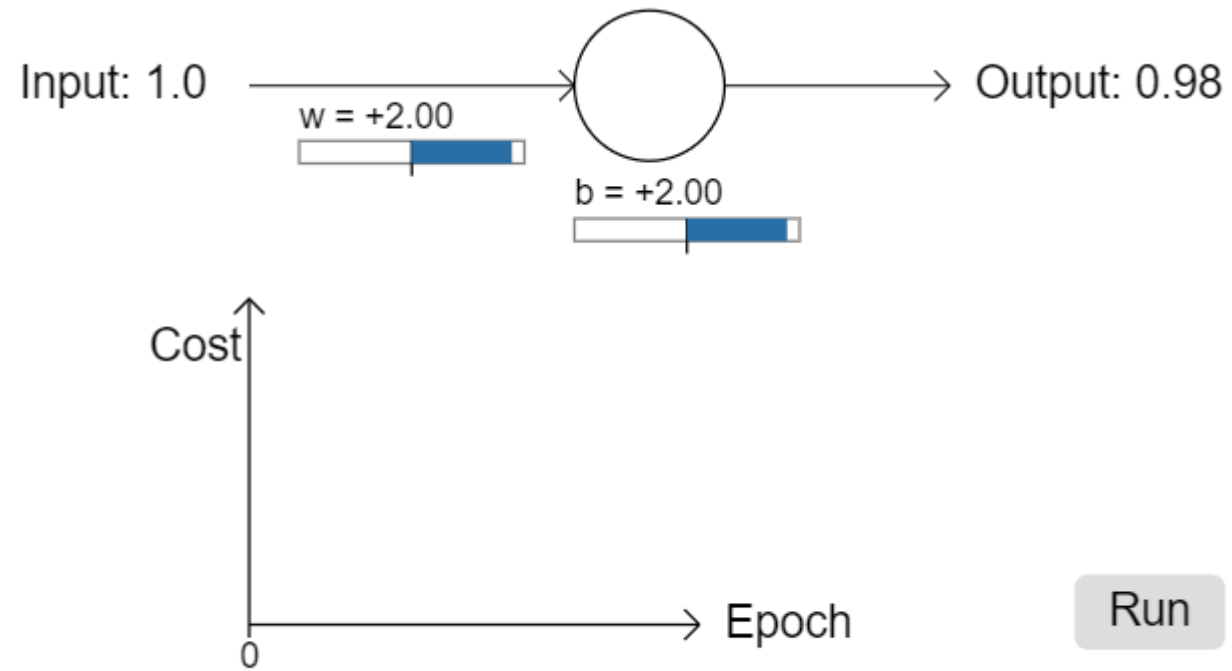


Loss:CE

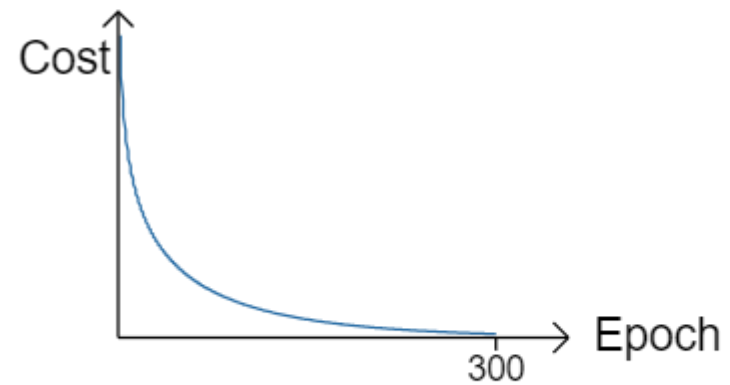
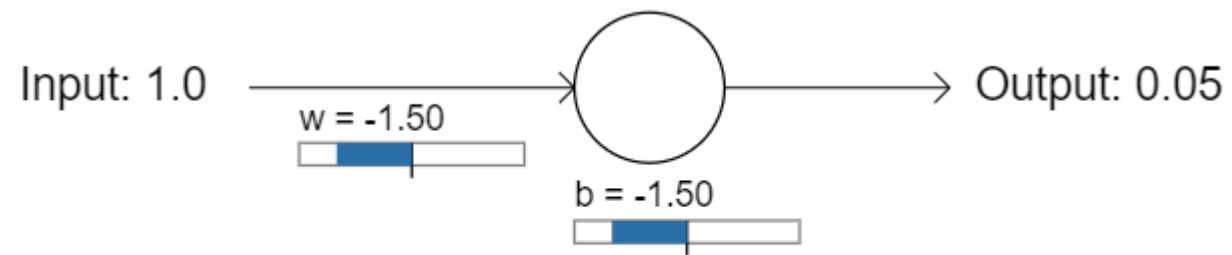


Run

Loss:CE



Loss:CE



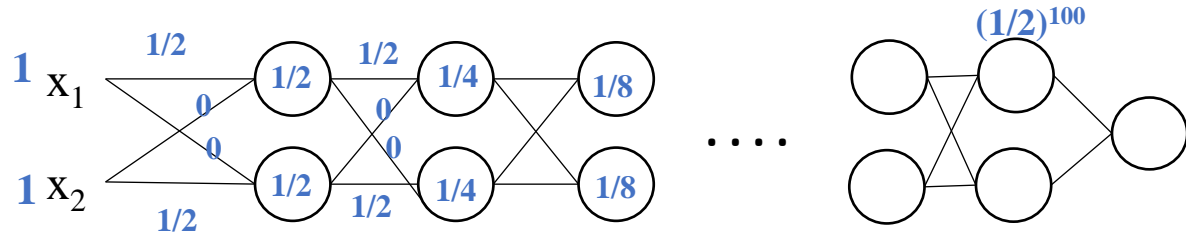
Run

Vanishing/exploding gradients

gradient clipping

- If $\|g\| > \text{threshold}$

$$g \leftarrow \frac{\text{threshold} \times g}{\|g\|}$$



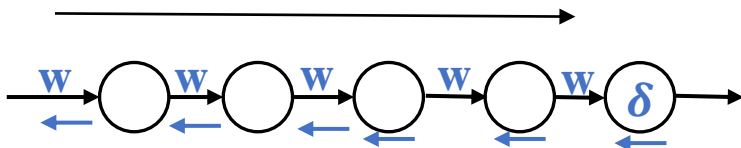
- $g(z) = z$
 $b^{[l]} = 0$

$$\hat{y} = w^{[l]} w^{[l-1]} \dots w^{[3]} w^{[2]} w^{[1]} x$$

$$w = \begin{bmatrix} 0.9 & 0 \\ 0 & 0.9 \end{bmatrix}$$

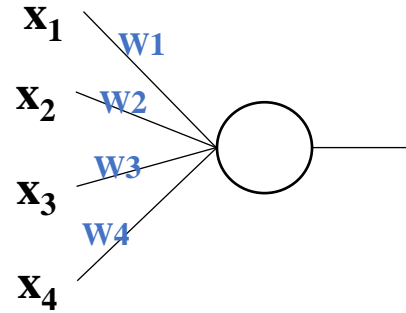
$$\hat{y} = w^{[l]} \begin{bmatrix} 0.9 & 0 \\ 0 & 0.9 \end{bmatrix}^{[l-1]} x$$

$$\begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}^{100} x$$

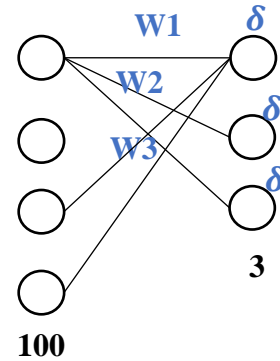


Single neuron example

- $W \approx N(0,1) * 0.01$



- *larger $n \rightarrow$ smaller w_i*



Xavier initialization

- $z = w_1x_1 + w_2x_2 + \dots + w_nx_n$ اثبات کنید.
- $W_i \approx N(0, \sigma^2)$ $z \approx N(0, \underbrace{n\sigma^2}_1)$
- $E(x_i) = 0$ $n\sigma^2 = 1$
- $var(x_i) = 1$ $\sigma^2 = 1/n$
- $w^{[l]} = np \cdot random.randn(shape) * np.sqrt\left(\frac{1}{n^{[l-1]}}\right)$
- $w^{[l]} = np \cdot random.randn(shape) * np.sqrt\left(\frac{2}{n^{[l-1]}}\right)$

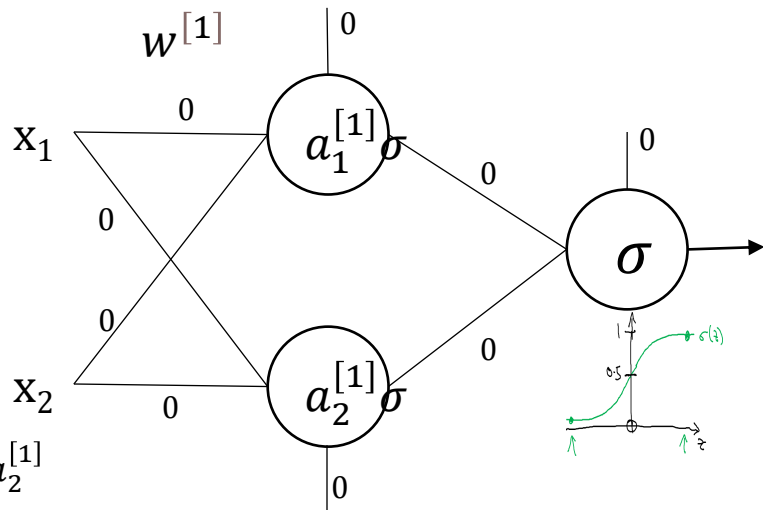
Xavier initialization

- tanh: $\sqrt{\frac{1}{n^{[l-1]}}}$
- Relu: $\sqrt{\frac{2}{n^{[l-1]}}}$
- Relu* = $\sqrt{\frac{2}{n^{[l-1]} + n^{[l]}}} \rightarrow \text{begin}$

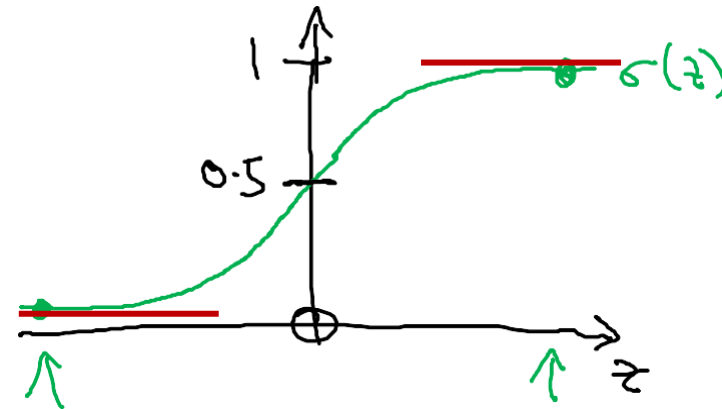
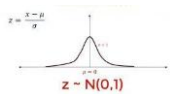
Random Initialization

- **Restricted Boltzmann Machine(RBM):** A **restricted Boltzmann machine** is a generative stochastic artificial neural network that can learn a probability distribution over its set of inputs.

• اگر همه وزن‌های اولیه و بایاس‌ها صفر باشند:



- $a_1^{[1]} = a_2^{[1]}$
- $w^{[1]} = np.random.randn((2,2)) * 0.01$
- $b^{[1]} = np.zeros((2,1))$
- $w^{[2]} = np.random.randn((1,2)) * 0.01$
- $b^{[2]} = 0$



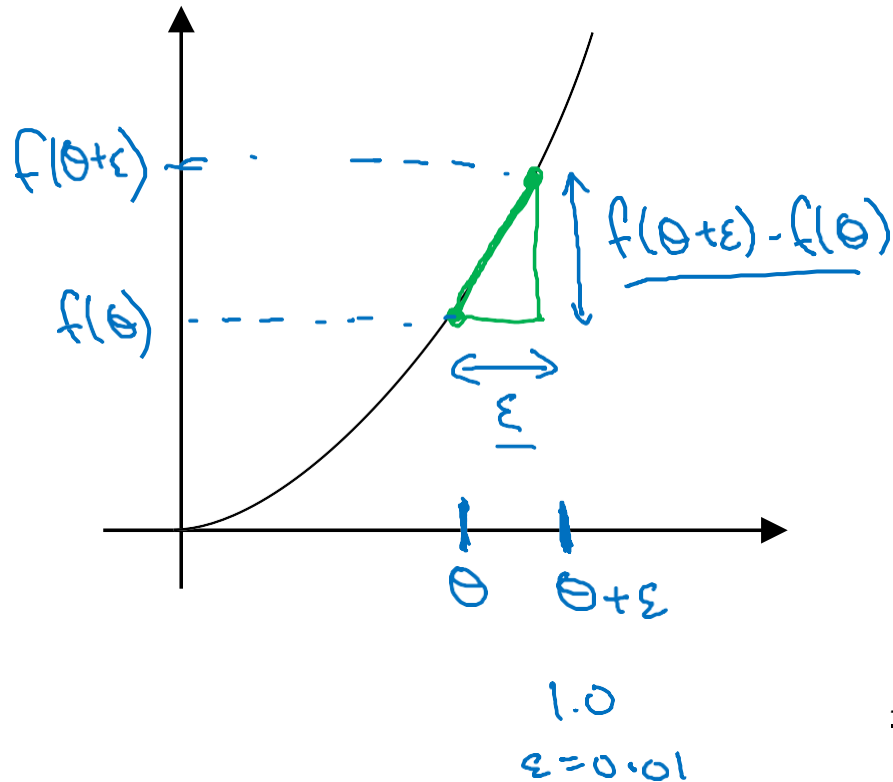
Vanishing
gradient

Numerical approximation of gradients

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

$$f(\theta) = \theta^3$$
$$\theta = 1$$

$$\frac{f(1.01)-f(0.99)}{2 \times 0.01} = \frac{(1.01)^3 - (0.99)^3}{0.02} = 3.0001$$



$$\theta = 1$$

$$\theta + \varepsilon = 1.01$$

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

Gradient Checking

- *for each i*

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

$$\begin{bmatrix} \times \\ \times \\ \times \\ \times \end{bmatrix}$$

- $d\theta_{approx}^{[i]} = \frac{dJ}{d\theta_i} \approx d\theta_i$

$$\frac{\|d\theta_{approx} - d\theta\|_2}{\|d\theta_{approx}\|_2 + \|d\theta\|_2} \ll 10^{-7} \qquad 10^{-3}$$

Mini-batch gradient descent

Batch, mini-batch

- $X = [x^{(1)}, x^{(2)}, \dots, x^{(1000)} | x^{(1001)} \dots x^{(2000)} | \dots x^{(m)}]$

$$X_{(n_x, 1000)}^{\{1\}}$$

$$X_{(n_x, 1000)}^{\{2\}}$$

$$X_{(n_x, m)}^{\{5000\}} \quad m=5000,000$$

$$Y = \left[\underbrace{y^{(1)}, y^{(2)} \dots, y^{(1000)}}_{Y^{\{1\}}} | \underbrace{y^{(1001)} \dots y^{(2000)}}_{Y^{\{2\}}} | \dots | \underbrace{\dots, y^{(m)}}_{Y^{\{5000\}}} \right]$$

X^t, Y^t

Mini-batch GD

- Repeat{

For t=1, ..., 5000 {

forward prop. on $x^{\{t\}}$

$$z^{[1]} = w^{[1]}x^{\{t\}} + b^{[1]}$$

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

.

.

.

$$A^{[L]} = g^{[L]}(z^{[L]})$$

compute cost $J^{\{t\}} = \frac{1}{1000} \sum_{i=1}^{1000} L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2 \times 1000} \sum_l \|w^{[l]}\|_F^2$

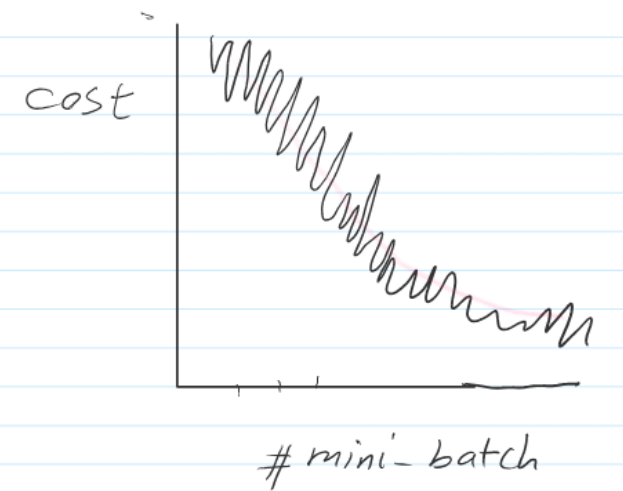
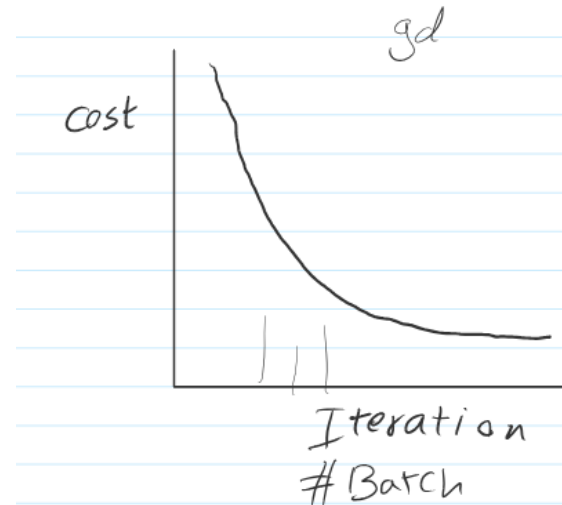
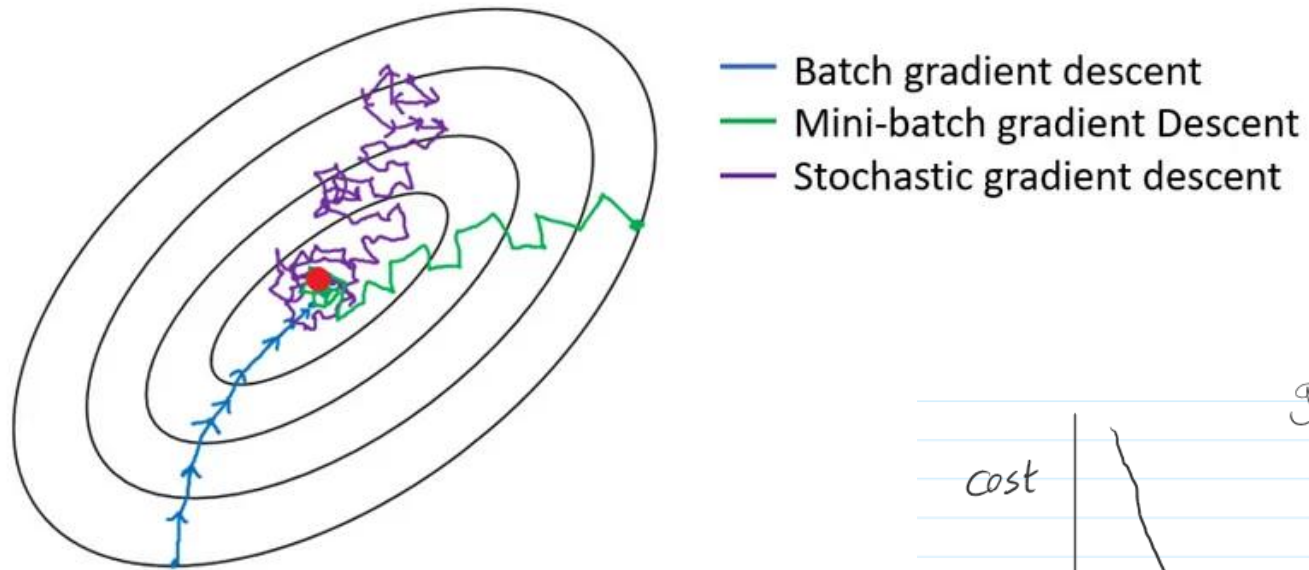
back propagation: $dw^{[l]}, db^{[l]}$

$$w^{[l]} = w^{[l]} - \alpha dw^{[l]}, b^{[l]} = b^{[l]} - \alpha db^{[l]}$$

}

} Until convergence

Mini-batch GD



Mini-batch GD

- min-batch size=1 stochastic gradient descent (SGD)

$$\{x^{(1)}, x^{(2)}, \dots, x^{(1000)}\}$$

$$\frac{1}{100} \sum_{i=1}^{100} L(y^{(i)}, \hat{y}^{(i)}) = \frac{1}{1000} \times 1000 L(y^{(1)}, \hat{y}^{(1)})$$

$$x^{(1)} = x^{(2)} = \dots = x^{(1000)}$$

$$\begin{matrix} x^{(1)} \\ x^{(2)} \\ \vdots \\ x^{(1000)} \end{matrix} \left. \vphantom{\begin{matrix} x^{(1)} \\ x^{(2)} \\ \vdots \\ x^{(1000)} \end{matrix}} \right\} w^{(1)} \text{ updated}$$

$$w^{(1)} = w^{*(1)}$$

SGD:

$$\begin{aligned} x^{(1)} &\rightarrow w^{*(1)} \text{ updated} \\ x^{(2)} &\rightarrow w^{*(2)} \text{ updated} \\ x^{(3)} &\rightarrow w^{*(3)} \text{ updated} \end{aligned}$$

\vdots

$$x^{(1000)} \rightarrow w^{*(1000)} \text{ updated}$$

1000 epoch

$$\begin{matrix} x^{(1)} \\ x^{(2)} \\ \vdots \\ x^{(1000)} \end{matrix} \left. \vphantom{\begin{matrix} x^{(1)} \\ x^{(2)} \\ \vdots \\ x^{(1000)} \end{matrix}} \right\} w^{(2)} \text{ updated}$$

\equiv

$$\begin{matrix} x^{(1)} \\ x^{(2)} \\ \vdots \\ x^{(1000)} \end{matrix} \left. \vphantom{\begin{matrix} x^{(1)} \\ x^{(2)} \\ \vdots \\ x^{(1000)} \end{matrix}} \right\} w^{(1000)} \text{ updated}$$

Mini-batch GD

SGD

Noisy

Lack of vectorization

mini-batch GD

Fastest learning

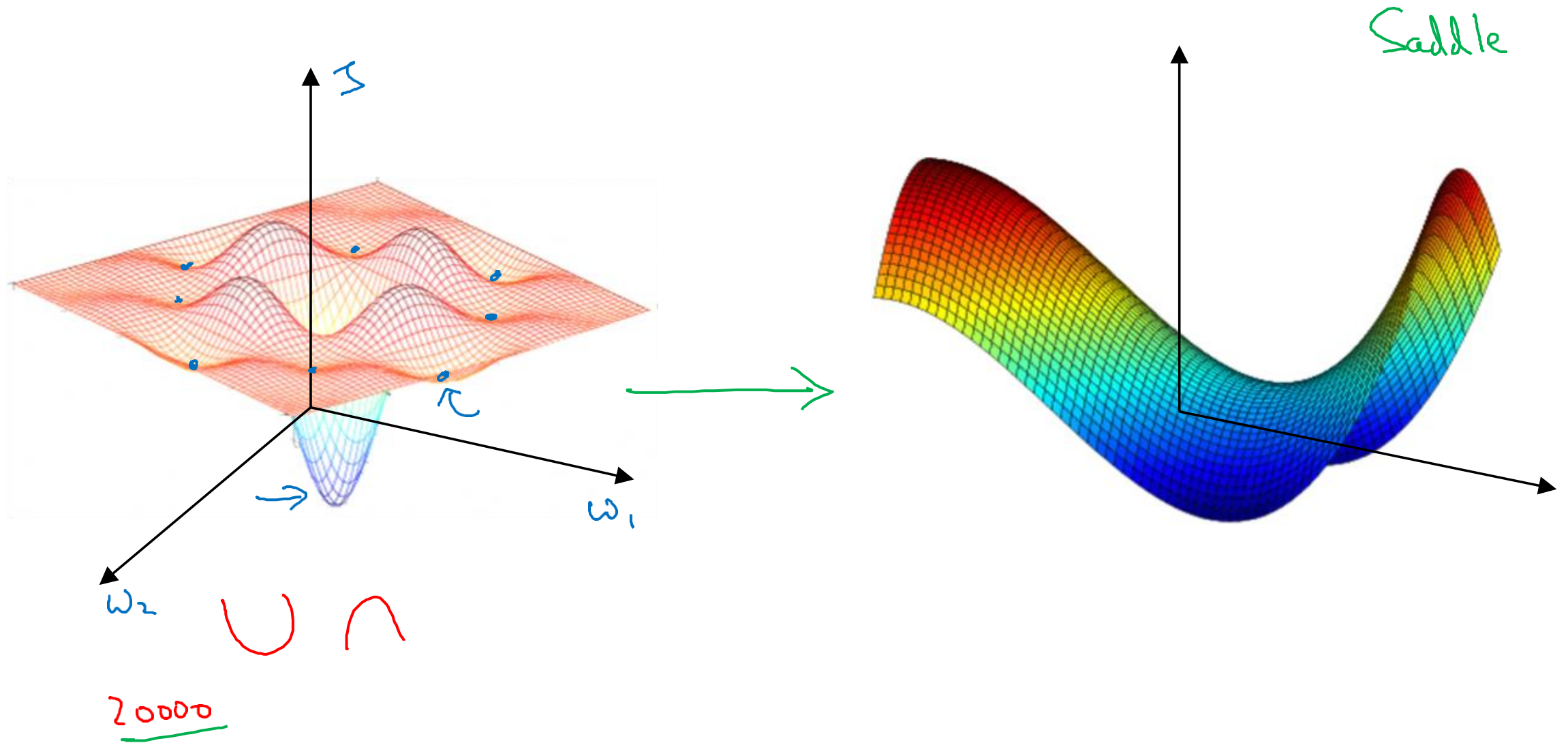
Batch GD

too long per iteration

$m \leq 2000$ Full batch

#mini batch sizes : 64, 128, 256, 512

Local optima in neural networks

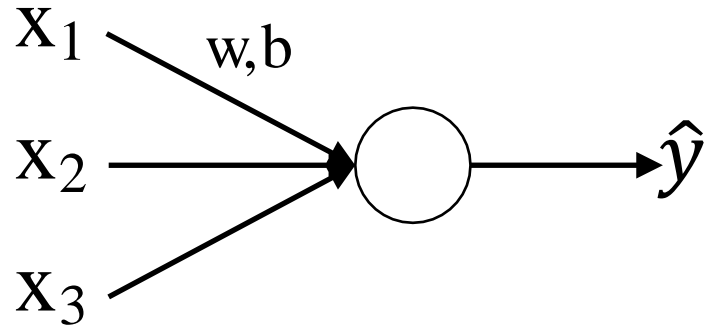


Batch Normalization

Normalizing activations
in a network

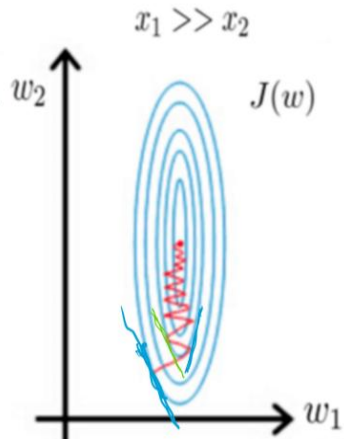
Batch Normalization

Normalizing inputs to speed up learning

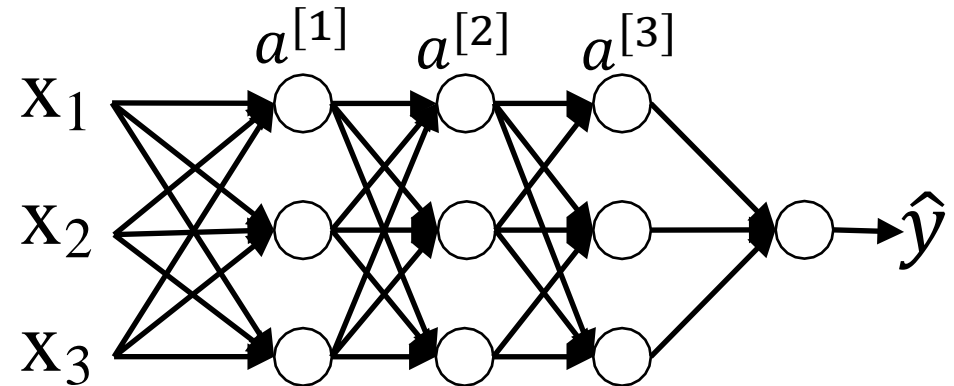
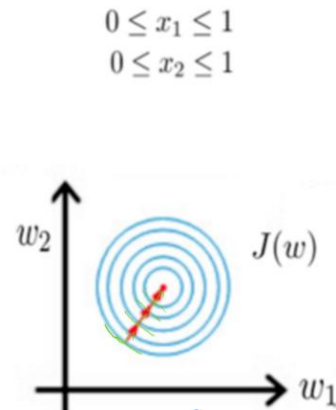


- $\mu = \frac{1}{m} \sum_{i=1}^m x^i$
- $X = X - \mu$
- $\sigma^2 = \frac{1}{m} \sum_{i=1}^m (x^{(i)})^2$
- $X = X / \sigma^2$

Gradient descent
without scaling



Gradient descent
after scaling variables



Batch Normalization

In an intermediate layer: Implementing Batch Norm

- $\mu = \frac{1}{m} \sum_{i=1}^m z^{(i)}$
- $\sigma^2 = \frac{1}{m} \sum_{i=1}^m (z^{(i)} - \mu)^2$
- $Z_{norm}^{(i)} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \epsilon}}$

$$\gamma = \sqrt{\sigma^2 + \epsilon}$$

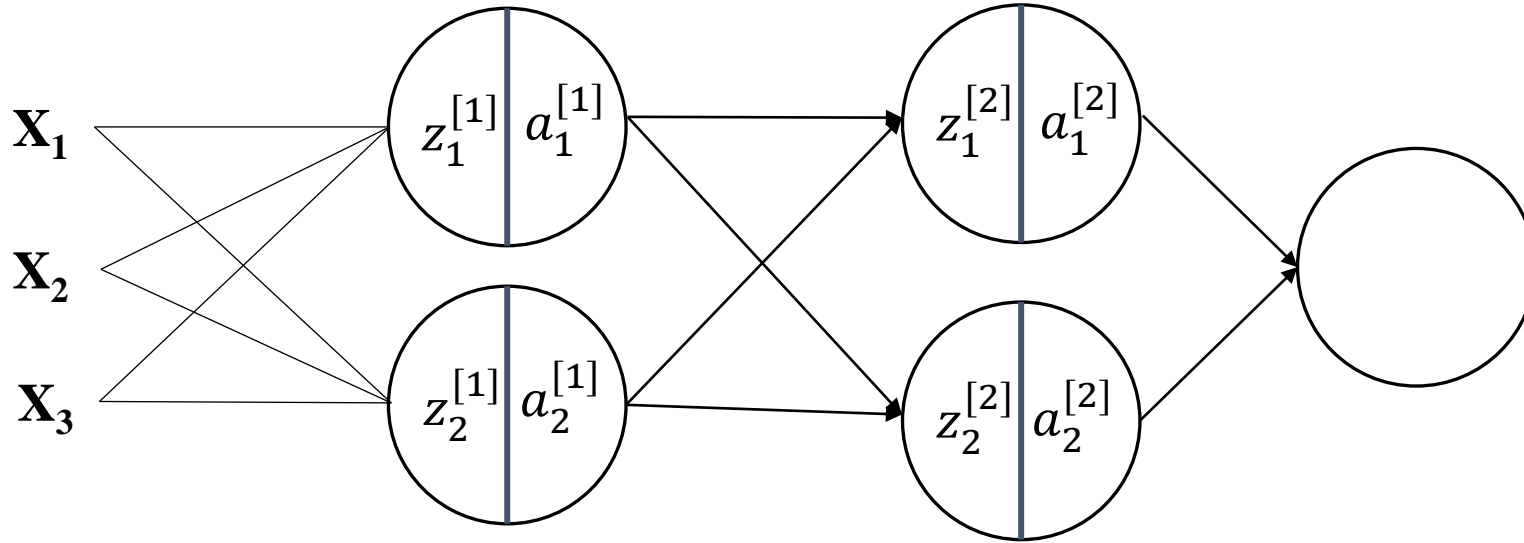
$$\beta = \mu$$

$$\tilde{z}^{(i)} = \gamma Z_{norm}^{(i)} + \beta$$

Learnable Parameters

Batch Normalization

Fitting Batch Norm into a neural network



$$\bullet \quad X \xrightarrow{w^{[1]}, b^{[1]}} Z^{[1]} \xrightarrow[\text{Batch Norm (BN)}]{\beta^{[1]}, \gamma^{[1]}} \tilde{Z}^{[1]} \longrightarrow a^{[1]} = g^{[1]}(\tilde{Z}^{[1]}) \xrightarrow{w^{[2]}, b^{[2]}} Z^{[2]} \xrightarrow[\text{BN}]{\beta^{[2]}, \gamma^{[2]}}$$

$$\tilde{Z}^{[2]} \longrightarrow a^{[2]}$$

$$w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]}, \dots, w^{[L]}, b^{[L]}$$

$$\beta^{[1]}, \gamma^{[1]}, \beta^{[2]}, \gamma^{[2]}, \dots, \beta^{[L]}, \gamma^{[L]}$$

$$d\beta^{[l]} \quad \beta^{[l]} = \beta^{[l]} - \alpha d\beta^{[l]}$$

Batch Normalization

Working with mini-batches

- $z^{[l]} = w^{[l]}a^{[l-1]} + \cancel{b^{[l]}}$

$$z^{[l]} = w^{[l]}a^{[l-1]}$$

$$z_{norm}^{[l]}$$

$$\tilde{z}^{[l]} = \gamma^{[l]}z_{norm}^{[l]} + \beta^{[l]}$$

$$w^{[l]}, \underbrace{\beta^{[l]}}_{(n^{[l]} \times 1)}, \underbrace{\gamma^{[l]}}_{(n^{[l]} \times 1)}$$

Implementing gradient descent

- for $t=1 \dots \text{\#num MiniBatches}$

forward Propagation on $X^{\{t\}}$

use BN in each layer to compute $\tilde{z}^{[l]}$

use Backprop to compute $dw^{[l]}, d\beta^{[l]}, d\gamma^{[l]}$

update parameters:

$$w^{[l]} = w^{[l]} - \alpha dw^{[l]}$$

$$\beta^{[l]} = \beta^{[l]} - \alpha d\beta^{[l]}$$

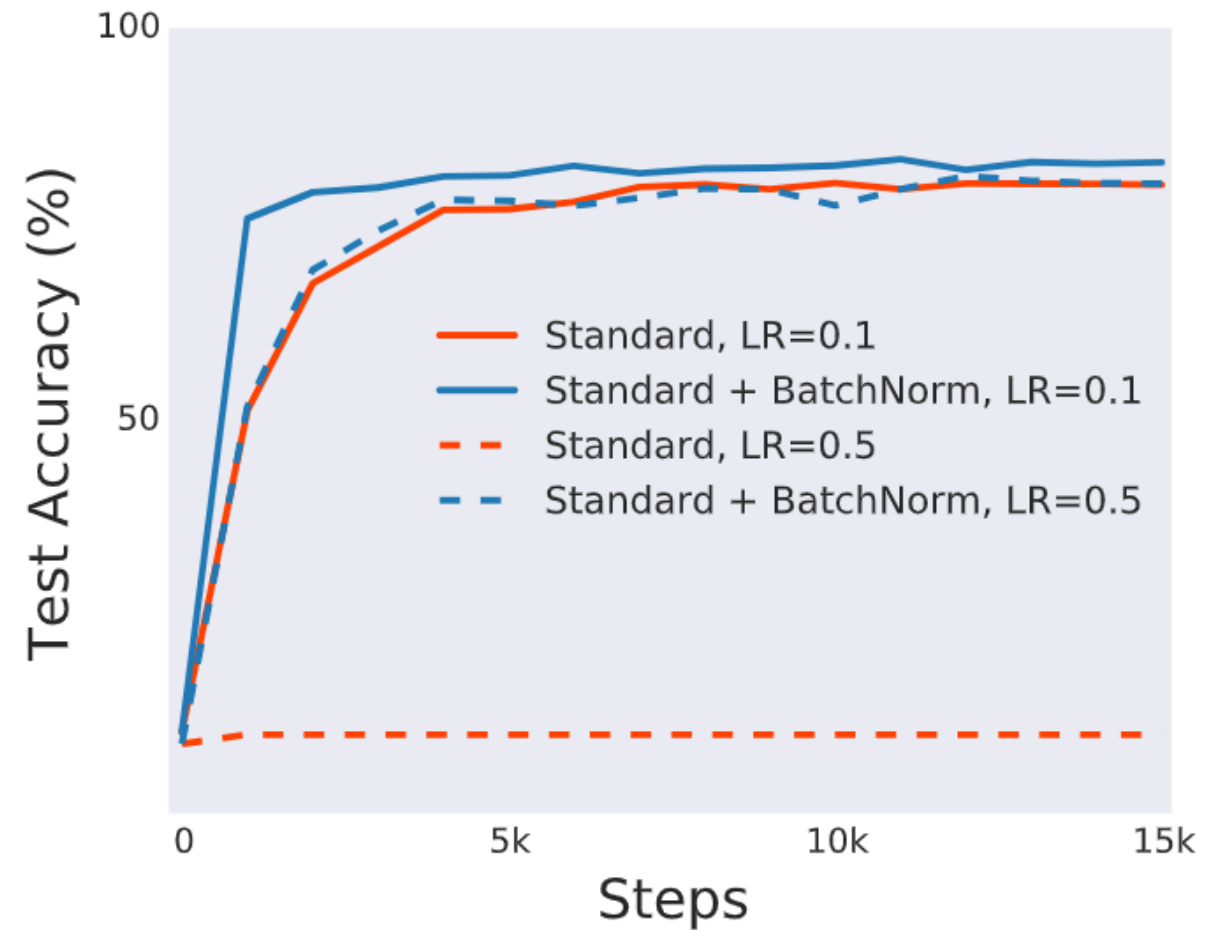
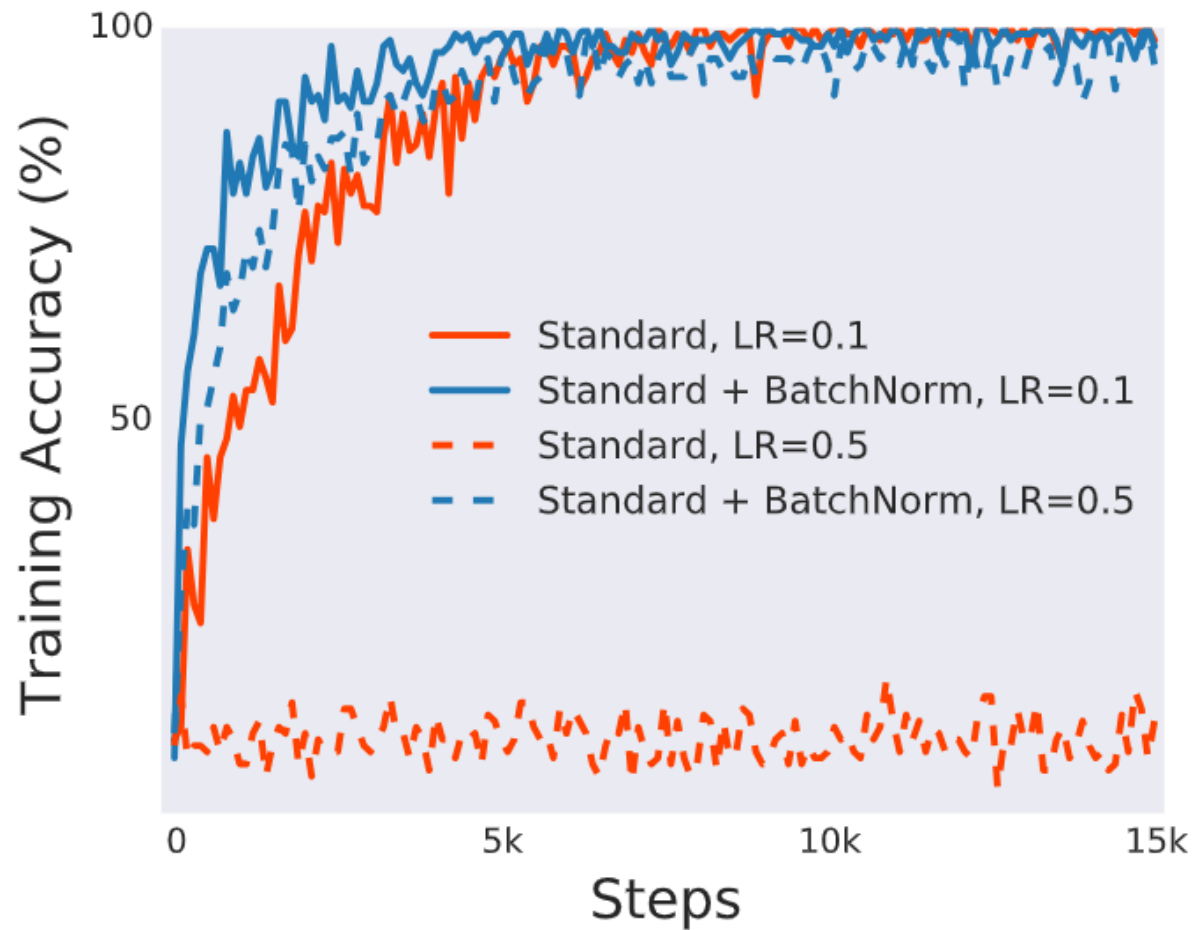
$$\gamma^{[l]} = \gamma^{[l]} - \alpha d\gamma^{[l]}$$

.

.

.

Batch normalization



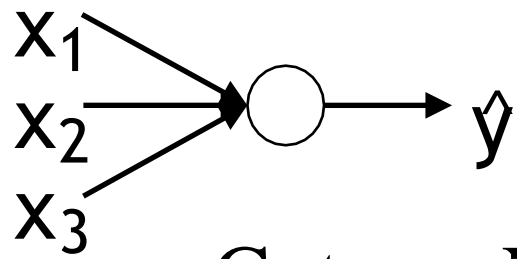
VGG network on CIFAR

<https://arxiv.org/pdf/1805.11604>, How Does Batch Normalization Help Optimization?

How Does Batch Normalization Help Optimization?

- prevention of exploding or vanishing gradients
- robustness to different settings of hyperparameters such as learning rate and initialization scheme
- keeping most of the activations away from saturation regions of non-linearities
- Smoothing effect of batch norm; the loss changes at a smaller rate and the magnitudes of the gradients are smaller too.
- the key implication of Batch Norm's reparametrization is that it makes the gradients more reliable and predictive.

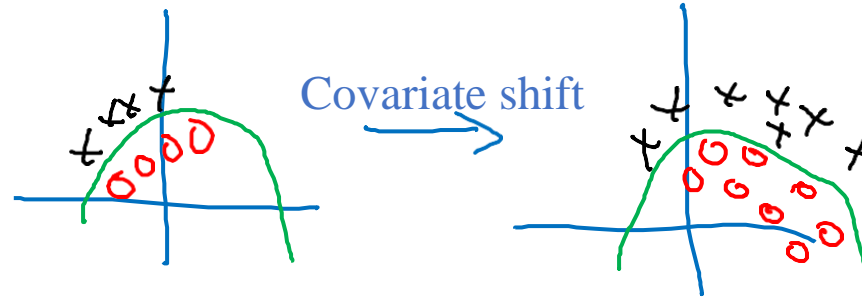
Learning on shifting input distribution



Cat
 $y = 1$



Non-Cat
 $y = 0$



$y = 1$

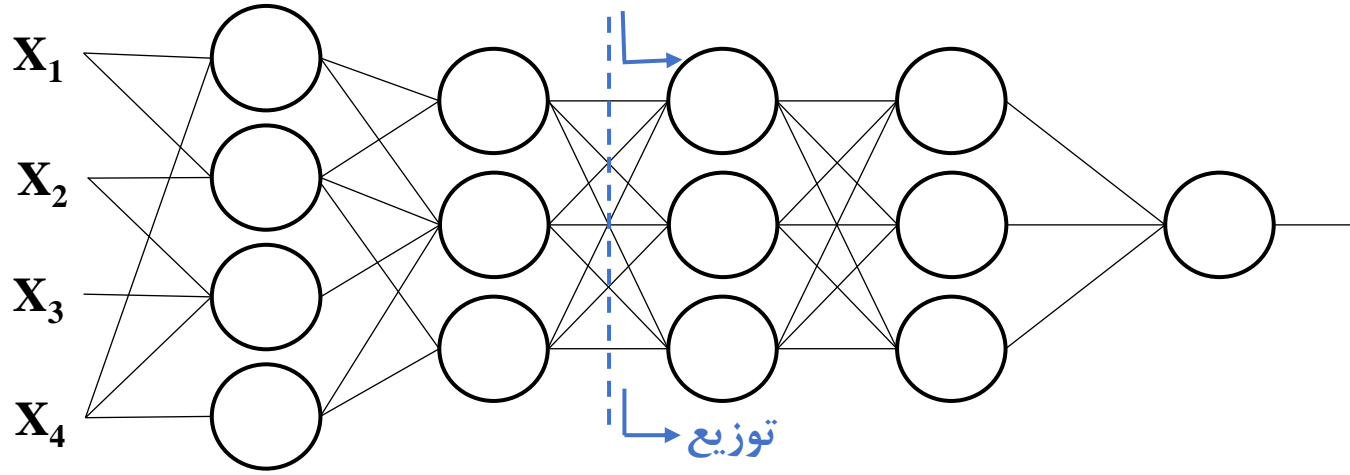


$y = 0$



Covariate shift->

Why this is a problem with neural networks?



Batch Norm as regularization

- Each mini-batch is scaled by the mean/variance computed on just that mini-batch.
- This adds some noise to the values $z^{[l]}$ within that minibatch. So similar to dropout, it adds some noise to each hidden layer's activations.
- This has a slight regularization effect.

$$\tilde{z} = \frac{z - \mu}{\sigma}$$

$$\bar{z} \times \varepsilon \quad \text{slight regularization}$$

Batch Norm at test time

- $\mu = \frac{1}{m} \sum_i z^i$
- $\sigma^2 = \frac{1}{m} \sum_i (z^i - \mu)^2$
- $z_{norm}^i = \frac{z^i - \mu}{\sqrt{\sigma^2 + \epsilon}}$
- $\tilde{z}^i = \underbrace{\gamma}_{scale} z_n^i + \underbrace{\beta}_{shift}$
- $\mu \xrightarrow{\text{در زمان آموزش}}$
 $\sigma^2 \xrightarrow{\hspace{1.5cm}}$

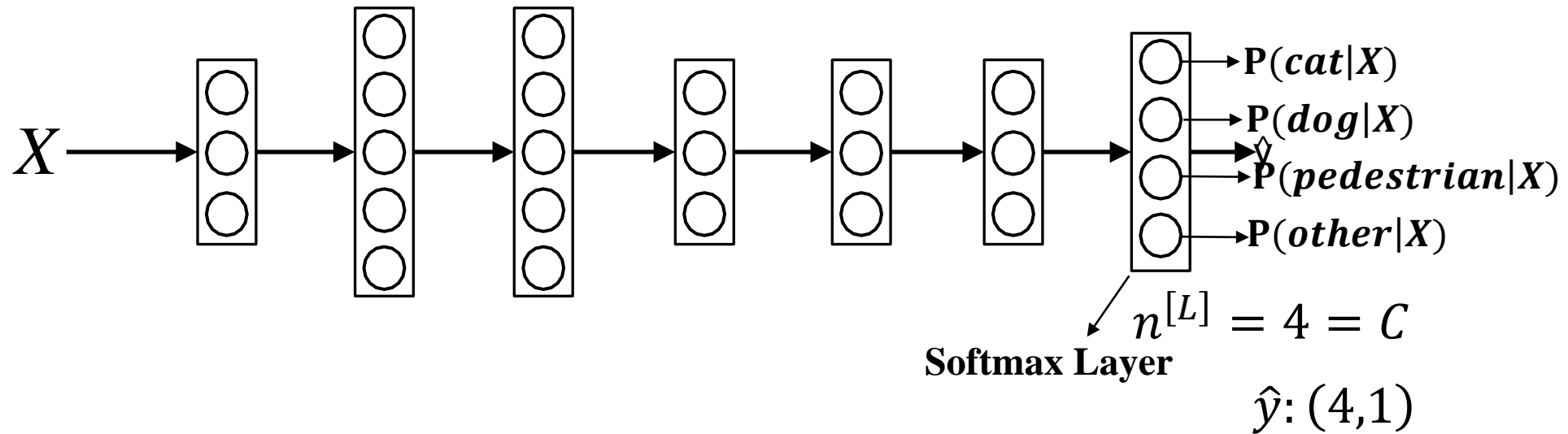
$\mu_1^{[l]}, \mu_2^{[l]}, \mu_3^{[l]} \dots \rightarrow \text{moving Average}$

$$Z_{norm}^{[l]} = \frac{Z^{[l]} - \mu^{[l]}}{\sqrt{\sigma^{[l]2} + \epsilon}}$$

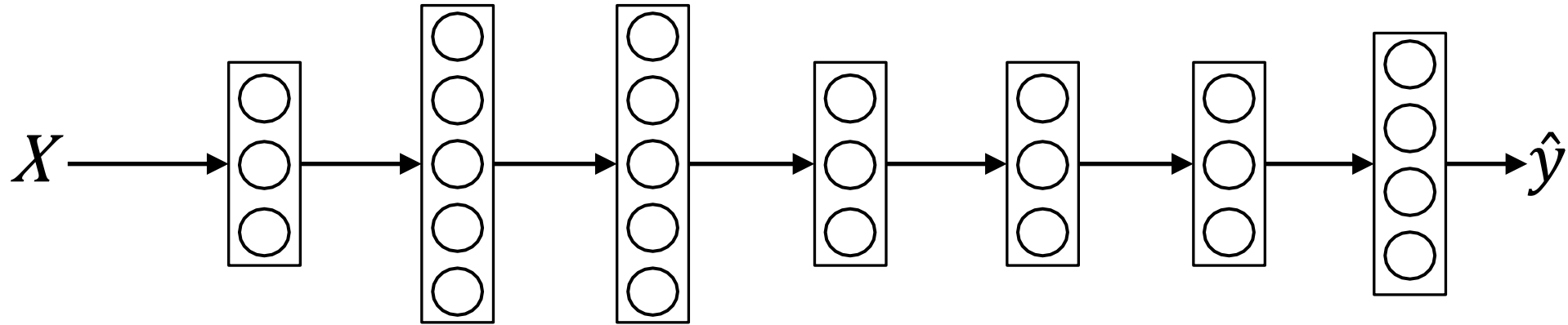
Multi-class classification

Softmax regression

- $C = \text{\#classes} = 4$



Softmax layer



- $z^{[L]} = w^{[L]}a^{[L-1]} + b^{[L]}$
- Softmax Activation function

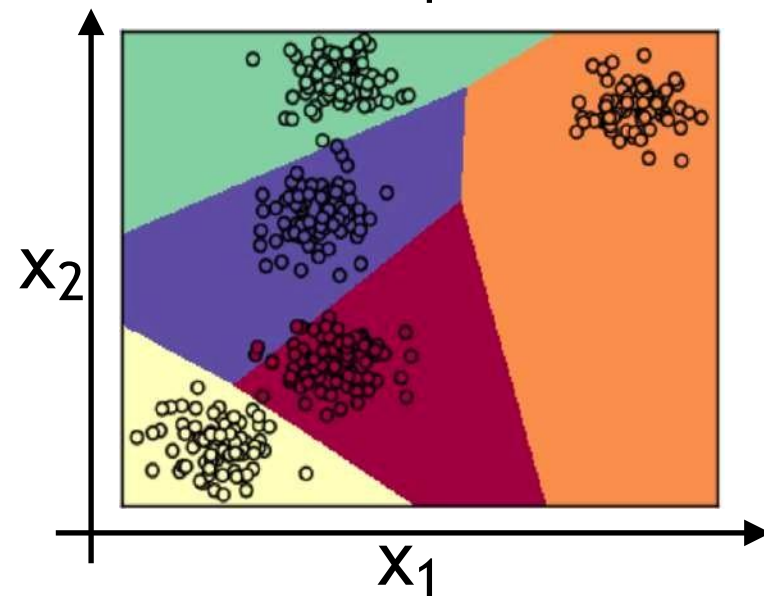
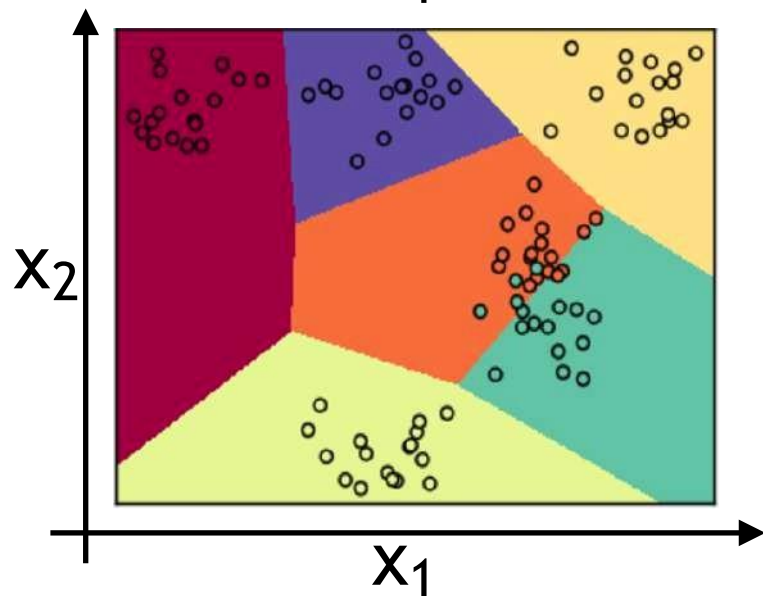
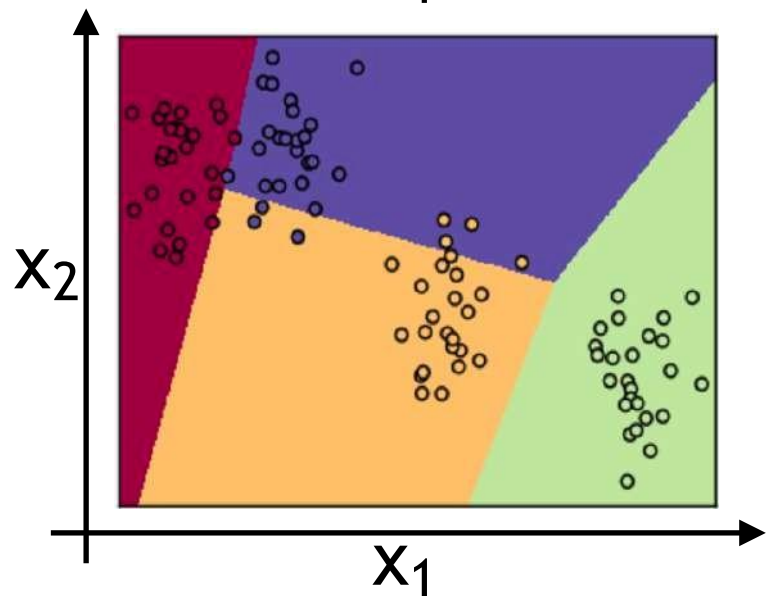
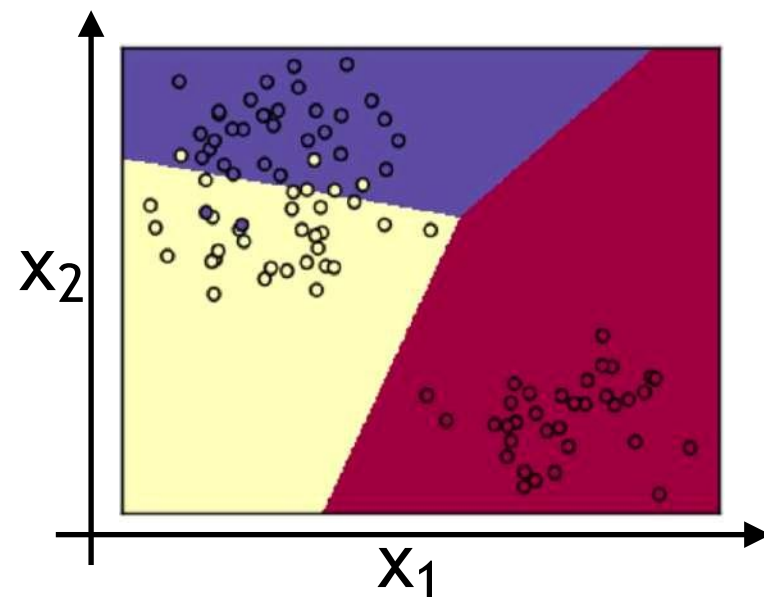
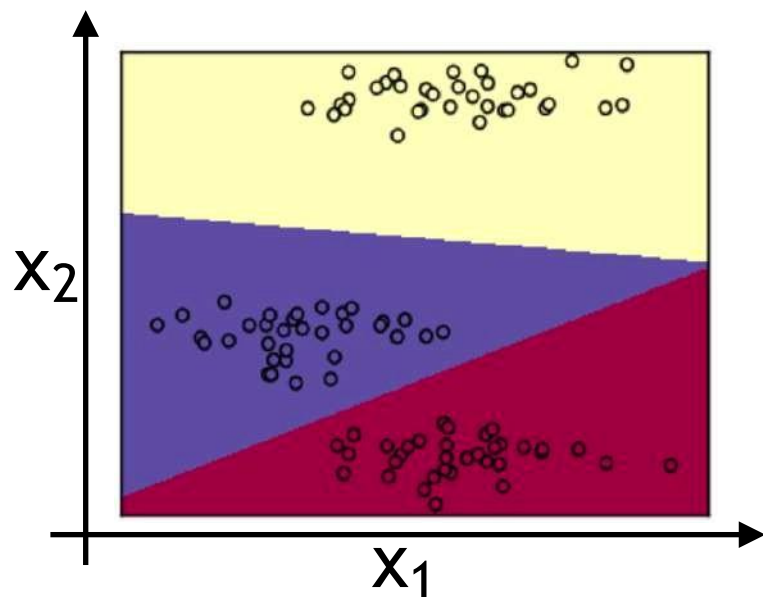
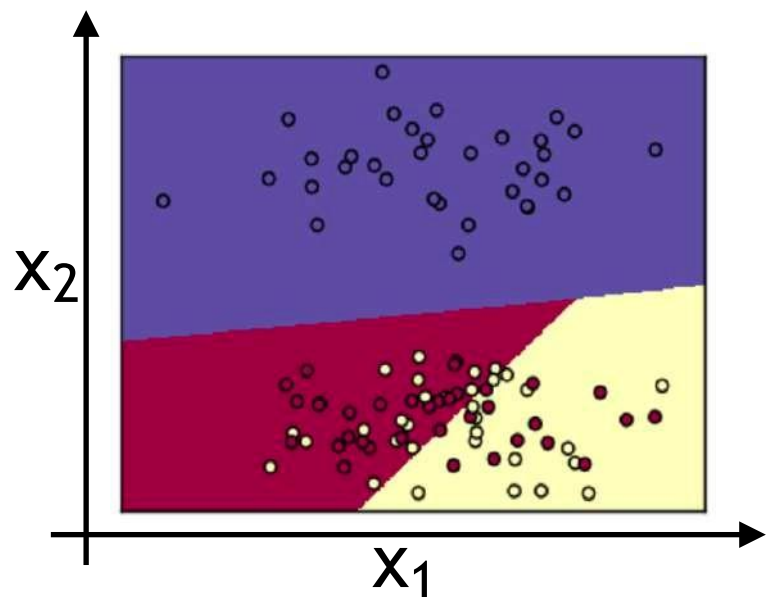
$$t = e^{(z^{[L]})} \quad (4,1)$$

$$(4,1) \quad a^{[L]} = \frac{e^{z^{[L]}}}{\sum_{j=1}^4 t_j} \Rightarrow a_i^{[L]} = \frac{t_i}{\sum_{j=1}^4 t_j}$$

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

Softmax Activation

Softmax examples



Softmax examples

$$\bullet z^{[L]} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix} \quad t = \begin{bmatrix} e^5 \\ e^2 \\ e^{-1} \\ e^3 \end{bmatrix} = \begin{bmatrix} 148.4 \\ 7.4 \\ 0.4 \\ 20.1 \end{bmatrix}$$

$$\sum_{j=1}^4 t_j = 176.3$$

$$a^{[L]} = \frac{t}{176.3}$$

$$\frac{1}{176.3} \begin{bmatrix} 148.4 \\ 7.4 \\ 0.4 \\ 20.1 \end{bmatrix} = \begin{bmatrix} 0.842 \\ 0.042 \\ 0.002 \\ 0.114 \end{bmatrix} = \hat{y}^{(i)}$$

$$y_{(4,1)}^{(i)} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

one hot vector

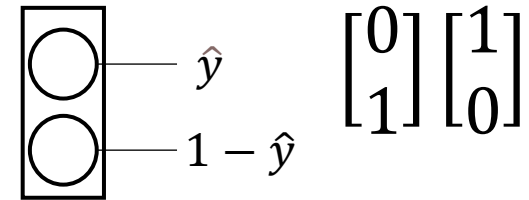
Loss function

- $-[y \log y + (1 - y) \log(1 - y)]$

- $L(\hat{y}, y) = -\sum_{j=1}^4 y_j \log \hat{y}_j$

$$-y_2 \log \hat{y}_2 = -\log \hat{y}_2$$

$$\bigcirc \quad \hat{y}_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

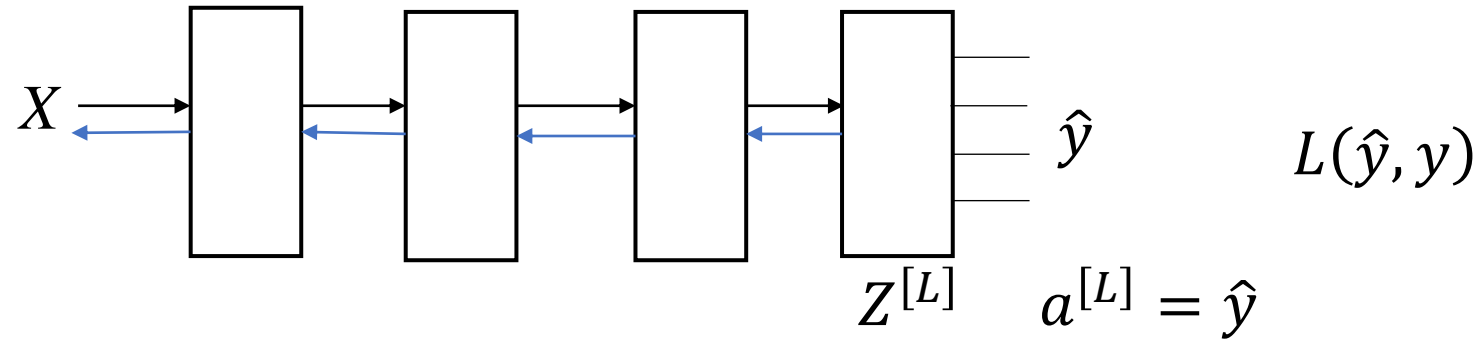


$$Y = [y^{(1)}, y^{(2)}, \dots, y^{(m)}] \quad \hat{Y} = [\hat{y}^{(1)}, \hat{y}^{(2)}, \dots, \hat{y}^{(m)}]$$

$$\begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \dots$$

$$\begin{bmatrix} 0.3 \\ 0.2 \\ 0.3 \\ 0.2 \end{bmatrix} \dots$$

Gradient descent with softmax



• اثبات کنید:

$$dz_{4 \times 1}^{[L]} = \hat{y}_{4 \times 1} - y_{4 \times 1}$$