Deep Learning 2 (Hyperparameters) week1 16 Feb Sal

Bias and Variance

Train set error:	1%	15%	15%	0.5%
Dev set error:	11%	16%	30%	1%
	high variance	high bias	high bias, high variance	low bias, low variance

Optimal (Bayes) error: 15%

Variance → overfitting

bias → underfitting

Basic recipe for machine learning

High bias →(Y) Bigger network

↓N

High variance → More data

Regularization

Bias variance tradeoff

↓ ↓

↓ ↑ → Deep learning de bu

Bigger network ile → low çözülmüş.

Regularization → Overfitting'i engeller? λ =||de büyük dursa over smooth high bias

$\min_{w,b} J(w,b)$ $w \in \mathbb{R}^{n_x}$, $b \in \mathbb{R}$ λ = regularization parameter

backward için

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\|w\|_2^2 + \frac{\lambda}{2m}b^2$$ (omit)

$$\frac{d}{dw}\left(\frac{\lambda}{2m}w^2\right) = \frac{\lambda}{m}w$$

L2 regularization $\|w\|_2^2 = \sum_{j=1}^{n_x} w_j^2 = w^T w$ ←

L1 regularization $\frac{\lambda}{2m}\sum_{j=1}^{n_x}|w_j| = \frac{\lambda}{2m}\|w\|_1$

→ cost'u arttırarak weightlerin küçük değerlerde kalmasını sağlar

$J(w^{[1]}, b^{[1]}, \ldots, w^{[L]}, b^{[L]}) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\sum_{l=1}^{L} \|w^{[l]}\|_F^2$

$\|w^{[l]}\|_F^2 = \sum_{i=1}^{n^{[l-1]}} \sum_{j=1}^{n^{[l]}} (w_{ij}^{[l]})^2$

$w: (n^{[l]}, n^{[l-1]})$

"Frobenius norm"

$\frac{d}{dw}\left(\frac{2\lambda}{m} w\right)$

$dw^{[l]} = (\text{from backward prop}) + \frac{\lambda}{m} w^{[l]}$

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

$\frac{dJ}{dw^{[l]}} = dw^{[l]}$

"weight decay"

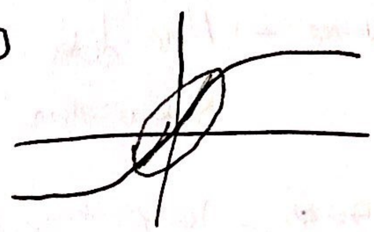
$w^{[l]} := w^{[l]} - \alpha\left[(\text{from backward}) + \frac{\lambda}{m} w^{[l]}\right]$

$\left(1 - \frac{\alpha\lambda}{m}\right) w^{[l]} = w^{[l]} - \frac{\alpha\lambda}{m} w^{[l]} - \alpha(\text{from backward prop})$

How does regularization prevent overfitting?

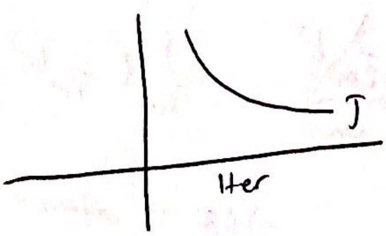
$\lambda \uparrow \quad w^{[l]} \downarrow \qquad \downarrow z^{[l]} = w^{[l]} a^{[l-1]} + b^{[l]}$

Every layer ≈ linear



z küçükse tanh linear'e yaklaşır.

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



16 Feb

Dropout regularization

→ Computer vision için sıklıkla kullanılır.

Implementing dropout ("Inverted dropout")

0.8 oranında 1
0.2 oranında 0 olur

Illustrate with layer $l=3$ keep-prop = 0.8

0.2 change eliminated

$d3 = np.random.rand(a3.shape[0], a3.shape[1]) < keep\_prop$

$a3 = np.multiply(a3, d3)$ # $a3 *= d3$
↳ elementwise

$a3 /= keep\_prob$

↳ $a3$ in shape i kadar %80 1 den %20 0 dan oluşan bir matris sonra $d3$ ile çarpılır.

50 units → 10 units shut off

$z^{[4]} = w^{[4]} \cdot a^{[3]} + b^{[4]}$

↑ reduced by 20%

$/= 0.8$

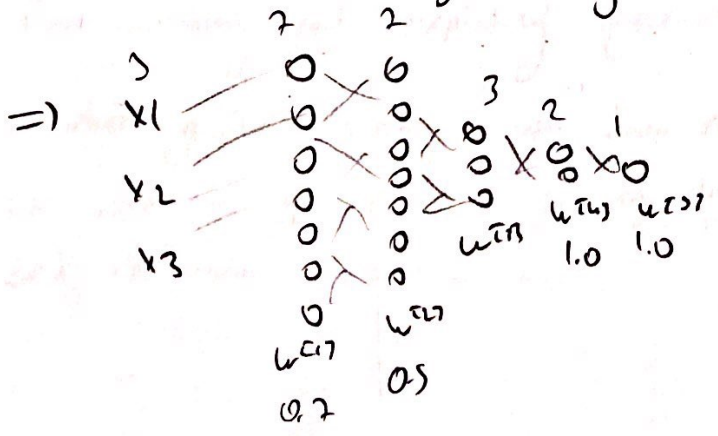
- → Regularization technique
- → only use dropout during training.
- → Applied both forward and backward

Making predictions at test time

No drop out → olası değerin random olmasını istemiyoruz.

Why does drop-out work?

Intuition: Can't rely on any one feature, so have to spread out weights



$w^{[1]} = (3,7)$

$w^{[2]} = (7,7)$ → w^2 ağırlıkları büyük olduğu için keep prop daha düşük olabilir. 0.5

$w^{[3]} = (7,7)$

$w^{[4]} = (3,2)$

$w^{[5]} = (2,1)$ diğerleri için 0.7

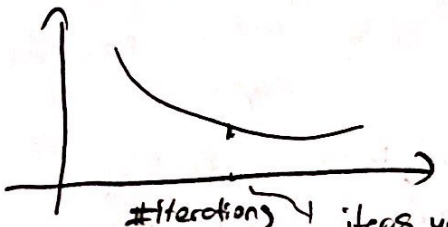
Aşırı öğrenme ihtimali de olan yüksek 1.0 da olabil

→ L2 and drop-out dan başka regularization tekniği

data augmentation → Bir cost regularization

- Resimlerin ters çevirmek
- Rastgele görüntüleri kırpma
- Zoomlama
- distorsiyon (bükülme)

2) Early stopping



iteras yonda durur.

Normalizing training sets → faster cost function optimize

Substract mean:

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

$$x := x - \mu$$

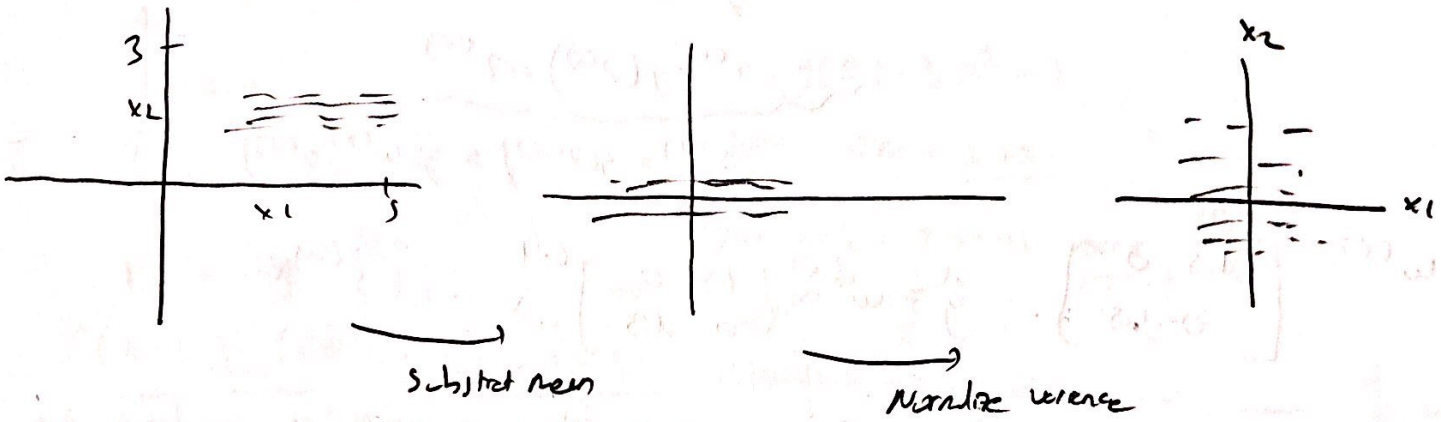
Normalize variance

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

↳ elementwise

$$x /= \sigma^2$$

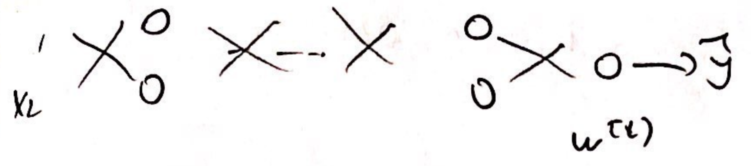
Use same μ σ^2 to normalize test set



If your input features came from very different scales, maybe some features are from 0 to 1, some from 1, to 1.000, then it's important to normalize your features.

Data vanishing and exploding gradients

=) When you're training a very deep network your derivatives or your slopes can sometimes get either very, very big or small, or even exponentially small, and this makes training difficult.



For sake of simplicity,

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

↳ linear

$\hat{y} = w^{[L]} \cdot w^{[L-1]} \cdots w^{[3]} w^{[2]} w^{[1]} x$ — ($w^{[1]} x = z^{[1]}$; the boxed part is $a^{[3]}$)

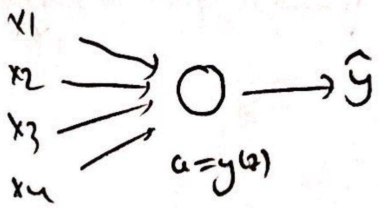
→ Bu değerler expo. olarak artar ve bu training etkilemesini zorlaştırır

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

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

$w^{[l]} = \begin{bmatrix} \cancel{1.5}\ 0.5 & 0 \\ 0 & \cancel{1.5}\ 0.5 \end{bmatrix} \quad \hat{y} = w^{[L]} \begin{bmatrix} \cancel{1.5}\ 0.5 & 0 \\ 0 & \cancel{1.5}\ 0.5 \end{bmatrix}^{L-1} \cdot x \rightarrow 1.5^{L-1} x$

Weight Initialization for Deep Network → vanishing or exploding gradients için weightlerin ayarlanması



$z = w_1 x_1 + w_2 x_2 + \dots + w_n x_n + \cancel{b}^{\,0}$

large n → smaller wi

$Var(w_i) = \cancel{\frac{1}{n}}\ \frac{2}{n}$

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

using Relu $\qquad g^{[l]}(z) = Relu(z)$

other variants,

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

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

He initialization

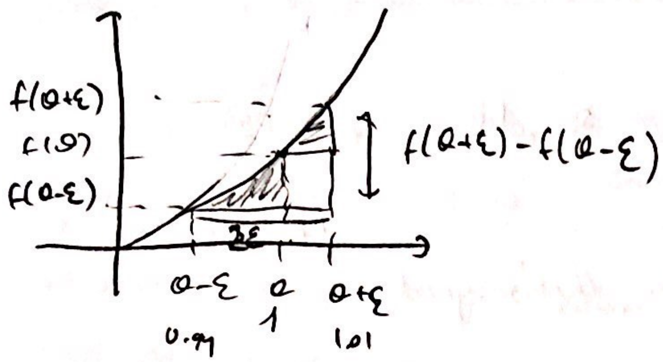
np.random.randn(layer_dims[l], layer_dims[l-1]) * np.sqrt(2./layer_dims[l-1])

17 Feb 2

Numerical approximation of gradients

Checking your derivative computation

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



$\varepsilon = 0.01$

g(1) için eşim denklemleri

→ eşim denklemi

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

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

$g(\theta) = 3\theta^2 = 3$

approx error = 0.0001

(per slide = 3.0301 error 0.03)

tekrarlı eğim kullanmadığımızda

$$\frac{f(\theta+\varepsilon) - f(\theta)}{\varepsilon} \quad \frac{(1.01)^3 - 1}{0.01} = 3.0301$$

$$f'(\theta) = \lim_{\varepsilon \to 0} \frac{f(\theta+\varepsilon) - f(\theta-\varepsilon)}{2\varepsilon} \quad O(\varepsilon^2) \quad 0.01 \quad 0.0001$$

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

Gradient checking

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

concatenate = θ de birleştir

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

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

concatenate

Gradient checking (Grad check) $J(\theta) = J(\theta_1, \theta_2, ..)$ → giant parameter

for each i:

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

$$\approx d\theta[i] = \frac{\partial J}{\partial \theta_i}$$ | $d\theta_{app} \approx d\theta$ $\|\ \|_2$ → np.linalg.norm(..)

Check $\frac{\|d\theta_{approx} - d\theta\|_2}{\|d\theta_{approx}\|_2 + \|d\theta\|_2}$ → uzaklık $\approx 10^{-7}$ or smaller that's great

10^{-5}

$\varepsilon = 10^{-7}$ 10^{-3} → worry

for find lots of bugs in implementation of neural nets,

- gradient checking eğitimde kullanılmaz !! Sadece hata ayıklamak için. only to debug
- $d\theta_{app}$ very far from $d\theta$.
- Remember regularization

def grad_check(x, theta, epsilon):
  $\theta^+ = \theta + \varepsilon$
  $\theta^- = \theta - \varepsilon$
  $J^+ =$ forward_prop(x, θ^+)
  $J^- =$ for... (x, θ^-)
  gradapp $= (J^+ - J^-) / (2 * \varepsilon)$] check et
  grad $=$ backward_prop(x, θ)]

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

bunu eklemeyi unutma

- gradient check dropout ile birlikte çalışmaz
  dropout olurken J maliyet fonksiyonu hesaplaması kolay değil
  keep-prob=1.0'a ayarlayıp öyle kontrol edilebilir.

Scanned with CamScanner

1) high variance versus — high bias versus

bigger network

- Get more training data
- Add regularization
- Get more test data

2) training set error 0.5%

dev set error 7% — Overfitting also,

- increase the regularization parameter lambda
- Get more training data

3) Weight decay

A regularization technique that results in gradient descent shrinking the weights on every iteration. (L2 regu such as)

cadence, term

4) ↑ λ regularization parameter weights close to 0

5) useful for reducing variance

- Dropout
- L2 regularization
- Data augmentation

6) Why do we normalize the input x?

It makes the cost function faster to optimize

Scanned with CamScanner

Model'i eğitirken weight'leri 0 yapmak fails to break symmetry dır.

random yapmak to large weights dır

random yapıp initialize etmek genel He

Xavier yöntemi

He → np.random.randn(L_d[l], L_d[l-1]) * np.sqrt(2./L_d[l-1])

Xavierde 2 yerine 1

→ Regularization will help you reduce overfitting

→ will drive your weights to lower values

→ L2 regularization and Dropout are two very effective regularization techniques