

Data 분리 -

1. Training Set

2. Hold-out CV set / Dev set ← 다양한 모델 중 어떤 모델이 좋은 성능을 내는지 확인함.

3. Test Set

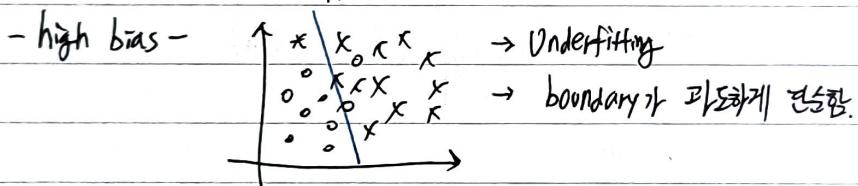
▷ Trend

- dataset이 100u 100000개 : 60:20:20
- 1000000개 이상 : 98:1:1 또는 더 높게.
- 요즘은 Train-test 분할과 다른 세트에서 트레이닝을 진행하기도 함.
- Test set이 없어도 가능할 수도 있음.
 - Unbiased한 추정치가 필요하지 않은 경우.

▷ Bias, Variance

Bias : 편향, Variance : 편차, 분산.

Bias-Variance trade-off



- high Variance \Rightarrow Overfitting

· boundary가 과도하게 복잡.

Training set error : 1%

15%

15%

Dev set error : 11%

16%

30%

training set에 의해
Overfit

↓

: high Variance

training Data에
잘 적합되지 않음

↓

\rightarrow Underfitting

high bias (Underfitting)

high Variance (Overfitting)

(worse)

: high bias

* 이러한 가정은 일반화 오류가 0%라고 가정함.

base error에 따라 다른 방법을 채택해야함.

▷ Basic 'Recipe'

- High bias \rightarrow Bigger NN
(Underfitting) $\downarrow N$

ML 알고리즘을 고차원적으로 바꿀.

- High Variance? \rightarrow 더 많은 Data
(overfitting) \downarrow
train Overfit

적합한, 아키텍처.

- 큰 NN 학습은 Bias를 거의 항상 줄여줌. (\leftrightarrow Bias-Vari tradeoff이 유용하지 않음)
- 데이터도 다양성. Variance를 항상 줄여줌.
- Regularization - Variance를 줄이는 효과적인 방법임.
(overfitting)

< Regularizing Neural Network >

Regularization

$$\text{비용함수 } J(w, b) = \frac{1}{m} \sum_{i=1}^m L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m} \|w\|_2^2 \quad (\text{L2 정규화})$$

$$L1: \|w\|_1 = \sum_{j=1}^{n_x} |w_j| \quad \text{// L1 norm}$$

$$L2: \|w\|_2^2 = \sum_{j=1}^{n_x} w_j^2 = w^T w \quad \text{// L2 norm}$$

- L2 정규화가 더 보편적으로 사용됨.

- L1 정규화: weight vector를 'sparse'하게 만듬. \rightarrow 모델 압축.

- λ : regularization hyperparameter. \Rightarrow 'Lambda'로 표현.

- Neural Network에는 다음과 같음.

$$J(w^{[0]}, b^{[0]}, \dots, w^{[L]}, b^{[L]}) = \frac{1}{m} \sum_{i=1}^m L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m} \sum_{l=1}^L \|w^{[l]}\|_F^2$$

\hookrightarrow "Frobenius norm", $\|w^{[l]}\|_F^2 = \sum_{i=1}^{n^{[l]}} \sum_{j=1}^{n^{[l+1]}} (w_{ij}^{[l]})^2$, $n^{[l]}$: layer l 에 있는 hidden units

$\therefore w = (n^{[0]}, n^{[1]}, \dots)$ shape으로, 그저 w 의 sum of squares를 포함.

- 역전파 + Regularization

$$dW^{[l]} = \left[\frac{1}{m} dZ^{[l]} \cdot A^{[l-1]T} + \frac{\lambda}{m} W^{[l]} \right] \rightarrow \text{regularization term gradient}$$

$$\rightarrow W^{[l]} := W^{[l]} - \alpha dW^{[l]}$$

- L2 Reg = "Weight Decay"

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

$$= (1 - \alpha \frac{\lambda}{m}) W^{[l]} - \alpha \left(\frac{1}{m} dZ^{[l]} \cdot A^{[l-1]T} \right)$$

≤ 1

역전파 (기준)

Weight의 1보다 작은값인 $(1 - \alpha \frac{\lambda}{m})$ 이 계산

곱해져서 'decay'를 표현.

Why Regularization reduces Overfitting?

$$\cdot J(w, b) = (\text{Sum of loss function}) + \frac{\lambda}{2m} \sum \|w^{[l]}\|_F^2$$

↳ 이 유의항이 weight matrix를

너무 큰 값을 갖지 못하게 penalize하기 때문.

- $\lambda \uparrow$: weight matrix 대부분이 0으로 됨. → NN 단순화. (logistic 모형과 비슷)
 - 각각의 hidden unit의 효과가 줄어드는 것.

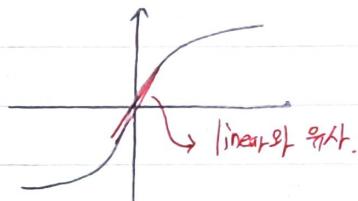
- tanh activation의 경우, $g(z) = \tanh(z)$

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

z 가 적은 범위를 가지므로, $G(z)$ 는 대략적인 선형 (roughly linear)가 됨!

⇒ linear regression과 유사해짐

비선형 정도가 약해지므로, 복잡한 결정 경계를
만들 수 없음.



▷ Dropout 정규화

- 각 iteration에서 일정 확률로 일부 뉴런을 제거함.

- 가장 보통적인 것은 Inverted Dropout

$$\text{keep_prob} = 0.8$$

$$l=3$$

$$\text{random matrix } d_3 = \text{np.random.rand}(a[3].\text{shape}[0], a[3].\text{shape}[1]) < \text{keep_prob}$$

$$a_3 = \text{np.multiply}(a_3, d_3) // \text{element-wise multiply}$$

Inverted dropout technique. $a_3 = a_3 / \text{keep_prob}$ // a_3 이 약 20% 정도는 제거되기 때문에, 기댓값을 변하지 않게 하도록 scaling.

▷ Understanding Dropout

효과) 곤란으로 노드를 삭제하기 때문에, 하나의 feature에 다른 노드가 의존하지 못하게 만들

→ Weight를 분산 (spread out weights)

- keep_prob을 충분히 다르게 하거나 가능.

- Overfitting이 걱정되는 layer에서 keep_prob을 낮기 성공함으로써 대응할 수 있음. $\approx L2 \text{ reg} \uparrow$

- Input layer에는 보통 1.

- 하지만 충분히 다르게 할 경우 나중 CV에서 성능 좋게 많아짐.

- CV에서는 Dropout이 혼란. input size가 매우 크고 보통은 data가 충분하지 않기 때문.

- 단점

- 비용 함수 J 가 잘 정의되지 못함 // 디버깅이 어려움.

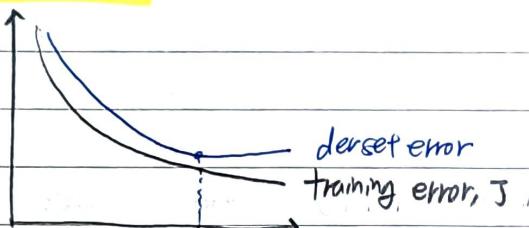
- 문제가 있다면, dropout을 고고 J 가 잘 감소하는지 체크.

▷ 다른 Regularization 테크닉

• Data Augmentation (증강)

- 랜덤하게 flip, 회전, 줄인/이웃, 짜그리프리기
- 새로운 Data를 얻기 힘들 때 사용.

• Early Stopping



: training cost와 devset cost가 초기점에서 iteration 중단.

- 단점: $J(w, b)$ 를 최소화하는 것과 Overfitting을 막는 것은 별개의 문제임.
early stopping은 중간에 중지하기 때문에 J 가 최소가 아닐 수 있음.

* 직교화 (Orthogonalization)

- Optimize J
: finding optimal w, b
- Prevent Overfitting
: 다른 방식... (L2 Reg)

▷ Normalizing (정규화)

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

$$\textcircled{2} \text{ Normalize Variance } \sigma^2 = \frac{1}{m} \sum_{i=1}^m (x^{(i)})^2 // \text{element-wise squaring}$$

$$\Rightarrow x := \frac{x - \mu}{\sigma}$$

- Normalize 하는 이유?

→ 비용함수의 모양이 일관되지 않아서, 최적화하는데 오랜 시간이 걸리기 때문

→ Normalize 시, 비용함수의 모양은 동그란 접시 모양, 즉 symmetric 해질.

Vanishing / Exploding gradient

- derivative (기울기) 매우 크거나 작아짐.

- 만약 $b = 0$ 일을 가정하면

$$\hat{y} = w^{[L]} w^{[L-1]} w^{[L-2]} \dots w^{[3]} w^{[2]} w^{[1]} x, \quad o^{[1]} = g(z^{[1]})$$

이는 층 layer 개수가 많을수록 gradient와 activation이 explode/vanish됨.

$W^{[L]} > I$: explode

$W^{[L]} < I$: vanish → 학습이 매우 느려짐.

$$z^{[1]} = W^{[1]} x$$

▷ Weight Initialization for Deep NN

(b를 제외하면) Single neuron model로 하기

$$Z = w_1x_1 + w_2x_2 + \dots + w_nx_n, n = \# \text{ of Input feature}$$

방법) 1. w_i 의 Variance를 $\frac{1}{n}$ 으로 설정.

2. ReLU에서는, w_i 의 Variance를 $2/n^{[l-1]}$ 으로 설정.

3. tanh에서는, w_i 의 Variance를 $\frac{1}{n^{[l-1]}}$ 또는 $\frac{2}{n^{[l-1]}+n^{[l]}}$ 로 설정.

• layer l 의 각각의 units는 $n^{[l-1]}$ 개의 input features를 지니는데,

이것들이 평균 0 분산 1로 되면 Z 가 비슷한 scale을 가지게됨.

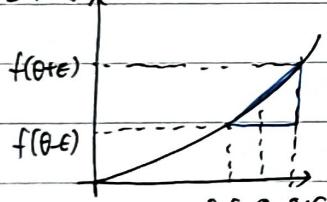
\Rightarrow weight matrix가 I에 균형하게 되고, 따라서 exploding과 vanishing을 어느정도 방지함.

$$\text{예) } np.random.rand(\text{shape}) + np.sqrt(\frac{1}{n^{[l-1]}})$$

▷ gradient의 수치근사

◦ 역전파를 알맞게 구현했는지 확인하기위함.

<중심차분법>



$$f'(\theta) = \lim_{\epsilon \rightarrow 0} \frac{f(\theta + \epsilon) - f(\theta - \epsilon)}{2\epsilon}$$

◦ 단방향 차별보다 느리게 작동함.

◦ 하지만 단방향 차별보다 정확함.

▷ Gradient Checking

1. 모델 안에 있는 모든 w, b 를 하나의 대형 vector θ 로 reshape. (Concatenate)

2. 비용함수는 $J(w^{[1]}, b^{[1]}, \dots, w^{[L]}, b^{[L]}) \Rightarrow J(\theta)$ 가됨.

3. $dW^{[1]}, db^{[1]}, \dots, dW^{[L]}, db^{[L]}$ 도 $d\theta$ 로 concatenate.

for each i :

$$d\theta_{\text{approx}}^{[i]} = \frac{J(\theta, \dots, \theta_i + \epsilon, \dots) - J(\theta, \dots, \theta_i - \epsilon, \dots)}{2\epsilon} \approx d\theta^{[i]} = \frac{\partial J}{\partial \theta^{[i]}}$$

4. 수치미분과 일치하는지 비교. $[d\theta_{\text{approx}}^{[i]} \approx d\theta]$

[assert shape]

• 유사도: 유물적적인 거리 사용.

$$= \frac{\|d\theta_{\text{approx}}^{[i]} - d\theta\|_2}{\|d\theta_{\text{approx}}^{[i]}\|_2 + \|d\theta\|_2} \quad \begin{cases} 10^{-7}: \text{Good} \\ 10^{-5}: \\ 10^{-3}: \text{Bad} \end{cases}$$

▷ Gradicheck에서 주의할 점

- Debug에서는 사용 (training X!)
- 경사검사에 실패했으면, 특정 원인도 뿐만 아니라 실패하는지 check.
- 정규화 항 까먹지 말기
- dropout은 끄기 (Random하게 0으로 만들기 때문에)

▷ Week 1 쓰임

• He Initialization ☆

→ random에 고려해줄 것.

- Xavier : $W^{[l]}$ 의 scaling factor $\sqrt{1/\text{layer.dims}[l-1]}$
- He : $\sqrt{2/\text{layer.dims}[l-1]}$, ReLU activation이나 추천.

- dim of $W^{[l]}$: (l번재 layer의 노드수, l-1번재 layer의 노드수)

- dim of $b^{[l]}$: (l번재 layer의 노드수, 1)

- Zero Initialization의 경우, Symmetry break를 실패함.

- Gradicheck

$$-\frac{\partial J}{\partial \theta} = \lim_{\epsilon \rightarrow 0} \frac{J(\theta + \epsilon) - J(\theta - \epsilon)}{2\epsilon}$$

'Gradient' (dict) → 'grad' (vector)로 바꾸기

<2주차>

* 표기법

▷ Mini-batch GD

$$X = (n_x, m), Y = (1, m)$$

- i번재 training set : $x^{(i)}$

- i번재 총의 Z값 : $Z^{(i)}$

- t번재 minibatch : $X^{[t]}, Y^{[t]}$

$$m = 5000,000$$

$$\text{minibatch size} = 1000$$

$$\rightarrow \text{Count} = 5000$$

- t번재 minibatch : $X^{[t]}, Y^{[t]}$ 를 한번에 처리.

- for $t=1, \dots, 5000$,

Forward Prop on $X^{[t]}$:

$$\begin{aligned} Z^{[t]} &= W^{[0]} X^{[t]} + b^{[0]} \\ A^{[t]} &= g^{[0]}(Z^{[t]}) \end{aligned} \quad \left. \right\} \text{1000개 minibatch}$$

$$A^{[t]} = g^{[0]}(Z^{[t]})$$

$$\text{Cost } J^{[t]} \equiv \underbrace{\frac{1}{1000} \sum_{i=1}^L}_{\text{미니배치 평균}} \underbrace{f(y^{(i)}, \hat{y}^{(i)})}_{\text{C } X^{[t]}, Y^{[t]} \text{ 대}} + \underbrace{\frac{\lambda}{2 \cdot 1000} \sum_{l=1}^L \|W^{[l]}\|_F^2}_{\text{정규화 항}}$$

미니배치 평균

Backprop: J^{ft} 에 대한 gradient 계산. (오직 $\Delta x^{ft}, \gamma^{ft}$) 사용.

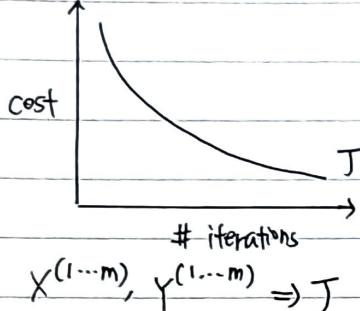
$$w^{[l+1]} := w^{[l]} - \alpha d_w^{[l]}, b^{[l+1]} = b^{[l]} - \alpha d_b^{[l]}$$

batch: 트레이닝 세트 1번 통과(1 epoch) : 1번의 gradient descent 수행

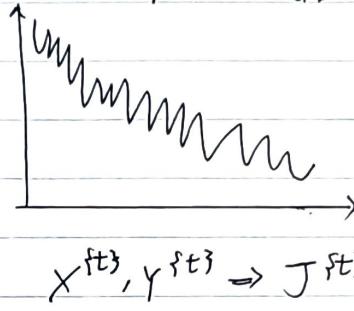
minibatch: 1 epoch \Rightarrow minibatch 수 만큼 GD 수행.

▷ 미니배치 경사하강법의 이해

Batch GD



Minibatch GD



• 미니배치의 Size

Size = m : Batch GD - iteration마다 너무 긴 시간 소요. 단점

Size = 1: Stochastic gradient descent, 모든 i 가 미니배치임.

모든 학습 데이터에 기울기 강화 적용.

- 광장학 |噪音| 해칠 수 있음, 절대 수렴하지는 않음.

- 또한, vectorization (벡터화)에서 대부분의 속도를 잃음.

단점

Size = 적당: • 적당한 벡터화

- 빠른 학습 속도

• 가이드라인

If 작은 training set ($2000 \downarrow$): batch GD

보통 $64 \sim 512$ 중에서 2의 배수형을 취할 때 이상적임.

$$\begin{matrix} \downarrow \\ 2^6 \end{matrix} \quad \begin{matrix} \downarrow \\ 2^9 \end{matrix}$$

▷ Exponentially Weighted averages (지수 가중평균)

- 최근의 데이터에 더 많은 영향을 끼친 데이터의 평균 흐름을 계산할 수 있음.

즉, 최근의 데이터 직접에 더 높은 가중치를 줌.

$$\theta_1 = 40^\circ\text{F}$$

$$V_t = t\text{번째 날의 EWA}$$

$$\theta_2 = 49^\circ\text{F}$$

$$V_0(\text{초기값}) = 0$$

$$V_t \approx \frac{1}{1-\beta} \text{ 일(최근) 동안의 평균 기온과 비슷.}$$

:

$$V_1 = \beta V_0 + (1-\beta)\theta_1$$

$$\beta = 0.9 \approx \text{최근 10일동안}$$

$$\theta_{180} = 60^\circ\text{F}$$

$$V_2 = \beta V_1 + (1-\beta)\theta_2$$

$$\beta = 0.98 \approx \text{5일동안 } //$$

$$\theta_{181} = 56^\circ\text{F}$$

(온도)

$$V_t = \beta(V_{t-1}) + (1-\beta)\theta$$