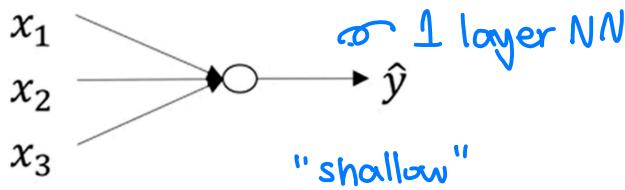
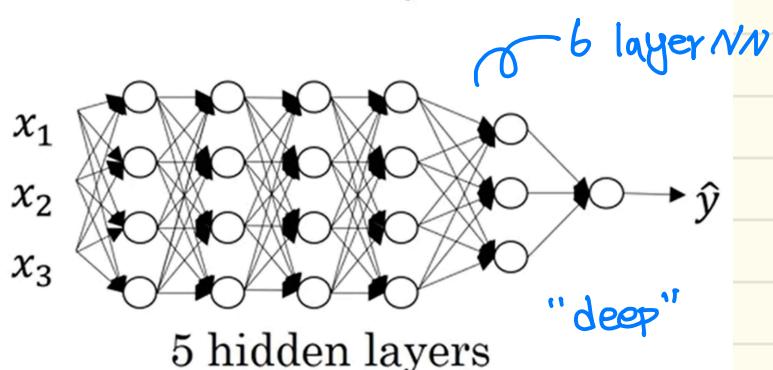
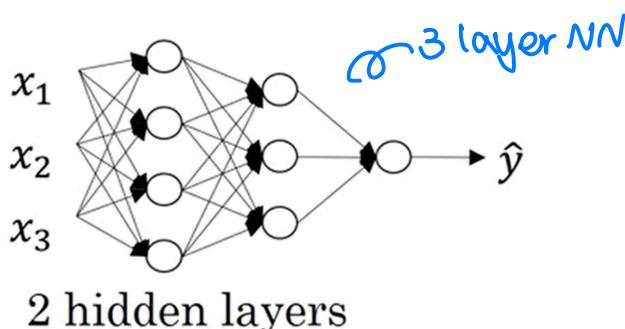
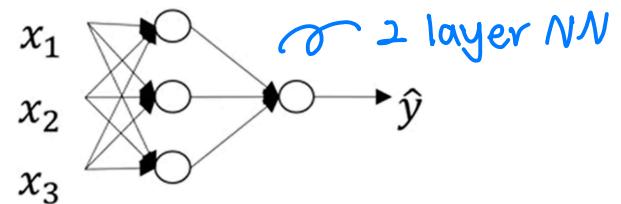


[4-1. Deep Neural Network]

<Deep L-layer Neural Network>

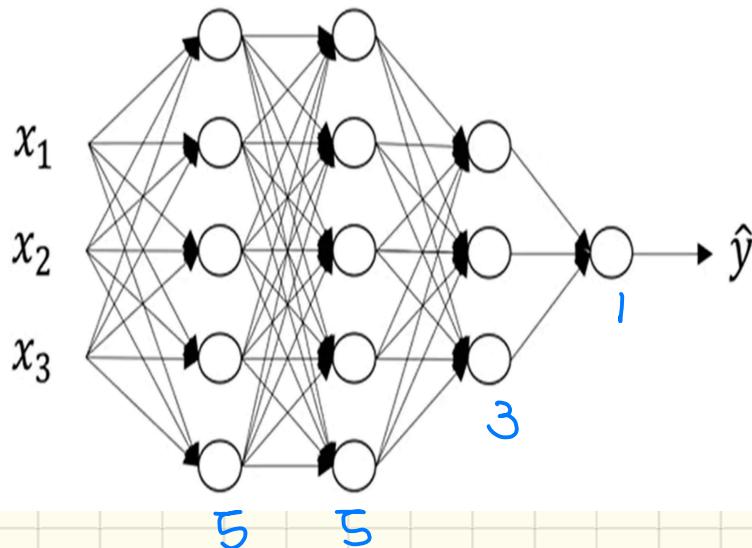


logistic regression



* 얼마나 깊은 신경망을 사용해야 하는지는 예측하기 어렵지만, 하이퍼파라미터를 조정하면서 깊이를 결정해야 한다.

ex) 4 layer NN



$$L = \# \text{layers} = 4$$

$$n^{[l]} = \# \text{ units in layer } l$$

$$n^{[0]} = 3, n^{[1]} = 5, n^{[2]} = 5$$

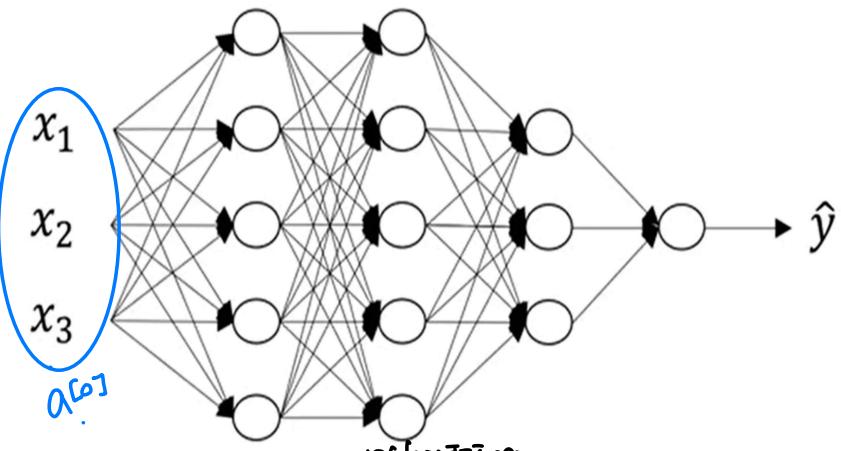
$$n^{[3]} = 3, n^{[4]} = n^{[L]} = 1$$

$$a^{[l]} = g^{[l]}(z^{[l]}) = \text{activation in layer } l$$

$$a^{[0]} = x, a^{[L]} = \hat{y}$$

$$w^{[l]}, b^{[l]}. \text{ weights for } z^{(l)}$$

<Forward Propagation In a Deep Network >



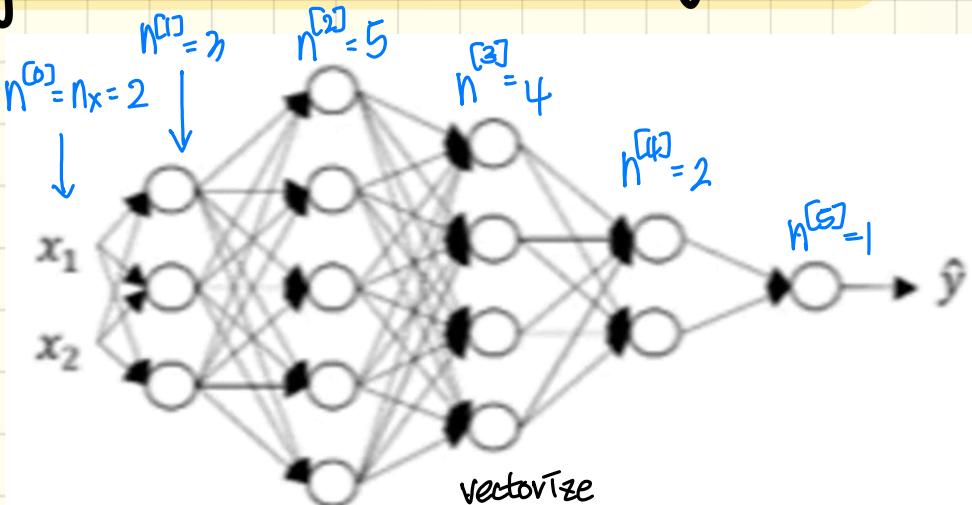
$$\begin{aligned}
 z^{[1]} &= w^{[1]} x + b^{[1]} \\
 a^{[1]} &= g^{[1]}(z^{[1]}) \\
 z^{[2]} &= w^{[2]} a^{[1]} + b^{[2]} \\
 a^{[2]} &= g^{[2]}(z^{[2]}) \\
 &\vdots \\
 \Rightarrow z^{[l]} &= w^{[l]} a^{[l-1]} + b^{[l]} \\
 a^{[l]} &= g^{[l]}(z^{[l]}) \\
 \end{aligned}$$

vectorizing

$$\begin{aligned}
 z^{[1]} &= W^{[1]} A^{[0]} + b^{[1]} \\
 A^{[1]} &= g^{[1]}(z^{[1]}) \\
 z^{[2]} &= W^{[2]} A^{[1]} + b^{[2]} \\
 A^{[2]} &= g^{[2]}(z^{[2]}) \\
 &\vdots \\
 \hat{y} &= g(z^{[4]}) = A^{[4]}
 \end{aligned}$$

} for $l=1$ to 4

<Getting your Matrix Dimensions Right>

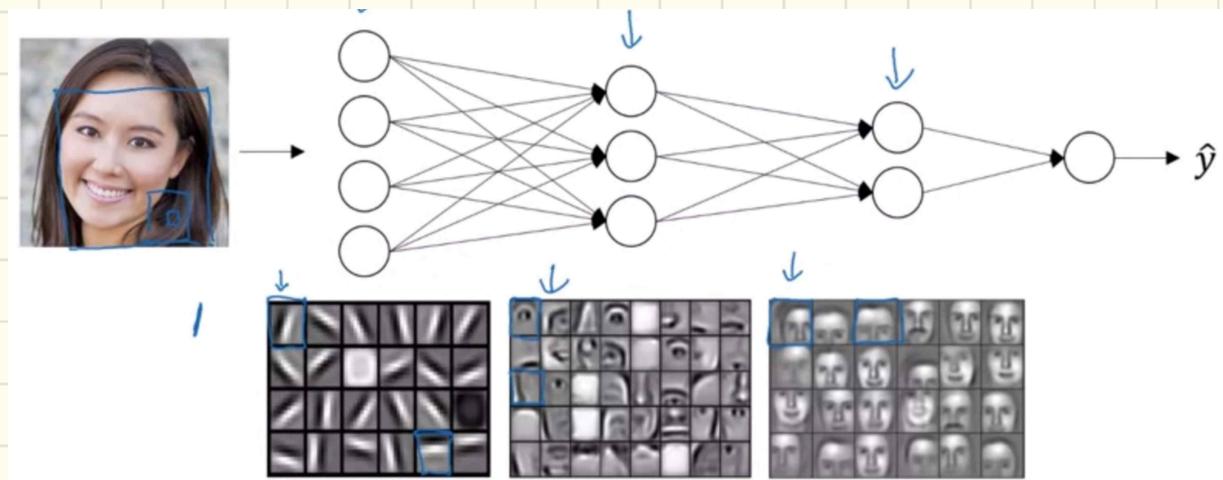


$$\begin{aligned}
 z^{[1]} &= W^{[1]} x + b^{[1]} \Rightarrow z^{[l]} = (n^{[l]}, 1) \\
 (3,1) \quad (3,2) \quad (2,1) \quad (3,1) & \quad W^{[l]} = (n^{[l]}, n^{[l+1]}) \\
 z^{[2]} &= W^{[2]} a^{[1]} + b^{[2]} \quad b^{[l]} = (n^{[l]}, 1) \\
 (5,1) \quad (5,2) \quad (3,1) \quad (3,1) & \quad dW^{[l]} = (n^{[l]}, n^{[l+1]}) \\
 d b^{[l]} &= (n^{[l]}, 1) \\
 \end{aligned}$$

$$\begin{aligned}
 z^{[1]} &= W^{[1]} x + b^{[1]} \\
 (3,m) \quad (3,2) \quad (2,m) \quad (3,1) & \quad \Rightarrow z^{[l]}, A^{[l]}, (n^{[l]}, m) \\
 \end{aligned}$$

$$dZ^{[l]}, dA^{[l]}, (n^{[l]}, m)$$

<Why Deep Representations?>



직관 1) 네트워크가 깊어질수록 더 많은 특징을 잡아낼 수 있다. 낮은 층에서는 간단한 특성을 찾고 깊은 층에서는 낮은 층에서 텁자한 간단한 것들을 종합해서 더 복잡한 특성을 찾아낼 수 있다.

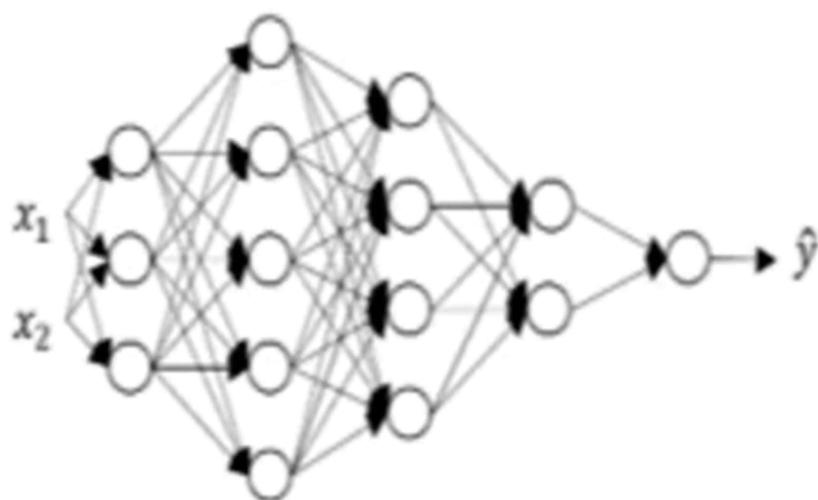
Circuit theory and deep learning

Informally: There are functions you can compute with a “small” L-layer deep neural network that shallower networks require exponentially more hidden units to compute.

직관 2) 순환이론에 따르면. 얕은 네트워크보다 깊은 네트워크에서 더 저산하기 쉬운 행위가 있다.

<Building Blocks of Deep Neural Networks>

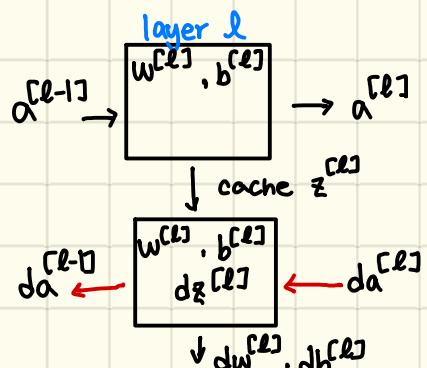
* Forward and Backward functions



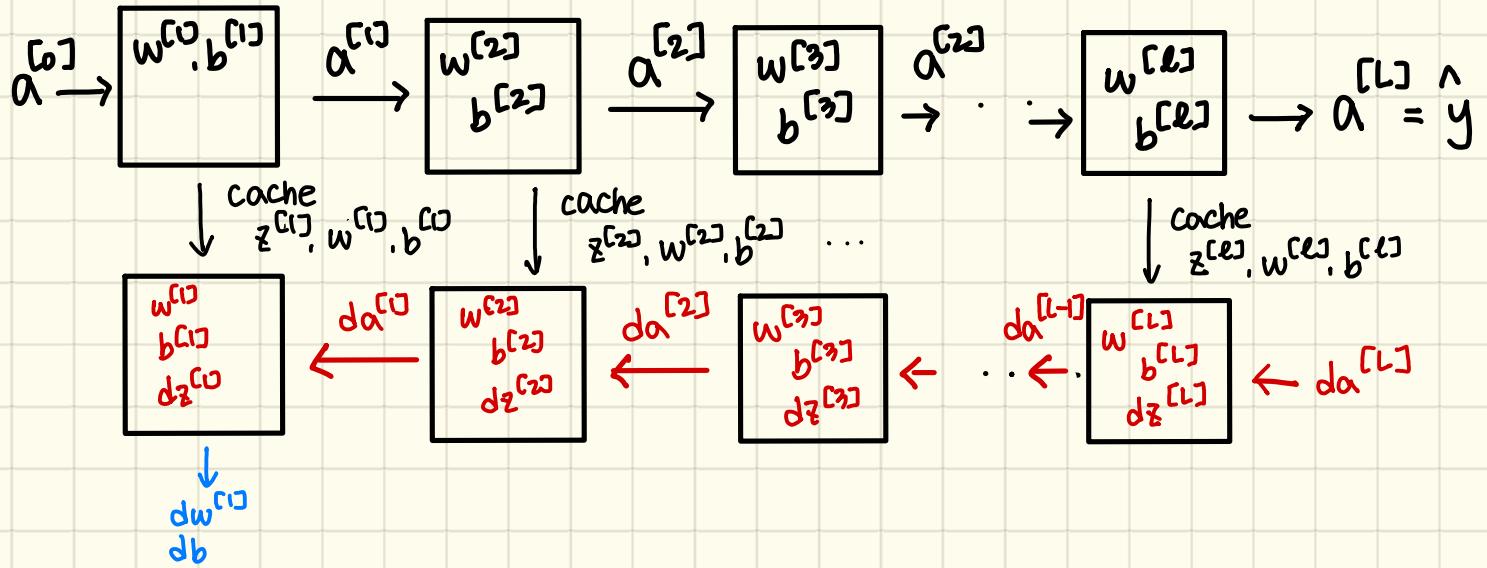
Layer l : $W^{[l]}, b^{[l]}$

Forward · Input $a^{[l-1]}$, Output $a^{[l]}$
cache $z^{[l]}$

Backward · Input $da^{[l]}$
Output $da^{[l-1]}, dw^{[l]}, db^{[l]}$



회상해보면,



<Forward and Backward Propagation>

* Forward propagation for layer l

- Input : $\alpha^{[l-1]}$
 - Output : $a^{[l]}$, cache ($z^{[l]}$) \oplus coding 관점에서 cache $w^{[l]}, b^{[l]}$
- $$z^{[l]} = w^{[l]} \cdot \alpha^{[l-1]} + b^{[l]} \quad \xrightarrow{\text{vectorize}} \quad z^{[l]} = W^{[l]} \cdot A^{[l-1]} + b^{[l]}$$
- $$a^{[l]} = g^{[l]}(z^{[l]}) \quad \quad \quad A^{[l]} = g^{[l]}(z^{[l]})$$

* Backward propagation for layer l

- Input : $da^{[l]}$
- Output : $da^{[l-1]}, dW^{[l]}, db^{[l]}$

$$dz^{[l]} = da^{[l]} * g^{[l]}'(z^{[l]})$$

$$dW^{[l]} = dz^{[l]} \cdot \alpha^{[l-1]T}$$

$$db^{[l]} = dz^{[l]}$$

$$da^{[l]} = W^{[l]T} \cdot dz^{[l]}$$

$$\rightarrow dz^{[l]} = W^{[l+1]T} \cdot dz^{[l+1]} * g^{[l]}'(z^{[l]})$$

$$dz^{[l]} = dA^{[l]} * g^{[l]}'(z^{[l]})$$

$$dW^{[l]} = \frac{1}{m} dz^{[l]} \cdot A^{[l-1]T}$$

$$db^{[l]} = \frac{1}{m} \text{np.sum}(dz^{[l]}, \text{axis}=1, \text{keepdims=True})$$

$$dA^{[l-1]} = W^{[l]T} \cdot dz^{[l]}$$

< Parameters VS Hyperparameters >

* What are hyperparameters?

- parameters : $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}, \dots$

- Hyperparameters : learning rate α

iterations

hidden layer L

hidden units $n^{[1]}, n^{[2]}, \dots$

Choice of activation functions

...

Improving Deep Neural Network

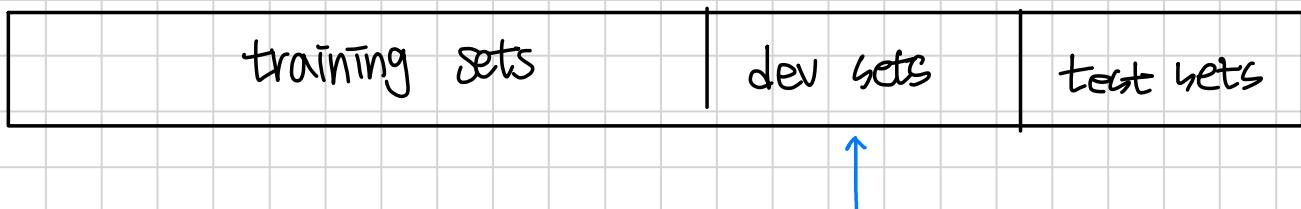
: Hyperparameter tuning, Regularization and Optimization

[I-1. Setting up your Machine Learning Application]

<Train / Dev / Test sets>

- 신경망이 몇개의 층을 가지는지, 각 층이 몇개의 은닉층을 가지는지, 학습률과 활성화 함수는 무엇인지 등을 결정해 신경망을 훈련시켜야 한다.
- 좋은 하이퍼파라미터를 찾기 위해 KFold 사이클을 여러번 반복해야 한다.

* train / dev / test sets



- 과정에서, $\text{train} : \text{test} = 70:1 : 30:1$ 로 나누면 좋음.
- Big data 환경에는, 예제들이 데이터가 1,000,000개가 있다고 하면 dev set과 test set은 1%만 사용해도 된다.

* Mismatched train / test distribution

en training sets: Cat pictures from webpages

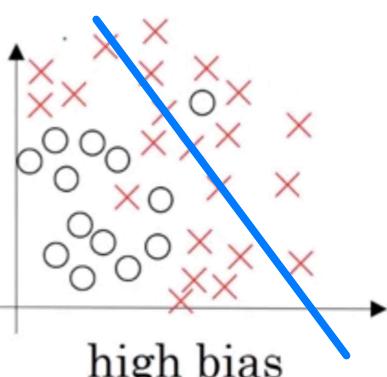
Dev / test sets: Cat pictures from users using your app

→ 훈련 세트와 테스트 세트가 같은 분포에서 나온지 확인해야 함

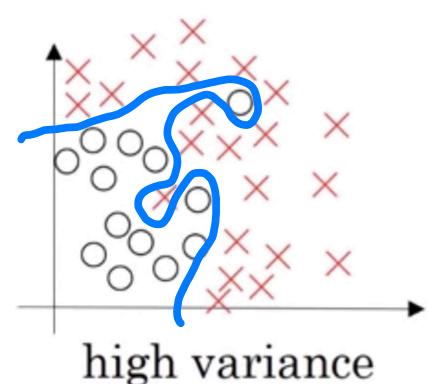
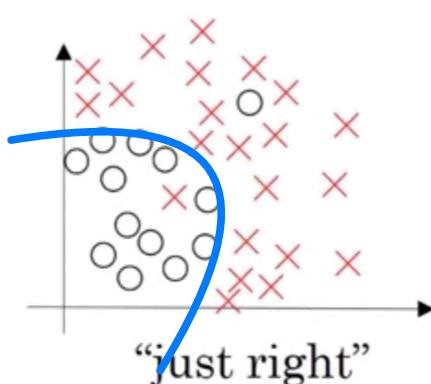
⊕ test set이 훈련 세트만 있어도 됨

<Bias and Variance>

* 편향과 분산의 트레이드 오프



↳ under fitting



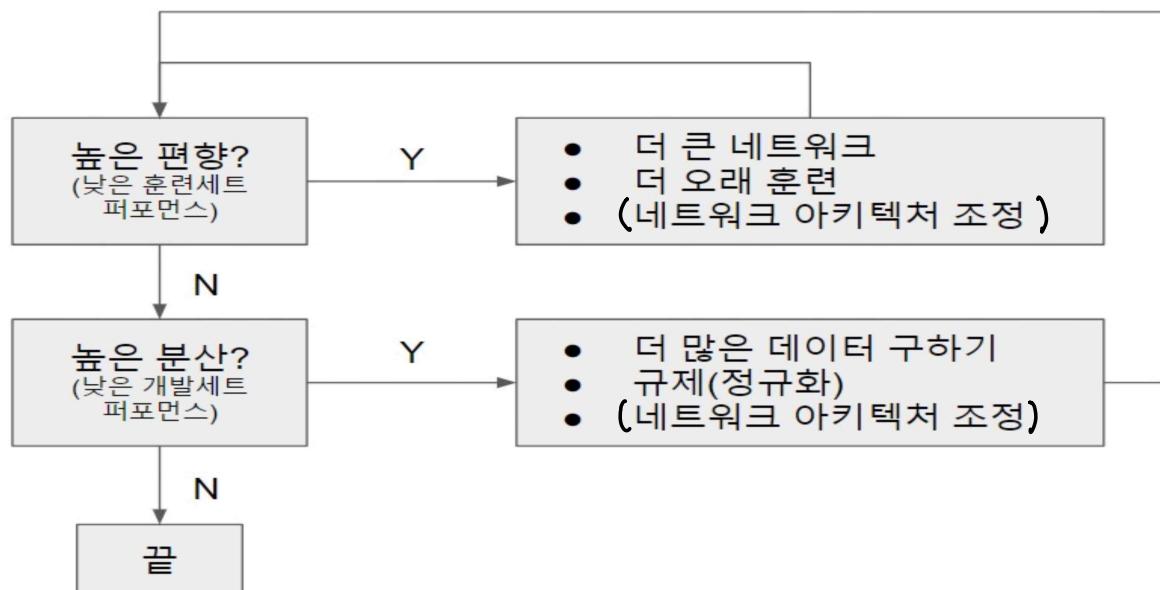
↳ over fitting

* 훈련 세트과 개발 세트의 관계

- 가능성 · 인간 수준의 성능이 기본이 되어야 한다. 일반적으로 말하면, 레이지안 회적 오차가 0.1% 이하

error	높은 분산 (과대적합)	높은 편향 (과소적합)	높은 편향 & 높은 분산	낮은 편향 & 낮은 분산
훈련 세트	1 %	15 %	15 %	0.5 %
개발 세트	11 %	11 %	30 %	1 %

<Basic Recipe for Machine Learning>



[1-2. Regularizing your Neural Network]

< Regularization >

* Logistic Regression

비용함수: $J(w, b) = \frac{1}{m} \sum_{i=1}^m L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m} \|w\|_2^2 + \frac{\lambda}{2m} b^2$, $w \in \mathbb{R}^{n_x}$, $b \in \mathbb{R}$

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

L1 regularization: $\frac{\lambda}{2m} \sum_{i=1}^m |w_i| = \frac{1}{2m} \|w\|_1$ → w will be sparse (= w 가 0을 가지고 있음)
 ⇒ 모델 압축에 도움이 될 수 있음.

* Neural Network

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

→ Frobenius norm: $\|w^{[l]}\|_F^2 = \sum_{i=1}^{n^{[l-1]}} \sum_{j=1}^{n^{[l]}} (w_{ij}^{[l]})^2$ ($\because w \in \mathbb{R}^{n^{[l-1]} \times n^{[l]}}$)

- L2 정규화가 weight decay라고 불리는 이유

$$\begin{aligned} w^{[l]} &:= w^{[l]} - \alpha d w^{[l]} = w^{[l]} - \alpha \{ \text{(from backprop)} + \frac{\lambda}{m} w^{[l]} \} \\ &= w^{[l]} - \frac{\alpha \lambda}{m} w^{[l]} - \alpha \{ \text{(from backprop)} \} \\ &= \underbrace{\left(1 - \frac{\alpha \lambda}{m}\right)}_{\text{즉, } 1\text{보다 작은 값인 }} w^{[l]} - \alpha \{ \text{(from backprop)} \} \end{aligned}$$

↳ 즉, 1보다 작은 값인 $(1 - \frac{\alpha \lambda}{m})$ 가 곱해지기 때문

< why Regularization Reduces Overfitting? >

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

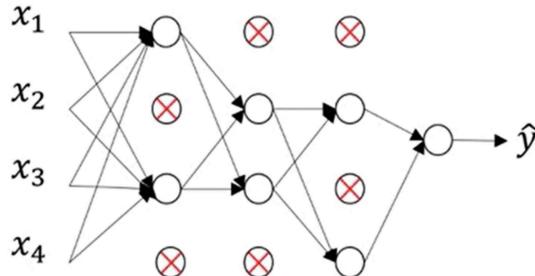
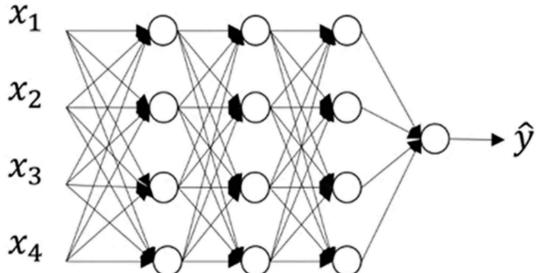
- 이때 λ 를 매우 크게 하면, $w^{[l]} \approx 0$ 이 되고, (\because 비용함수 최소화)

⇒ 그 결과 hidden units의 개수가 적어져서

⇒ 작은 신경망이 되어, 과대적합이 일어나지 않는다.

< Dropout Regularization >

* Dropout Regularization



- 드롭아웃 방식: 신경망의 각각의 층에 대해 노드를 삭제하는 확률을 설정하는 것. 삭제할 노드를 선정한 후, 삭제된 노드의 들어가는 링크와 나가는 링크를 모두 삭제함.
→ 더 적고 간소화된 네트워크가 만들어지고, 이 라이전 네트워크로 훈련함.

* Implementing dropout

ex) $\text{layer} = 3$, $\text{keep_prob} = 0.8$

$d_3 = \text{np.random.rand}(a_3.\text{shape}[0], a_3.\text{shape}[1]) < \text{keep_prob}$

$a_3 = \text{np.multiply}(a_3, d_3)$ # $a_3 *= d_3$

$a_3 /= \text{keep_prob}$ → Inverted dropout technique

: dropout을 적용하기 전과 동일하게 활성화 값의 기대값으로 맞춰주기 위해 노드를 삭제후 같은 활성화 값에 keep_prob (삭제하지 않을 확률)을 나온다.

⊕ test time에서는 dropout을 진행하지 않는다.

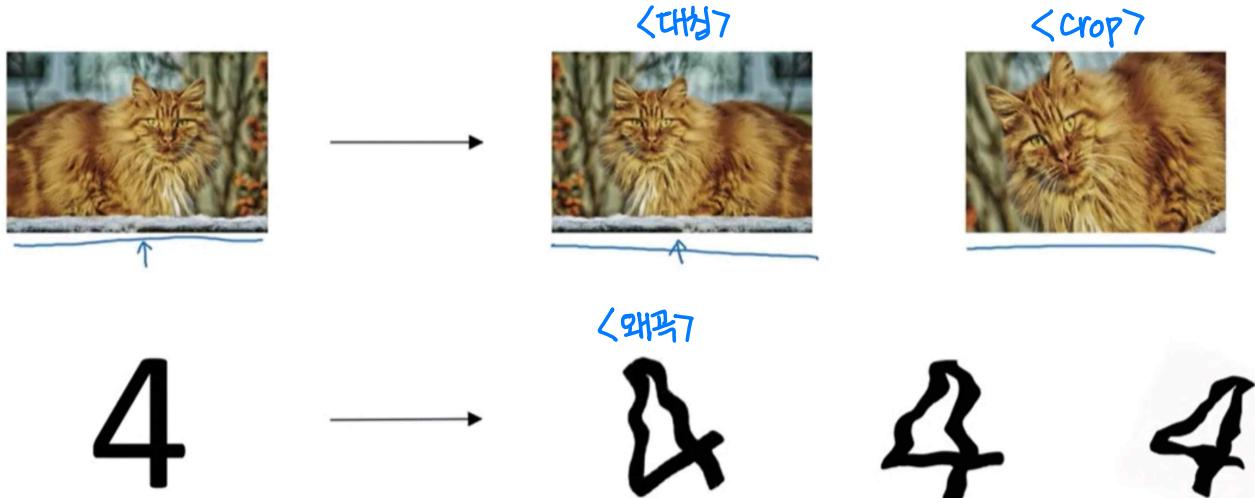
< Understanding Dropout >

- 드롭아웃은 랜덤으로 노드를 삭제하기 때문에, 하나의 특성에 의존하지 못하게 만들면서 기종차를 다른곳에 보낸다
- 드롭아웃의 keep_prob 확률은 층마다 다르게 설정 가능
- 비용함수가 단조 감소인지 우선 확인한 후, 드롭아웃을 사용해야 함

<Other Regularization Methods>

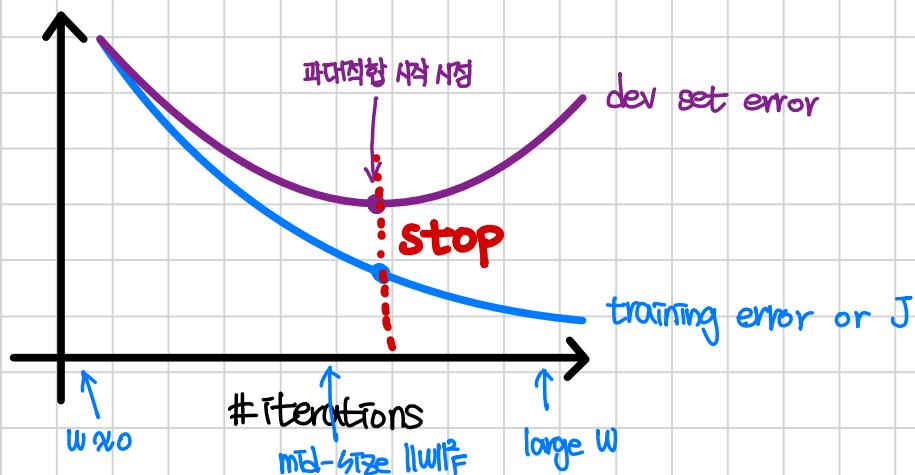
* Data augmentation

Data augmentation



- 이미지의 경우, 더 많은 트레이닝셋을 사용하여 과대적합을 해결할 수 있음
- 대침, 확대, 왜곡, 회전 등을 이용하여 새로운 데이터 생성
- 이런 가짜이미지는 원래의 새로운 생플보다 더 적은 정보를 추가하지만, 비용이 들지 않는 장점

* Early stopping



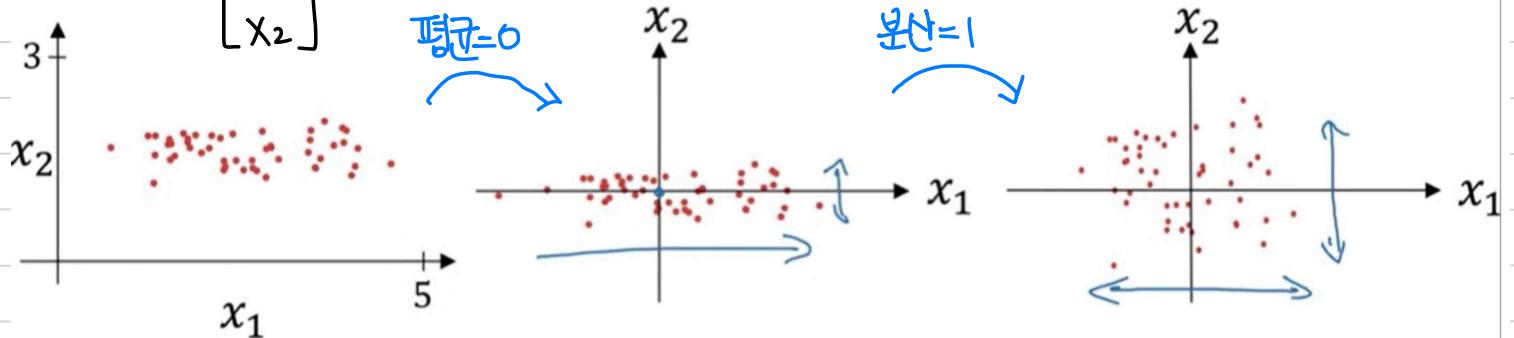
- early stopping: 신경망이 dev set의 오차 저점 부근, 즉 가장 잘 작동하는 점에서 훈련 끝
- 단점: training 시. ① 비용 함수 미적분 ② 과대적합 하지 않도록 이 두 가지는 별개의 일 (Orthogonalization) 이므로 다른 접근법을 사용해야 하지만, early stopping은 두 가지를 뒤에서 미적분의 조건을 못 찾을 수도 있음.

[1-3. Setting Up your Optimization Problem]

< Normalizing Inputs >

* Normalizing training sets

$$\text{ex) } \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$



① 평균을 0으로 만든다

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

$$x := x - \mu$$

② 분산을 1로 만든다.

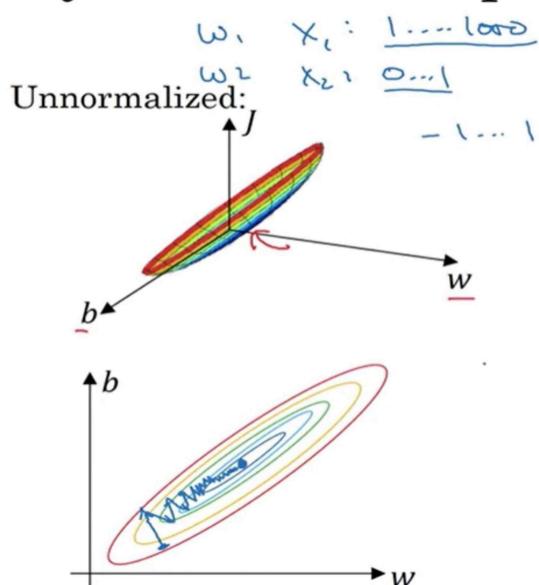
$$\sigma^2 = \frac{1}{m} \sum_{i=1}^m (x^{(i)})^2$$

$$x := \frac{x}{\sigma^2}$$

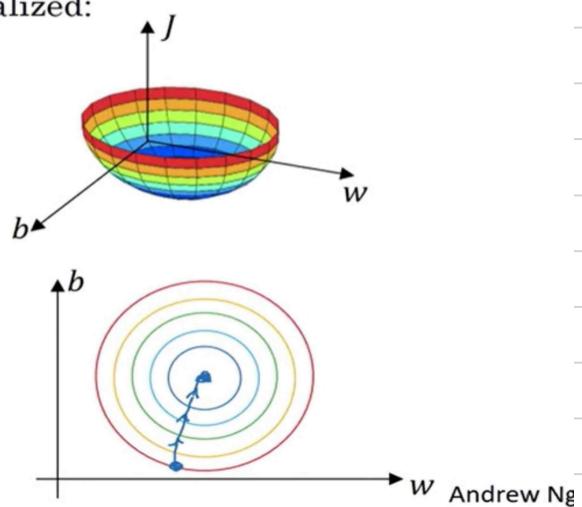
- 테스트 셋을 정규화할 때, train set에 사용한 μ, σ 를 사용해야 함

Why normalize inputs?

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

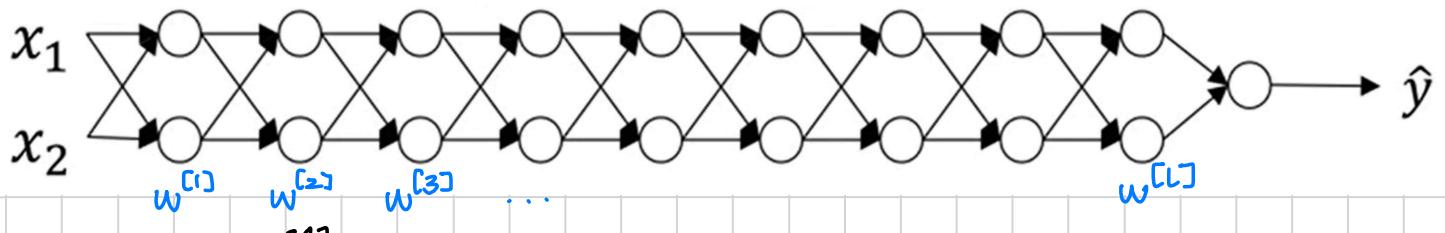


Normalized:



→ 정규화를 통해 비율항수의 영향이 더 둻끌고 확장화하기 때문에 학습 알고리즘이 빨리 실행됨

<Vanishing / Exploding Gradients>



$$\text{if } g(z) = \sum b^{[l]} = 0$$

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

① $w = 1.5E \rightarrow \underbrace{w^{[l]}}_{1.5^{[L-1]} E x} > I$

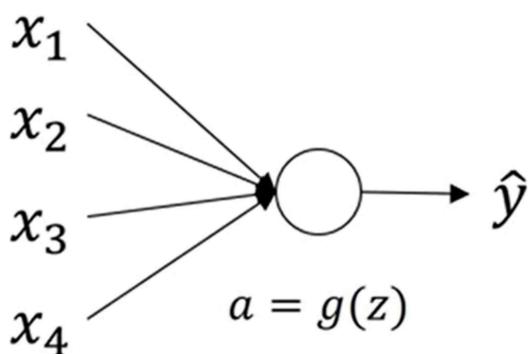
: $\hat{y} = w^{[L]} \cdot 1.5^{[L-1]} E x \rightarrow$ 신경망이 급격히 \hat{y} 가 과정적으로 "커짐"

② $w = 0.5E \rightarrow \underbrace{w^{[l]}}_{0.5^{[L-1]} E x} < I$

: $\hat{y} = w^{[L]} \cdot 0.5^{[L-1]} E x \rightarrow$ 신경망이 급격히 \hat{y} 가 과정적으로 "작아짐"

<Weight Initialization for Deep Networks>

Single neuron example



$$z = w_1 x_1 + w_2 x_2 + \dots + w_n x_n + b$$