

Practical Aspect of Deep Learning

Setting up ML application

1. Train/Dev/Test sets

training set	dev set	test set
--------------	---------	----------

- Hold-out cross validation
- Development set "dev"

Previous: 100 or 1000 or 10,000 data

train 70%/ test 30% or train 60%/ dev 20%/ test 20%

↓

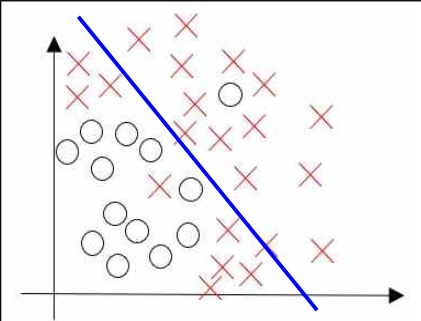
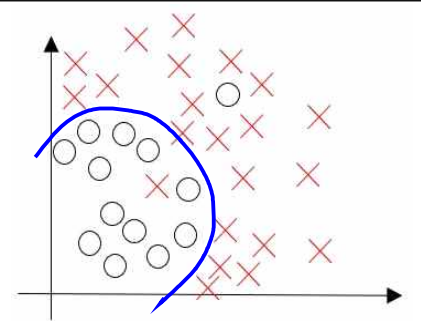
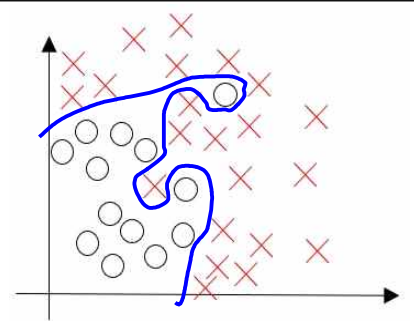
Big Data: 10,000 or 100,000 or 1,000,000 or more data

train 98%/ dev 1%/ test 1% or train 99.5%/ dev 0.4%/ test 0.1%

※ Make sure dev and test sets come from same distribution.

Not having a test set might be okay.(Only dev set)

2. Bias/Variance

		
high bias underfitting	"just right"	high variance overfitting

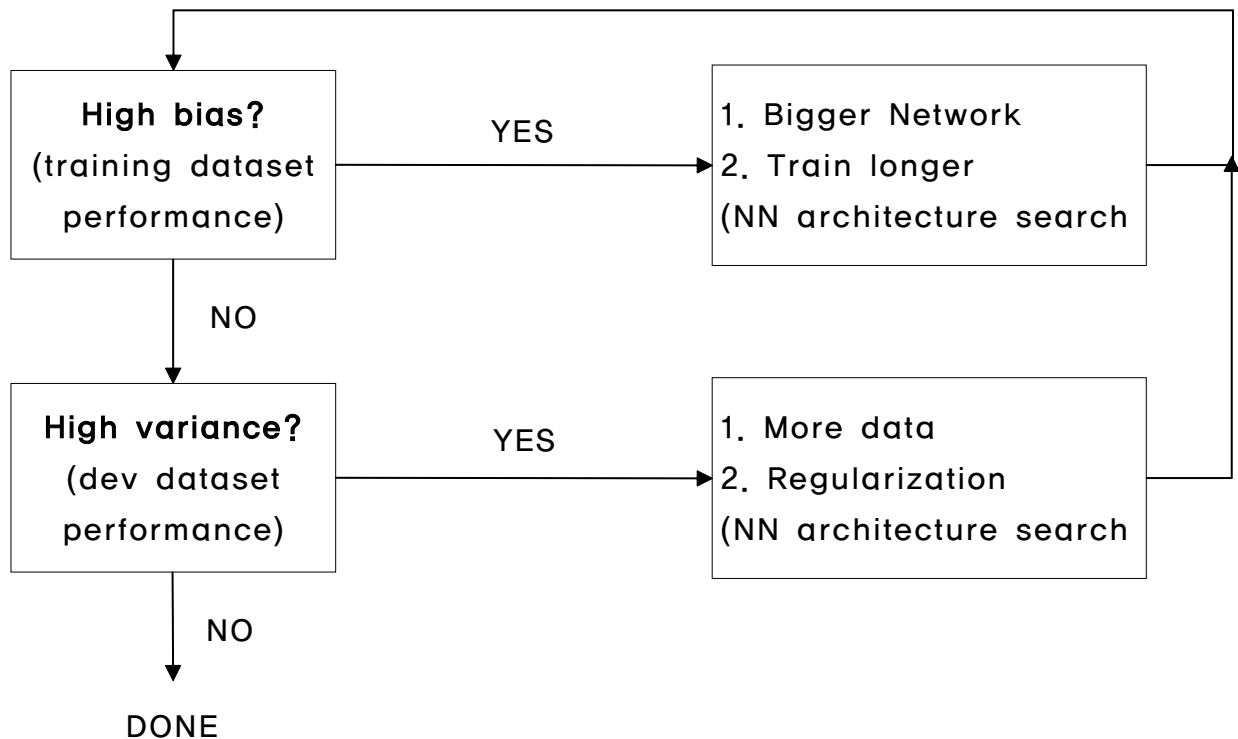
Train set error	1%	15%	15%	0.5%
Dev set error	11%	16%	30%	1%
	high variance	high bias	high bias & high variance	low bias & low variance

Optimal (Bayes) error: $P(Y|X)$ 에 대한 확률 분포를 안다고 가정했을 때

이론적으로 도달할 수 있는 최소 error

$$Bayes\ Error = \int \sum_y \min [P(y_1|x), P(y_2|x)] P(x, y) dx$$

Basic "Recipe" for Machine Learning



※ Bias 와 Variance는 약간의 trade-off 관계이다.

Regularizing neural network

1. Regularization

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

→ b 에 대한 규제도 넣어도 된다. But W 보다 b 의 차원($n^{[l]}, 1$)은 매우 작아 무시할 수 있다.

$$L_2 \text{ Regularization: } \|W\|_2^2 = \sum_{j=1}^{n_x} w_j^2 = W^T W$$

$$L_1 \text{ Regularization: } \frac{\lambda}{2m} \sum_{j=1}^{n_x} |w_j| = \frac{\lambda}{2m} \|W\|_1$$

→ W will be sparse : L_1 Regularization를 하면 W 의 많은 요소가 0으로 변한다.
 W 의 요소가 압축되면서 성능이 좋아진다는 의견도 있지만 실제로 그런지는 확실하진 않다.

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

$$W^{[l]}: (n^{[l]}, n^{[l-1]}) \rightarrow \|W^{[l]}\|_F^2 = \sum_{i=1}^{n^{[l]}} \sum_{j=1}^{n^{[l-1]}} (w_{i,j}^{[l]})^2 \rightarrow$$

L_2 Regularization == Frobenius Norm == "Weight Decay"

$$dW^{[l]} = (\text{from backprop}) + \frac{\lambda}{m} W^{[l]}, \quad \frac{\partial J}{\partial W^{[l]}} = dW^{[l]}$$

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

$$\begin{aligned} W^{[l]} &:= W^{[l]} - \alpha \left((\text{from backprop}) + \frac{\lambda}{m} W^{[l]} \right) \\ &= \left(1 - \frac{\alpha \lambda}{m} \right) W^{[l]} - \alpha (\text{from backprop}) \\ \left(1 - \frac{\alpha \lambda}{m} \right) &< 1 \end{aligned}$$

How does regularization prevent overfitting?

$W^{[l]} \approx 0 \Rightarrow \lambda$ 가 커질수록 W 는 0에 가까워지도록 업데이트된다.

그러면서 신경망은 심플해지고 작아진다. high variance \rightarrow high bias

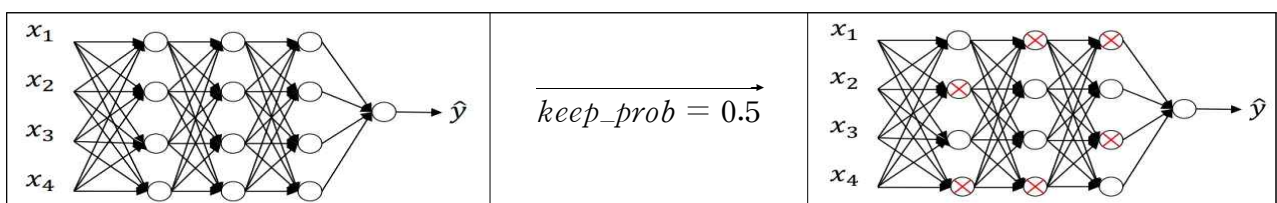
※ hidden units이 사라지는 것이 아니라 효과가 줄어드는 것이다. W 가 0이 되는 것은 아니기 때문이다.

$$\lambda \uparrow \rightarrow W^{[l]} \downarrow \rightarrow Z^{[l]} \downarrow = W^{[l]} \downarrow a^{[l-1]} = b^{[l]}$$

모든 레이어가 선형 레이어처럼 변한다. 선형 함수에만 의미 있고 비선형 의사결정은 불가능.

activation function이 tanh일때 상대적으로 선형 함수가 될 것이다. 전체 신경망의 산출값은 선형 함수의 산출값과 크게 다르지 않을 것이다. 비선형 함수보다는 선형 함수에 가까워 더 심플하므로 훨씬 덜 overfitting하는 것이다.

2. Dropout



Implementing dropout: "Inverted Dropout"

illustrate with layer $l=3$, $keep_prob = 0.8$

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

$$a3 = np.multiply(a3, d3) \quad \# a3 = a3 * d3: d3 \text{는 boolean이지만}$$

$$a3 = a3 / keep_prob \quad np.multiply \text{는 boolean을 0과 1로 바꾸어 계산한다.}$$

50 units 中 10 units(20%)는 비활성화

$z^{[4]}$ 의 기대값이 감소하지 않기 위해 $a3 = a3 / keep_prob$ 을 해줘야 한다. $a3$ 의 기대값이 변하지 않으려면 조정하거나 dropout할 $(1 - keep_prob)$ 확률만큼 올라야하기 때문이다. 이를 Inverted Dropout 테크닉이라고 한다.

효과: $keep_prob$ 을 어떻게 설정하더라도 a 에 $keep_prob$ 를 나눠줘서 a 의 기대값이 동일하게 유지된다. 이 효과로 scale 문제가 덜 하기 때문에 test를 쉽게 해준다.

Making predictions at test time

You do not apply dropout (do not randomly eliminate units) and do not keep the $1/keep_prob$ factor in the calculations used in training

테스트할 때는 dropout을 사용하지 않는다.

이미 학습된 가중치에 노이즈가 되기 때문이다.

$1/keep_prob$ 의 역dropout 효과는 테스트에서 dropout을 구현하지 않아도 된다.

activation 기대값의 크기는 변하지 않기 때문에 테스트할 때 스케일링 매개 변수를 추가하지 않아도 된다.

Why does drop-out work?

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

→ weights가 줄어든다

특정 입력에 모든 것을 걸지 않기 때문에 특정 입력에 유난히 큰 가중치를 부여하지 않고 입력 각각으로 가중치를 분산한다. 분산하여 가중치의 노름의 제공값이 줄어든다.

효과: 가중치를 줄이는 것이고 과대적합을 막는다.

L_2 규제와 비슷한 효과를 보여주지만 L_2 규제가 다른 가중치에 적용된다는 것과 서로 다른 크기의 입력에 더 잘 적응한다는 것이 다르다.

Increasing the parameter $keep_prob$

1. Reducing the regularization effect

2. Causing the neural network to end up with a lower training set error

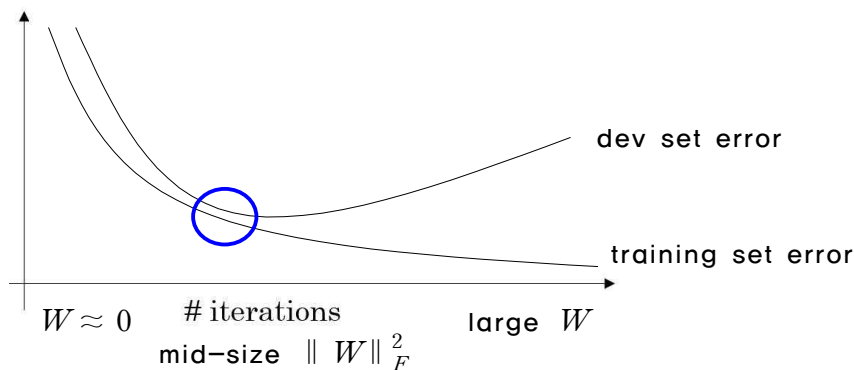
3. Data Augmentation

: 데이터 셋 확장 → 가짜 추가 트레이닝 샘플 만들기

1. 가로로 뒤집기
2. 회전하고 줄인
3. 찌그리거나 변형주기

이러한 가짜 샘플은 완전 새로운 데이터를 추가하는 것보다는 도움이 되지 않는다. 하지만 무료이기 때문에 사용하고 일반화를 통해 과대적합을 결과적으로 줄일 수 있다. 가로로 뒤집힌 고양이를 보더라도 고양이라고 분류할 수 있을 것이다. 하지만 거꾸로 서 있는 고양이를 인식하고 싶진 않기 때문에 세로로 뒤집지는 않는다.

4. Early Stopping



- Optimize cost function $J(W, b)$: Gradient descent, ...
- Not overfit: Regularization, ...

Early Stopping을 하면 이 두개의 별개 문제를 단독으로 풀 수 없어진다.

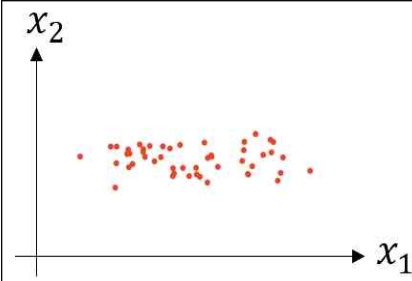
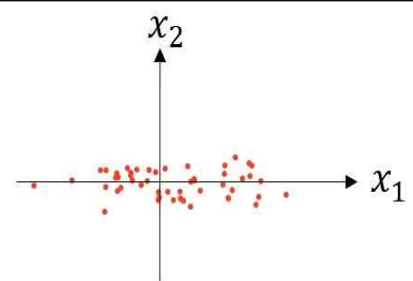
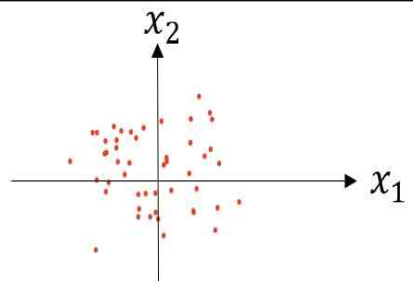
not overfit하기 위해 optimize cost function $J(W, b)$ 를 하는 도중에 stop하게 된다. 두 방법을 다른 도구로 하는 것이 아니라 한가지 도구에 두가지가 약간 섞인 것이기 때문이다.

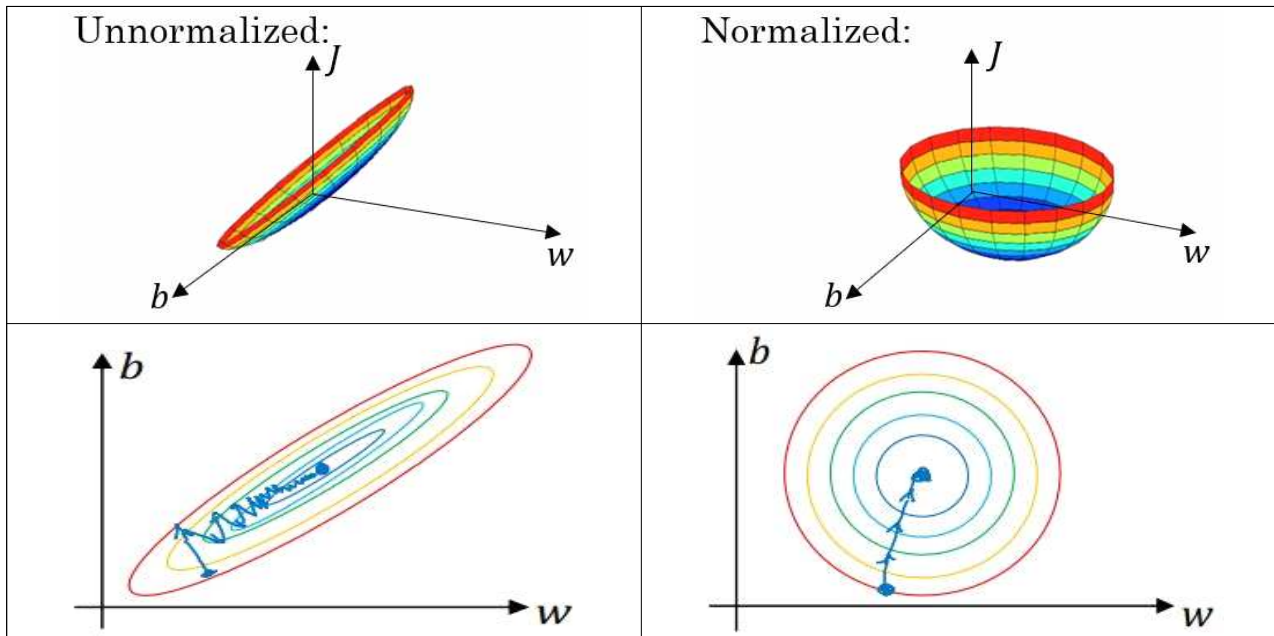
early stopping 대신 L_2 규제를 사용하는 방법이 있다. 이 경우 신경망을 최대한 길게 학습시키면 된다. 하이퍼 파라미터의 서치 범위를 더 분해시켜서 서치하기 쉽다. 하지만 다양한 일반화와 람다 값들을 시도해봐야 해야 한다.

early stopping의 장점: L_2 의 λ 을 시도해볼 필요도 없이 gradient descent를 한 번만 실행해도 여러 크기의 W 를 한 번에 시도해 볼 수 있다.

Setting up optimization problem

1. Normalizing inputs

					
subtract mean: $\mu = \frac{1}{m} \sum_{i=1}^m x^{(i)}$ $X := X - \mu$		normalize variance: $\sigma^2 = \frac{1}{m} \sum_{i=1}^m (x^{(i)})^2$ $X = X / \sigma^2$			
		use same μ and σ^2 to normalize test set.			



정규화를 사용하면 gradient descent를 빠르게 할 수 있다.

왼쪽 사진처럼 왔다 갔다 하지 않아도 큰 스텝으로 전진할 수 있다.

정규화는 어떠한 손해도 없기 때문에 데이터의 스케일 차이가 크든 작든 하는 것을 추천.

2. Vanishing/ Exploding gradient

$$\begin{aligned} \text{if } g(z) = z \text{ and } b^{[l]} = 0, \hat{y} &= (W^{[L]}(W^{[L-1]} \dots (W^{[2]}(W^{[1]}X)))) \\ &= (W^{[L]}(W^{[L-1]} \dots (W^{[2]}(a^{[1]})))) \\ &= (W^{[L]}(W^{[L-1]} \dots (a^{[2]}))) \\ &\vdots \end{aligned}$$

$$W^{[l]} > I \rightarrow \text{exploding gradient}$$

$$W^{[l]} < I \rightarrow \text{Vanishing gradient}$$

3. Weight Initialization for deep neural networks

$z = w_1x_1 + w_2x_2 + \dots + w_nx_n$: large $n \rightarrow$ smaller w_i

① ReLU의 경우: He initialization

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

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

② tanh의 경우: Xavier initialization

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

$$\textcircled{3} \quad W^{[l]} = np.random.randn(shape) * np.sqrt\left(\frac{2}{n^{[l-1]} + n^{[l]}}\right)$$

만약 activation의 입력 특성이 대략 $mean = 0$ 과 $variance = 1$ 이면 이는 Z 와 비슷한 scale을 갖게 할 것이므로 문제가 해결되지 않을 것이다. 하지만 vanishing과 exploding gradient 문제를 도와주는 한다. 왜냐하면 W 를 1보다 너무 크지 않고 1보다 너무 작지 않게 초기화해서 너무 빨리 explode하거나 vanish하지 않게 한다.

4. Numerical approximation of gradients

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

two-side differential을 gradient checking과 backward propagation에 사용하는 경우 one-side differential보다 2배 정도 느리나 훨씬 더 정확도가 높아 two-side를 쓰는 것을 추천한다. one-side: $error = O(\epsilon)$, $\epsilon < 1$ 가 two-side: $error = O(\epsilon^2)$, $\epsilon < 1$ 보다 훨씬 크다.

5. Gradient Checking

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

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

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

\Rightarrow Is $d\theta$ the gradient of $J(\theta)$?

for each i :

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

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

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

$$\text{check } \epsilon = 10^{-7}, \frac{\|d\theta_{approx} - d\theta\|_2}{\|d\theta_{approx}\|_2 + \|d\theta\|_2} \approx 10^{-7} \text{ 이면 great}$$
$$\approx 10^{-5}$$
$$\approx 10^{-3} \text{이면 worry}$$

forward prop하고 backward prop하고 grad check하여 버그가 있는지 확인한다.
 10^{-3} 정도 보다 크면 버그가 있을 가능성이 높다.

Gradient Checking implementation notes

1. Don't use in training - only to debug

매 학습마다 grad check하면 너무 느려진다.

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

3. Remember regularization.

4. Doesn't work with dropout.

dropout 없이 알고리즘이 옳은지 grad check를 하고

학습할 때 dropout을 다시 켜는 것을 권장한다.

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

만약 W 와 b 가 0에 가깝다면 gradient descent가 옳을 수 있다.

따라서 W 와 b 를 0에서 멀어지게 함으로써 해결할 수 있다.