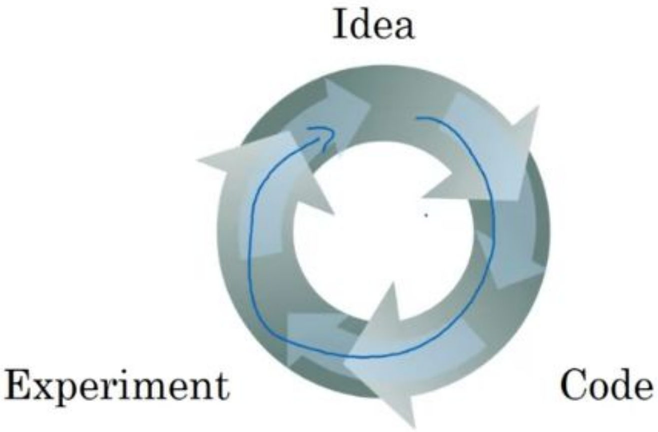
Improving Deep Neural Networks: Hyperparameter Tuning, Regularization and Optimization

27/03/2021 Minsung Kim

Setting up your machine learning application

Applied ML is a highly iterative process

- # layers
- # hidden units
- learning rates
- activation functions
- ...



NLP, Vision, Speech, Structural data

Ads   Search   Security   logistic ....

Setting up your machine learning application

Test/dev/test sets

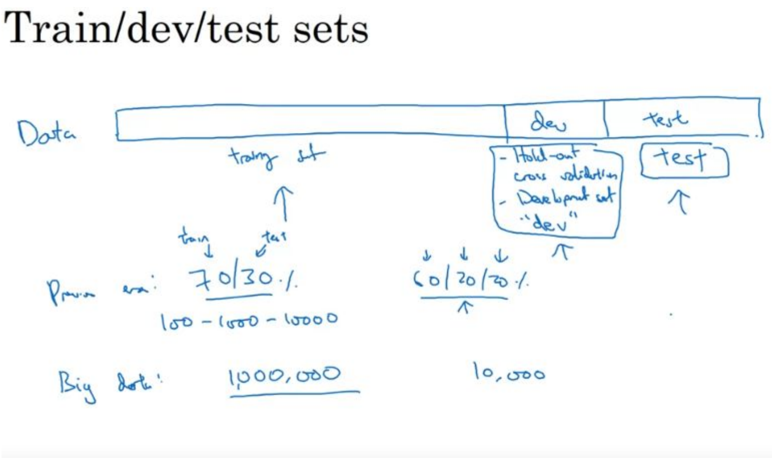
Previous era: 70/30 or 60/20/20 - 데이터의 양이 작았기에

Modern era: dev/test -> 데이터가 많아서 굳이 이전처럼 나눌필요 없이 적당히

ex) total 1,000,000 -> Dev 10,000 Test 10,000

98/1/1

99/0.5/0.5



Setting up your machine learning application

Mismatched train/test distribution

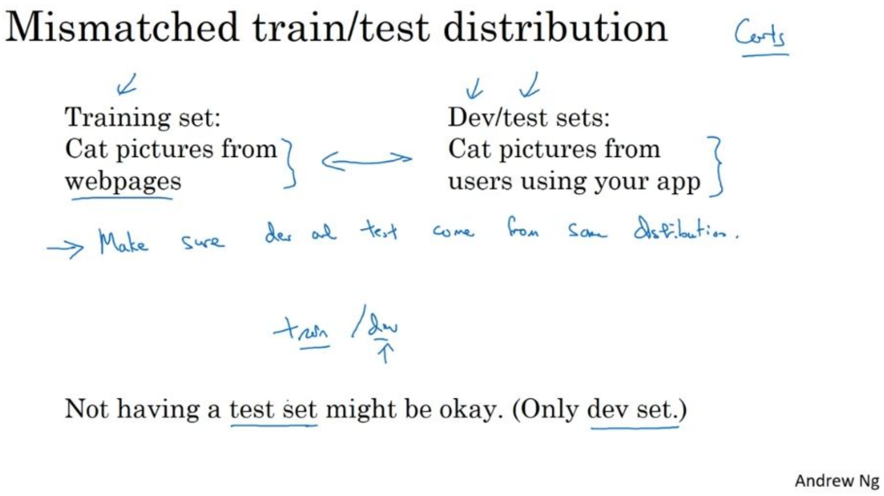
Training vs. Dev/Test

서로 다른 분포에서 나오는 데이터들을 다룸.

Test와 Dev는 같은 분포에서 나오도록 할것.

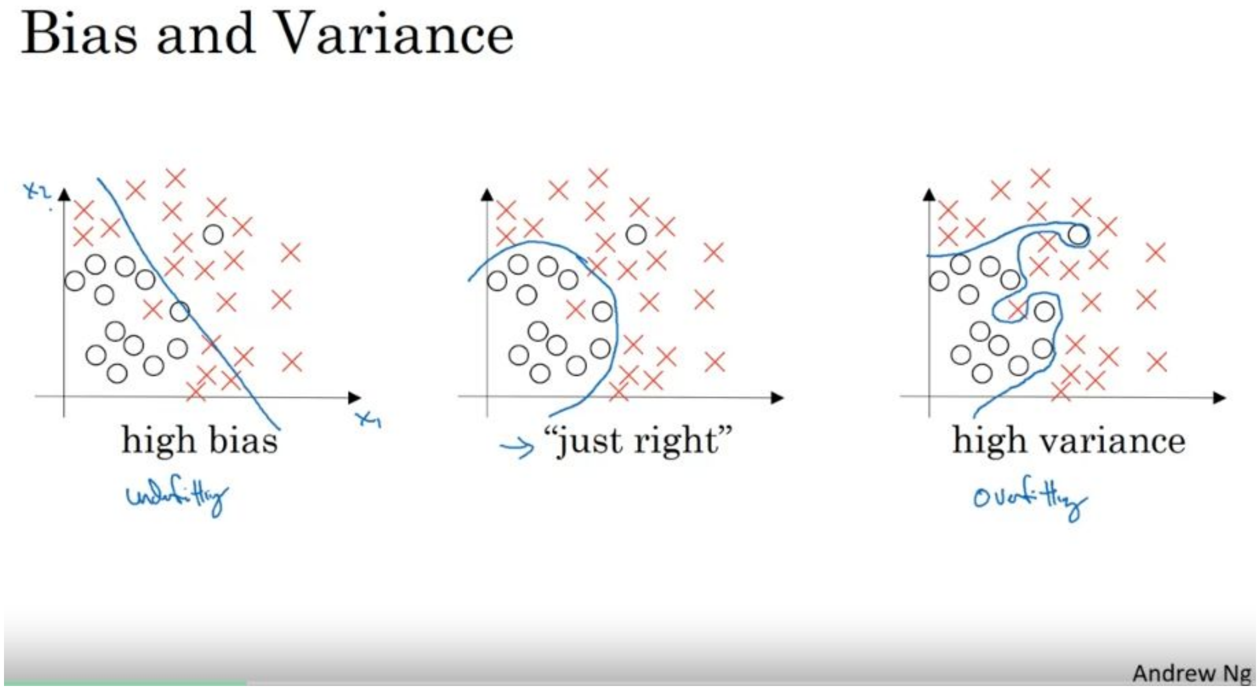
Training은 상관없나?

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



Setting up your machine learning application

Bias/Variance



Setting up your machine learning application

Bias/Variance

Bias and Variance

Cat classification

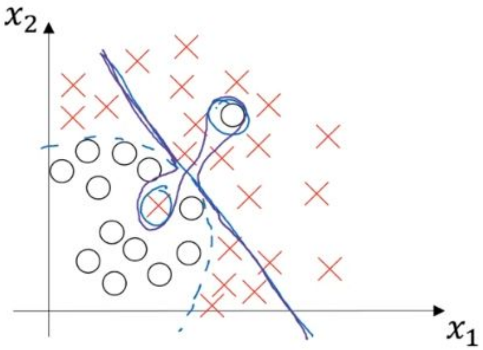
$y=1$

$y=0$

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

Human: ≈0%

Optimal (Bayes) error: ~~≈0%~~ 15%   Blurry images

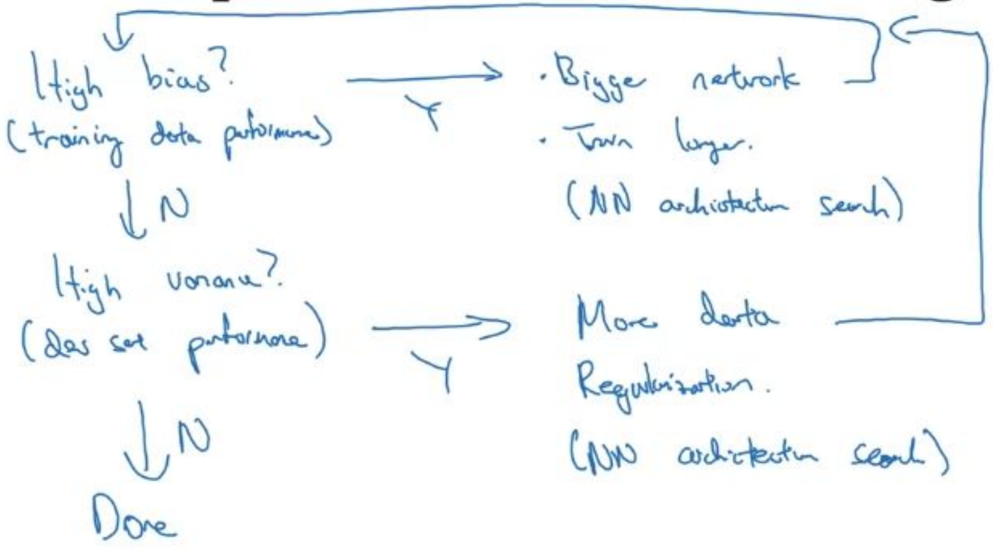


High bias and high variance

Andrew Ng

Setting up your machine learning application

Basic recipe for machine learning



Setting up your machine learning application

Basic recipe for machine learning

High bias?
(training data performance)

Y →
→ $\boxed{\text{Bigger network}}$
→ Train longer.
(NN architecture search)

↓ N

High variance?
(dev set performance)

Y →
→ $\boxed{\text{More data}}$
→ Regularization
(NN architecture search)

↓ N

Done

Bias variance "tradeoff"

Regularizing your neural network

Logistic regression

$w \in \mathbb{R}^{n_x}, \quad b \in \mathbb{R}$

λ = regularization parameter

lambda lambd

$$\min_{w,b} J(w,b)$$

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

omit

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

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

w will be sparse

Regularizing your neural network

Neural network

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

"Frobenius norm"

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

Regularizing your neural network

Gradient descent?

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

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

Before regularization

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

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

$$\left(1 - \frac{\alpha\lambda}{m}\right)$$

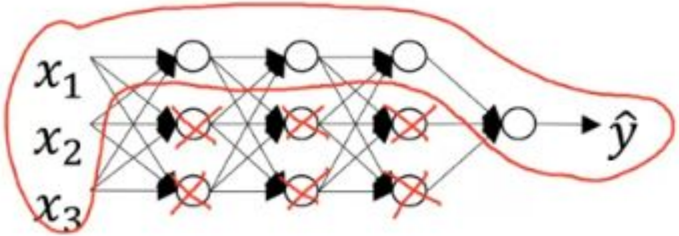
After regularization

Regularizing your neural network

Regularization은 왜 오버피팅을 감소시킬까?

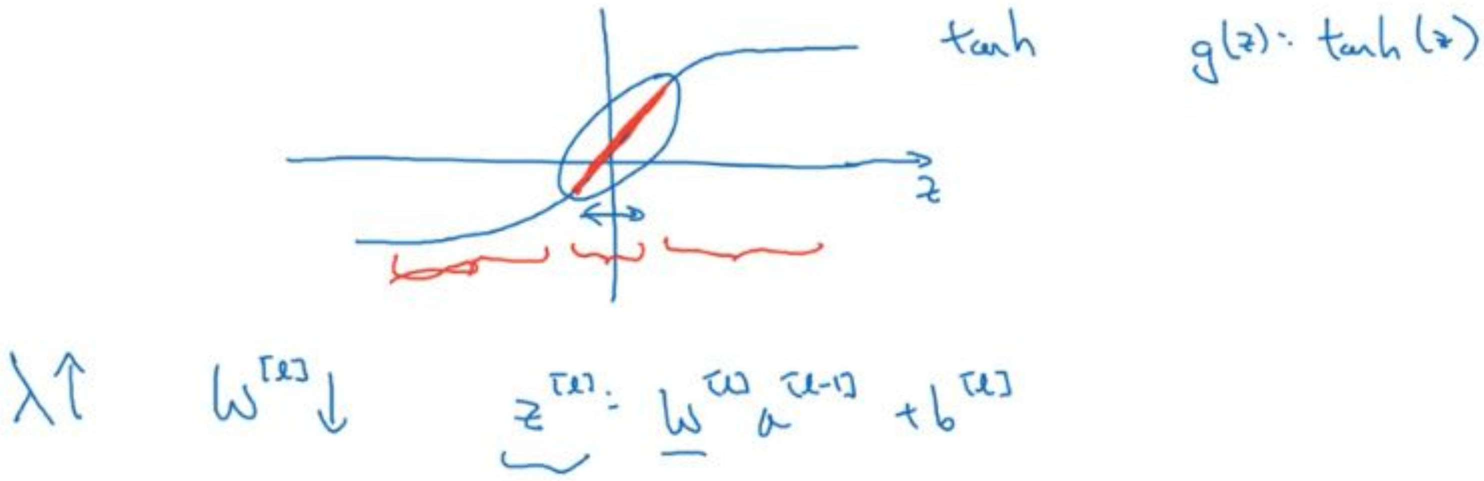
$$J(w^{[l]}, b^{[l]}) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\sum_{l=1}^{L} \underbrace{\|w^{[l]}\|_F^2}$$

$$w^{[l]} \approx 0$$



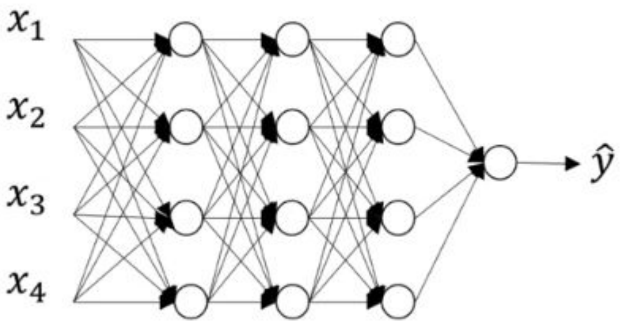
Regularizing your neural network

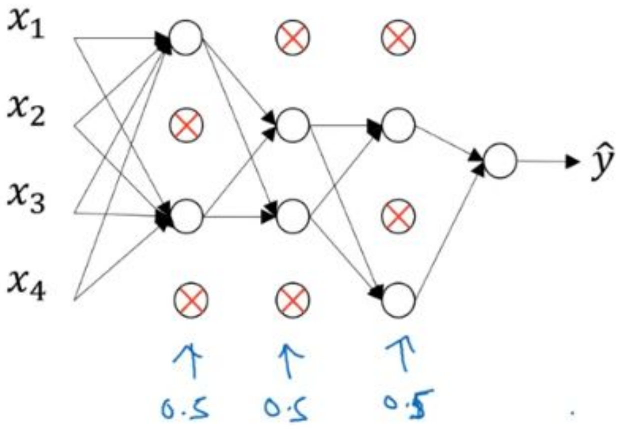
Regularization은 왜 오버피팅을 감소시킬까?



Regularizing your neural network

Dropout regularization





Andrew Ng

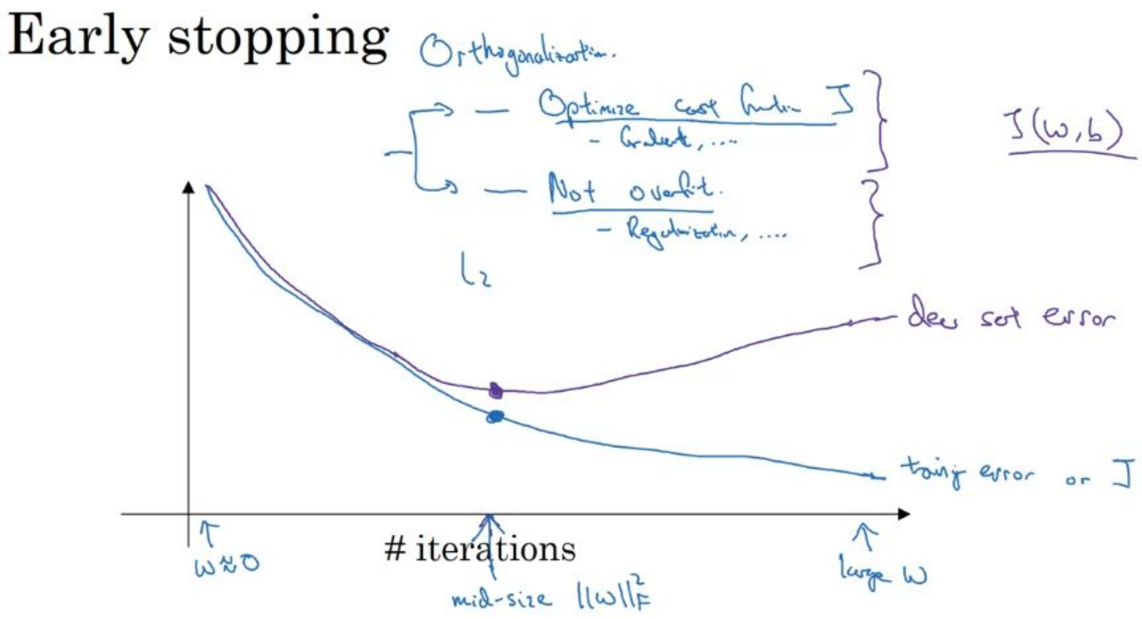
Regularizing your neural network

다른 Regularization 방법들

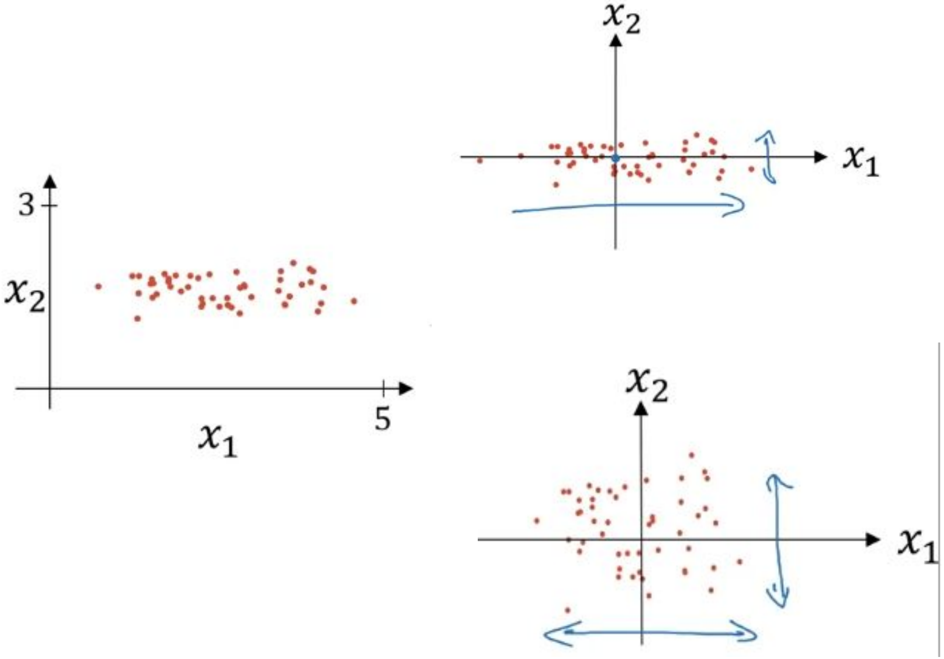
Data augmentation

Regularizing your neural network

다른 Regularization 방법들



Setting up your optimization problem



Subtract mean:

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

$$x := x - \mu$$

Normalize variance

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

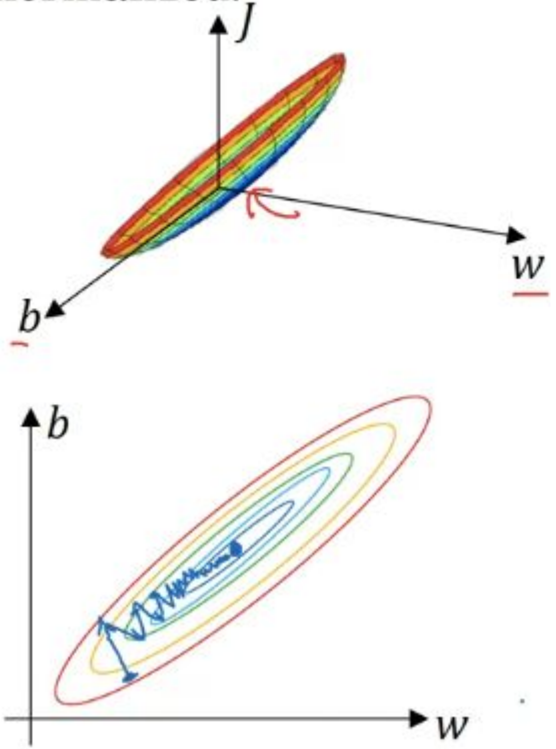
← element-wise

$$\frac{x - \mu}{\sigma}$$

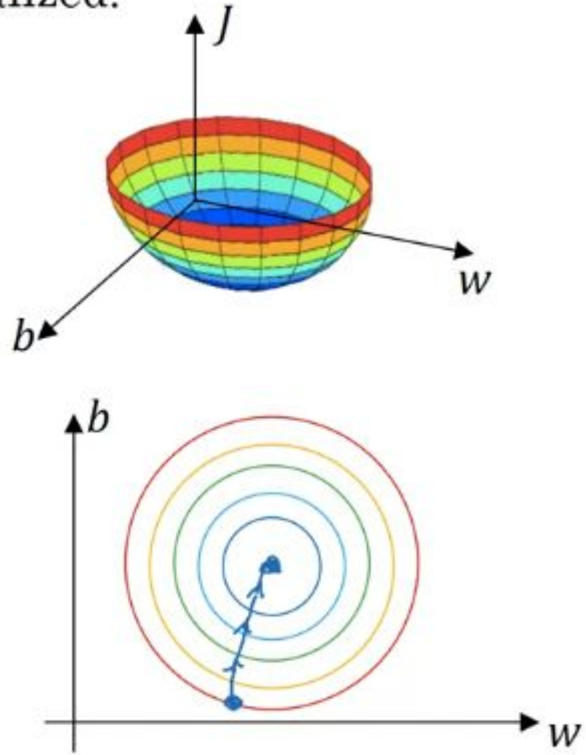
Setting up your optimization problem

w_2 x_2, 0...1

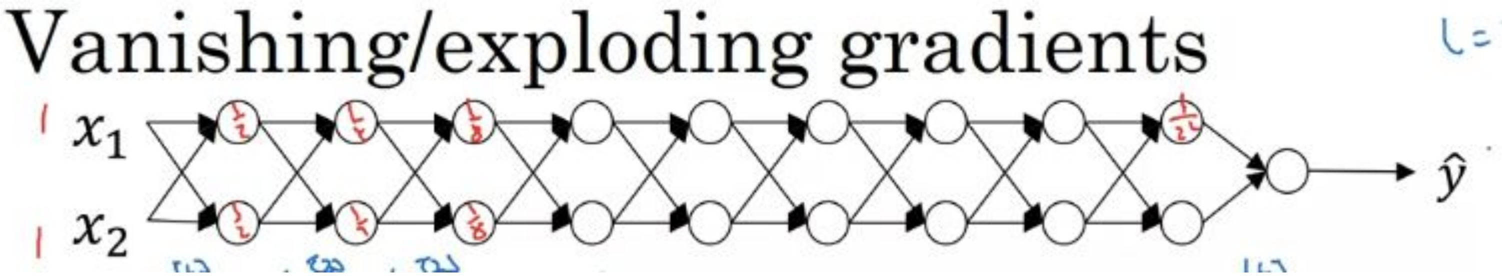
Unnormalized:



Normalized:



Setting up your optimization problem



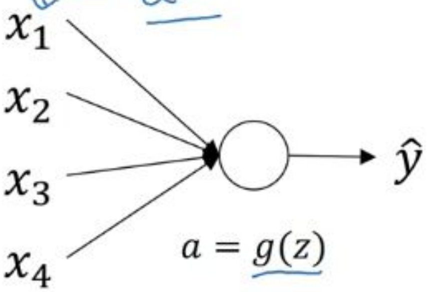
$W^{[l]} > I$ -> W 커지면서 Exploding

$W^{[l]} < I$ -> W가 작아지면서 Vanishing

Setting up your optimization problem

Single neuron example

$a^{[l-1]}$



$w^{[l]}$

$z = w_1x_1 + w_2x_2 + \cdots + w_nx_n \cancel{+b}$

Large n $\rightarrow$ Smaller w_i

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

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

ReLU $\quad g^{[l]}(z) = \text{ReLU}(z)$

Other variants:

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

Xavier initialization

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

Setting up your optimization problem

Checking your derivative computation

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

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

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

Is $d\theta$ the gradient of J

Setting up your optimization problem

for each i:

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

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

$$d\theta_{approx} \overset{?}{\approx} d\theta$$

Check $\rightarrow \dfrac{\| d\theta_{approx} - d\theta \|_2}{\| d\theta_{approx} \|_2 + \| d\theta \|_2}$ $\approx 10^{-7}$ – great!

10^{-5}

$\rightarrow 10^{-3}$ – worry.

$\underline{\varepsilon = 10^{-7}}$

Setting up your optimization problem

Gradient checking implementation notes

- Don't use in training – only to debug

$d\theta_{approx}[i] \longleftrightarrow d\theta[i]$

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

$db^{[l]}$ $dw^{[l]}$

- Remember regularization.

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

$d\theta$ = grad of J w.r.t. θ

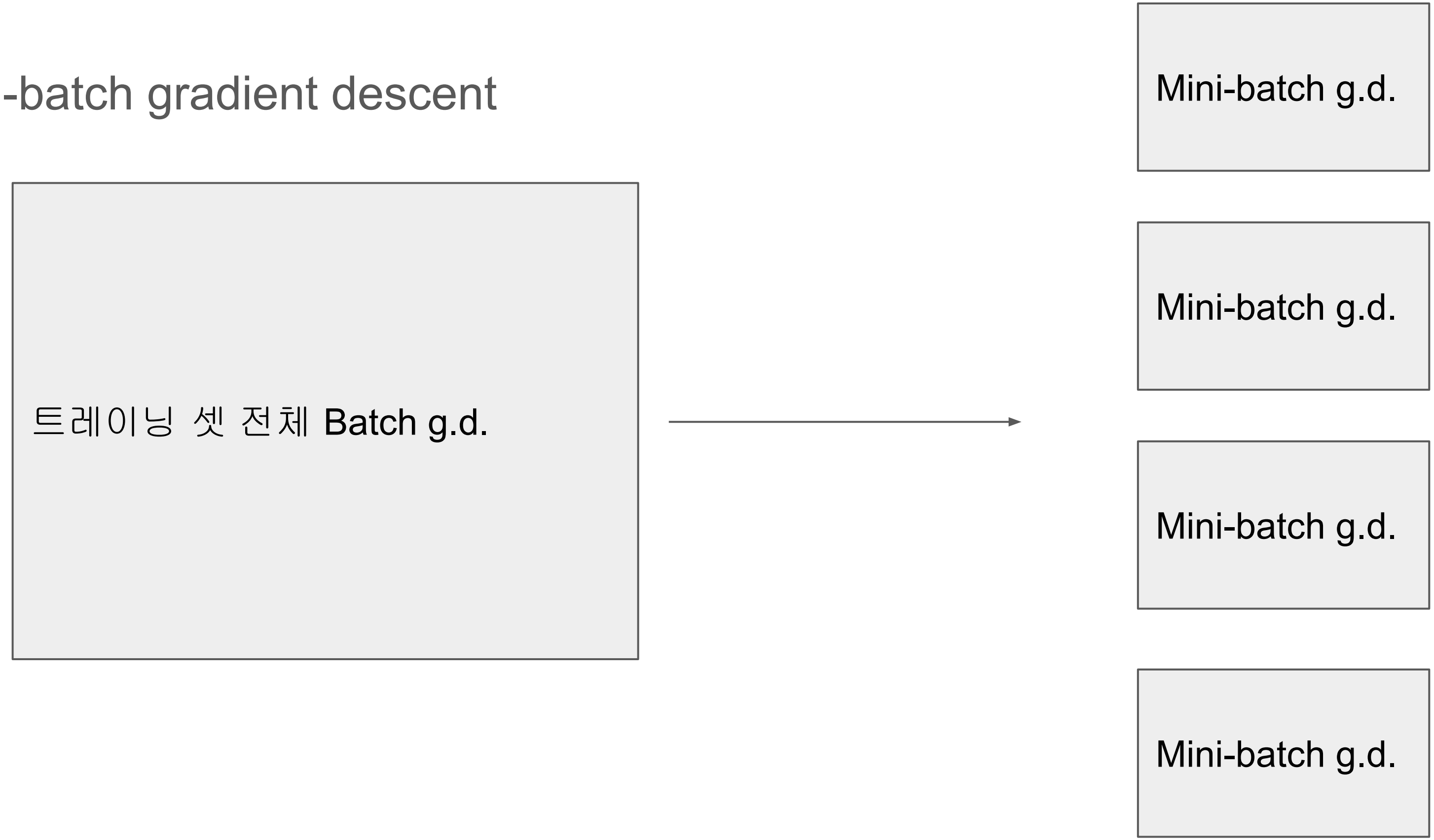
- Doesn't work with dropout. J keep-prob = 1.0

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

Andrew Ng

Optimization algorithms

Mini-batch gradient descent



Optimization algorithms

Mini-batch gradient descent

repeat {
for $t = 1, \ldots, 5000$ {

Forward prop on $X^{\{t\}}$.

$$Z^{[1]} = W^{[1]} X^{\{t\}} + b^{[1]}$$

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

$$\vdots$$

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

Vectorized implementation (1000 examples)

1 step of grad descent using $X^{\{t\}}, Y^{\{t\}}$. (as if m=1000)

X, Y

Compute cost $J^{\{t\}} = \frac{1}{1000} \sum_{i=1}^{l} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2 \cdot 1000} \sum_{l} \|W^{[l]}\|_F^2$.

from $X^{\{t\}}, Y^{\{t\}}$.

Backprop to compute gradients wrt $J^{\{t\}}$ (using $(X^{\{t\}}, Y^{\{t\}})$)

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

}

}

"1 epoch"

pass through training set.

Andrew Ng

Optimization algorithms

Mini-batch gradient descent

repeat { for $t = 1, \dots, 5000$ {

Forward prop on $X^{\{t\}}$.

$$\left.\begin{aligned} Z^{[1]} &= W^{[1]} X^{\{t\}} + b^{[1]} \\ A^{[1]} &= g^{[1]}(Z^{[1]}) \\ &\vdots \\ A^{[L]} &= g^{[L]}(Z^{[L]}) \end{aligned}\right\} \text{Vec}$$

Compute cost $J^{\{t\}} = \frac{1}{1000} \sum_{i=1}^{l} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2 \cdot 1000} \sum_{l} \|W^{[l]}\|_F^2$.

Backprop to compute gradients wrt $J^{\{t\}}$ (using $(X^{\{t\}}, Y^{\{t\}})$)

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

}

}

1 step of gradient descent using $X^{\{t\}}, Y^{\{t\}}$. (as if $m = 1000$)

X, Y

"1 epoch" – pass through training set.

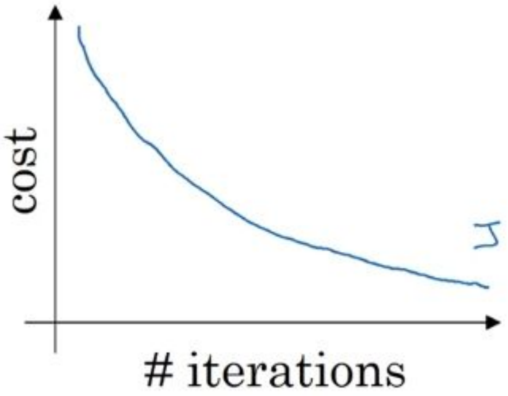
- Epoch means one pass over the full training set
- Batch means that you use all your data to compute the gradient during one iteration.
- Mini-batch means you only take a subset of all your data during one iteration.

Andrew Ng

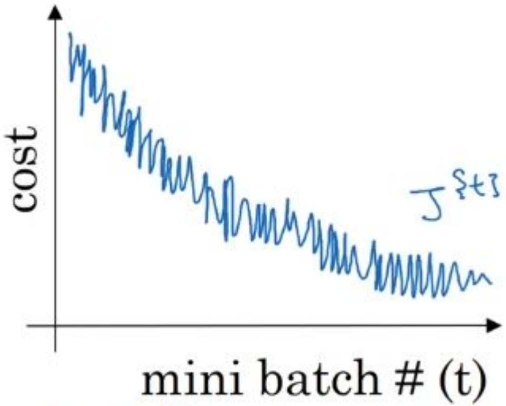
Optimization algorithms

Training with mini batch gradient descent

Batch gradient descent



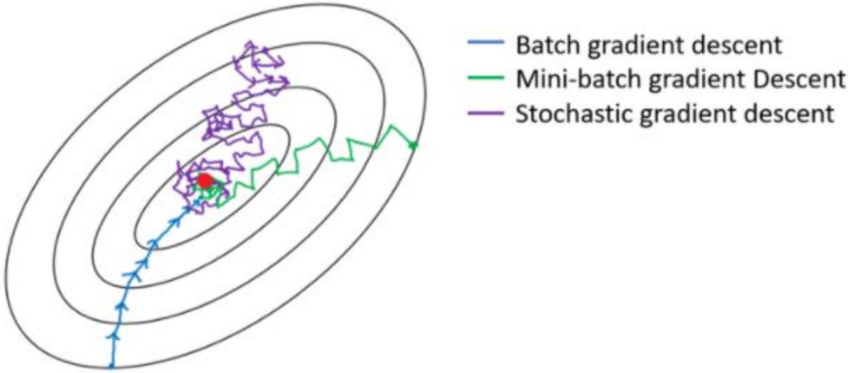
Mini-batch gradient descent



Plot $J^{\{t\}}$ computed using $X^{\{t\}}, Y^{\{t\}}$

Andrew Ng

Optimization algorithms



Stochastic gradient Descent

Lose speedup from vectorization

In-between (mini-batch size not too big/small)

Fastest learning.

- Vectorization. (~1,000)

Batch gradient descent (mini-batch size = m)

Too long per iteration

Typical mini-batch sizes:

$\underbrace{\underset{2^6}{64}, \underset{2^7}{128}, \underset{2^8}{256}, \underset{2^9}{512}}$ $\quad \underset{2^{10}}{1024}$

Make sure mini-batch fit in CPU/GPU memory.

$X^{\{t\}}, Y^{\{t\}}$.

Optimization algorithms

Exponentially weighted averages

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

$\beta = 0.9$: ≈ 10 days' temperature.

$\beta = 0.98$: ≈ 50 days

V_t as approximately average over $\approx \frac{1}{1-\beta}$ days' temperature.

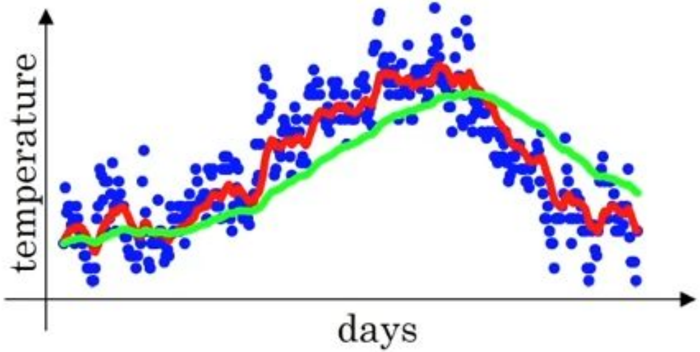
$$\frac{1}{1-0.98} = 50$$

$$SMA = \frac{A_1 + A_2 + \ldots + A_n}{n}$$

where:

A = Average in period n

n = Number of time periods



Andrew Ng

Optimization algorithms

Exponentially weighted averages

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

$\beta = 0.9$: $\approx$ 10 days' temperature.

$\beta = 0.98$: $\approx$ 50 days

V_t as approximately average over $\approx \frac{1}{1-\beta}$ days' temperature.

$$\frac{1}{1-0.98} = 50$$

$$SMA = \frac{A_1 + A_2 + \ldots + A_n}{n}$$

where:

A = Average in period n

n = Number of time periods

What Is an Exponential Moving Average (EMA)?

An exponential moving average (EMA) is a type of moving average (MA) that places a greater weight and significance on the most recent data points. The exponential moving average is also referred to as the exponentially weighted moving average.

Optimization algorithms

Exponentially weighted averages

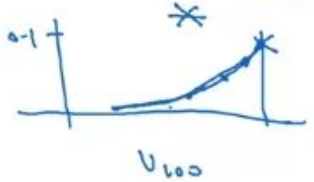
$$v_t = \beta v_{t-1} + (1-\beta)\theta_t$$

$$v_{100} = 0.9v_{99} + 0.1\theta_{100}$$

$$\boxed{v_{99} = 0.9v_{98} + 0.1\theta_{99}}$$

$$\boxed{v_{98} = 0.9v_{97} + 0.1\theta_{98}}$$

$$\cdots$$



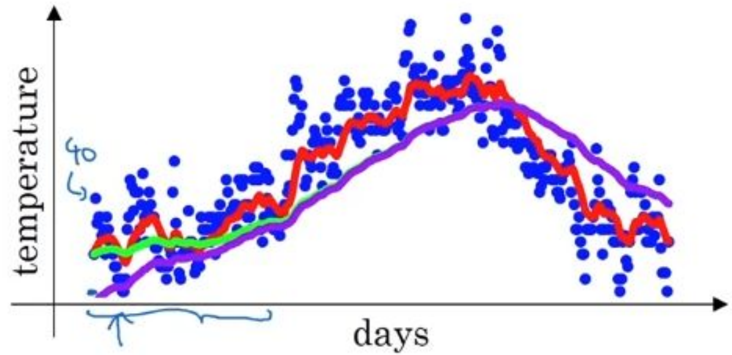
$$0.1\,\theta_{98} + 0.9\,v_{97}$$

$$V_{100} = 0.1\,\theta_{100} + 0.9\,\cancel{v_{99}}\,(0.1\,\theta_{99} + 0.9\,\boxed{\cancel{v_{98}}})$$

$$= 0.1\,\theta_{100} + 0.1 \times 0.9 \cdot \theta_{99} + 0.1\,(0.9)^2\,\theta_{98} + 0.1\,(0.9)^3\,\theta_{97} + 0.1\,(0.9)^4\,\theta_{96} + \cdots$$

Optimization algorithms

Bias correction



$\beta = 0.98$

$\rightarrow v_t = \beta v_{t-1} + (1-\beta)\theta_t$

$V_0 = 0$

$V_1 = \cancel{0.98 V_0 +} \; 0.02\,\theta_1$

$V_2 = 0.98\,V_1 + 0.02\,\theta_2$

$= 0.98 \times 0.02 \times \theta_1 + 0.02\,\theta_2$

$= 0.0196\,\theta_1 + 0.02\,\theta_2$

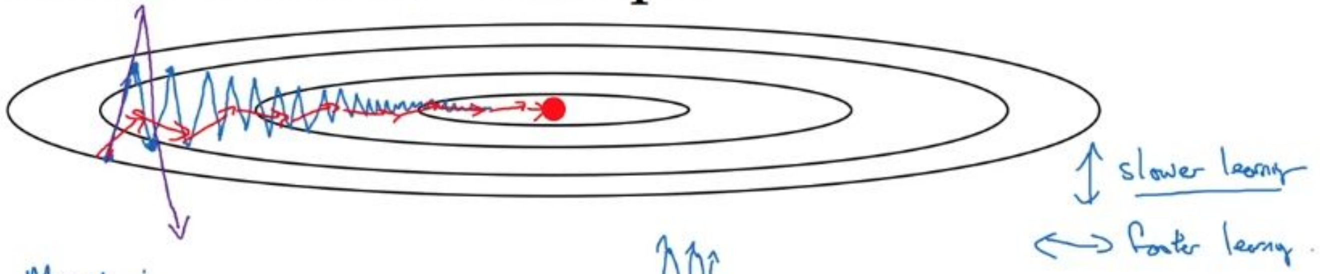
$\dfrac{V_t}{1-\beta^t}$

$t=2: \quad 1-\beta^t = 1-(0.98)^2 = 0.0396$

$\dfrac{V_2}{0.0396} = \dfrac{0.0196\,\theta_1 + 0.02\,\theta_2}{0.0396}$

Optimization algorithms

Gradient descent example



Momentum:

On iteration t:

Compute dW, db on current mini-batch.

$$V_{dW} = \beta V_{dW} + (1-\beta) dW$$

$$V_{db} = \beta V_{db} + (1-\beta) db$$

"$V_\theta = \beta V_\theta + (1-\beta) \theta_t$"

$$W := W - \alpha V_{dW}, \quad b := b - \alpha V_{db}$$

Optimization algorithms

Implementation details

On iteration t:

Compute dW, db on the current mini-batch

$$v_{dW} = \beta v_{dW} + (1-\beta)dW$$

$$v_{db} = \beta v_{db} + (1-\beta)db$$

$$W = W - \alpha v_{dW}, \;\; b = b - \alpha v_{db}$$

Hyperparameters: α, β $\beta = 0.9$

Andrew Ng