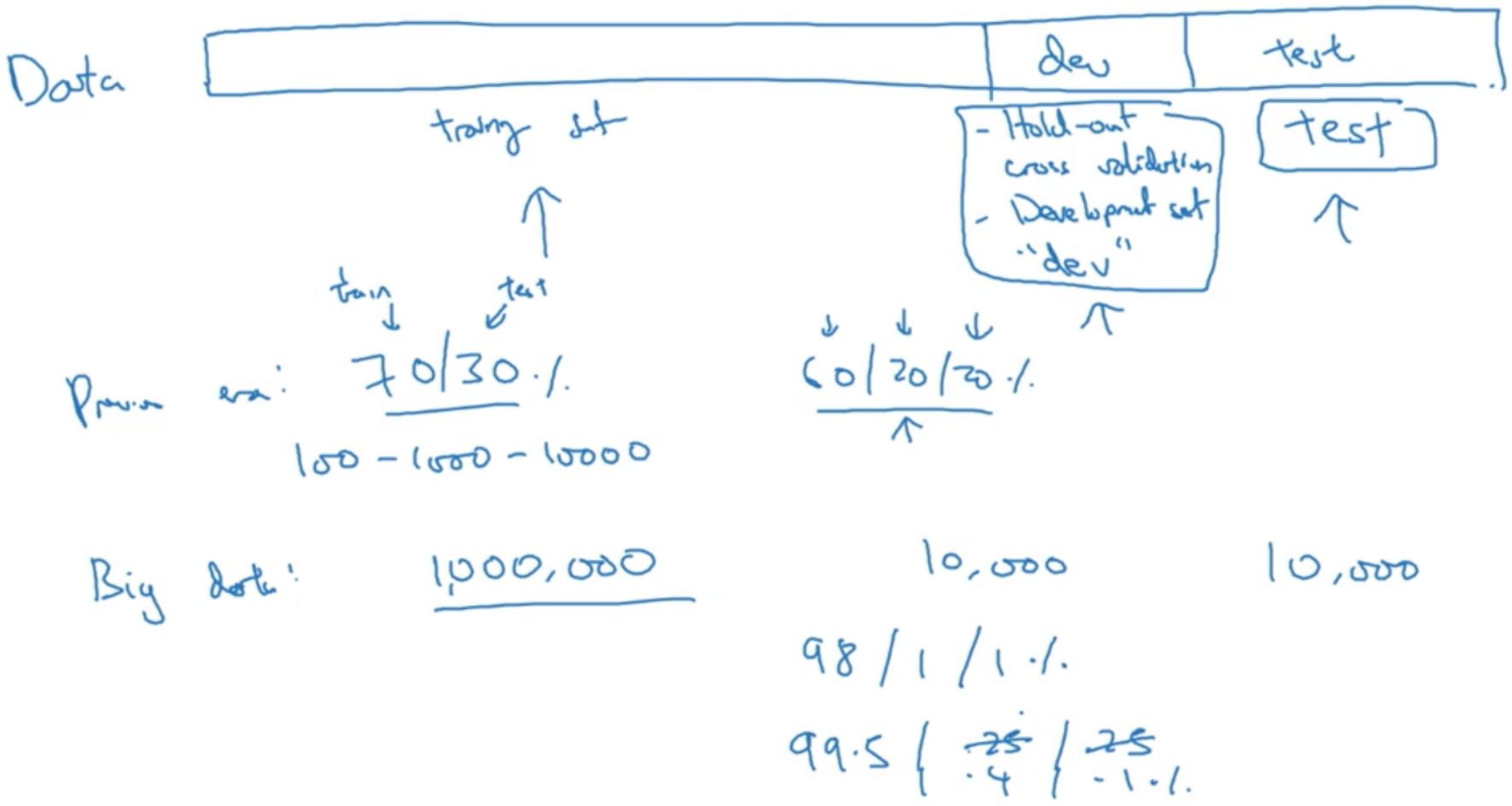
Train/dev/test sets



Andrew Ng

Mismatched train/test distribution

Certs

Training set:
Cat pictures from webpages

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

→ Make sure dev and test come from same distribution.

"test"

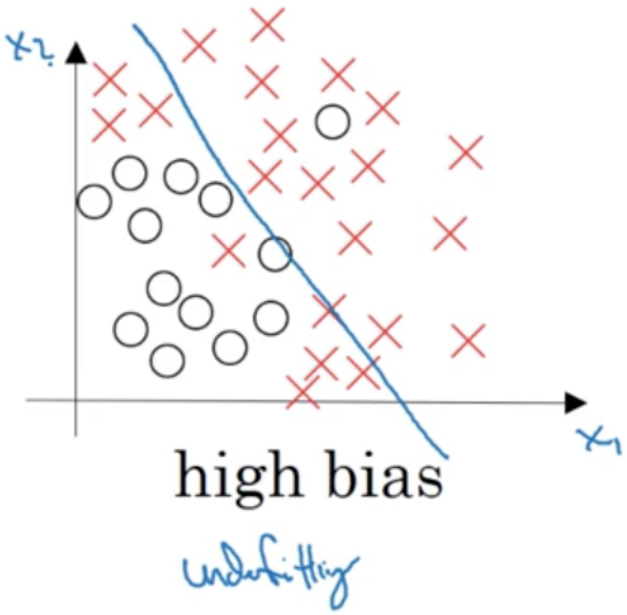
train / dev

train / test

train / dev

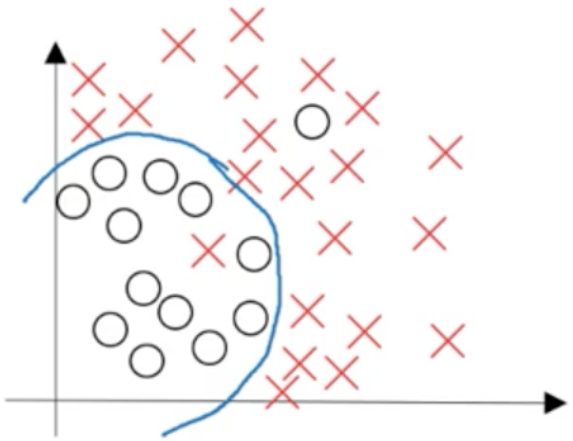
Not having a test set might be okay. (Only dev set.)

Bias and Variance

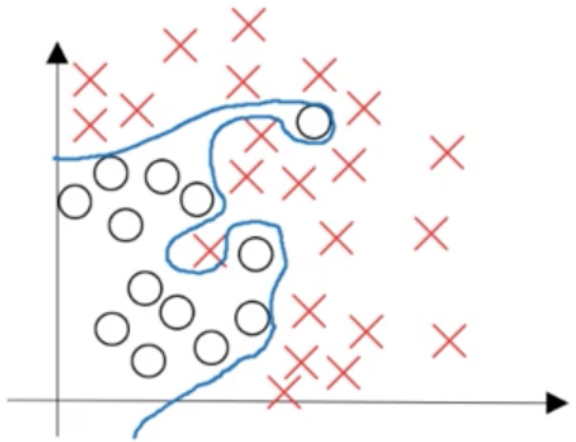


high bias

underfitting



→ "just right"



high variance

overfitting

Bias and Variance

Cat classification

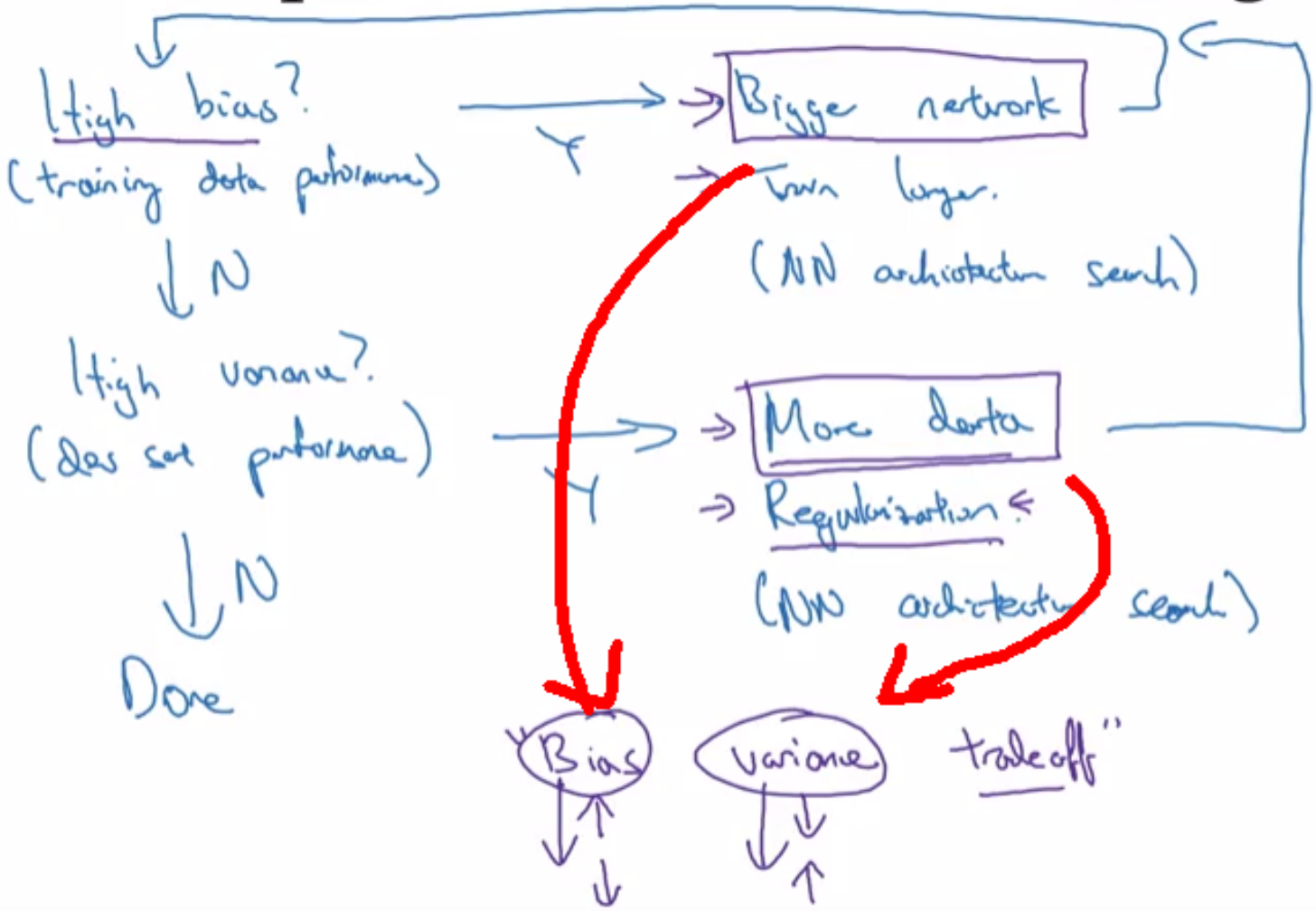
Train set error:	1%	15%	15%	0.5%
Dev set error:	11%	16%	30%	1%
	high variance	high bias	high bias & high variance	low bias low variance

Human: ≈0%

Optimal (Bayes) error: ~~≈0%~~ 15%

Blurry images

Basic recipe for machine learning



Logistic regression

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

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

λ = regularization parameter

lambda

lambd

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

L_2 regularization $\quad \|w\|_2^2 = \sum_{j=1}^{n_x} w_j^2 = w^T w$ ←

Popular!

L_1 regularization $\quad \frac{\lambda}{2m}\sum_{j=1}^{n_x}|w_j| = \frac{\lambda}{2m}\|w\|_1$ ← w will be sparse

Neural network

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

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

"Frobenius norm"

$\|\cdot\|_2^2$

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

$\|\cdot\|_F^2$

?

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

$$\frac{\partial J}{\partial w^{[l]}} = dw^{[l]}$$

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

"Weight decay"

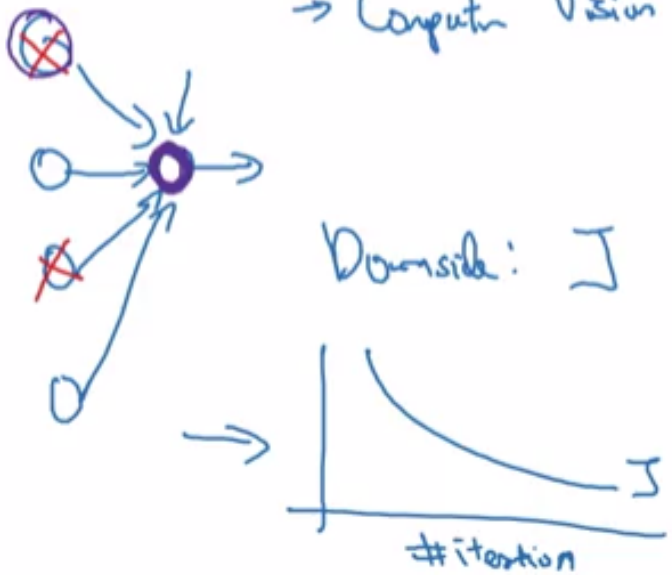
$$w^{[l]} := w^{[l]} - \alpha \left[(\text{from backprop}) + \frac{\lambda}{m} w^{[l]} \right]$$

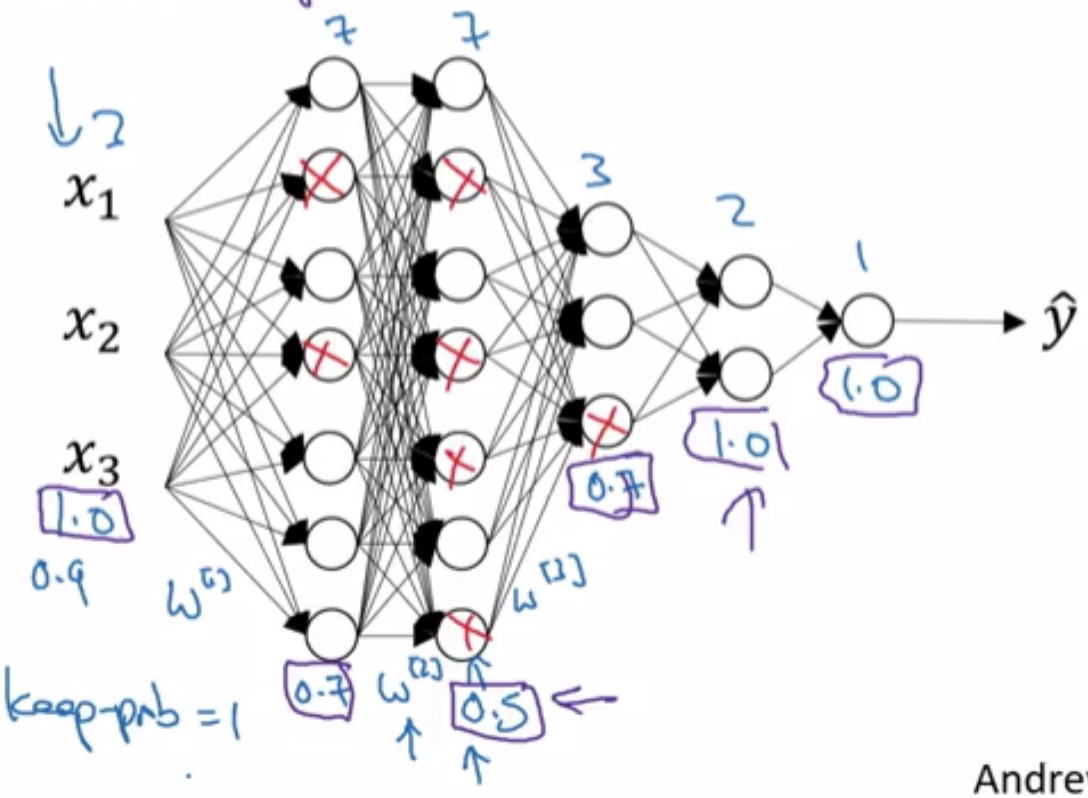
$$= \underbrace{\left(1 - \frac{\alpha\lambda}{m}\right)}_{} w^{[l]} - \alpha\, (\text{from backprop})$$

Andrew Ng

Why does drop-out work?

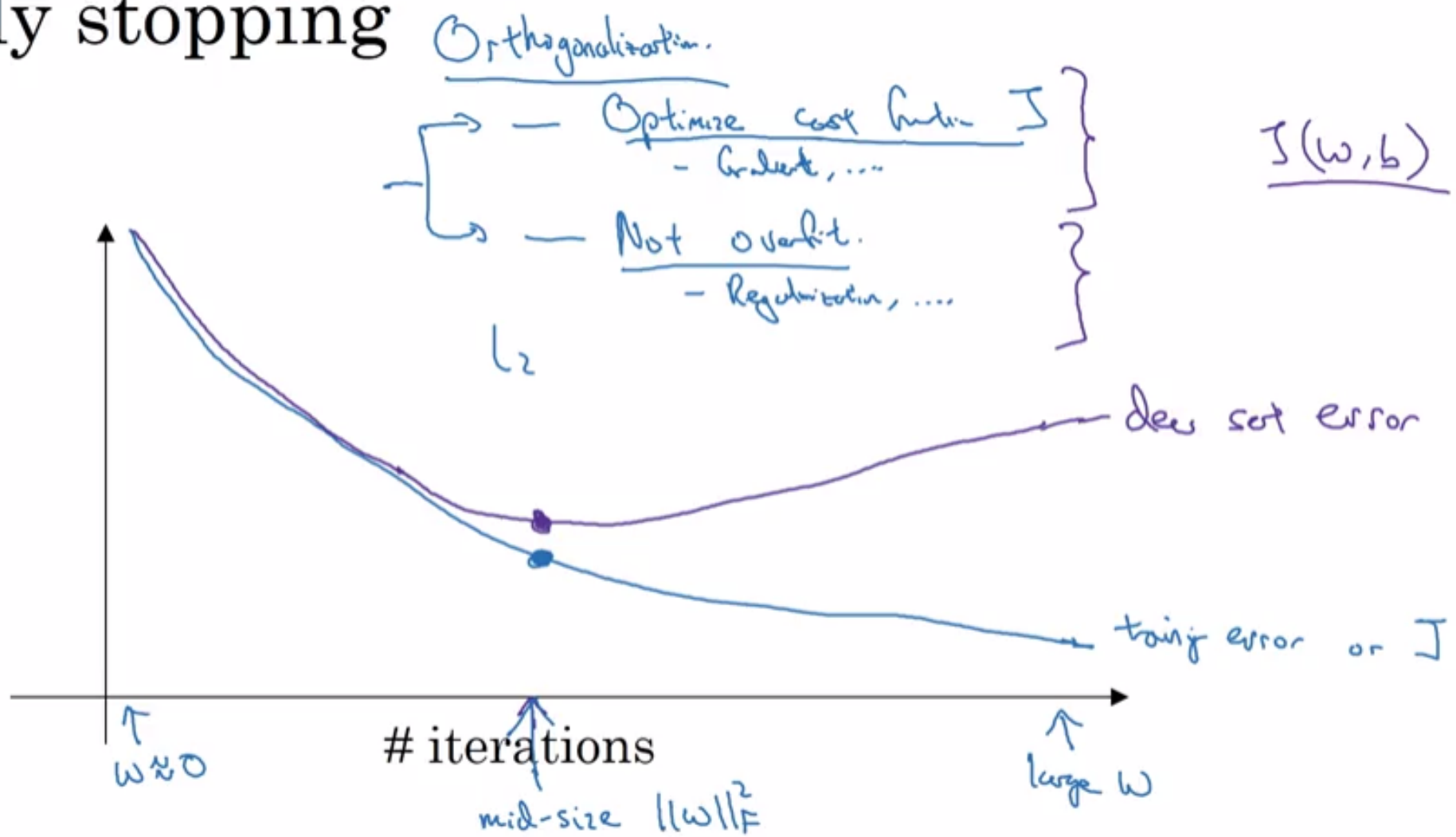
Intuition: Can't rely on any one feature, so have to spread out weights. ⇝ Shrink weights. L_2





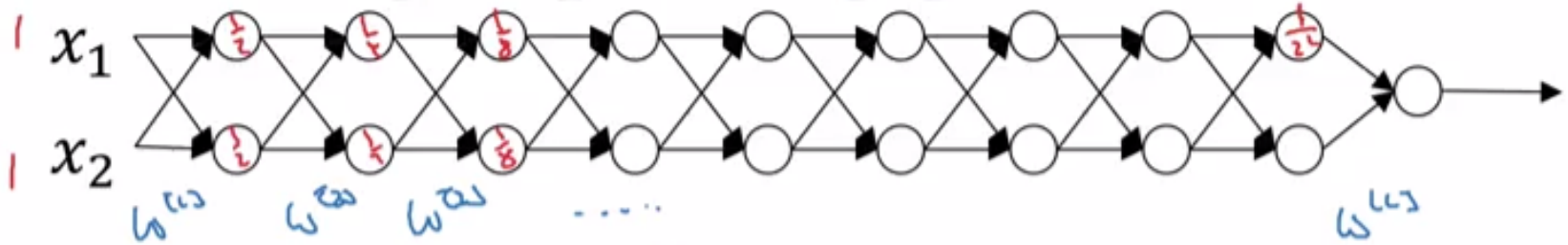
Andrew Ng

Early stopping



Andrew Ng

Vanishing/exploding gradients



$L = 150$

L

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

$\hat{y} = w^{[L]} w^{[L-1]} w^{[L-2]} \cdots x$

$= w^{[L]} w^{[L-1]} w^{[L-2]} \cdots w^{[3]} w^{[2]} w^{[1]} x$

$a^{[3]}$

1.5^L

0.5^L

$w^{[l]} > I$

$w^{[l]} < I$ $\begin{bmatrix} 0.9 & \\ & 0.9 \end{bmatrix}$

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

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

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

$w^{[2]} = \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}$ $\begin{matrix} 0.5 \\ 0.5 \end{matrix}$

$\hat{y} = w^{[L]} \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}^{L-1} x$ $\begin{matrix} 0.5 \\ 0.5 \end{matrix}$

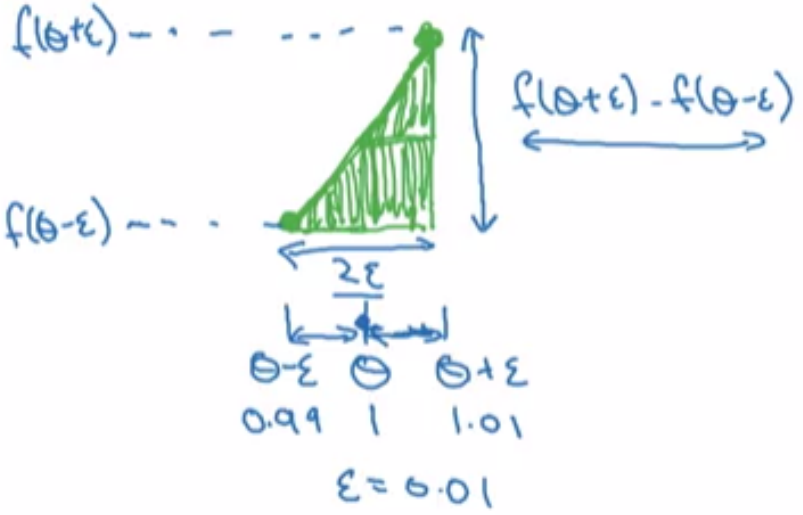
$1.5^{L-1} x$

$0.5^{L-1} x$

Andrew Ng

Checking your derivative computation

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



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

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

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

approx error: 0.0001

(prev slide: 3.0301. error: 0.03)

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

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

Andrew Ng

Mini-batch gradient descent

1 step of gradient descent using $X^{\{t\}}, Y^{\{t\}}$.

X, Y

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)

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

Exponentially weighted (moving) averages

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

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

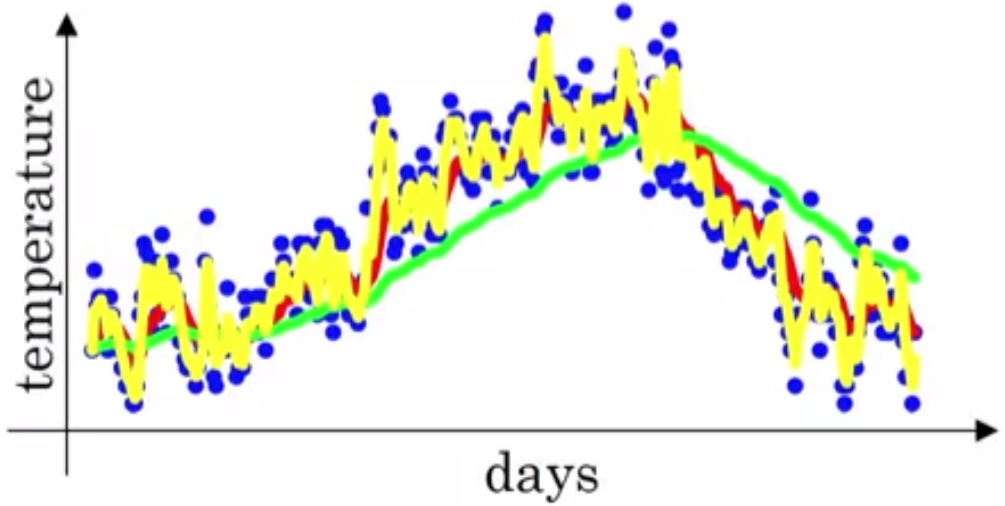
$\beta = 0.98$: $\approx$ 50 days

$\beta = 0.5$: $\approx$ 2 days

V_t as approximately average over

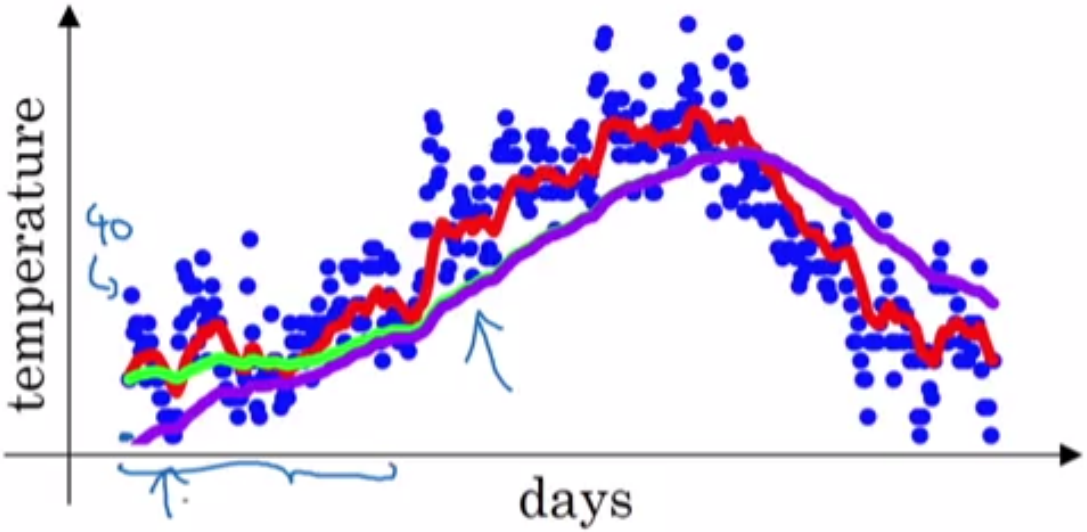
$\rightarrow \approx \frac{1}{1-\beta}$ days' temperature.

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



Andrew Ng

Bias correction



$\beta = 0.98$

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

$v_0 = 0$

$v_1 = 0.98 v_0 + 0.02 \theta_1$

$v_2 = 0.98 v_1 + 0.02 \theta_2$

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

$= 0.0196 \theta_1 + 0.02 \theta_2$

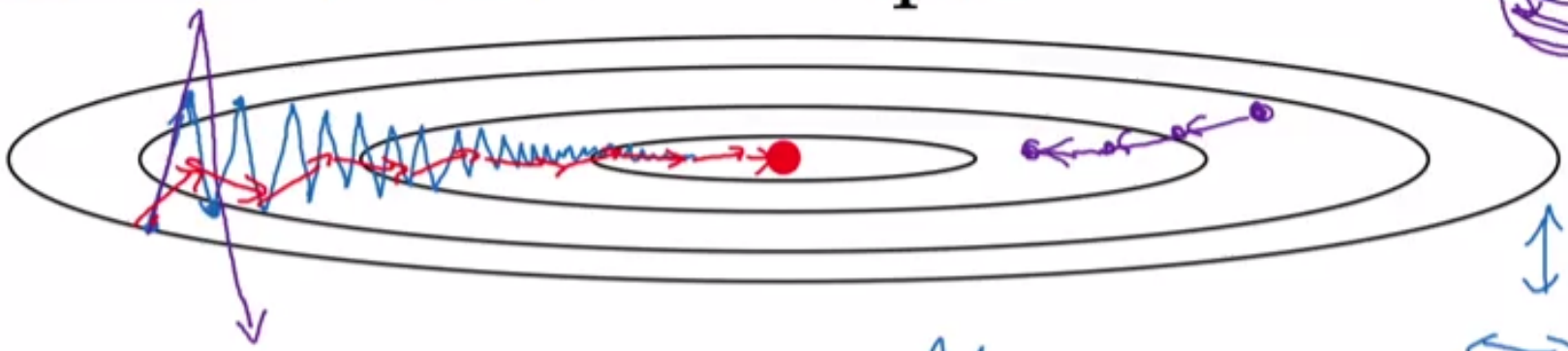
$\frac{v_t}{1-\beta^t}$

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

$\frac{v_2}{0.0396} = \frac{0.0196\theta_1 + 0.02\theta_2}{0.0396}$

Andrew Ng

Gradient descent example



slower learning

faster learning.

Momentum:

On iteration t:

Compute dW, db on current mini-batch.

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

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

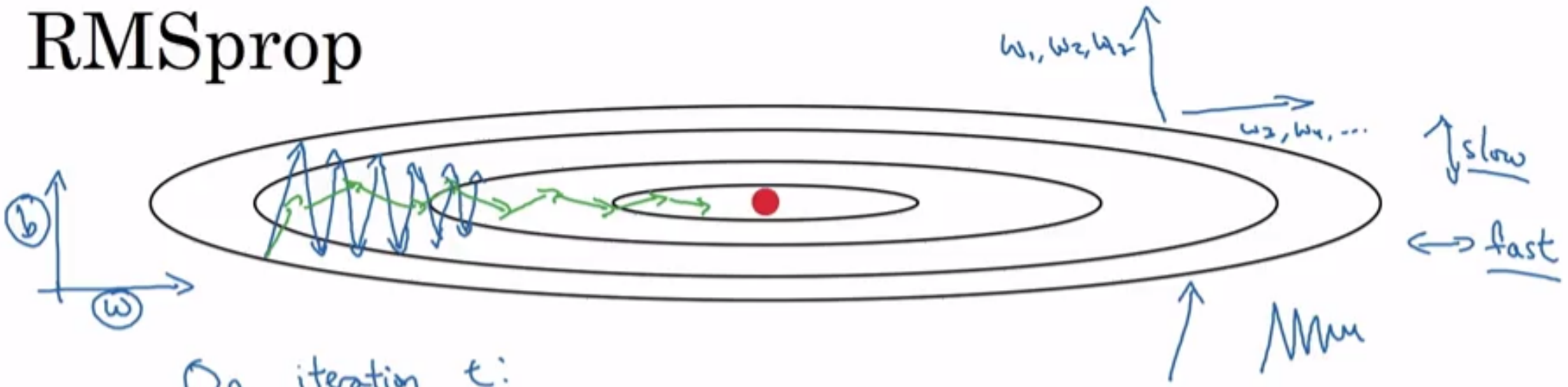
"$v_\theta = \beta v_{\theta-1} + (1-\beta)\theta_t$"

friction — velocity — acceleration

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

Andrew Ng

RMSprop



On iteration t:

Compute dW, db on current mini-batch

element-wise

$S_{dW} = \beta_2 S_{dW} + (1-\beta_2) dW^2$ ← small

→ $S_{db} = \beta_2 S_{db} + (1-\beta_2) db^2$ ← large

can be relatively larger

$$W := W - \alpha \frac{dW}{\sqrt{S_{dW}} + \varepsilon} \qquad b := b - \alpha \frac{db}{\sqrt{S_{db}} + \varepsilon}$$

$\varepsilon = 10^{-8}$

Andrew Ng

Adam optimization algorithm

$V_{dw} = 0$, $S_{dw} = 0$. $V_{db} = 0$, $S_{db} = 0$

On iteration t:

Compute dW, db using current mini-batch

$V_{dw} = \beta_1 V_{dw} + (1-\beta_1) dW$, $V_{db} = \beta_1 V_{db} + (1-\beta_1) db$ ← "momentum" β_1

$S_{dw} = \beta_2 S_{dw} + (1-\beta_2) dW^2$, $S_{db} = \beta_2 S_{db} + (1-\beta_2) db$ ← "RMSprop" β_2

$V_{dw}^{corrected} = V_{dw} / (1-\beta_1^t)$, $V_{db}^{corrected} = V_{db} / (1-\beta_1^t)$

$S_{dw}^{corrected} = S_{dw} / (1-\beta_2^t)$, $S_{db}^{corrected} = S_{db} / (1-\beta_2^t)$

$$W := W - \alpha \frac{V_{dw}^{corrected}}{\sqrt{S_{dw}^{corrected}} + \varepsilon} \qquad b := b - \alpha \frac{V_{db}^{corrected}}{\sqrt{S_{db}^{corrected}} + \varepsilon}$$

Andrew Ng

Hyperparameters choice:

→ α : needs to be tune

→ β_1 : 0.9 → (dw)

→ β_2 : 0.999 → (dw^2)

→ ε : 10^{-8}

Adam: Adaptive moment estimation

Adam Coates

Andrew Ng

Learning rate decay

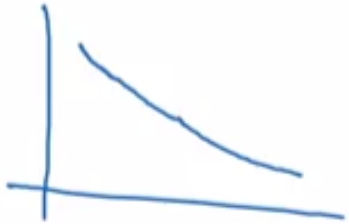
1 epoch = 1 pass through data.

$$\alpha = \frac{1}{1 + \text{decay\_rate} * \text{epoch\_num}} \alpha_0$$

$\alpha_0 = 0.2$

decay_rate = 1

Epoch	α
1	0.1
2	0.67
3	0.5
4	0.4
...	...



Andrew Ng

Other learning rate decay methods

formula

$\alpha = 0.95^{\text{epoch-num}} \cdot \alpha_0$ — exponentially decay.

$\alpha = \frac{k}{\sqrt{\text{epoch-num}}} \cdot \alpha_0$ or $\frac{k}{\sqrt{t}} \cdot \alpha_0$

discrete staircase

Manual decay.

Implementing Batch Norm

Given some intermediate values in NN $z^{(1)}, \ldots, z^{(m)}$

$z^{[l](i)}$

$$\mu = \frac{1}{m} \sum_i z^{(i)}$$

$$\sigma^2 = \frac{1}{m} \sum_i (z_i - \mu)^2$$

$$z^{(i)}_{norm} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \varepsilon}}$$

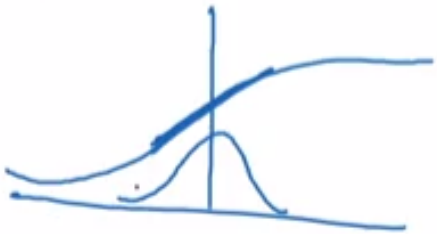
$$\tilde{z}^{(i)} = \gamma z^{(i)}_{norm} + \beta$$

learnable parameters of model.

If $\gamma = \sqrt{\sigma^2 + \varepsilon}$, $\beta = \mu$, then $\tilde{z}^{(i)} = z^{(i)}$

X

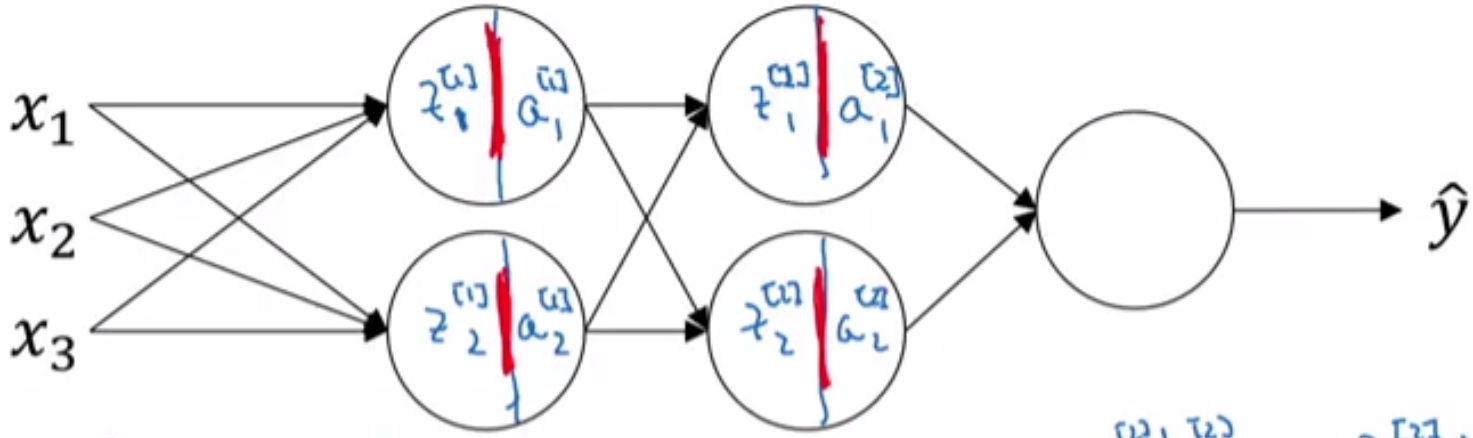
$z^{(i)}$



Use $\tilde{z}^{[l](i)}$ instead of $z^{[l](i)}$.

Andrew Ng

Adding Batch Norm to a network



$X \xrightarrow{W^{[1]}, b^{[1]}} z^{[1]} \xrightarrow[\text{Batch Norm (BN)}]{\beta^{[1]}, \gamma^{[1]}} \tilde{z}^{[1]} \rightarrow a^{[1]} = g^{[1]}(\tilde{z}^{[1]}) \xrightarrow{W^{[2]}, b^{[2]}} z^{[2]} \xrightarrow[BN]{\beta^{[2]}, \gamma^{[2]}} \tilde{z}^{[2]} \rightarrow a^{[2]} \rightarrow \dots$

Parameters: $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}, \dots, W^{[L]}, b^{[L]}, \beta^{[1]}, \gamma^{[1]}, \beta^{[2]}, \gamma^{[2]}, \dots, \beta^{[L]}, \gamma^{[L]}$

$d\beta^{[2]}$

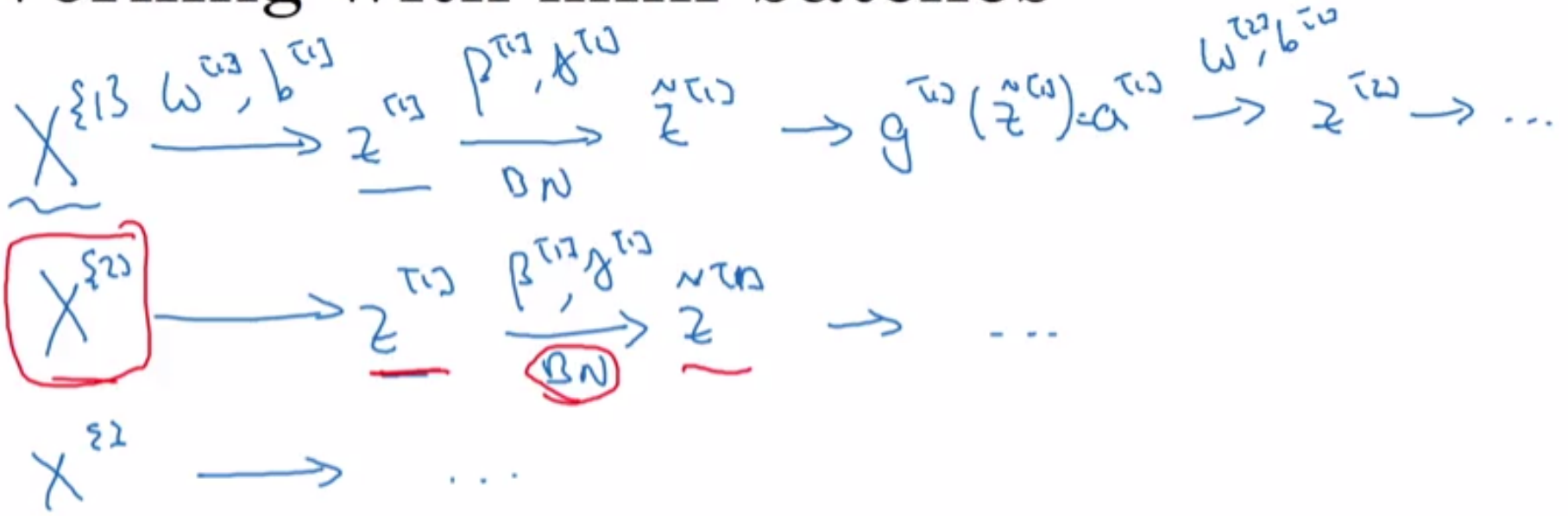
$\beta^{[2]} = \beta^{[2]} - \alpha\, d\beta^{[2]}$

tf.nn.batch_normalization

β

Andrew Ng

Working with mini-batches



Parameters: $w^{[l]}$, ~~$b^{[l]}$~~, $\beta^{[l]}$, $\gamma^{[l]}$.

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

$z^{[l]}$

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

$z^{[l]} = w^{[l]} a^{[l-1]} + b^{[l]}$

$z^{[l]} = w^{[l]} a^{[l-1]}$

$z^{[l]}_{norm}$

$z^{[l]} = \gamma^{[l]} z^{[l]}_{norm} + \beta^{[l]}$

Andrew Ng

Implementing gradient descent

for $t = 1 \ldots$ numMiniBatches

Compute forward prop on $X^{\{t\}}$.

In each hidden layer, use BN to replace $z^{[l]}$ with $\tilde{z}^{[l]}$.

Use backprop to compute $dW^{[l]}$, ~~$db^{[l]}$~~, $d\beta^{[l]}$, $d\gamma^{[l]}$

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

$\beta^{[l]} := \beta^{[l]} - \alpha\, d\beta^{[l]}$

$\gamma^{[l]} := \ldots$

Works w/ momentum, RMSprop, Adam.

Andrew Ng