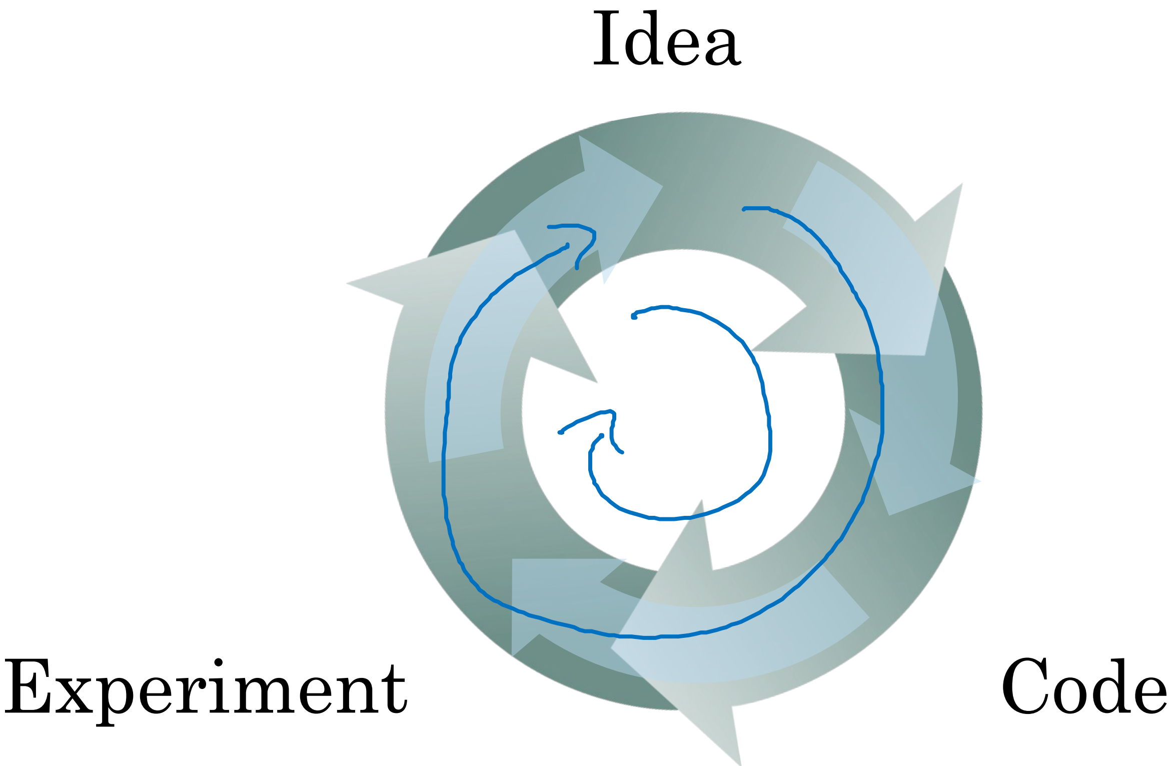
deeplearning.ai

Setting up your ML application

Train/dev/test sets

Applied ML is a highly iterative process

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



NLP, Vision, Speech, Structural data

Ads search Security logistics ....

Andrew Ng

Train/dev/test sets

Data | | dev | test
--- | --- | ---
training set | - Hold-out cross validation
- Development set "dev" | test

Previous era: 70/30 % (train / test)

100 - 1000 - 10000

60/20/20 %

Big data: 1,000,000   10,000   10,000

98/1/1 %

99.5 / ~~.25~~ .4 / ~~.25~~ .1 %

Mismatched train/test distribution

Cats

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

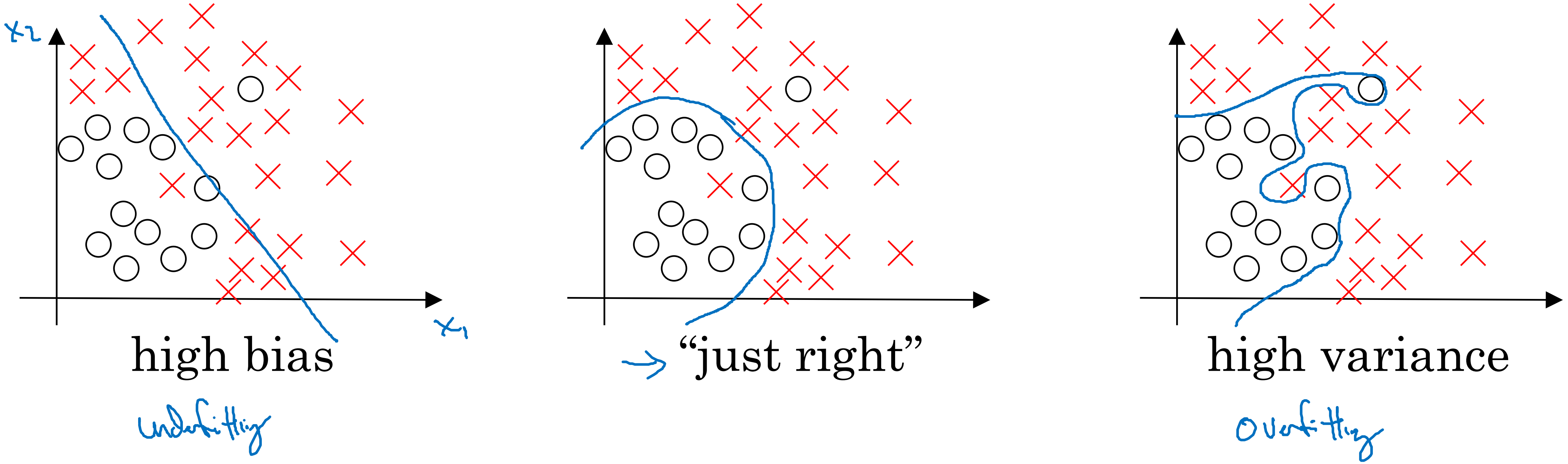
Not having a test set might be okay. (Only dev set.)

deeplearning.ai

Setting up your ML application

Bias/Variance

Bias and Variance



Andrew Ng

Bias and Variance

Cat classification

y=1

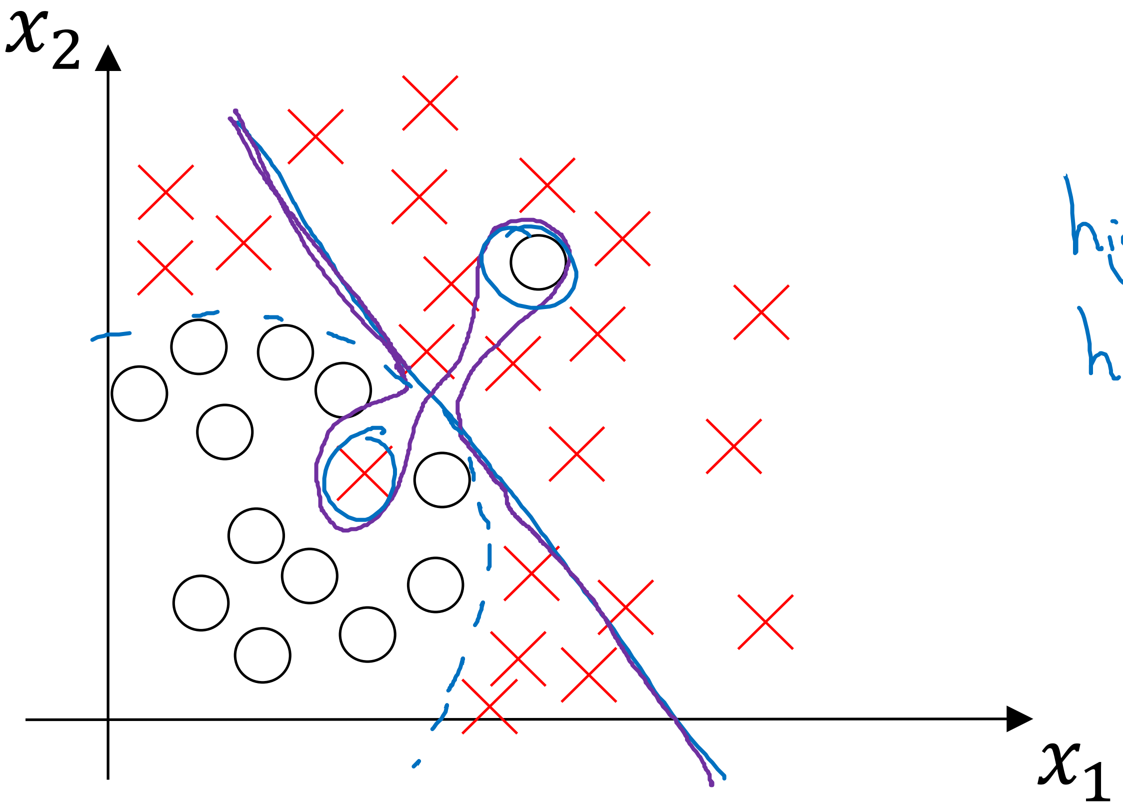
y=0

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%

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

High bias and high variance



high bias
high variance

Andrew Ng

deeplearning.ai

Setting up your ML application

Basic "recipe" for machine learning

Basic “recipe” for machine learning

Basic recipe for machine learning

High bias?
(training data performance)

Y → Bigger network
→ Train longer
(NN architecture search)

↓ N

High variance?
(dev set performance)

Y → More data
→ Regularization
(NN architecture search)

↓ N

Done

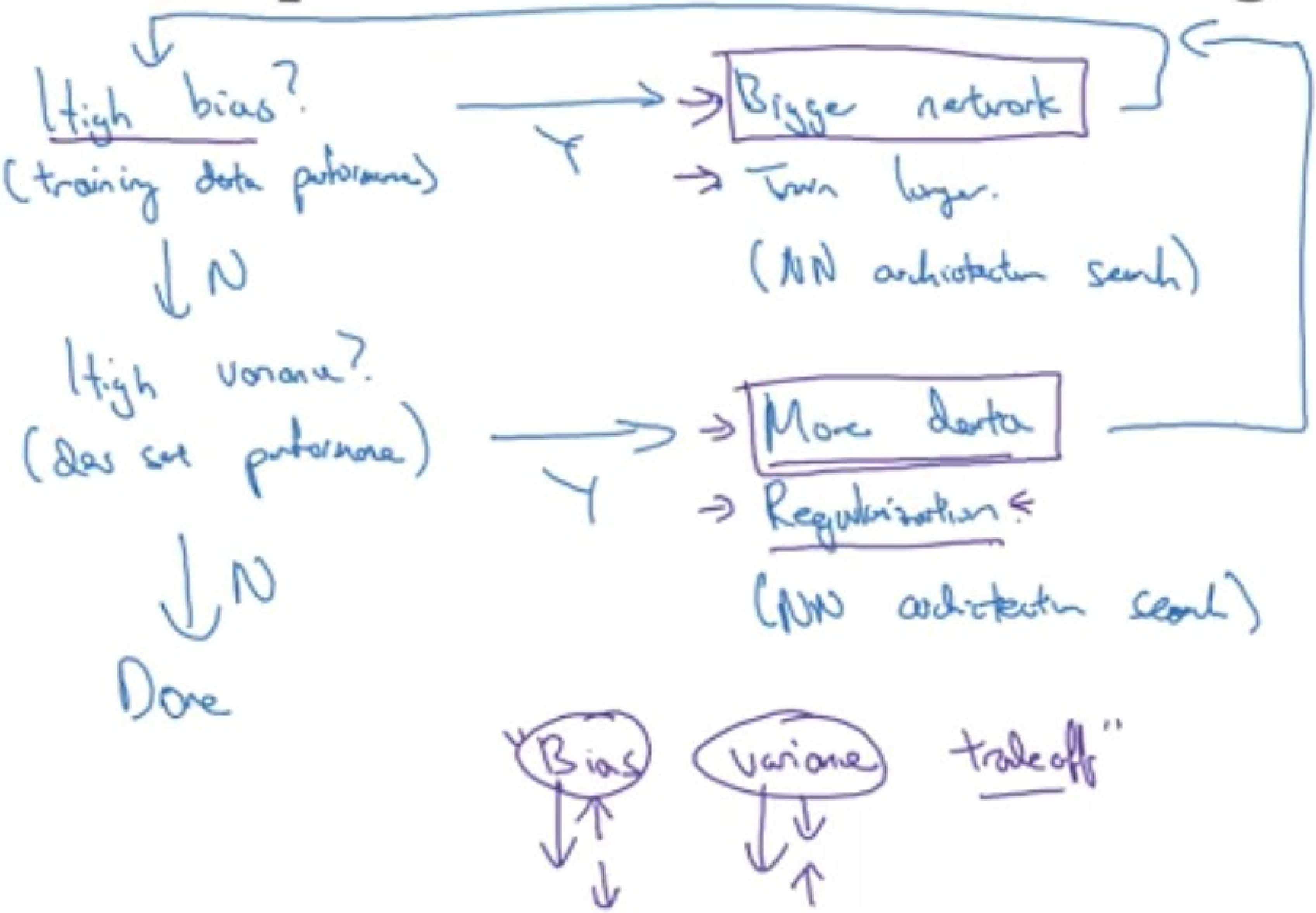
Bias variance "tradeoff"

deeplearning.ai

Setting up your ML application

Basic "recipe" for machine learning

Basic recipe for machine learning



deeplearning.ai

Regularizing your neural network

Regularization

Logistic regression

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

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

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

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

w will be sparse

Andrew Ng

Neural network

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

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

"Frobenius norm" $\qquad \|\cdot\|_2^2 \qquad \|\cdot\|_F^2$

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

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

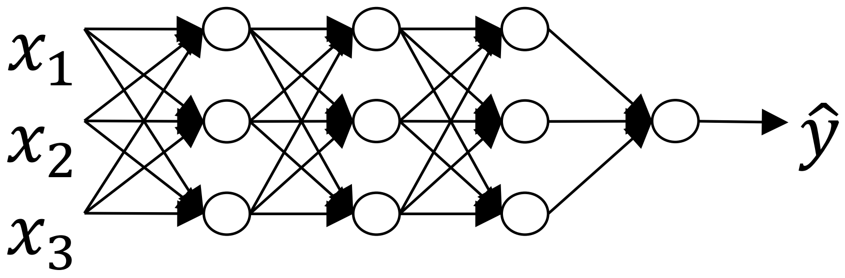
"Weight decay"

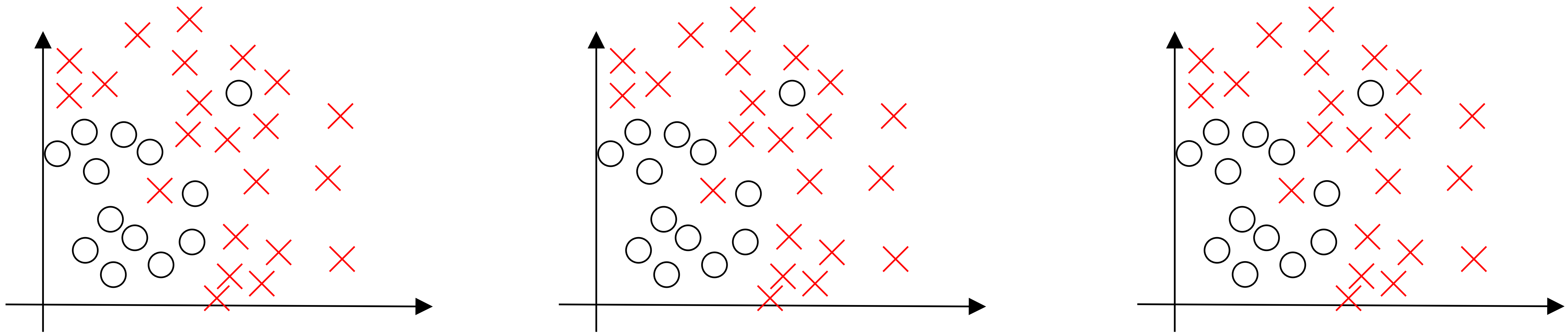
$$w^{[l]} := w^{[l]} - \alpha\left[(\text{from backprop}) + \frac{\lambda}{m} w^{[l]}\right]$$

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

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

How does regularization prevent overfitting?





Andrew Ng

How does regularization prevent overfitting?

deeplearning.ai

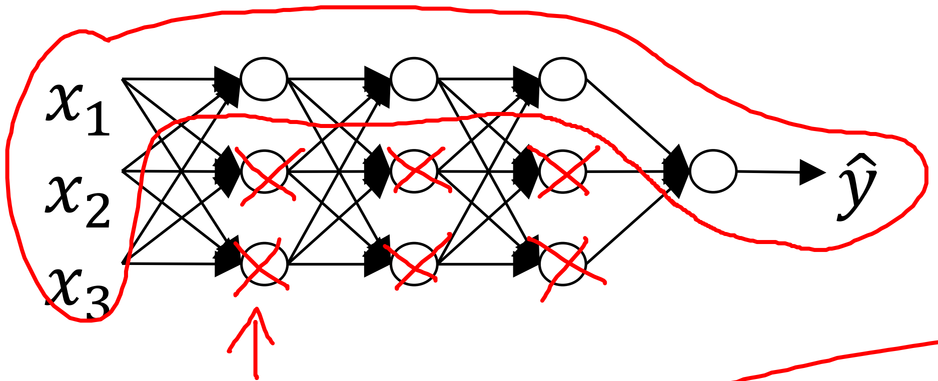
Regularizing your neural network

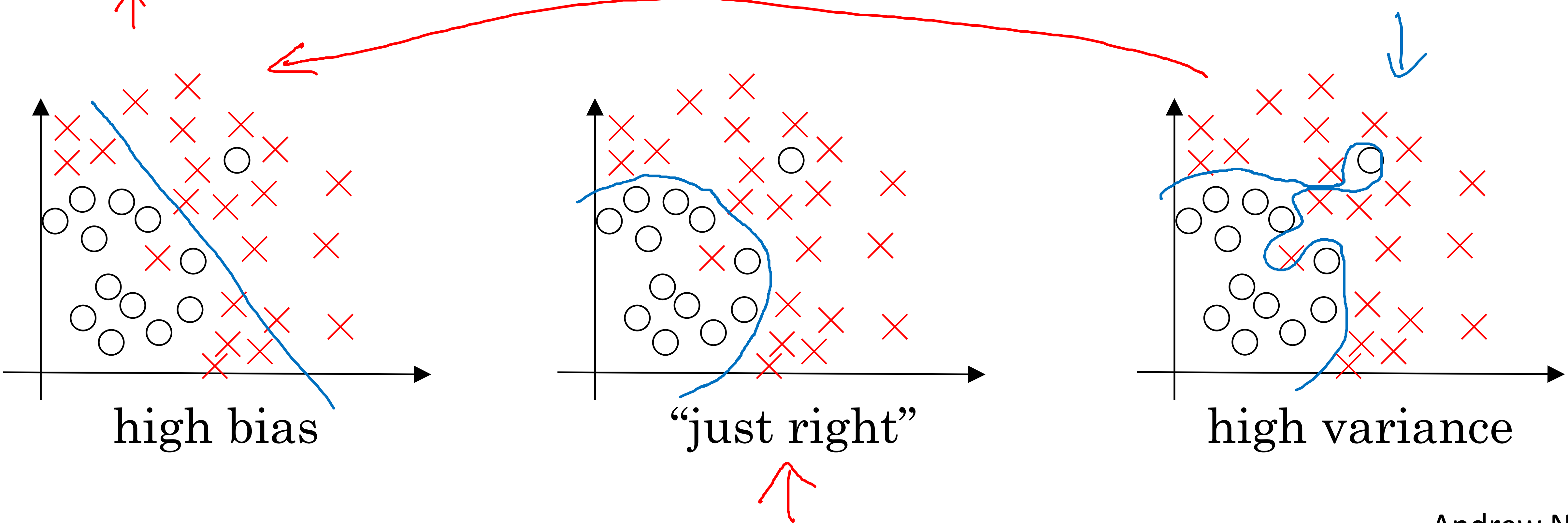
Why regularization reduces overfitting

How does regularization prevent overfitting?

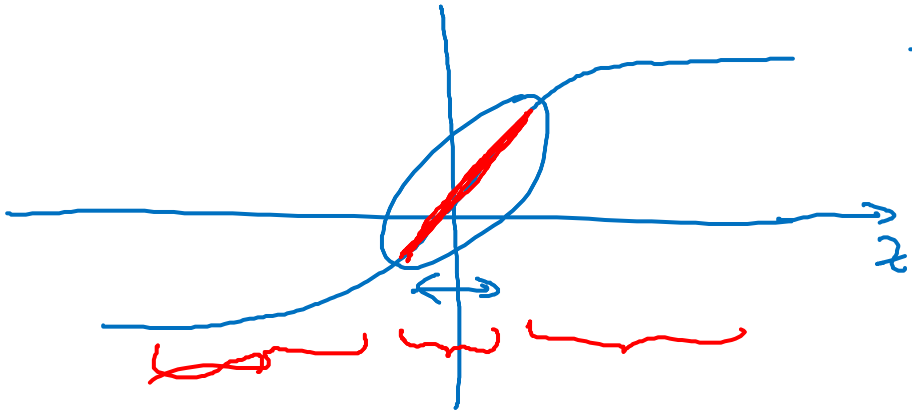
$$J(w^{[l]}, b^{[l]}) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\sum_{l=1}^{L} \underbrace{\|w^{[l]}\|_F^2}$$

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



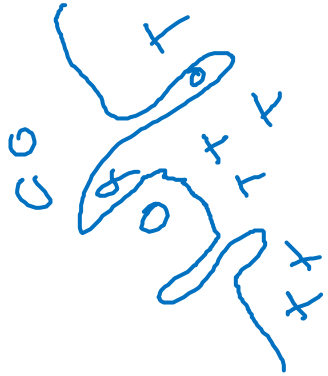


How does regularization prevent overfitting?



tanh

$g(z) = \tanh(z)$

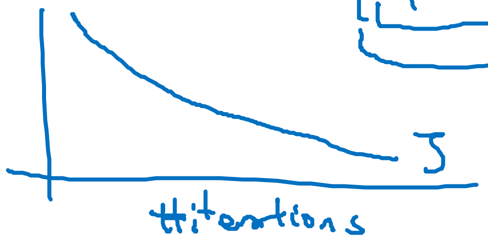


$\lambda \uparrow \qquad W^{[l]} \downarrow$

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

Every layer $\approx$ linear.

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



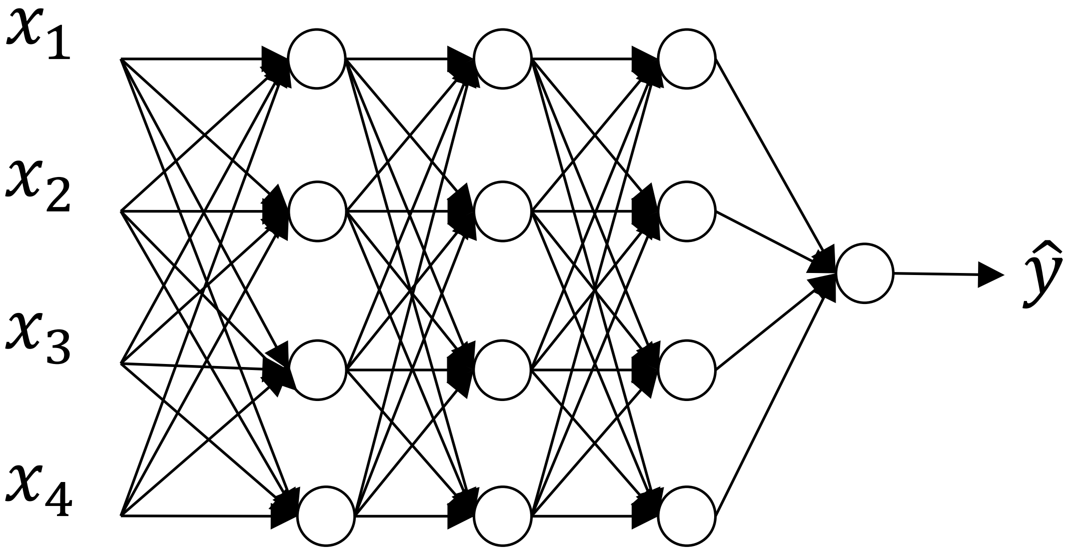
Andrew Ng

deeplearning.ai

Regularizing your neural network

Dropout regularization

Dropout regularization



0.5 0.5 0.5

Andrew Ng

Implementing dropout ("~~Inverted dropout~~")

Illustrate with layer $l=3$. keep-prob $= 0.8$ 0.2

→ $\boxed{d3} = $ np.random.rand(a3.shape[0], a3.shape[1]) < keep-prob

a3 = np.multiply(a3, d3) # a3 *= d3.

→ $\boxed{\text{a3 /= } \cancel{0.8} \text{ keep-prob}}$ ←

50 units. ⟿ 10 units shut off

$$z^{[4]} = w^{[4]} \cdot a^{[3]} + b^{[4]}$$

reduced by 20%. Test

/= 0.8

Andrew Ng

Making predictions at test time

$a^{[0]} = X$

No drop out.

$z^{[1]} = W^{[1]} \underline{a^{[0]}} + b^{[1]}$

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

$z^{[2]} = W^{[2]} \underline{a^{[1]}} + b^{[2]}$

$a^{[2]} = \ldots$

$\downarrow$

$\hat{y}$

$/= \text{keep-prob}$

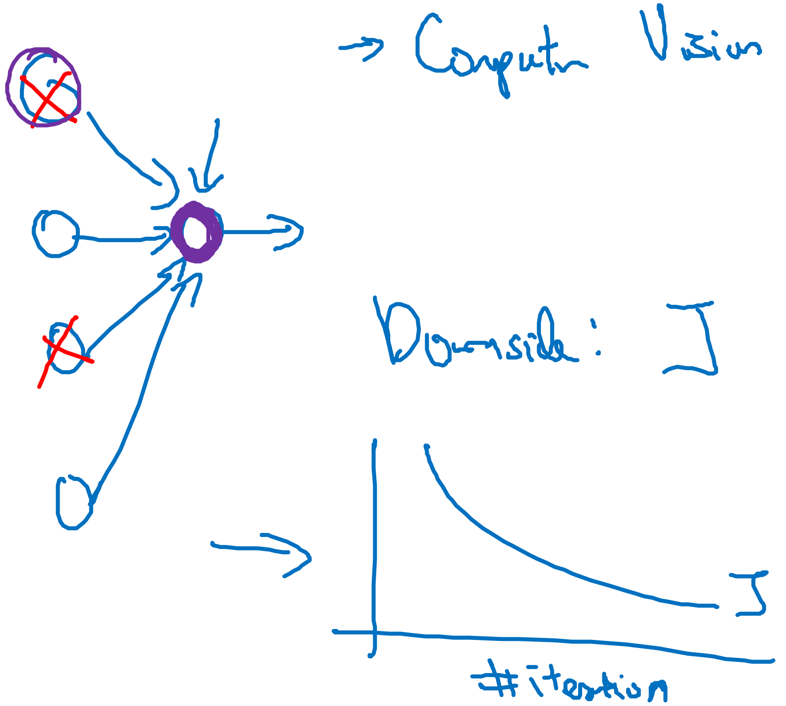
deeplearning.ai

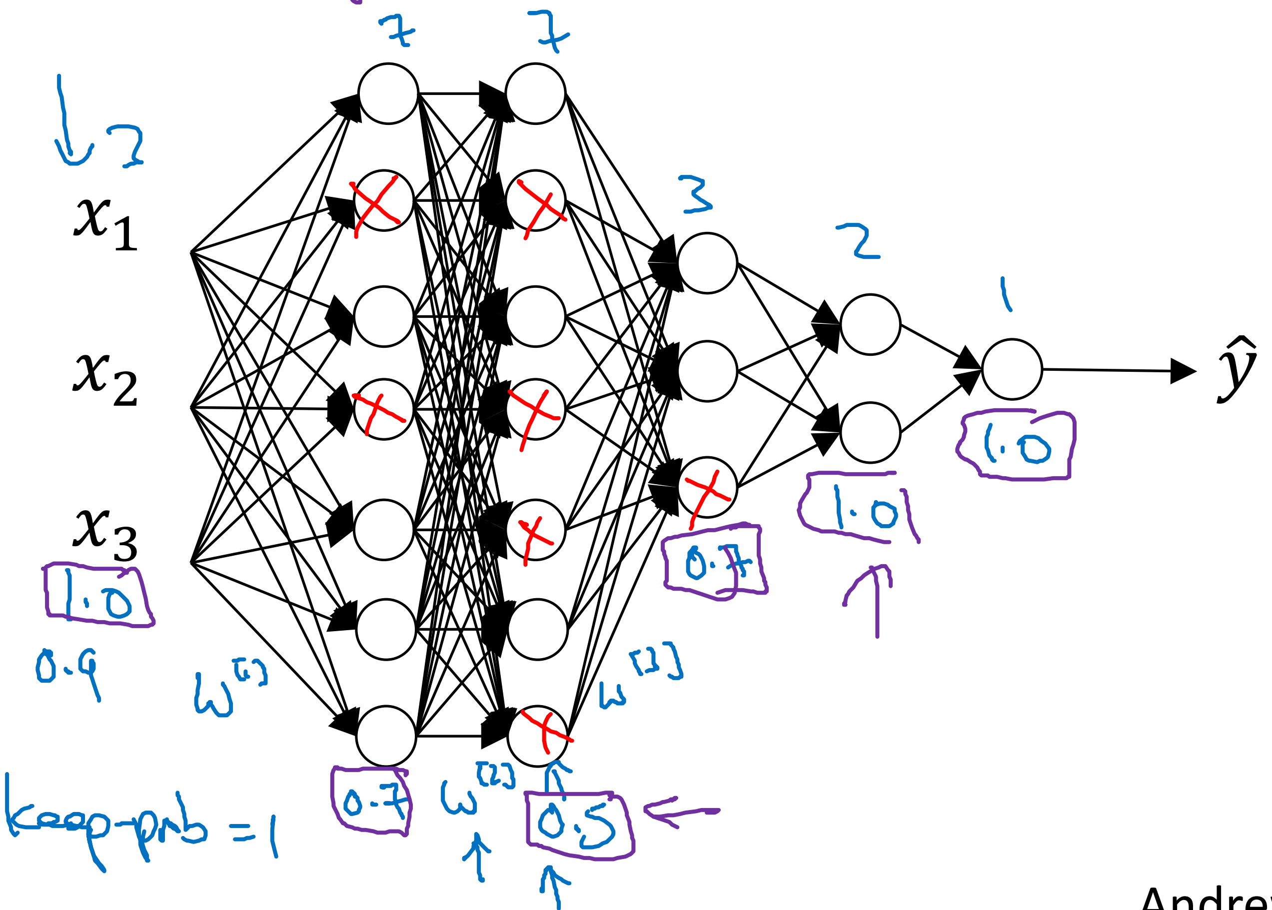
Regularizing your neural network

Understanding dropout

Why does drop-out work?

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





deeplearning.ai

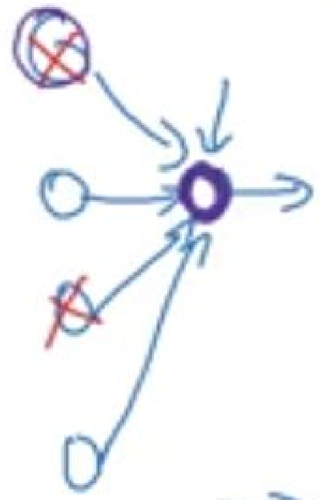
Regularizing your neural network

Understanding dropout

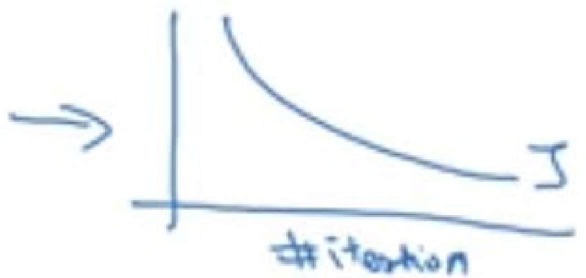
Why does drop-out work?

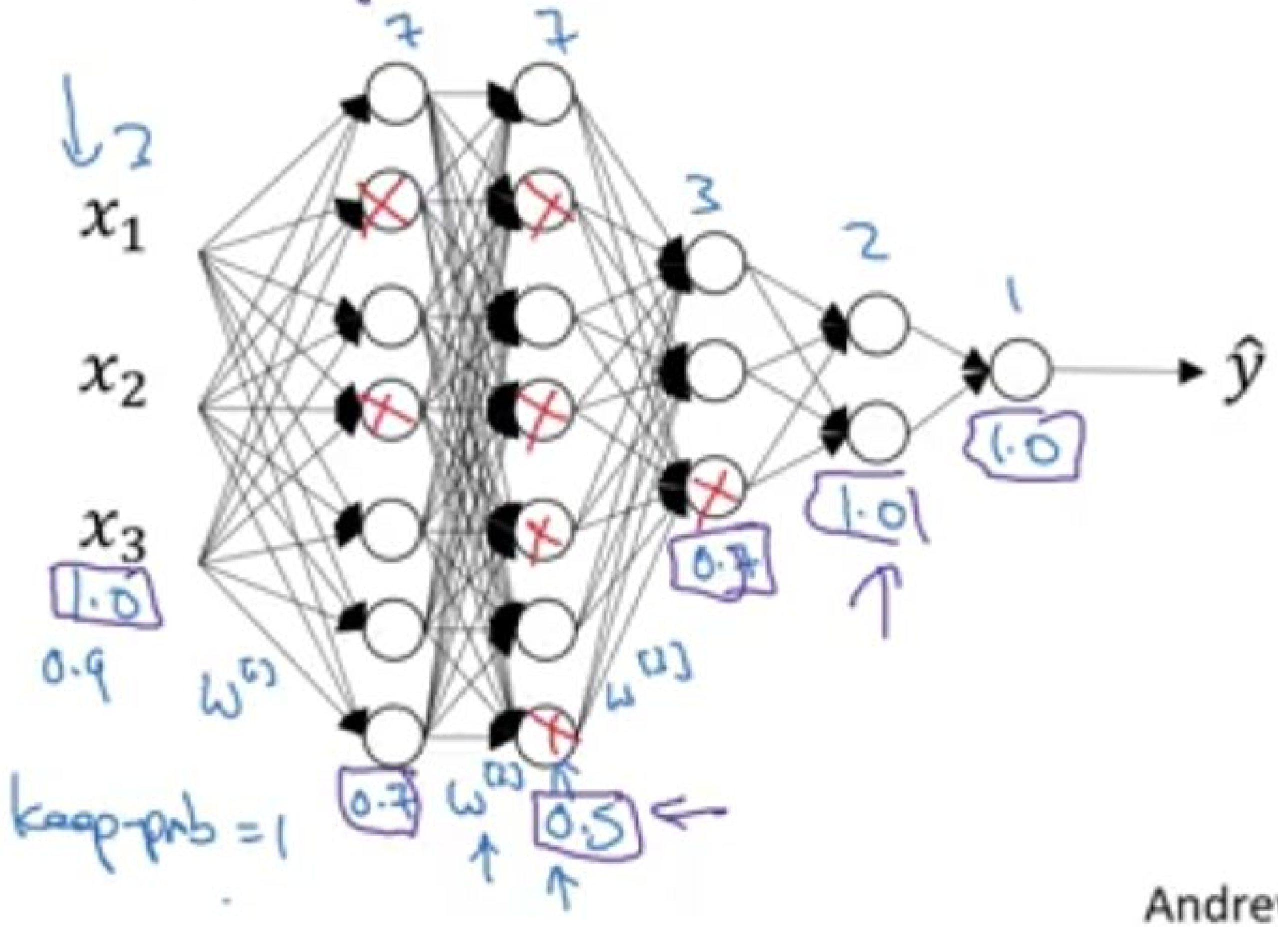
Intuition: Can't rely on any one feature, so have to spread out weights. ⇝ Shrink weights. L_2

→ Computer Vision



Downside: J





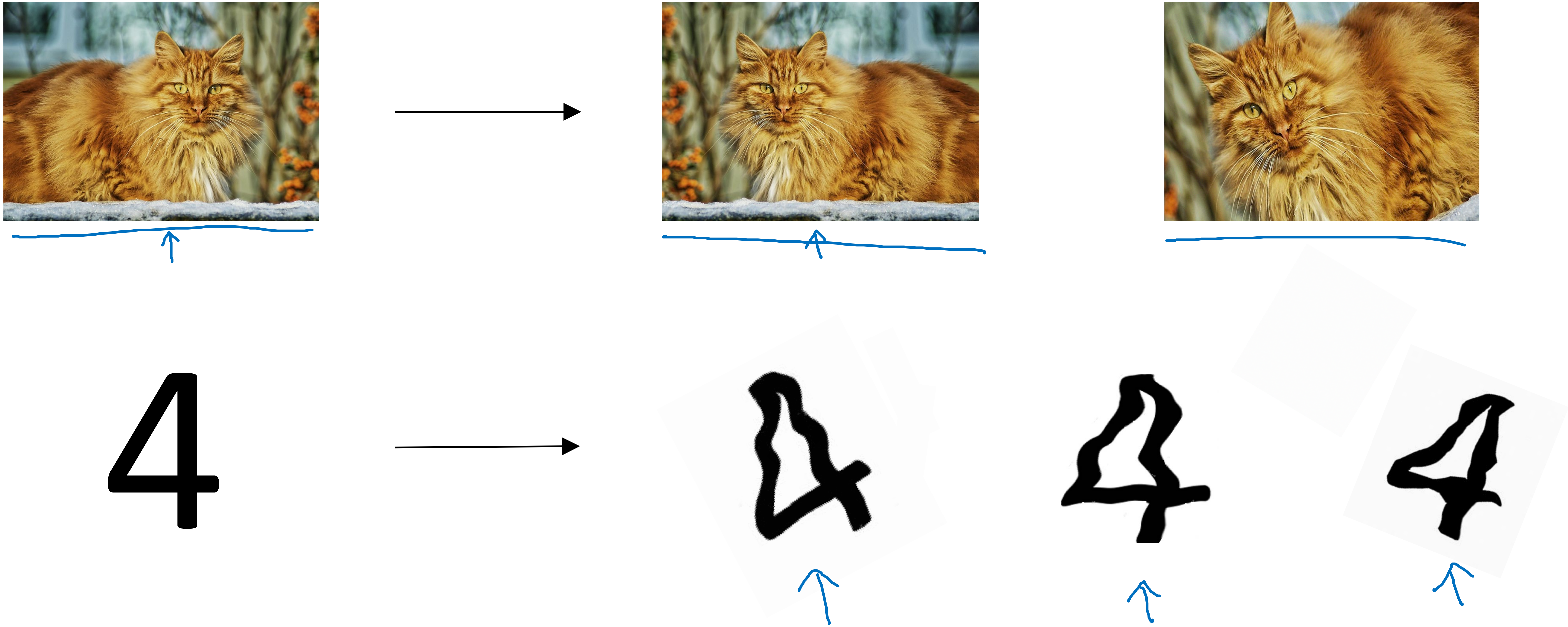
Andrew Ng

deeplearning.ai

Regularizing your neural network

Other regularization methods

Data augmentation



Andrew Ng

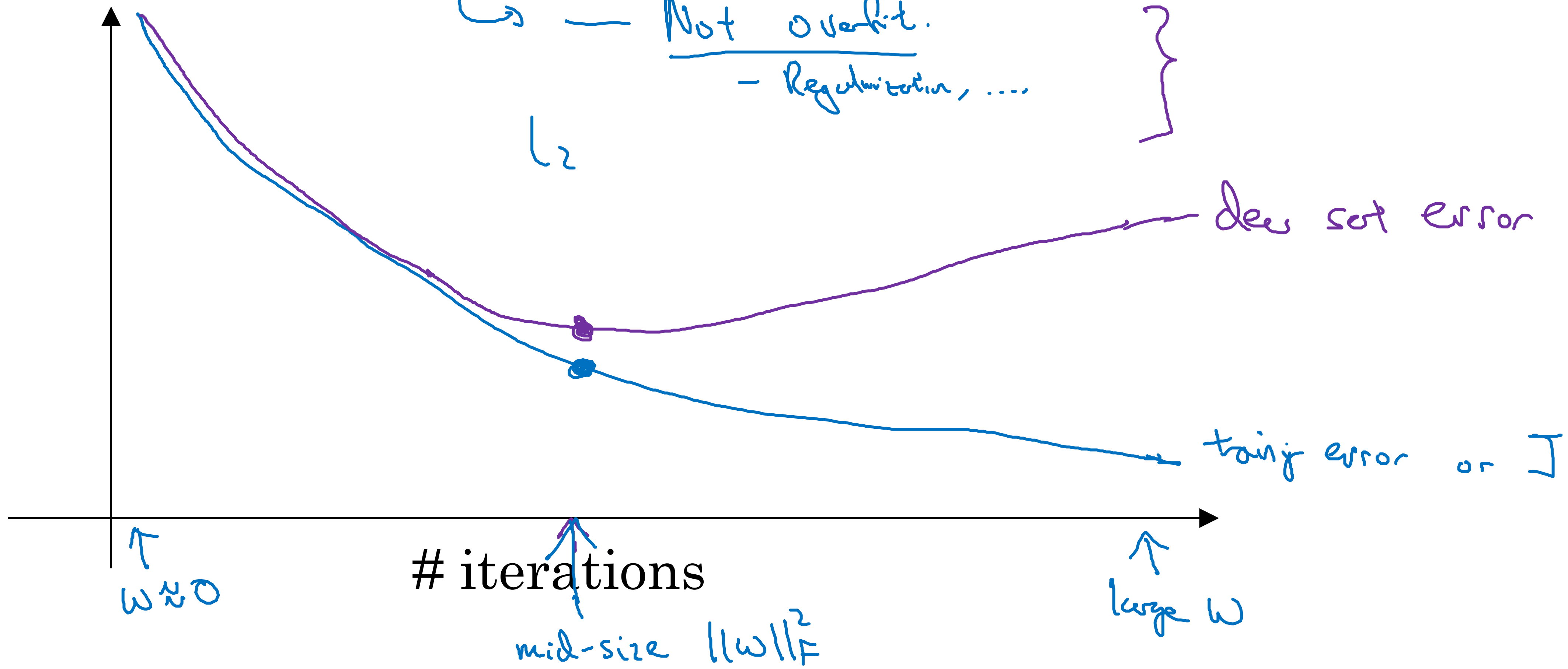
Early stopping

Orthogonalization.

- Optimize cost function J
  - Gradient, ...
- Not overfit.
  - Regularization, ...

$J(w,b)$

L_2



Andrew Ng

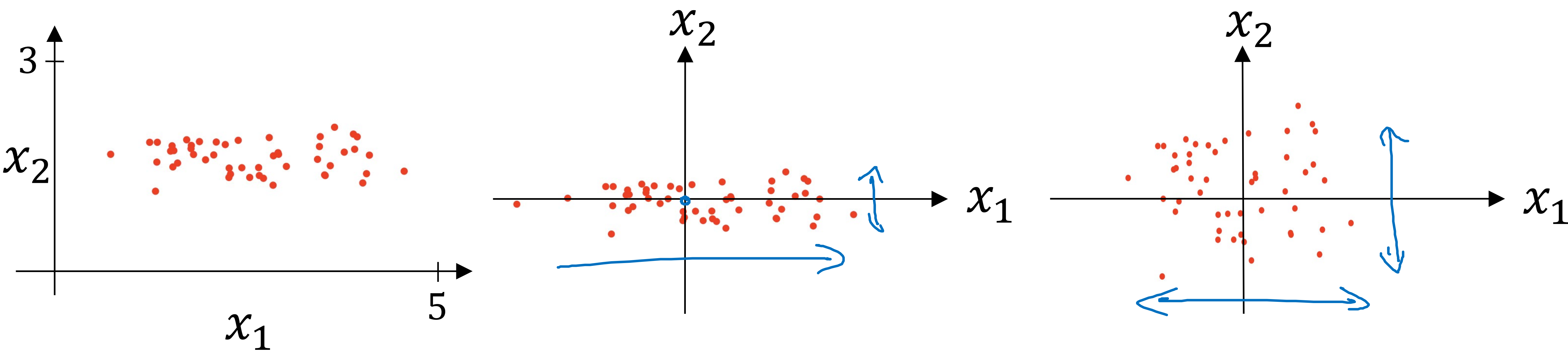
deeplearning.ai

Setting up your optimization problem

Normalizing inputs

Normalizing training sets

$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$



Subtract mean:

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

$$\boxed{x := x - \mu}$$

Normalize variance

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

← element-wise

$$\boxed{x \mathrel{/}= \sigma^2}$$

Use same μ, σ^2 to normalize test set.

Why normalize inputs?

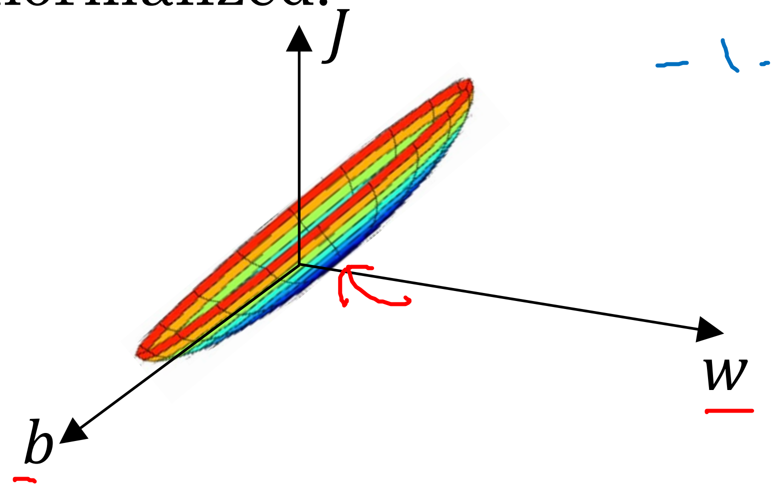
$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

w_1 x_1: $1 \ldots 1000$ ←

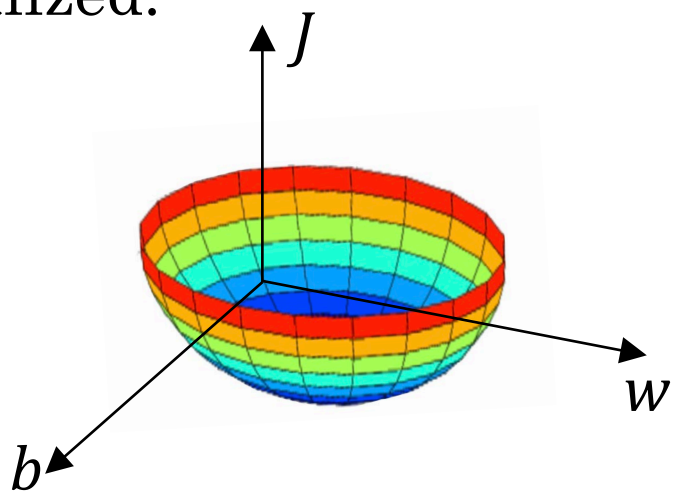
w_2 x_2: $0 \ldots 1$ ←

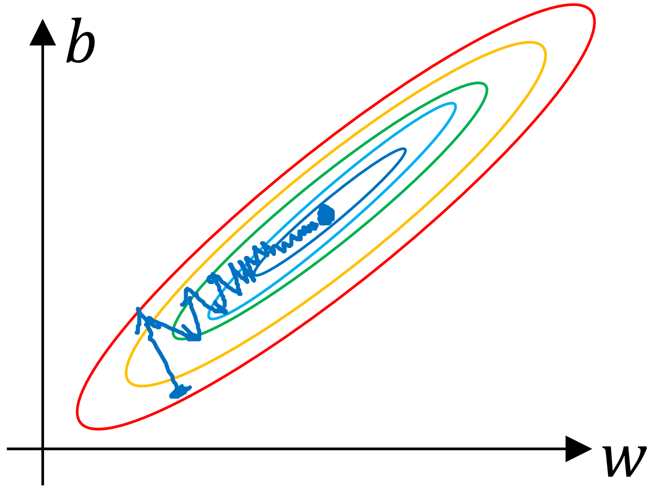
$-1 \ldots 1$

Unnormalized:



Normalized:

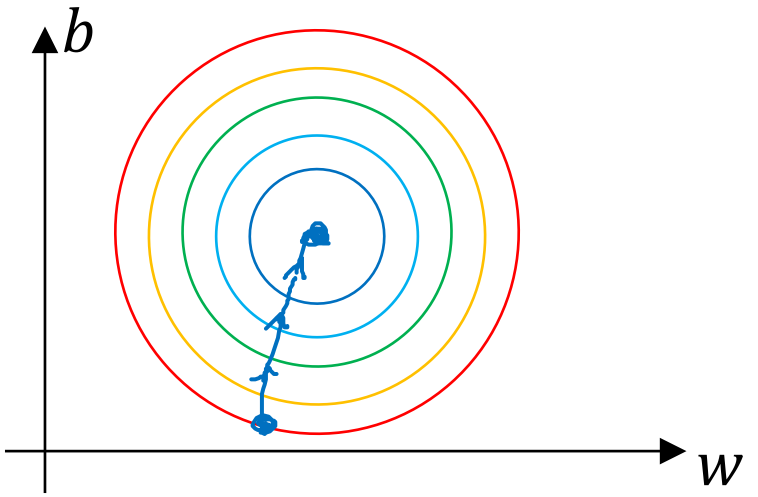




x_1: $0 \ldots 1$

x_2: $-1 \ldots 1$

x_3: $1 \ldots 2$



Andrew Ng

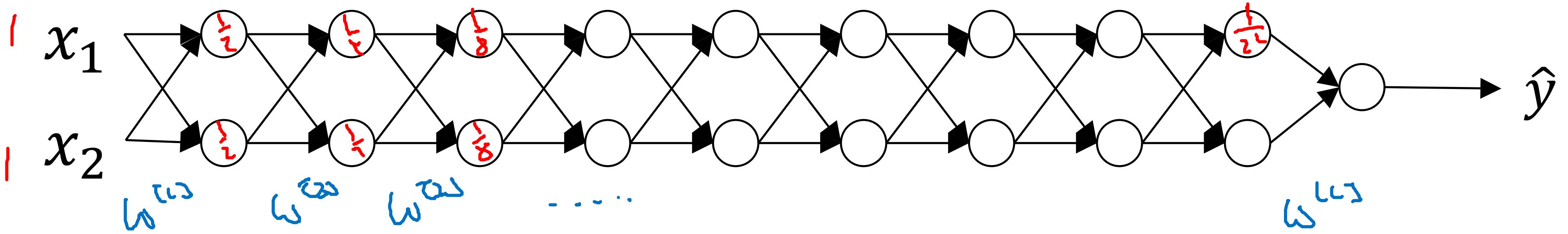
deeplearning.ai

Setting up your optimization problem

Vanishing/exploding gradients

Vanishing/exploding gradients

$L=150$



L

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

$$\hat{y} = W^{[L]}\underbrace{W^{[L-1]}W^{[L-2]}\cdots W^{[3]}W^{[2]}W^{[1]}x}$$

$a^{[3]}$

1.5^L

0.5^L

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

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

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

$W^{[l]} > I$

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

$$W^{[l]} = \begin{bmatrix}\cancel{1.5}\ 0.5 & 0 \\ 0 & \cancel{1.5}\ 0.5\end{bmatrix}$$

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

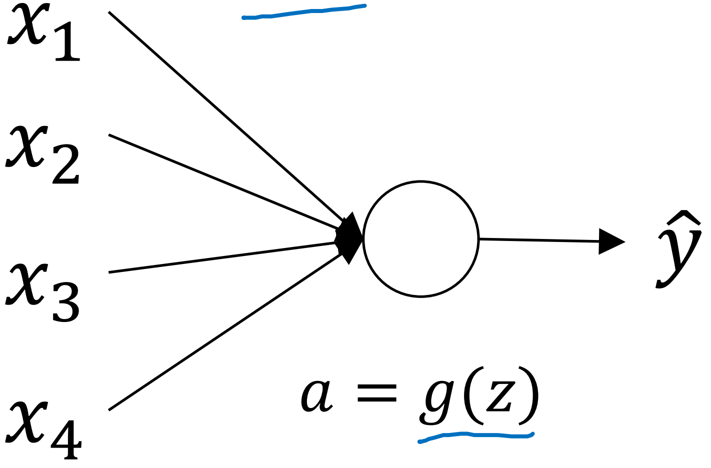
$1.5^{L-1}x$

$0.5^{L-1}x$

Andrew Ng

Single neuron example

$a^{[l]}$



$a = g(z)$

$W^{[l]}$

$z = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + \cancel{b}$

Large n → smaller w_i

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

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

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

Other variants:

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

Xavier initialization

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

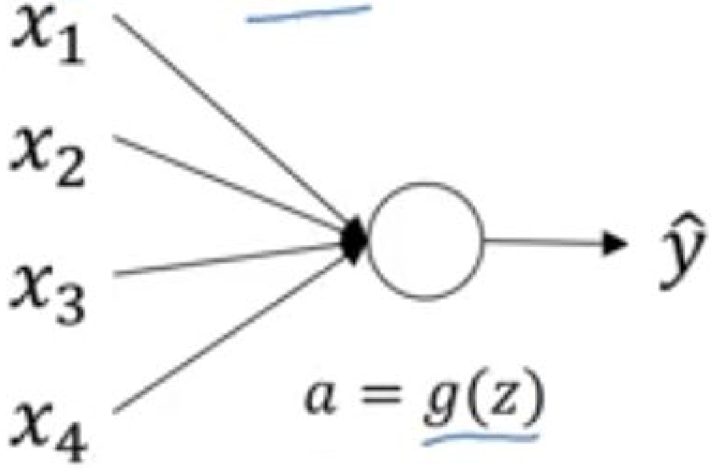
Andrew Ng

deeplearning.ai

Setting up your optimization problem

Weight initialization for deep networks

Single neuron example



$a^{[l]}$

$w^{[l]}$

$z = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n \cancel{+b}$

large n → smaller w_i

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

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

ReLU $\qquad g^{[l]}(z) = \mathrm{ReLU}(z)$

Other variants:

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

Xavier initialization

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

Andrew Ng

deeplearning.ai

Setting up your optimization problem

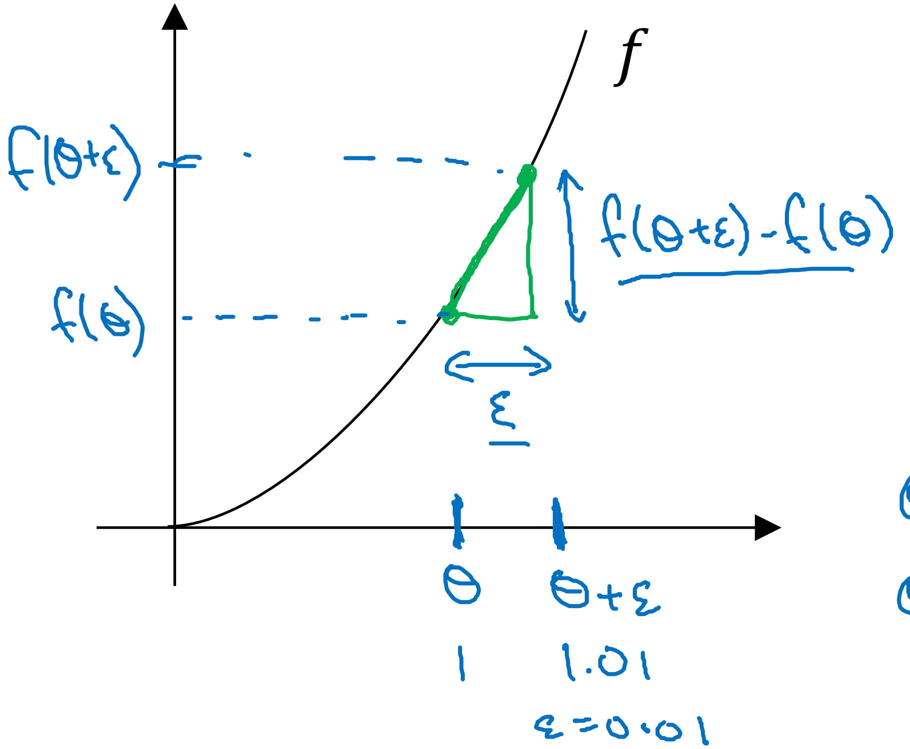
Numerical approximation of gradients

Checking your derivative computation

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

J

$\theta \in \mathbb{R}$



$g(\theta) = \frac{d}{d\theta} f(\theta) = f'(\theta)$

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

dw

db

$g(\theta) = 3 \cdot (1)^2 = 3$

when $\theta = 1$

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

$$\frac{(1.01)^3 - 1^3}{0.01} = 3.0301 \approx 3$$

$\theta = 1$

$\theta + \varepsilon = 1.01$

0.0301

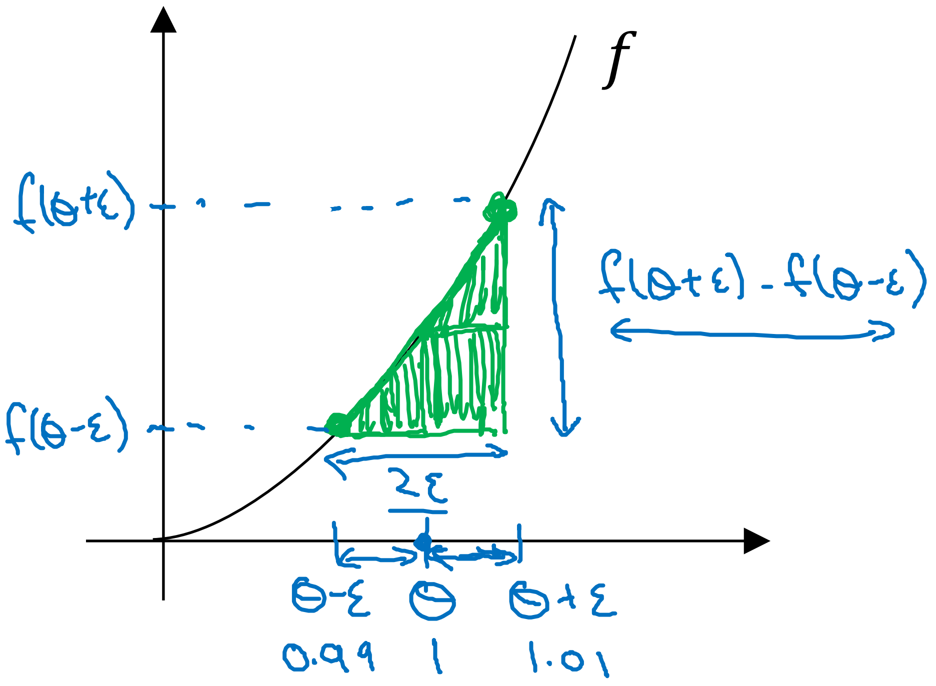
3.1

3.2

Andrew Ng

Checking your derivative computation

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



$\varepsilon = 0.01$

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

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

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

approx error: 0.0001

(prev slide: 3.0301. error: 0.03)

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

0.01

0.0001

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

0.01

Andrew Ng

deeplearning.ai

Setting up your optimization problem

Gradient Checking

Gradient check for a neural network

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

concatenate

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

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

concatenate

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

Andrew Ng

Gradient checking (Grad check)

$J(\theta) = J(\theta_1, \theta_2, \theta_3, \ldots)$

for each i:

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

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

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

Check

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

10^{-5}

$\rightarrow 10^{-3}$ - worry.

$\varepsilon = 10^{-7}$

Andrew Ng

deeplearning.ai

Setting up your optimization problem

Gradient Checking implementation notes

Gradient checking implementation notes

- Don't use in training – only to debug

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

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

$\underline{db^{[l]}} \qquad \underline{dW^{[l]}}$

- Remember regularization.

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

$d\theta$ = gradient of J w.r.t. θ

- Doesn't work with dropout. $\quad J \qquad$ $\underline{\text{keep-prob} = 1.0}$

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

$\underline{W, b \approx 0}$

Andrew Ng

deeplearning.ai

Setting up your optimization problem

Gradient Checking implementation notes

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.

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

$d\theta$ = gradient 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.

$W, b \approx 0$

Andrew Ng

deeplearning.ai

Optimization Algorithms

Mini-batch gradient descent

Batch vs. mini-batch gradient descent

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

Vectorization allows you to efficiently compute on m examples.

$$X = [\underbrace{x^{(1)}\ x^{(2)}\ x^{(3)} \dots x^{(1000)}}_{X^{\{1\}}\ (n_x,1000)} \mid \underbrace{x^{(1001)} \dots x^{(2000)}}_{X^{\{2\}}\ (n_x,1000)} \mid \dots \mid \underbrace{\dots x^{(m)}}_{X^{\{5,000\}}\ (n_x,1000)}]$$

(n_x, m)

$$Y = [\underbrace{y^{(1)}\ y^{(2)}\ y^{(3)} \dots y^{(1000)}}_{Y^{\{1\}}\ (1,1000)} \mid \underbrace{y^{(1001)} \dots y^{(2000)}}_{Y^{\{2\}}\ (1,1000)} \mid \dots \mid \underbrace{\dots y^{(m)}}_{Y^{\{5,000\}}\ (1,1000)}]$$

$(1, m)$

What if $m = 5{,}000{,}000$?

5,000 mini-batches of 1,000 each

Mini-batch t: $X^{\{t\}}, Y^{\{t\}}$

$x^{(i)}$

$z^{[l]}$

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

Andrew Ng

Mini-batch gradient descent

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

X, Y

repeat {

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

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

$$\left.\begin{aligned} Z^{[1]} &= W^{[1]} X^{\{t\}} + b^{[1]} \\ A^{[1]} &= g^{[1]}(Z^{[1]}) \\ &\vdots \\ A^{[L]} &= g^{[L]}(Z^{[L]}) \end{aligned}\right\}$$ Vectorized implementation (1000 examples)

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

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 epoch" — pass through training set.

Andrew Ng

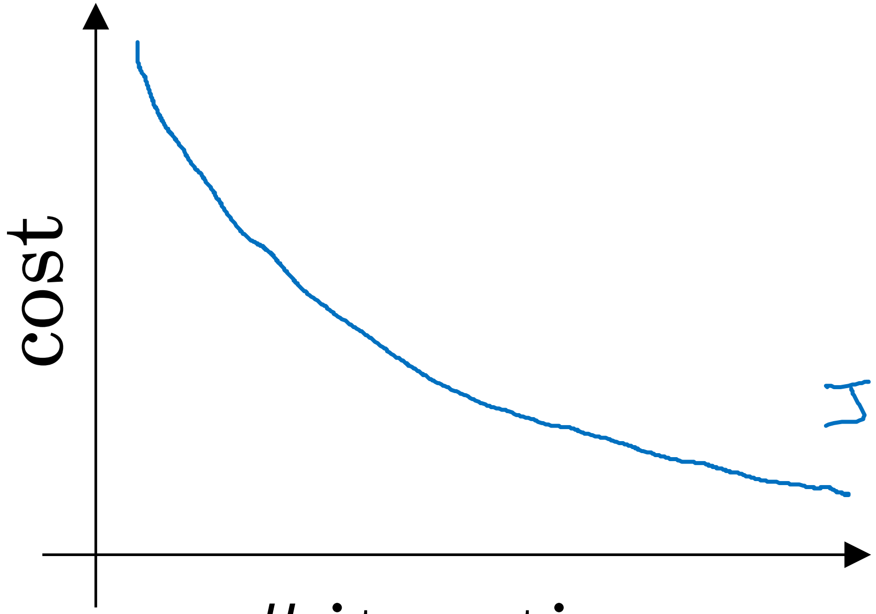
deeplearning.ai

Optimization Algorithms

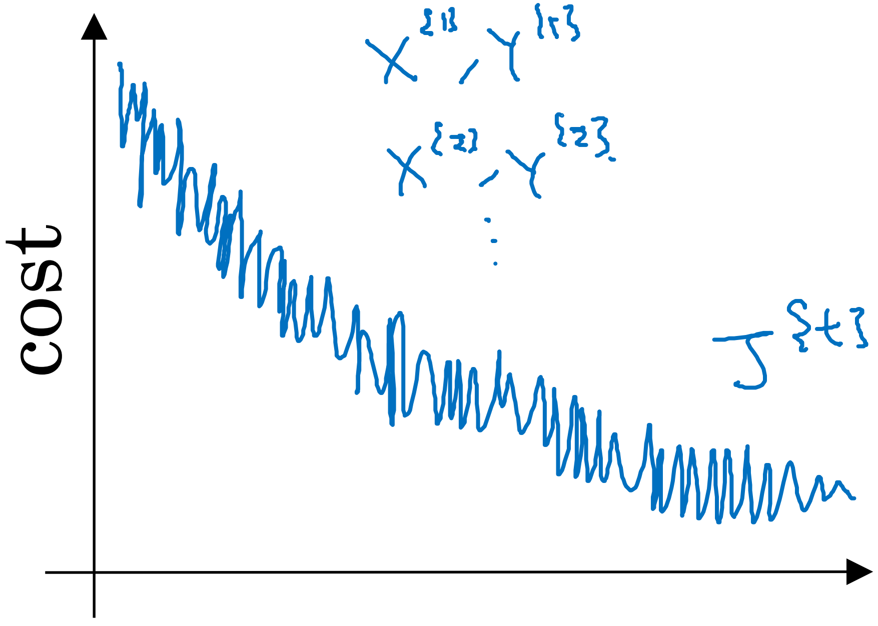
Understanding mini-batch gradient descent

Training with mini batch gradient descent

Batch gradient descent



Mini-batch gradient descent



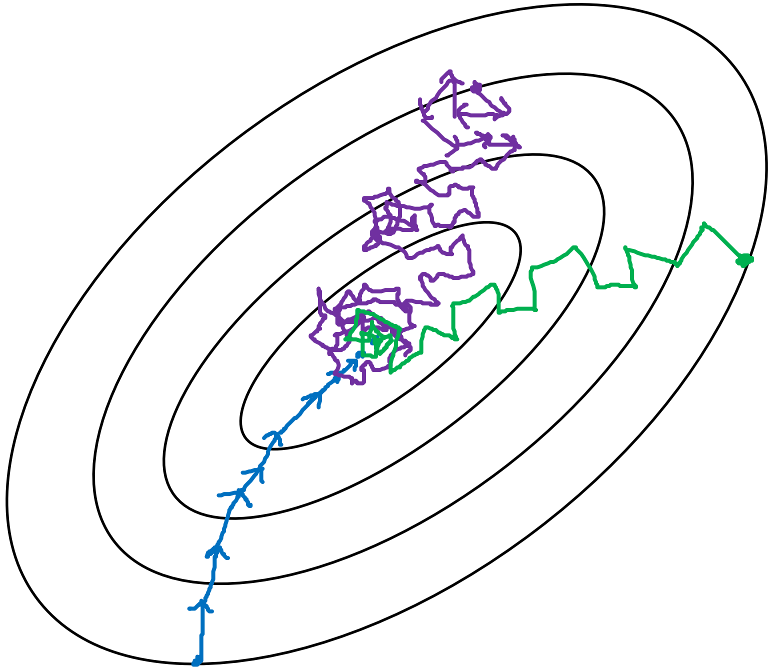
Plot $J^{\{t\}}$ computed using $\underline{X^{\{t\}}, Y^{\{t\}}}$

Choosing your mini-batch size

→ If mini-batch size = m : Batch gradient descent. $(X^{\{1\}}, Y^{\{1\}}) = (X, Y)$.

→ If mini-batch size = 1 : Stochastic gradient descent. Every example is it own
$(X^{\{1\}}, Y^{\{1\}}) = (x^{(1)}, y^{(1)})$ ... $(x^{(2)}, y^{(2)})$ mini-batch.

In practice: Somewh in-between 1 and m



Stochastic gradient descent
↓
Lose speedup from vectorization

In-between (mini-batch size not too big/small)
↓
Fastest learning.
- Vectorization. (~1000)
- Make progress without processing entire training set.

Batch gradient descent (mini-batch size = m)
↓
Too long per iteration

Andrew Ng

Choosing your mini-batch size

If small training set: Use batch gradient descent.
($m \leq 2000$)

Typical mini-batch sizes:

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

Make sure mini-batch fit in CPU/GPU memory.
$X^{\{t\}}, Y^{\{t\}}$

Andrew Ng

deeplearning.ai

Optimization Algorithms

Exponentially weighted averages

Temperature in London

$\theta_1 = 40°F$ 4°C ←

$\theta_2 = 49°F$ 9°C

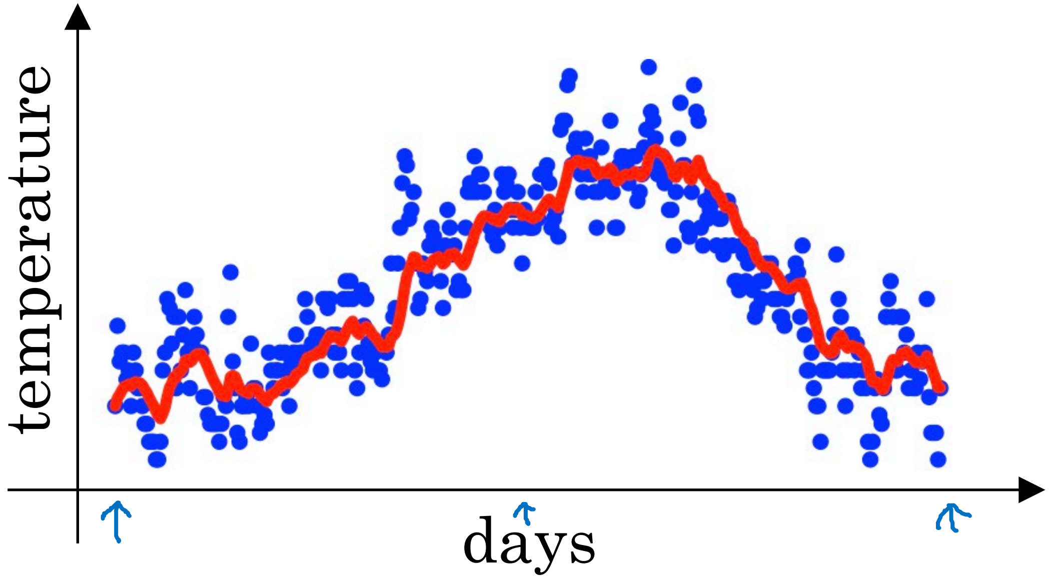
$\theta_3 = 45°F$ ⋮

⋮

$\theta_{180} = 60°F$ 15°C

$\theta_{181} = 56°F$ ⋮

⋮



$V_0 = 0$

$V_1 = 0.9\, V_0 + 0.1\, \theta_1$

$V_2 = 0.9\, V_1 + 0.1\, \theta_2$

$V_3 = 0.9\, V_2 + 0.1\, \theta_3$

⋮

$$\boxed{V_t = 0.9\, V_{t-1} + 0.1\, \theta_t}$$

Andrew Ng

Exponentially weighted (moving) averages

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

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

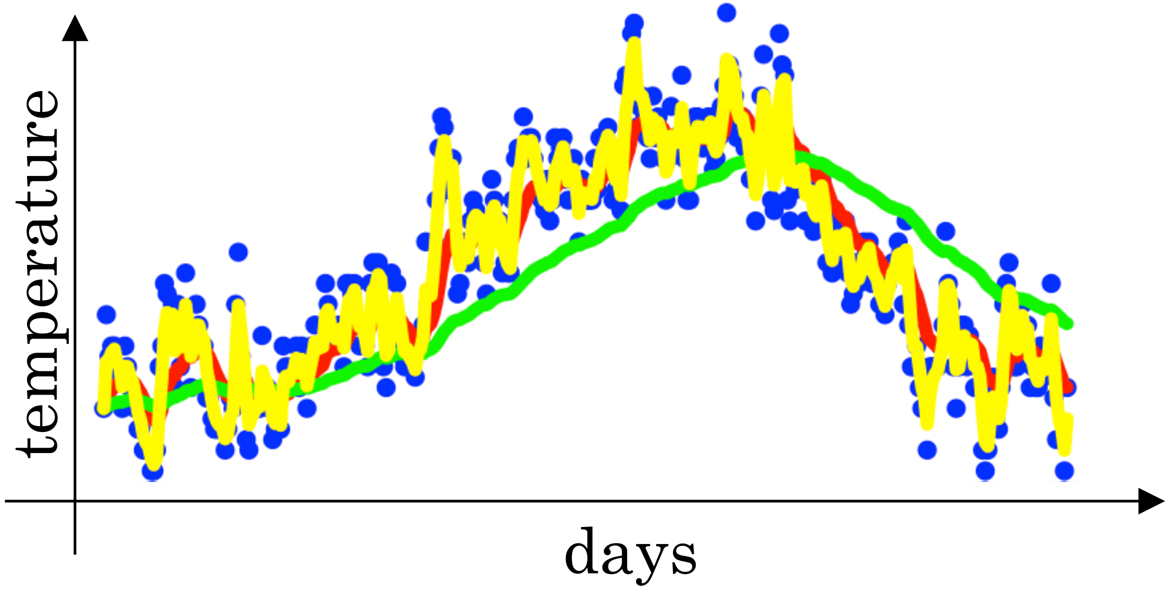
$\beta = 0.98$: ≈ 50 days

$\beta = 0.5$: ≈ 2 days

V_t as approximately average over

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

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



Andrew Ng

deeplearning.ai

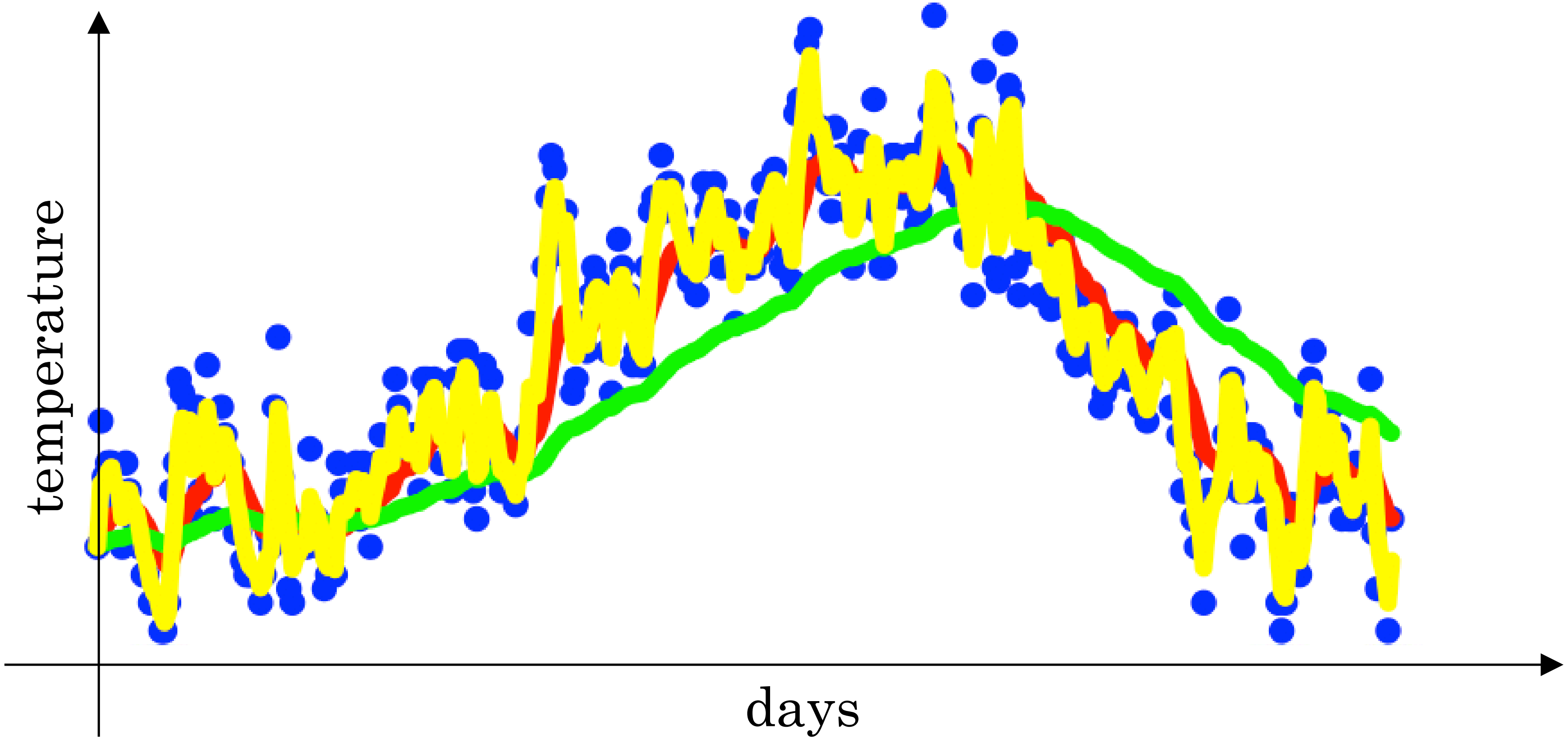
Optimization Algorithms

Understanding exponentially weighted averages

Exponentially weighted averages

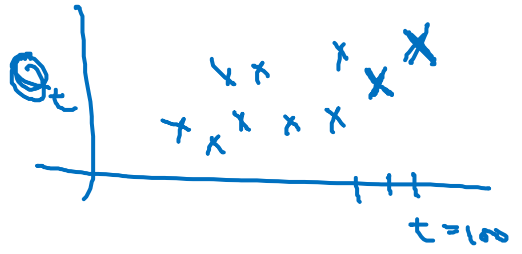
$$\boxed{v_t = \beta v_{t-1} + (1-\beta)\theta_t}$$

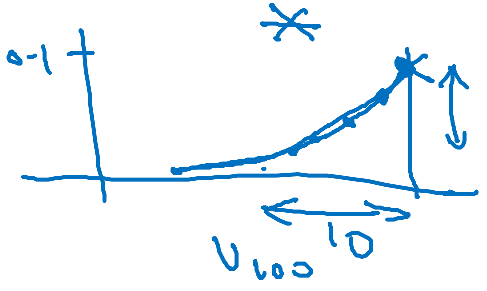
$\beta = 0.9$ 0.98 0.5



Exponentially weighted averages

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





$\approx \frac{1}{1-\beta}$

$\varepsilon = 1-\beta$

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

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

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

...

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

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

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

$0.9^{10} \approx 0.35 \approx \frac{1}{e}$

$\underbrace{(1-\varepsilon)}_{0.9}{}^{1/\varepsilon} = \frac{1}{e}$

0.98 ?

$\varepsilon = 0.02 \to 0.98^{50} \approx \frac{1}{e}$

Andrew Ng

Implementing exponentially weighted averages

$$v_0 = 0$$
$$v_1 = \beta v_0 + (1-\beta)\,\theta_1$$
$$v_2 = \beta v_1 + (1-\beta)\,\theta_2$$
$$v_3 = \beta v_2 + (1-\beta)\,\theta_3$$
$$\ldots$$

$V_\theta := 0$

$V_\theta := \beta V + (1-\beta)\,\theta_1$

$V_\theta := \beta V + (1-\beta)\,\theta_2$

$\vdots$

$V_\theta = 0$

Repeat {

Get next θ_t

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

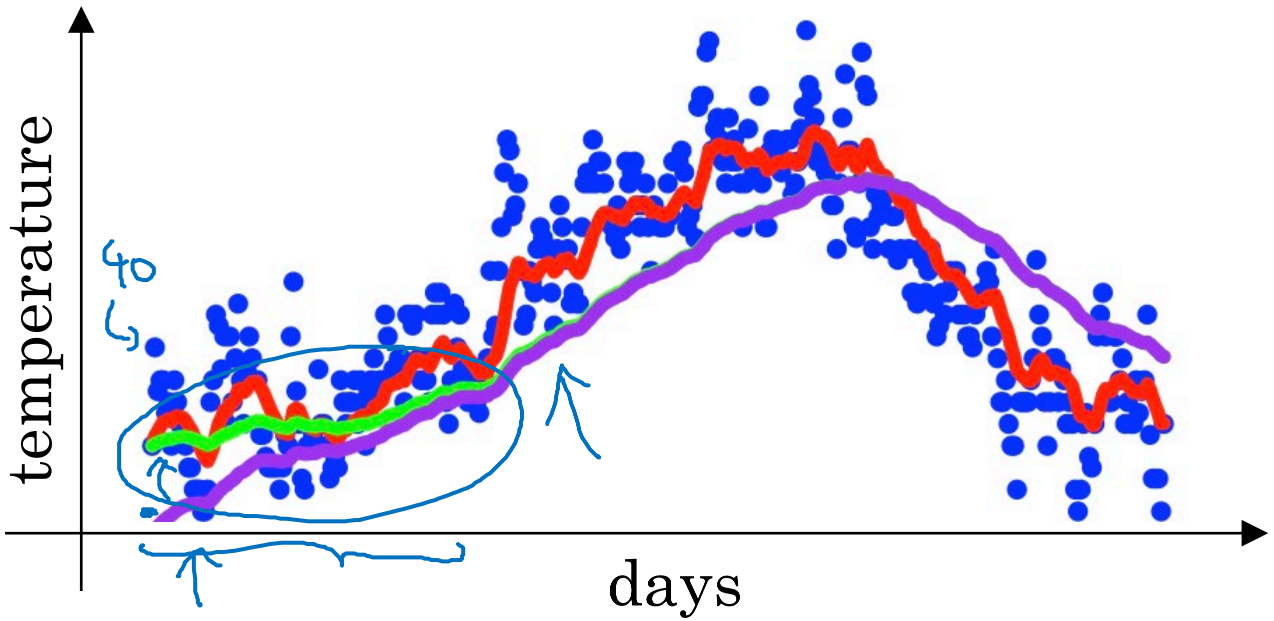
}

deeplearning.ai

Optimization Algorithms

Bias correction in exponentially weighted average

Bias correction



$\beta = 0.98$

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

$v_0 = 0$

$v_1 = \cancel{0.98 v_0 +} \; 0.02\,\theta_1$

$v_2 = 0.98\, v_1 + 0.02\,\theta_2$

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

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

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

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

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

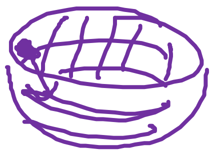
Andrew Ng

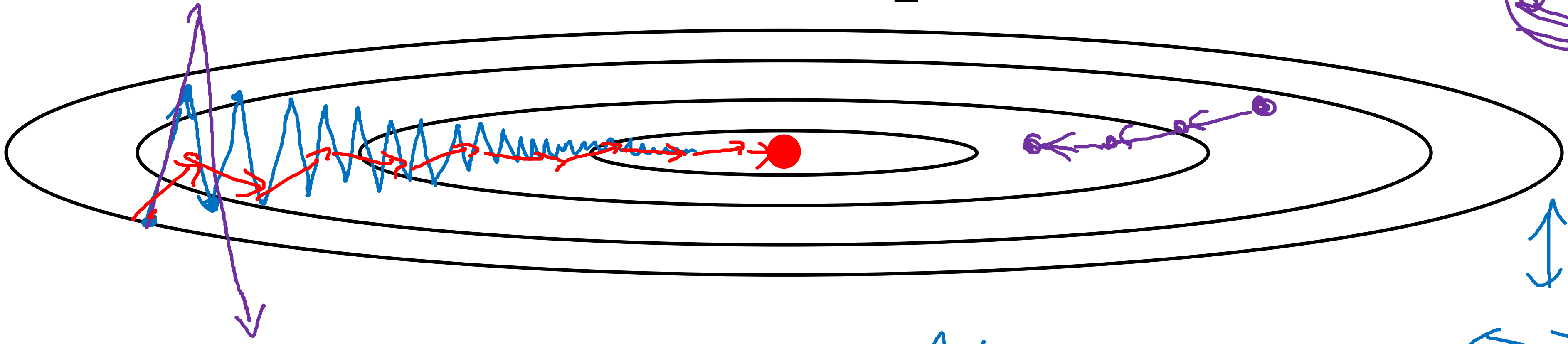
deeplearning.ai

Optimization Algorithms

Gradient descent with momentum

Gradient descent example





$\updownarrow$ slower learning

$\leftrightarrow$ faster learning.

Momentum:

On iteration t:

Compute dW, db on current mini-batch.

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

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

friction, velocity, acceleration

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

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

Andrew Ng

Implementation details

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

On iteration t:

Compute dW, db on the current mini-batch

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

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

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

$V_{dw} = \beta V_{dw} + dW \leftarrow$

$\cancel{\dfrac{V_{dw}}{1-\beta^t}}$

Hyperparameters: α, β

$\beta = 0.9$

average over last ≈ 10 gradients

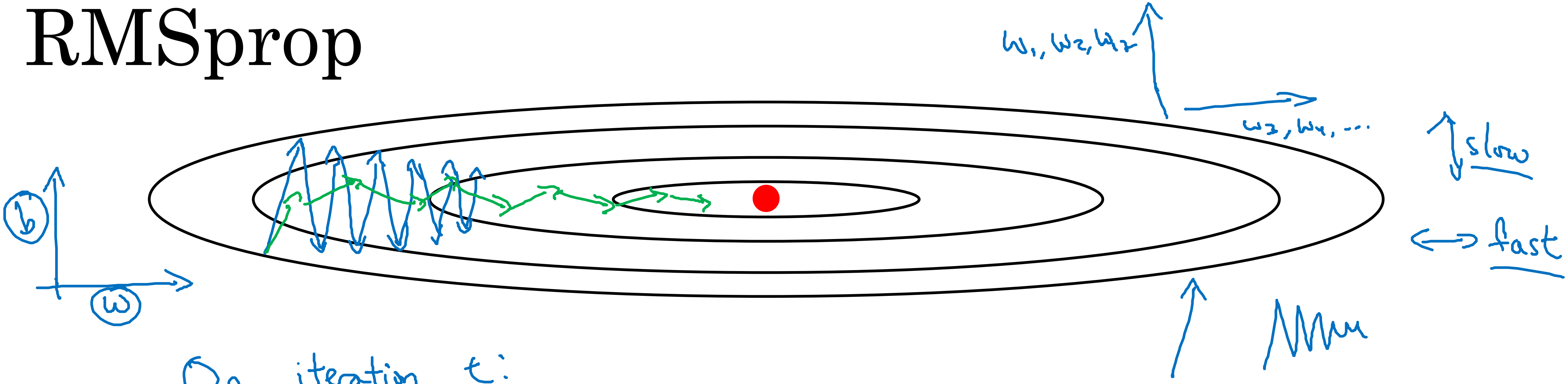
Andrew Ng

deeplearning.ai

Optimization Algorithms

RMSprop

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

$W := W - \alpha \frac{dW}{\sqrt{S_{dW}+\varepsilon}}$ ← ←

$b := b - \alpha \frac{db}{\sqrt{S_{db}+\varepsilon}}$ ← ←

$\varepsilon = 10^{-8}$

Andrew Ng

deeplearning.ai

Optimization Algorithms

Adam optimization algorithm

Adam optimization algorithm

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

On iteration t:

Compute dW, db using current mini-batch

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

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

yhat = np.array([.9, 0.2, 0.1, .4, .9])

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

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

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

Hyperparameters choice:

→ α : needs to be tune

→ β_1 : 0.9 → $(\underline{dw})$

→ β_2 : 0.999 → $(\underline{dw^2})$

→ ε : 10^{-8}

Adam: Adaptive moment estimation

Adam Coates

Andrew Ng

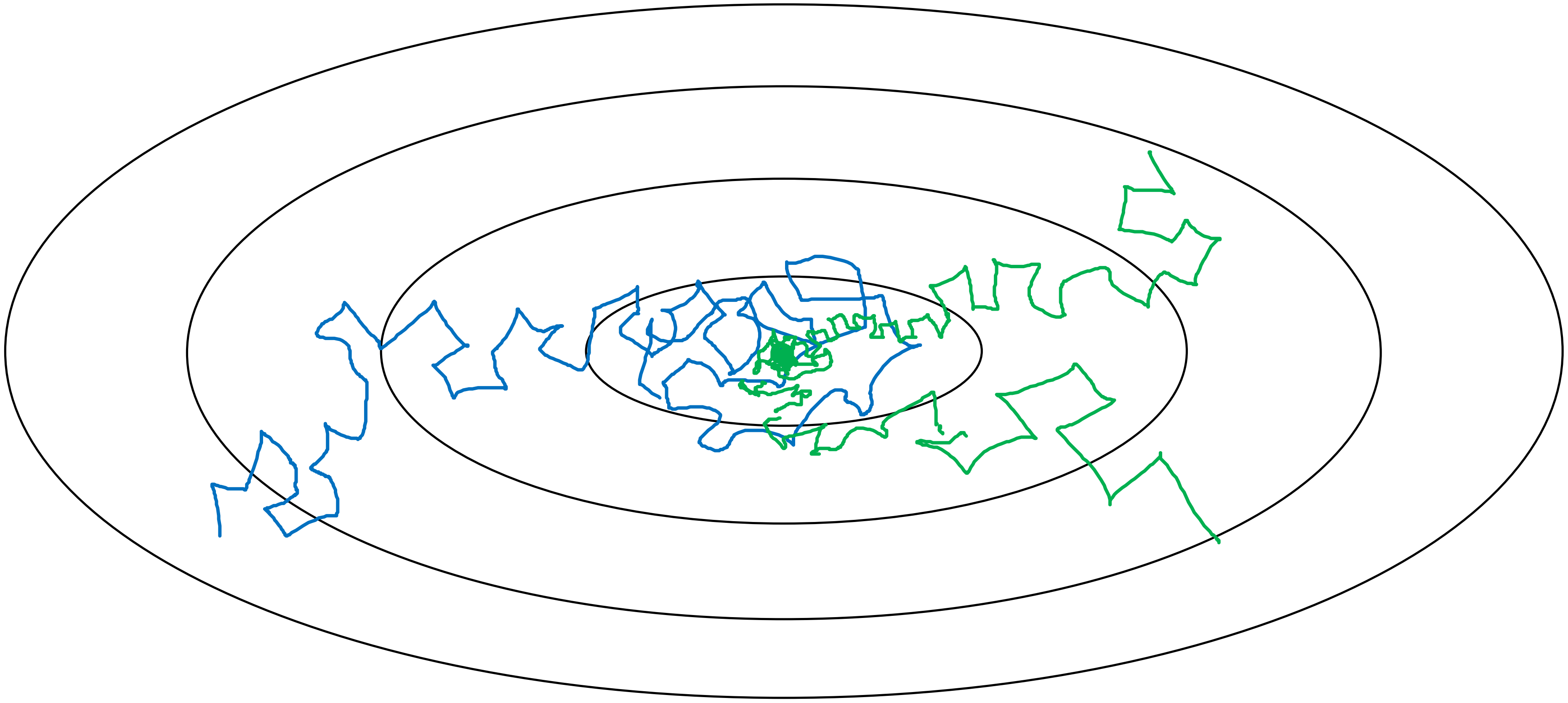
deeplearning.ai

Optimization Algorithms

Learning rate decay

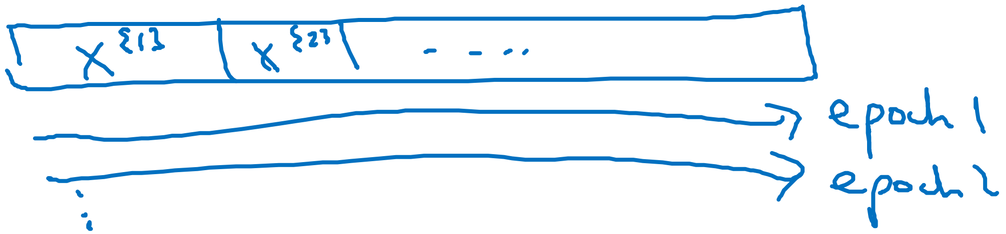
Learning rate decay

Slowly reduce α



Learning rate decay

1 epoch = 1 pass through data.

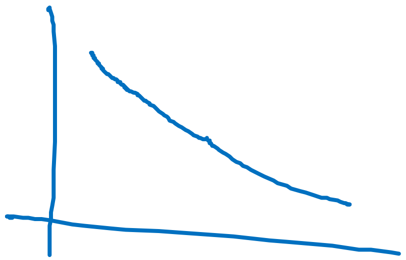


$$\alpha = \frac{1}{1 + \text{decay-rate} * \text{epoch-num}} \alpha_0$$

Epoch	α
1	0.1
2	0.67
3	0.5
4	0.4
⋮	⋮

$\alpha_0 = 0.2$

decay.rate = 1



Other learning rate decay methods

formula

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

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

discrete staircase

Manual decay.

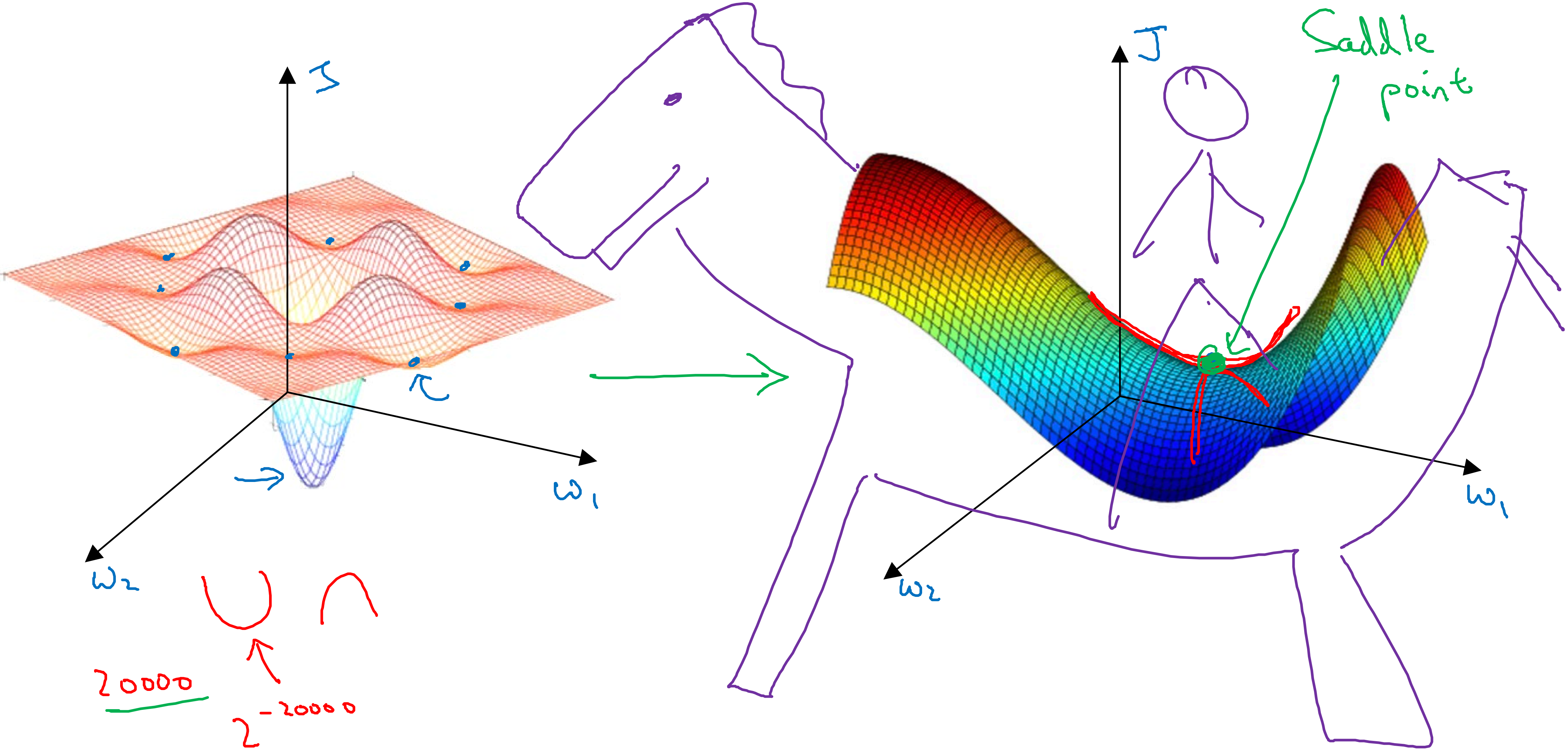
Andrew Ng

deeplearning.ai

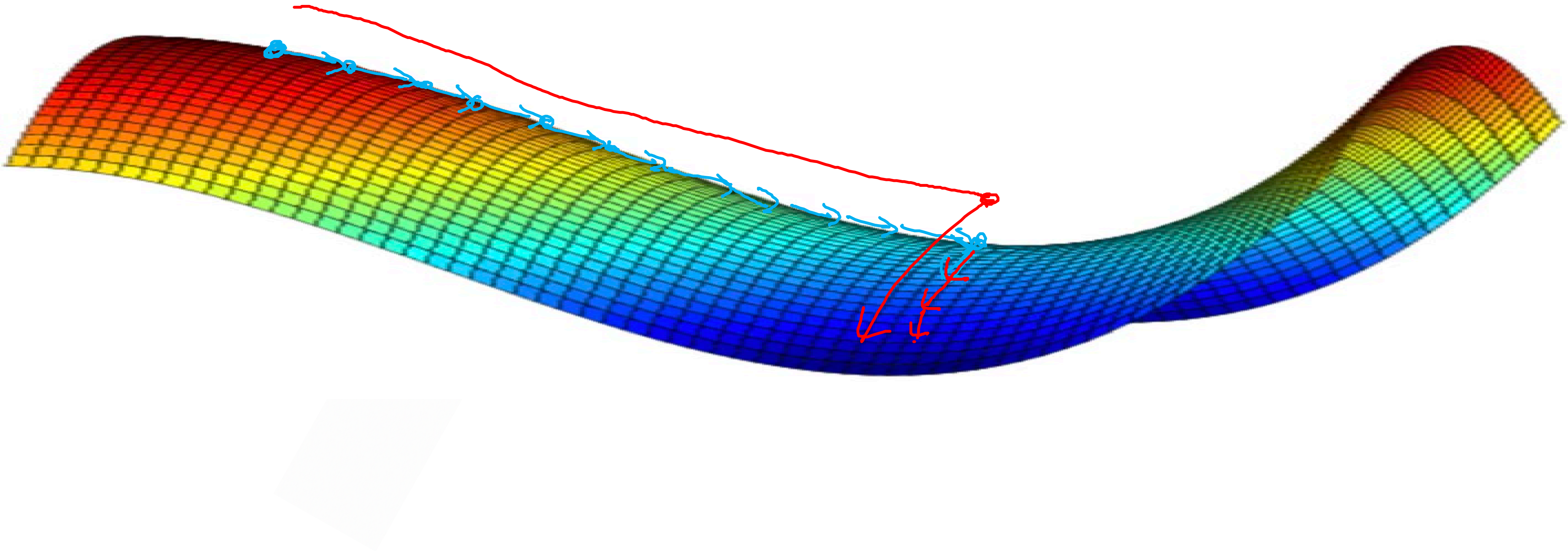
Optimization Algorithms

The problem of local optima

Local optima in neural networks



Problem of plateaus



- Unlikely to get stuck in a bad local optima
- Plateaus can make learning slow

deeplearning.ai

Hyperparameter tuning

Tuning process

Hyperparameters

$\rightarrow \boxed{\alpha}$

$\boxed{\beta}$ ~0.9

$\underset{0.9}{\beta_1}$, $\underset{0.999}{\beta_2}$, $\underset{10^{-8}}{\varepsilon}$

layers

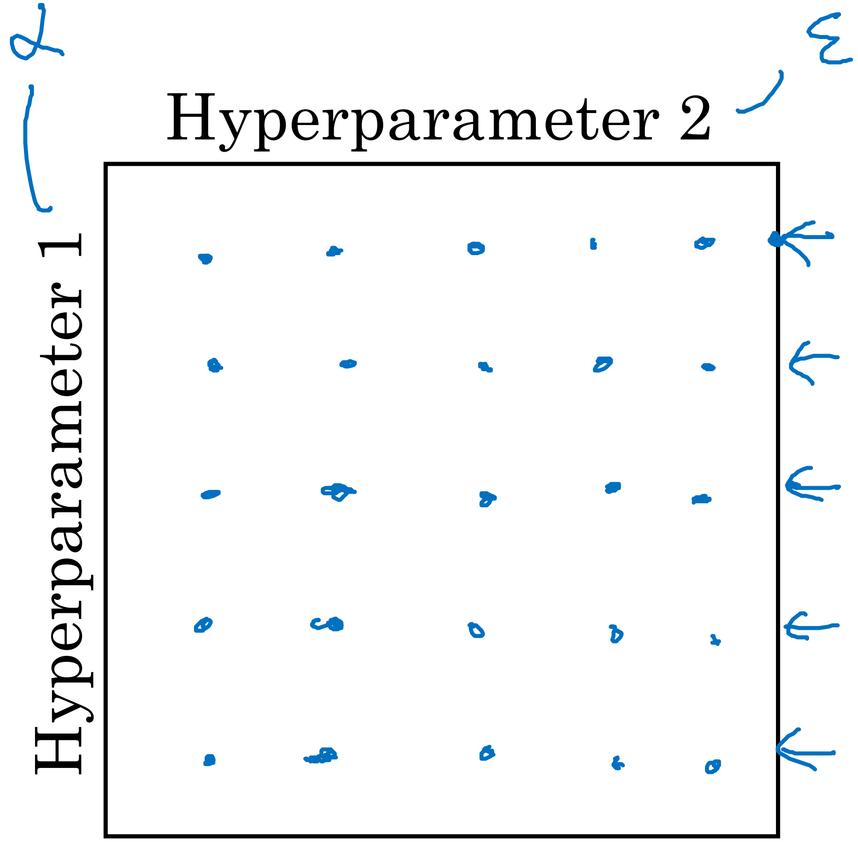
hidden units

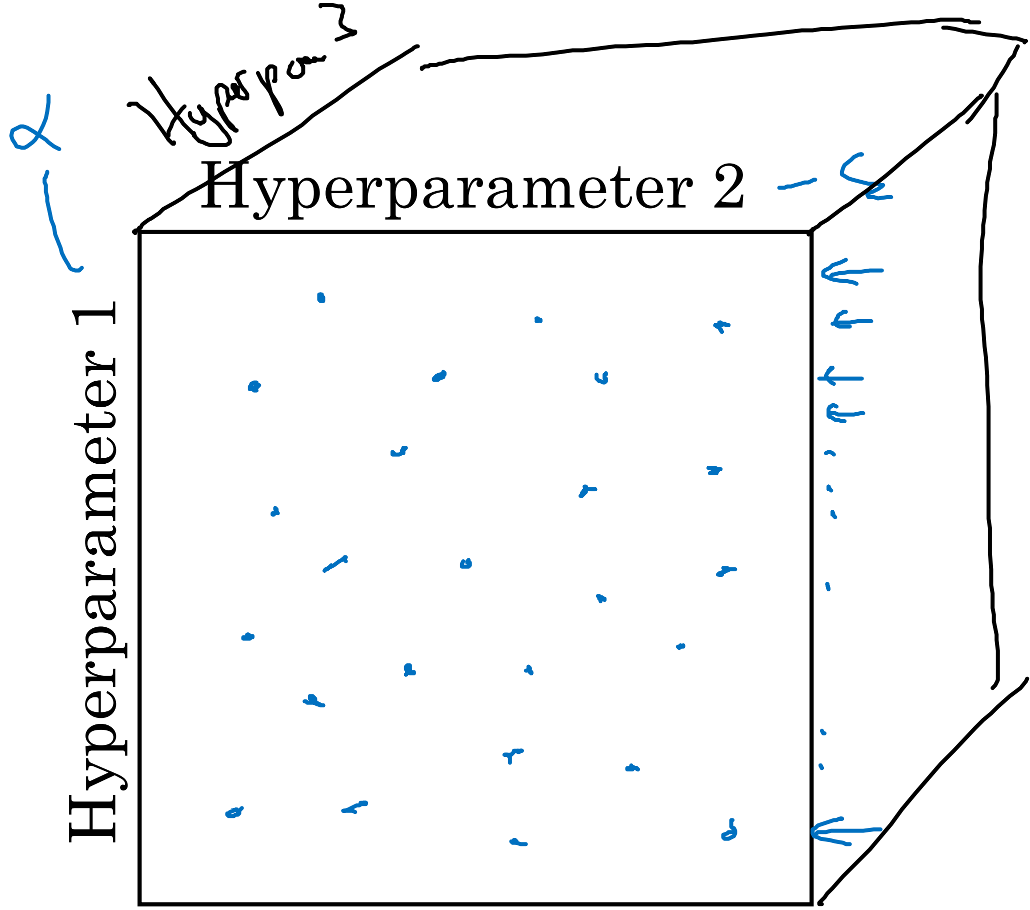
learning rate decay

mini-batch size

Andrew Ng

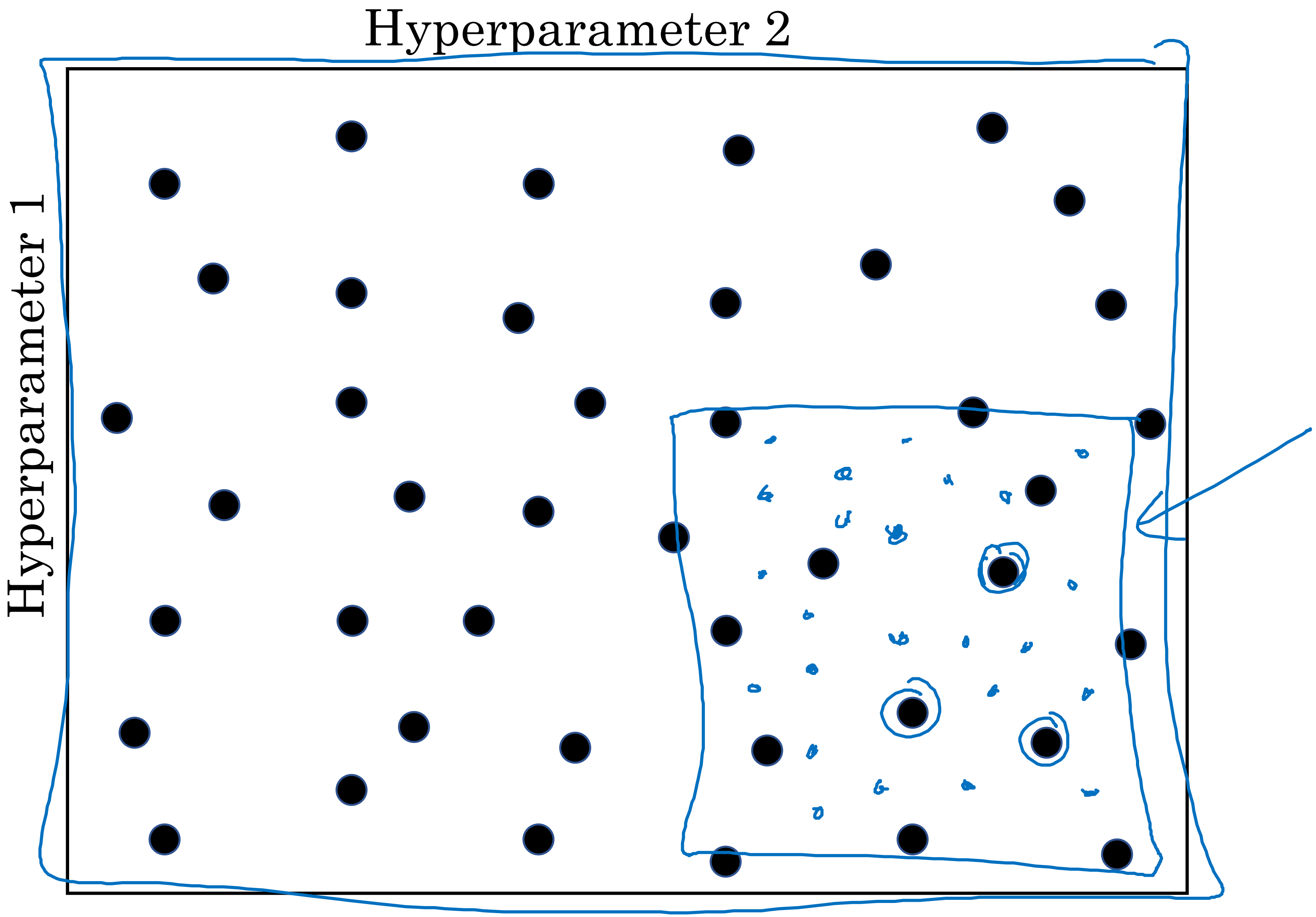
Try random values: Don't use a grid





Andrew Ng

Coarse to fine



Andrew Ng

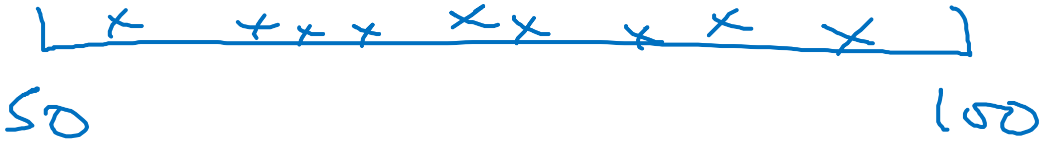
deeplearning.ai

Hyperparameter tuning

Using an appropriate scale to pick hyperparameters

Picking hyperparameters at random

→ $n^{[l]} = 50, \ldots, 100$



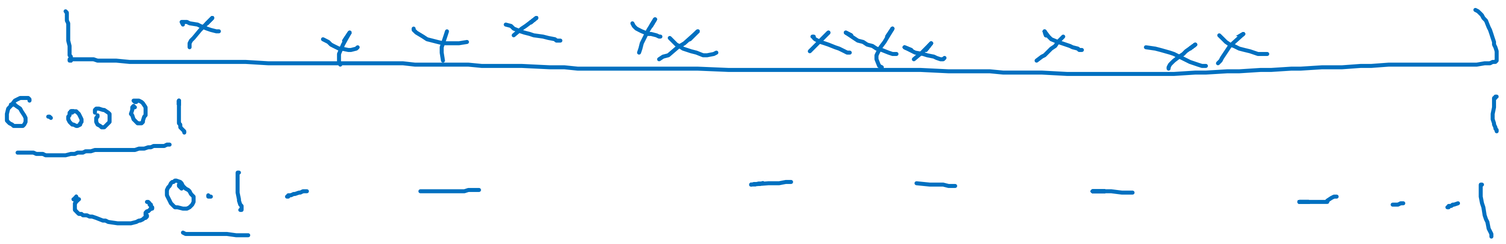
→ #layers L: 2 – 4

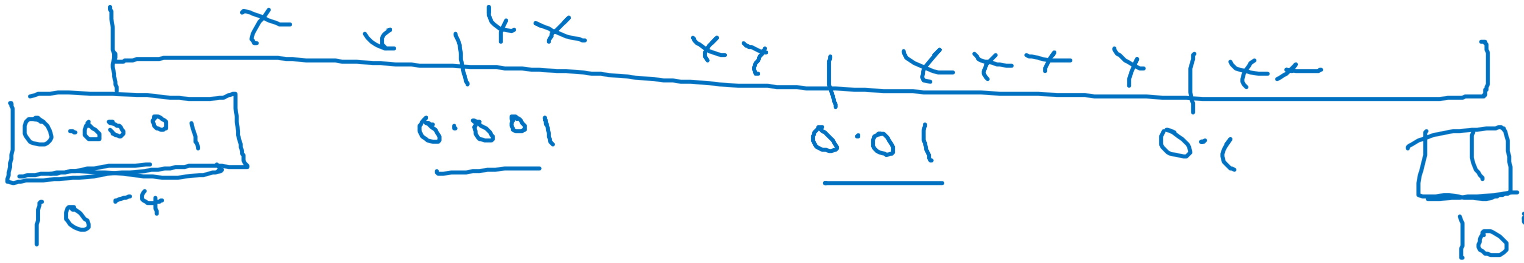
2, 3, 4

Andrew Ng

Appropriate scale for hyperparameters

$\alpha = 0.0001, \ldots, 1$





10^a $\quad$ 10^b

$a = \log_{10} 0.0001 = -4$ $\qquad$ $b = \log_{10} 1 = 0$

$r = -4 * np.random.rand()$ $\leftarrow$ $r \in [-4, 0]$

$\alpha = 10^r$ $\leftarrow$ $10^{-4} \ldots 10^{0}$

$10^a \ldots 10^b$ $\qquad$ $r \in [a, b]$ $\quad$ $[-4, 0]$ $\qquad$ $\alpha = 10^r$

Andrew Ng

Hyperparameters for exponentially weighted averages

$\beta = 0.9 \quad \ldots \quad 0.999$

$\downarrow \qquad\qquad \downarrow$

$10 \qquad\qquad 1000$

$1-\beta = 0.1 \quad \ldots \quad 0.001$

$\beta: 0.9000 \rightarrow 0.9005 \Big\} \sim 10$

$\beta: 0.999 \rightarrow 0.9995$

$\sim 1000 \qquad \sim 2000$

$\frac{1}{1-\beta}$

$0.9 \qquad\qquad 0.999$

$0.9 \qquad 0.99 \qquad 0.999$

$0.1 \qquad 0.01 \qquad 0.001$

$10^{-1} \qquad\qquad 10^{-3}$

$r \in [-3, -1]$

$1-\beta = 10^r$

$\beta = 1-10^r$

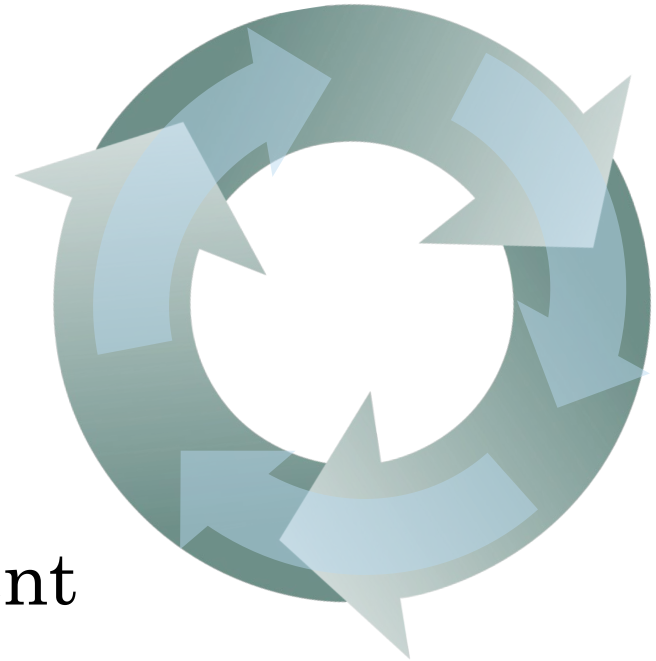
Andrew Ng

deeplearning.ai

Hyperparameters tuning

Hyperparameters tuning in practice: Pandas vs. Caviar

Re-test hyperparameters occasionally

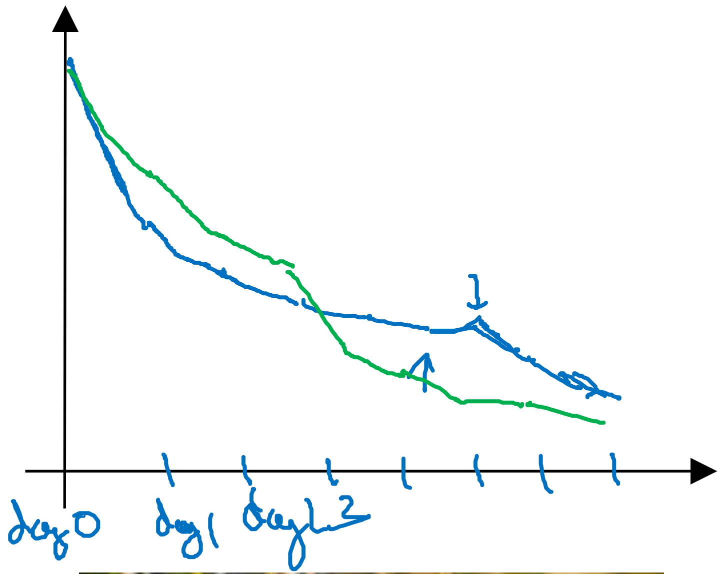


- NLP, Vision, Speech, Ads, logistics, ...

- Intuitions do get stale. Re-evaluate occasionally.

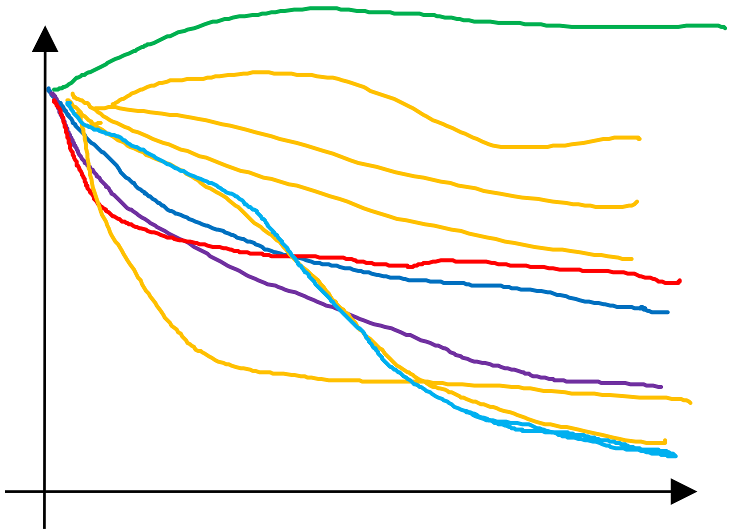
Andrew Ng

Babysitting one model



Panda

Training many models in parallel



Caviar

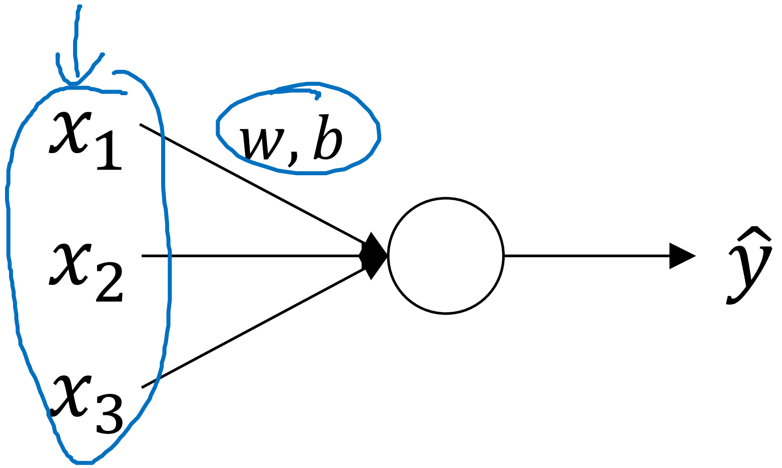
Andrew Ng

deeplearning.ai

Batch Normalization

Normalizing activations in a network

Normalizing inputs to speed up learning

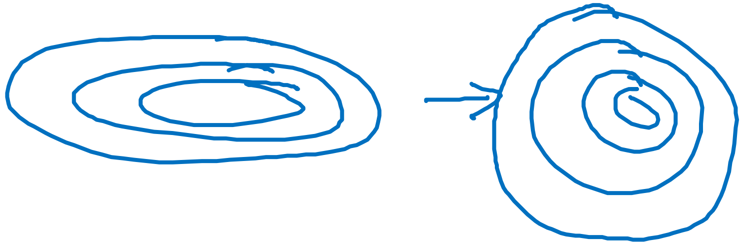


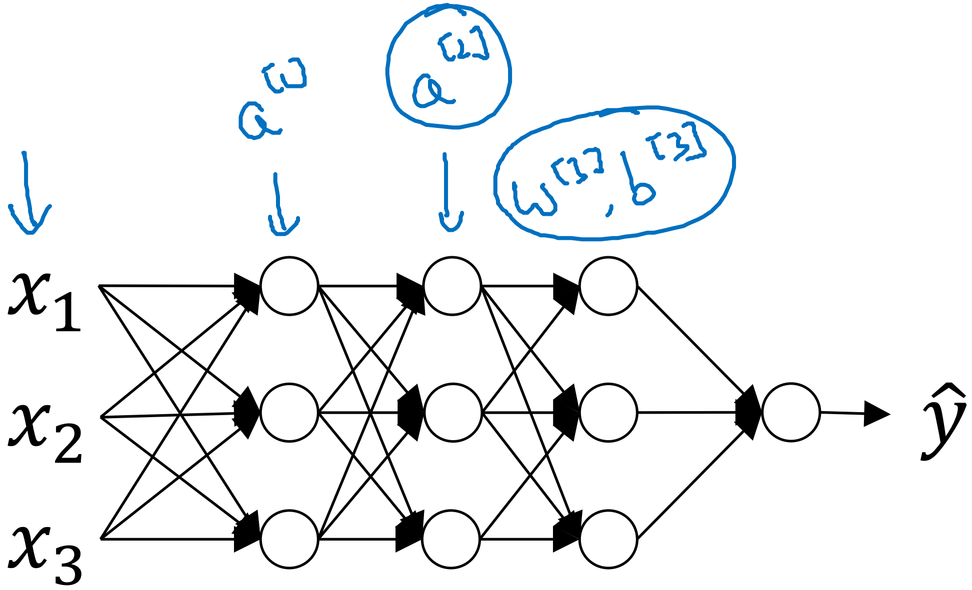
$\mu = \frac{1}{m} \sum_i x^{(i)}$

$X = X - \mu$ ← element-wise

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

$X = X / \sigma^2$





Can we normalize $\underline{a^{[2]}}$ so as to train $W^{[3]}, b^{[3]}$ faster

Normalize $\underline{z^{[2]}}$

Implementing Batch Norm

Given some intermediate values in NN $\underbrace{z^{(1)}, \ldots, z^{(m)}}_{z^{[l](i)}}$

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

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

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

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

learnable parameters of model.

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

If

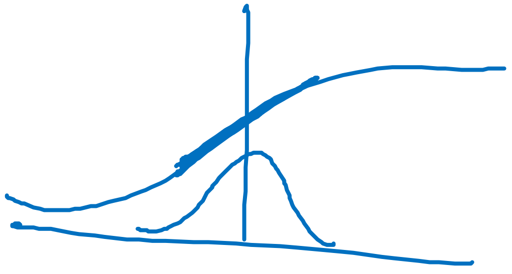
$$\gamma = \sqrt{\sigma^2 + \varepsilon}$$

$$\beta = \mu$$

then $\tilde{z}^{(i)} = z^{(i)}$

X

$z^{(i)}$

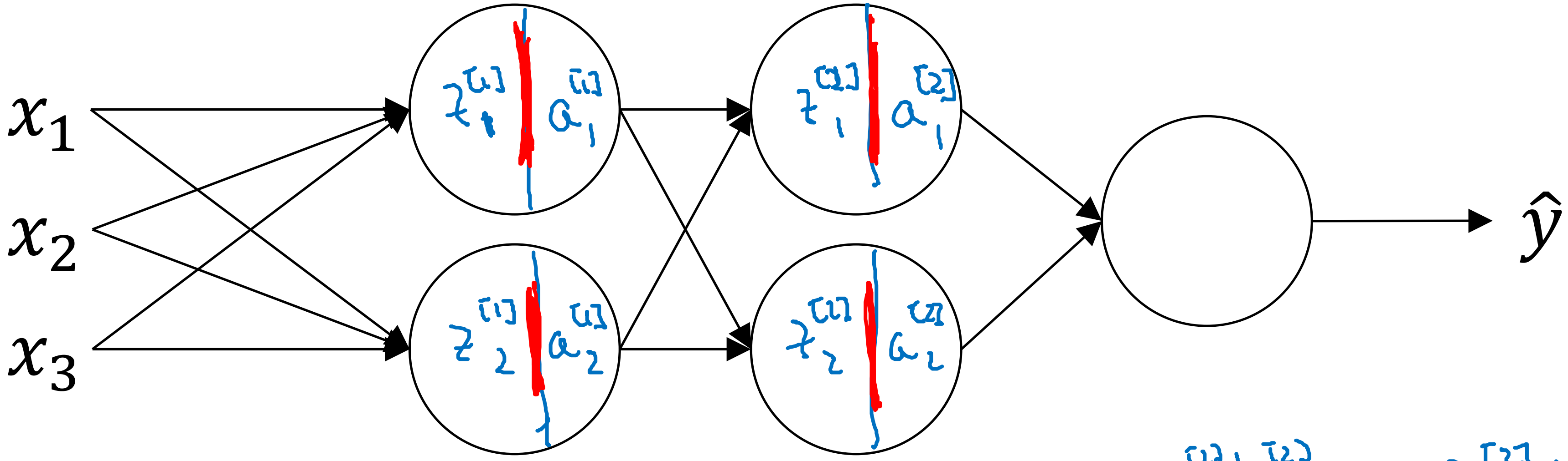


deeplearning.ai

Batch Normalization

Fitting Batch Norm into a neural network

Adding Batch Norm to a network



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

Parameters: $\left.\begin{array}{l} W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}, \ldots, W^{[L]}, b^{[L]}, \\ \rightarrow \beta^{[1]}, \gamma^{[1]}, \beta^{[2]}, \gamma^{[2]}, \ldots, \beta^{[L]}, \gamma^{[L]} \end{array}\right]$

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

tf.nn.batch_normalization $\Leftarrow$

$\rightarrow \beta$

Andrew Ng

Working with mini-batches

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

$$\boxed{X^{\{2\}}} \rightarrow z^{[1]} \xrightarrow[BN]{\beta^{[1]},\, \gamma^{[1]}} \tilde{z}^{[1]} \rightarrow \cdots$$

$$X^{\{3\}} \rightarrow \cdots$$

Parameters: $W^{[l]}, \cancel{b^{[l]}}, \beta^{[l]}, \gamma^{[l]}$.

$\cancel{b^{[l]}}$: $(n^{[l]}, 1)$, $\beta^{[l]}$: $(n^{[l]}, 1)$, $\gamma^{[l]}$: $(n^{[l]}, 1)$

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

$$z^{[l]} = W^{[l]} a^{[l-1]} + \boxed{\cancel{b^{[l]}}}$$

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

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

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

Andrew Ng

Implementing gradient descent

for $t = 1 \ldots$ numMiniBatches

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

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

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

Update params

$$\left.\begin{aligned} W^{[l]} &:= W^{[l]} - \alpha\, dW^{[l]} \\ \beta^{[l]} &:= \beta^{[l]} - \alpha\, d\beta^{[l]} \\ \gamma^{[l]} &:= \ldots \end{aligned}\right\} \leftarrow$$

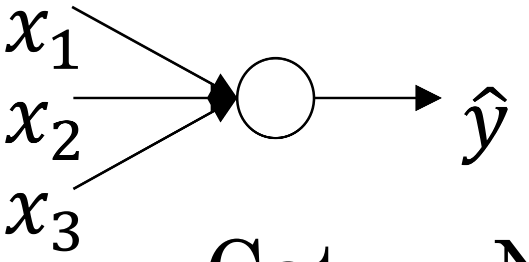
Works w/ momentum, RMSprop, Adam.

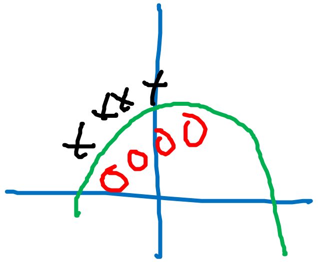
deeplearning.ai

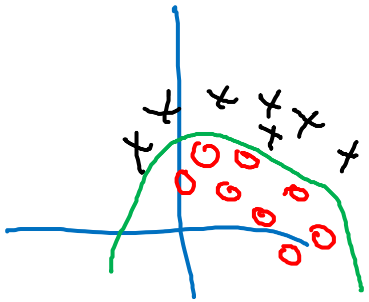
Batch Normalization

Why does Batch Norm work?

Learning on shifting input distribution





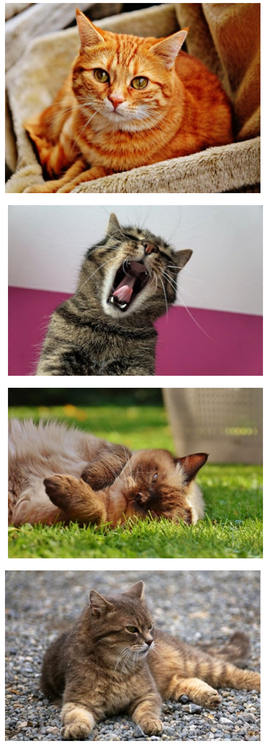


Cat	Non-Cat
$y = 1$	$y = 0$

"Covariate shift"

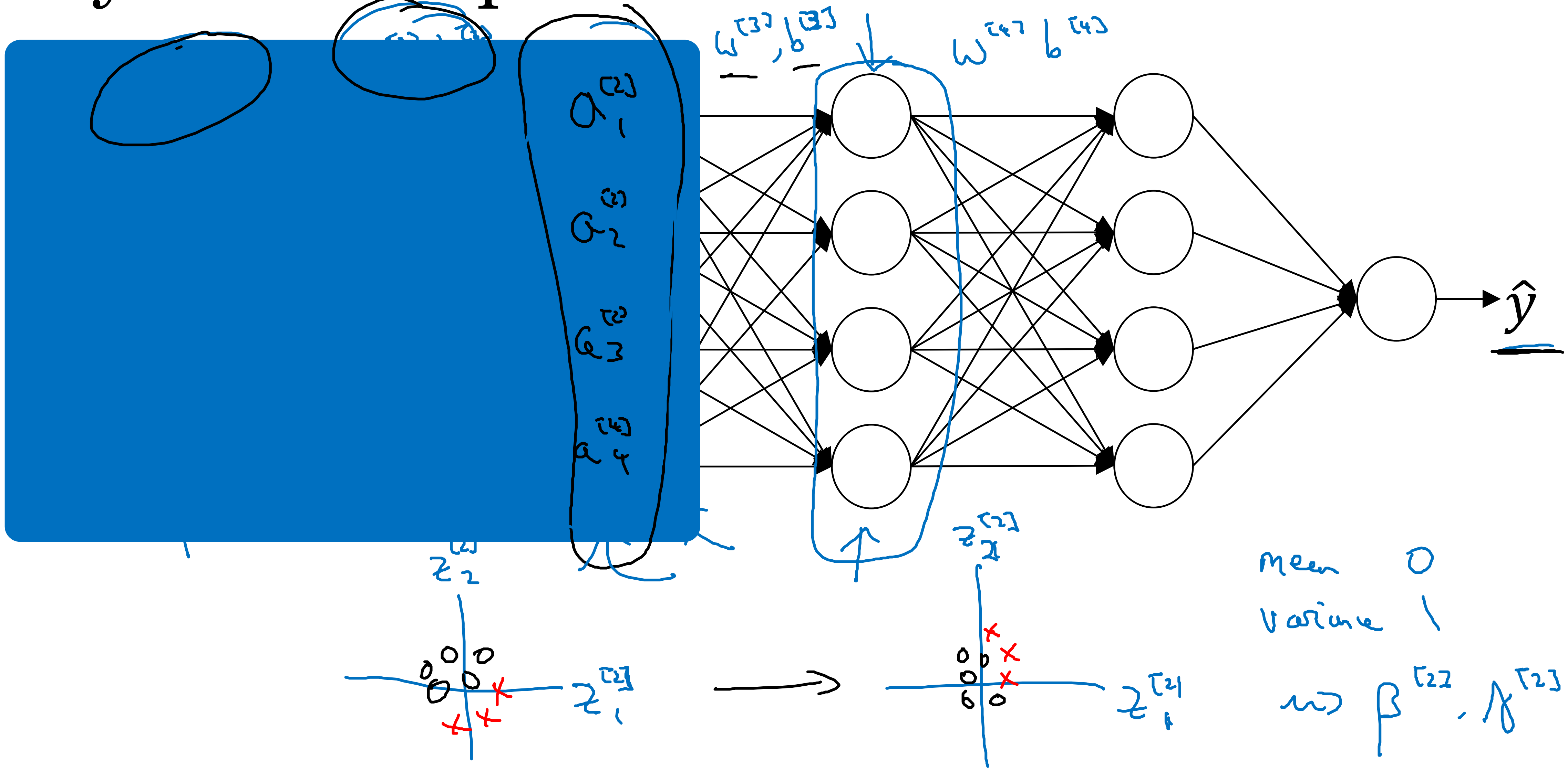
$X \rightarrow y$

$y = 1$ $\quad$ $y = 0$



Andrew Ng

Why this is a problem with neural networks?



Andrew Ng

Batch Norm as regularization

- Each mini-batch is scaled by the mean/variance computed on just that mini-batch.
- This adds some noise to the values $z^{[l]}$ within that minibatch. So similar to dropout, it adds some noise to each hidden layer's activations.
- This has a slight regularization effect.

X

$X^{\{t\}}$

$\tilde{z}^{[l]}$

64, 128

$z^{[l]}$

μ, σ^2

mini-batch : 64 $\longrightarrow$ 512

Andrew Ng

deeplearning.ai

Batch Normalization

Batch Norm at test time

Batch Norm at test time

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

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

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

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

μ, σ^2: estimate using exponentially weighted average (across mini-batches).

$X^{\{1\}}, X^{\{2\}}, X^{\{3\}}, \ldots$

$\cancel{\mu^{[l]}}$

$\mu^{\{1\}[l]}$ $\quad \mu^{\{2\}[l]}$ $\quad \mu^{\{3\}[l]}$ $\longrightarrow \mu$

θ_1 $\quad \theta_2$ $\quad \theta_3$ $\qquad \sigma^2$

$\sigma^{2\{1\}[l]}$ $\quad \sigma^{2\{2\}[l]}$ $\quad \ldots$

$z_{\text{norm}} = \dfrac{z - \mu}{\sqrt{\sigma^2 + \varepsilon}}$ $\qquad \tilde{z} = \gamma z_{\text{norm}} + \beta$

Andrew Ng

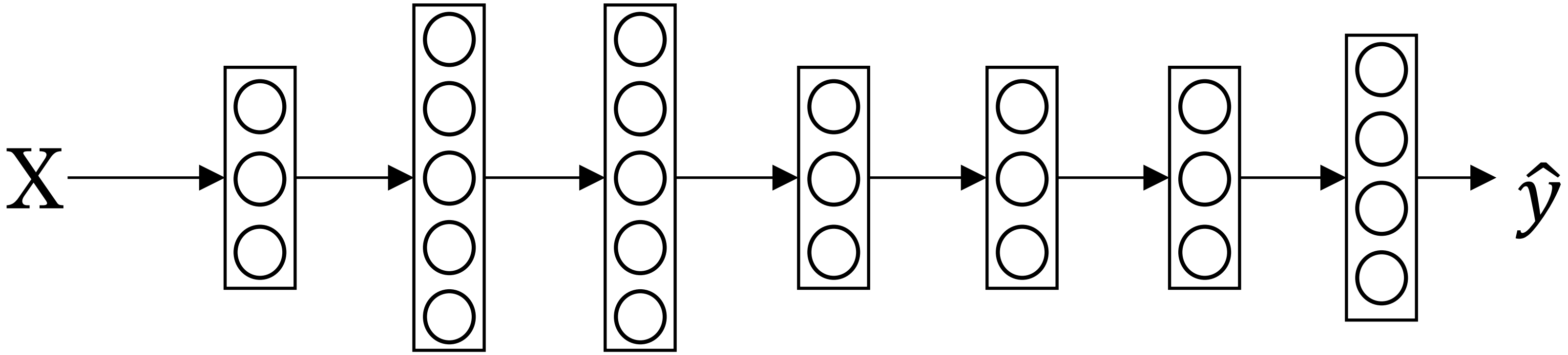
deeplearning.ai

Multi-class classification

Softmax regression

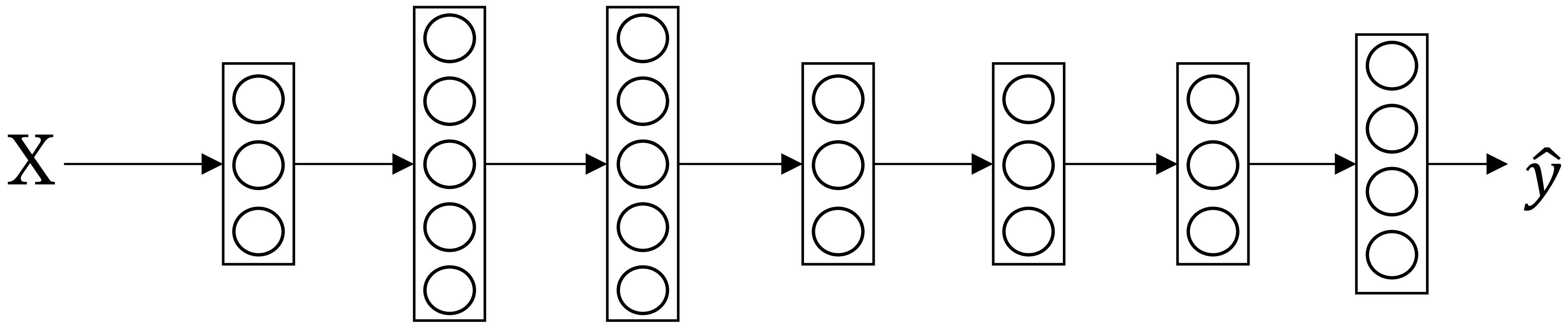
Recognizing cats, dogs, and baby chicks

3 1 2 0 3 2 0 1

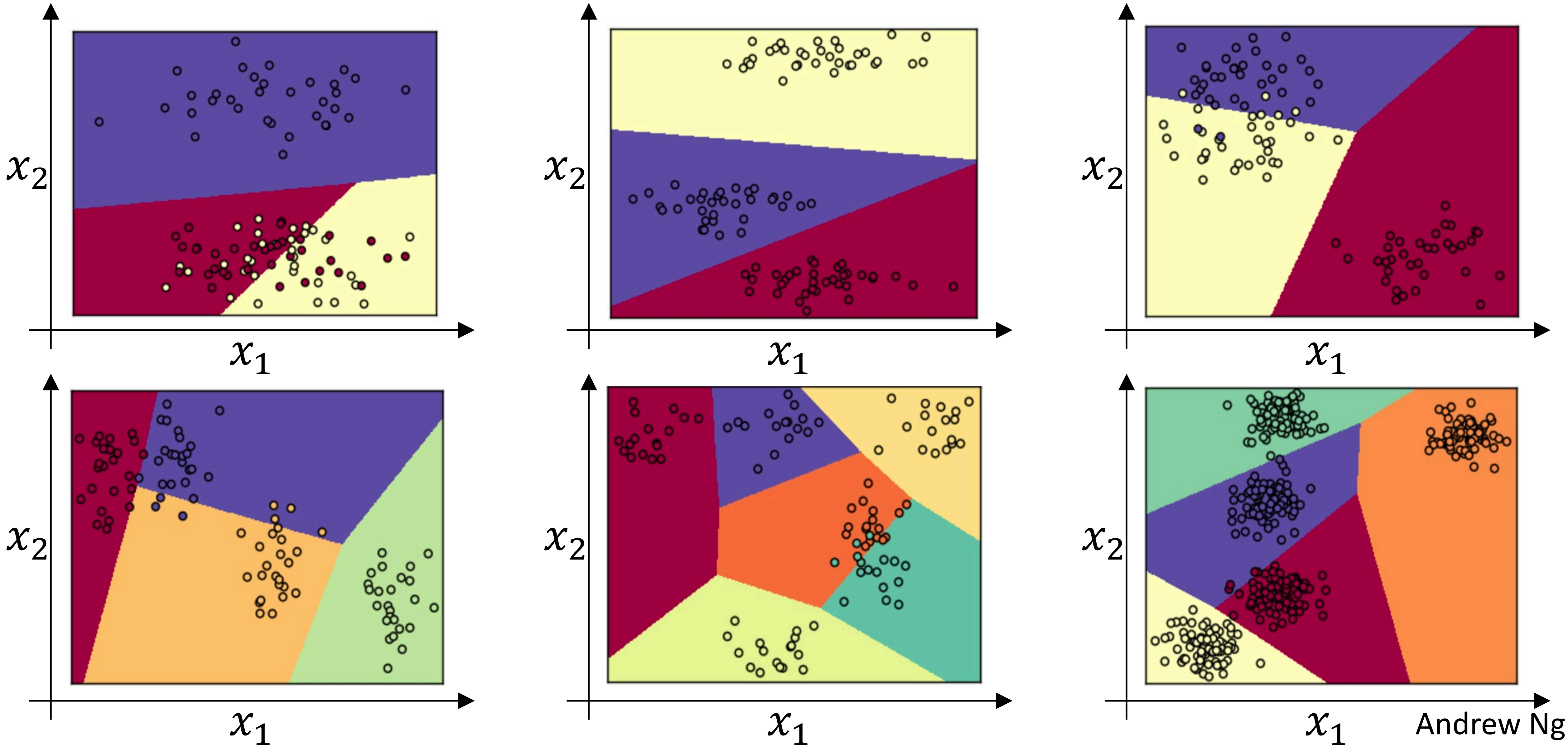


Andrew Ng

Softmax layer



Softmax examples



Andrew Ng

deeplearning.ai

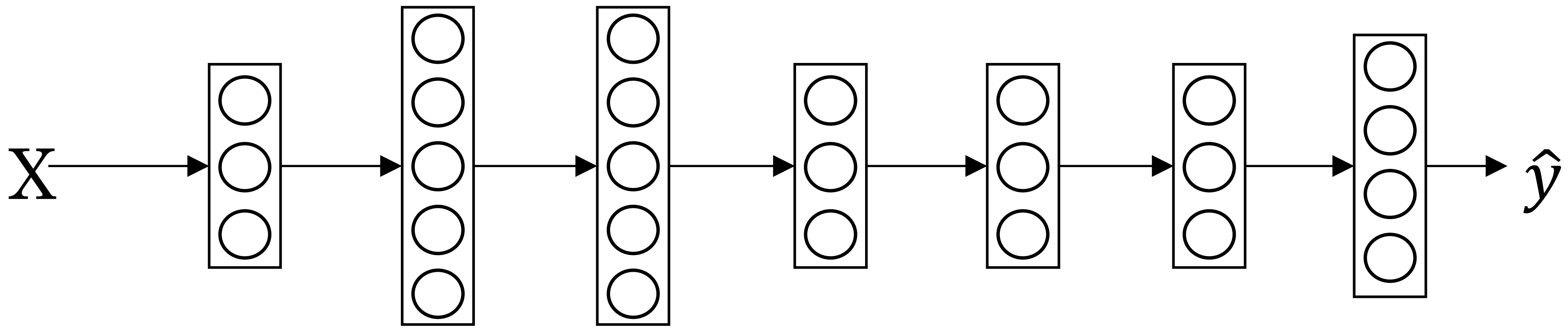
Multi-class classification

Trying a softmax classifier

Understanding softmax

Loss function

Summary of softmax classifier



deeplearning.ai

Programming Frameworks

Deep Learning frameworks

Deep learning frameworks

- Caffe/Caffe2
- CNTK
- DL4J
- Keras
- Lasagne
- mxnet
- PaddlePaddle
- TensorFlow
- Theano
- Torch

Choosing deep learning frameworks

- Ease of programming (development and deployment)
- Running speed
- Truly open (open source with good governance)

deeplearning.ai

Programming Frameworks

TensorFlow

Motivating problem

$J(w) = \boxed{w^2 - 10w + 25}$
(cost)

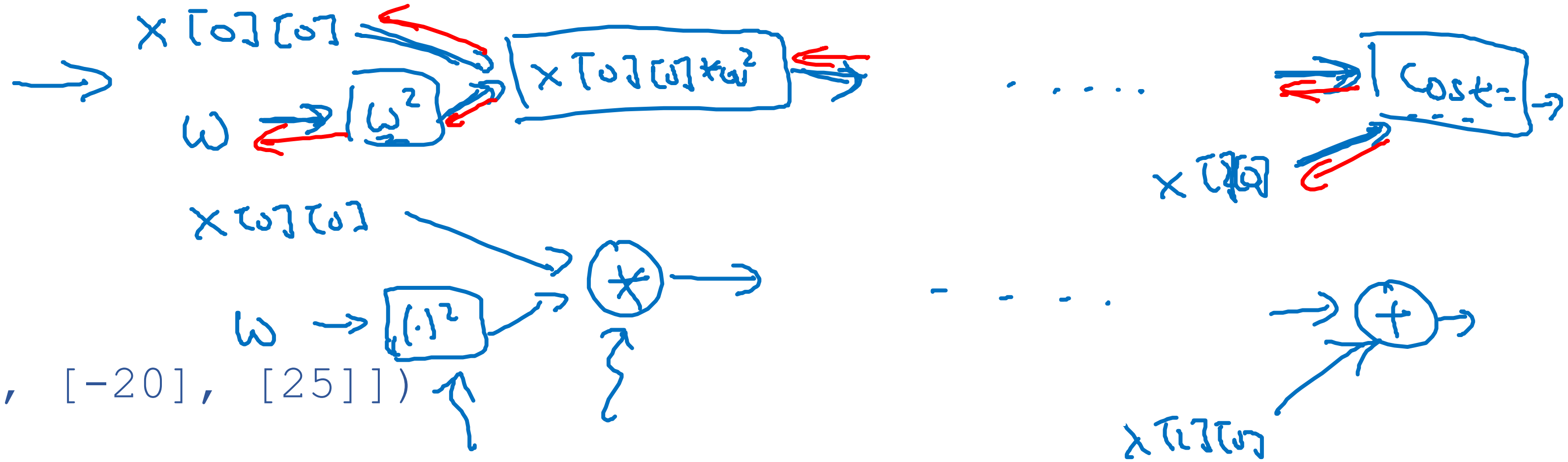
$(w-5)^2$

$w = 5$

$J(w, b)$
$\uparrow \quad \uparrow$

Andrew Ng

Code example




```
import numpy as np
import tensorflow as tf

coefficients = np.array([[1], [-20], [25]])

w = tf.Variable([0],dtype=tf.float32)
x = tf.placeholder(tf.float32, [3,1])
cost = x[0][0]*w**2 + x[1][0]*w + x[2][0]     # (w-5)**2
train = tf.train.GradientDescentOptimizer(0.01).minimize(cost)
init = tf.global_variables_initializer()
session = tf.Session()
session.run(init)
print(session.run(w))
```

```
with tf.Session() as session:
    session.run(init)
    print(session.run(w))
```

```
for i in range(1000):
    session.run(train, feed_dict={x:coefficients})
print(session.run(w))
```

Andrew Ng