UC San Diego
JACOBS SCHOOL OF ENGINEERING

ECE/SIOC 228 Machine Learning for Physical Applications
Lecture 4: Optimization and Regularization

Yuanyuan Shi

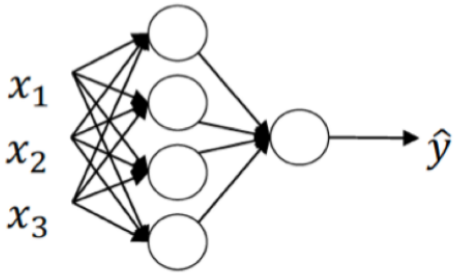
Assistant Professor, ECE

University of California, San Diego

Reference & Acknowledgement: Stanford CS230 Deep Learning Lecture Note, Andrew Ng

Review: Forward and Backward Computation of Neural Network

UC San Diego



1 sample (x, y), $x \in R^3$, $y \in \{0, 1\}$

$L(\hat{y}, y) = -[y \log \hat{y} + (1-y) \log(1-\hat{y})]$

x, $w^{[1]}$, $b^{[1]}$ → $z^{[1]} = w^{[1]} x + b^{[1]}$ → $a^{[1]} = \sigma(z^{[1]})$ → $z^{[2]} = w^{[2]} a^{[1]} + b^{[2]}$ → $a^{[2]} = \sigma(z^{[2]})$ → $L(y, a^{[2]})$

$w^{[2]}$, $b^{[2]}$

$dz^{[1]}$ → $dw^{[1]}$, $db^{[1]}$

$da^{[1]}$

$dz^{[2]}$ → $dw^{[2]}$, $db^{[2]}$

$da^{[2]}$

Weight update: $w^{[l]} \leftarrow w^{[l]} - \eta dw^{[l]}$, $b^{[l]} \leftarrow b^{[l]} - \eta db^{[l]}$, $l = 1$ or 2

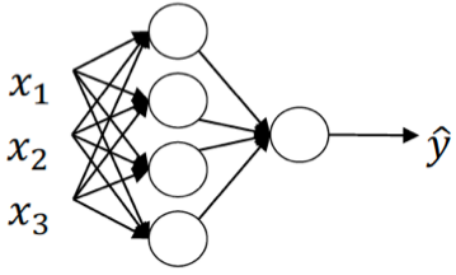
Today's Lecture

UC San Diego

- Forward and Backward Computation across Multiple Examples
- Optimization Algorithms
- Regularization

Vectorizing across Multiple Examples

UC San Diego



for one sample (x, y), we have:

$$z^{[1]} = W^{[1]}x + b^{[1]}, \quad a^{[1]} = \sigma(z^{[1]})$$

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}, \quad a^{[2]} = \sigma(z^{[2]})$$

What if we have m samples?

$x^{(1)} \rightarrow \hat{y}^{(1)}$

$x^{(2)} \rightarrow \hat{y}^{(2)}$

$x^{(m)} \rightarrow \hat{y}^{(m)}$

for $i = 1$ to m (layer 1, sample i)

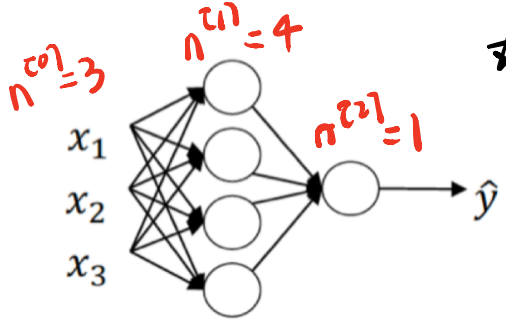
$$z^{[1](i)} = W^{[1]}x^{(i)} + b^{[1]}, \quad a^{[1](i)} = \sigma(z^{[1]})$$

$$z^{[2](i)} = W^{[2]}a^{[1](i)} + b^{[2]}, \quad a^{[2](i)} = \sigma(z^{[2](i)})$$

$$\hat{y}^{(i)} = a^{[2](i)}$$

Vectorizing across Multiple Examples

UC San Diego



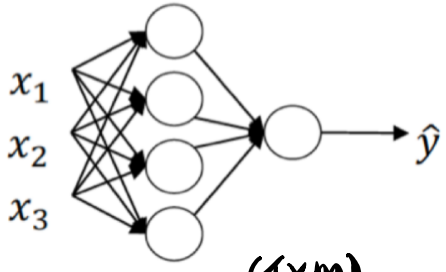
☆ Speedup computation: avoid using explicit for-loop (GPU/CPU have parallelizations)

☆ $z^{[1](i)} = w^{[1]}x^{(i)} + b^{[1]}$. suppose $b^{[1]}$ is zero vector

$$W^{[1]}\underbrace{\begin{bmatrix} | & | & | & \\ x^{(1)} & x^{(2)} & x^{(3)} & \cdots \\ | & | & | & \end{bmatrix}}_{\text{input matrix}} = \begin{bmatrix} | & | & | & \\ w^{[1]}x^{(1)} & w^{[1]}x^{(2)} & w^{[1]}x^{(3)} & \cdots \\ | & | & | & \end{bmatrix} = \underbrace{\begin{bmatrix} | & | & | & \\ z^{[1](1)} & z^{[1](2)} & z^{[1](3)} & \cdots \\ | & | & | & \end{bmatrix}}_{Z^{[1]}}$$

Vectorizing across Multiple Examples

UC San Diego



$$\underbrace{W^{[1]}}_{(4\times 3)}\begin{bmatrix} | & | & | & \\ x^{(1)} & x^{(2)} & x^{(3)} & \cdots \\ | & | & | & \end{bmatrix}_{(3\times m)} + \begin{bmatrix} | \\ b^{[1]} \\ | \end{bmatrix}_{(4\times 1)??}$$

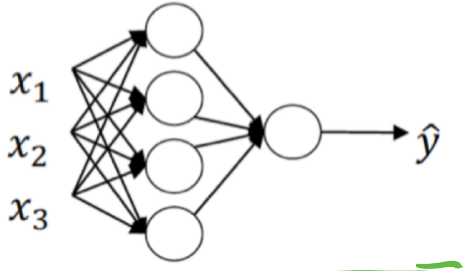
(not valid sum!)

$$\Rightarrow \underbrace{\begin{bmatrix} | & | & | & \\ \bullet & \bullet & \bullet & \cdots \\ \bullet & \bullet & \bullet & \\ \bullet & \bullet & \bullet & \\ | & | & | & \end{bmatrix}}_{(4\times m)} + \begin{bmatrix} | & | & & | \\ b^{[1]} & b^{[1]} & \cdots & b^{[1]} \\ | & | & & | \end{bmatrix}_{(4\times m)} = \begin{bmatrix} | & | & | & \\ z^{[1](1)} & z^{[1](2)} & z^{[1](3)} & \cdots \\ | & | & | & \end{bmatrix}_{(4\times m)}$$

$W^{[1]}x^{(1)}$ $W^{[1]}x^{(2)}$

automatic

broadcasting in python



$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}$$

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

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

$$Z^{[2]} = W^{[2]} A^{[1]} + b^{[2]}$$

$$A^{[2]} = \sigma(Z^{[2]})$$

Make sure dimension is right!

$X : (3 \times m)$ $\quad W^{[1]} : (4 \times 3)$ ($n^{[1]} \times n^{[0]}$)

$b^{[1]} : (4 \times 1)$

$Z^{[1]} : (4 \times m)$, $A^{[1]} : (4 \times m)$

$W^{[2]} : (1 \times 4)$ ($n^{[2]} \times n^{[1]}$), $b^{[2]}$ (1×1)

$Z^{[2]} : (1 \times m)$, $A^{[2]} = (1 \times m)$

Vectorizing across Multiple Examples

UC San Diego

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) = \frac{1}{m}\sum_{i=1}^{m} -\left[y^{(i)}\log\hat{y}^{(i)} + (1-y^{(i)})\log(1-\hat{y}^{(i)})\right]$$

for i = 1 to m

$z^{[1](i)} = W^{[1]}x^{(i)} + b^{[1]}$, $a^{[1](i)} = \sigma(z^{[1](i)})$

$z^{[2](i)} = W^{[2]}a^{[1](i)} + b^{[2]}$, $a^{[2](i)} = \sigma(z^{[2](i)})$

$\hat{y}^{(i)} = a^{[2](i)}$

$J \mathrel{+}= -\left[y^{(i)}\log a^{[2](i)} + (1-y^{(i)})\log(1-a^{[2](i)})\right]$

$dz^{[2](i)} = a^{[2](i)} - y^{(i)}$

$dW^{[2]} \mathrel{+}= dz^{[2](i)}\, a^{[1](i)T}$

$db^{[2]} \mathrel{+}= dz^{[2](i)}$

$dW^{[1]} \mathrel{+}= dW^{[1](i)}$, $db^{[1]} \mathrel{+}= db^{[1](i)}$

← loop over all samples

$J \leftarrow \frac{1}{m}J$

$dW^{[1]} \leftarrow \frac{1}{m}dW^{[1]}$

$db^{[1]} \leftarrow \frac{1}{m}db^{[1]}$

$dW^{[2]} \leftarrow \frac{1}{m}dW^{[2]}$

$db^{[2]} \leftarrow \frac{1}{m}db^{[2]}$

Vectorizing across Multiple Examples

UC San Diego

$$L(Y,\hat{Y}) = \frac{1}{m}\sum_{i=1}^{m} L(Y^{(i)},\hat{Y}^{(i)}) = \frac{1}{m}\sum_{i=1}^{m} -\left[Y^{[i]}\log\hat{Y}^{[i]} + (1-Y^{[i]})\log(1-\hat{Y}^{[i]})\right]$$

for $i = 1$ to m

$z^{[1](i)} = W^{[1]}x^{(i)} + b^{[1]}$, $a^{[1](i)} = \sigma(z^{[1]})$

$z^{[2](i)} = W^{[2]}a^{[1](i)} + b^{[2]}$, $a^{[2](i)} = \sigma(z^{[2](i)})$

$\hat{y}^{(i)} = a^{[2](i)}$

$L+= -\left[y^{(i)}\log a^{[2](i)} + (1-y^{(i)})\log(1-a^{[2](i)})\right]$

- $dz^{[2](i)} = a^{[2](i)} - y^{(i)}$
- $dw^{[2]} += dz^{[2](i)}a^{[1](i)T}$
- $db^{[2]} += dz^{[2](i)}$

$dw^{[1]} += dw^{[1](i)}$, $db^{[1]} += db^{[1](i)}$

$$dz^{[2]} = \left[a^{[2](1)} - y^{(1)},\ a^{[2](2)} - y^{(2)},\ \cdots,\ a^{[2](m)} - y^{(m)}\right]$$

$$= A^{[2]} - Y$$

$$db^{[2]} = \frac{1}{m}\sum_{i=1}^{m} dz^{[2](i)}$$

$$dw^{[2]} = \frac{1}{m}\sum_{i=1}^{m} dz^{[2](i)}\ a^{[1](i)T}$$

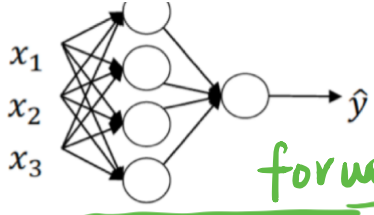
(1×4)

$$= \frac{1}{m}\left[dz^{[2](1)}\cdots dz^{[2](m)}\right]\begin{bmatrix} -a^{[1](1)}- \\ \vdots \\ -a^{[1](m)}- \end{bmatrix}$$

1×m, m×4

$$= \frac{1}{m}\,dz^{[2]}\left(A^{[1]}\right)^{T}$$

$$L(y,\hat{y}) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{[i]}\log\hat{y}^{[i]} + (1-y^{[i]})\log(1-\hat{y}^{[i]})\right]$$



forward

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}$$

$$Z^{[1]} = W^{[1]}X + b^{[1]},$$
$$A^{[1]} = \sigma(Z^{[1]})$$
$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$
$$A^{[2]} = \sigma(Z^{[2]})$$

Backpropagation

$$dZ^{[2]} = A^{[2]} - Y$$
$$dW^{[2]} = \frac{1}{m}dZ^{[2]}(A^{[1]})^T$$
$$db^{[2]} = \frac{1}{m}\sum_{i=1}^{m} dZ^{[2](i)}$$
$$dZ^{[1]} = W^{[2]T}dZ^{[2]} * \underbrace{g^{[1]\prime}(Z^{[1]})}_{\text{elementwise}}$$
$$dW^{[1]} = \frac{1}{m}dZ^{[1]}X^T$$
$$db^{[1]} = \frac{1}{m}\sum_{i=1}^{m} dZ^{[1](i)}$$

Mini-batch and Stochastic Gradient Descent

Batch vs. Mini-batch Gradient Descent

UC San Diego

- Vectorization allows you to efficiently compute on m examples

$$X = \begin{bmatrix} | & | & | & & | \\ x^{(1)} & x^{(2)} & x^{(3)} & \cdots & x^{(m)} \\ | & | & | & & | \end{bmatrix}$$

What if m is very large?

$$Y = \begin{bmatrix} y^{(1)} & y^{(2)} & y^{(3)} & \cdots & y^{(m)} \end{bmatrix}$$

e.g. $m = 1{,}000{,}000$

Mini-batch

$X = [x^{(1)}, x^{(2)}, \cdots, x^{(1000)}]\ [x^{(1001)}, \cdots, x^{(2000)}] \cdots [\cdots x^{(m)}]$

$Y = [y^{(1)}, y^{(2)}, \cdots y^{(1000)}],\ [y^{(1001)}, \cdots, y^{(2000)}],\ \cdots [\cdots y^{(m)}]$

repeat { ← epoch

for $t = 1, \cdots, 1000$ { iterate over all mini-batches

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

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

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

$$\vdots$$

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

Compute cost $J^{\{t\}} = \frac{1}{1000} \sum_{i=1}^{l} L(\hat{y}^{(i)}, y^{(i)})$

backprop to compute gradient using $(X^{\{t\}}, Y^{\{t\}})$

η : learning rate

$$W^{[l]} = W^{[l]} - \eta \, dW^{[l]}$$

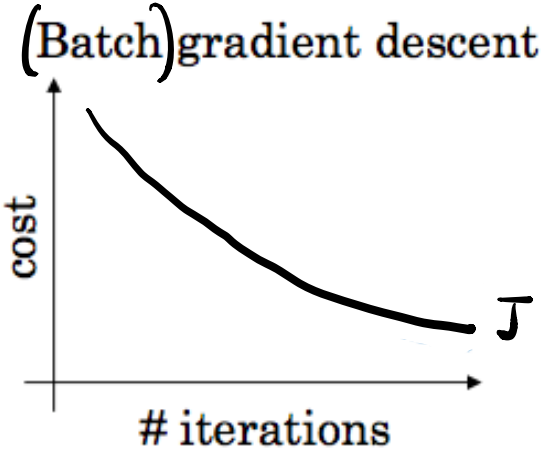
$$b^{[l]} = b^{[l]} - \eta \, db^{[l]}$$

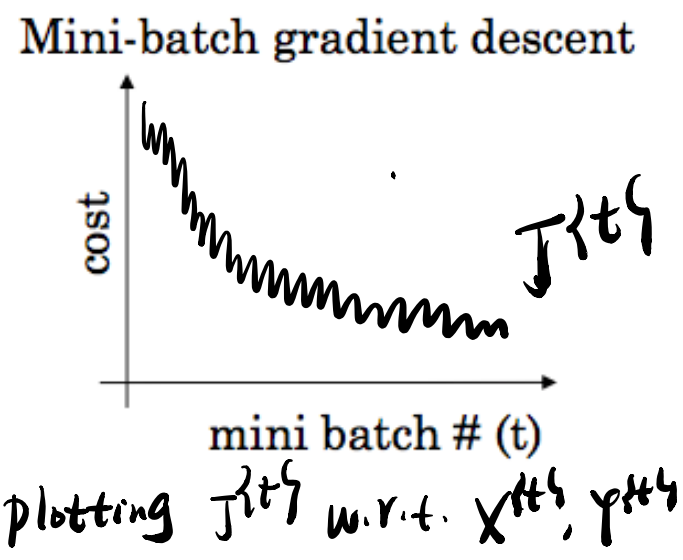
}

}

Understanding Mini-batch Gradient Descent

UC San Diego





Choosing Mini-batch Size

UC San Diego

→ If mini-batch size = m : (Batch) gradient descent

→ If mini-batch size = 1 : stochastic gradient descent

→ If mini-batch size $1 < b < m$: mini-batch gradient descent

SGD	Mini-batch gradient descent typical size: 64, 128, 256 ... (power of 2)	Batch gradient descent

$$\min_w L(w) = \frac{1}{m}\sum_{i=1}^{m} L_i(w)$$

$L_i(w)$ is the loss on sample $(x^{(i)}, y^{(i)})$

Gradient descent will do:

$$w \leftarrow w - \eta \cdot \frac{1}{m}\sum_{i=1}^{m} \nabla L_i(w)$$

SGD : $w \leftarrow w - \eta_k \cdot \nabla L_{i_k}(w)$

$i_k \in \{1, 2, \cdots, m\}$

✡ Randomized rule is common in practice:
Choose $i_k \in \{1, 2, \cdots m\}$ uniformly at random

$$E[\nabla L_{i_k}(w)] = \nabla L(w)$$

- SGD is using an unbiased estimate of the gradient at each step (Gradient descent + zero-mean noise)
- Big saving in memory & computation

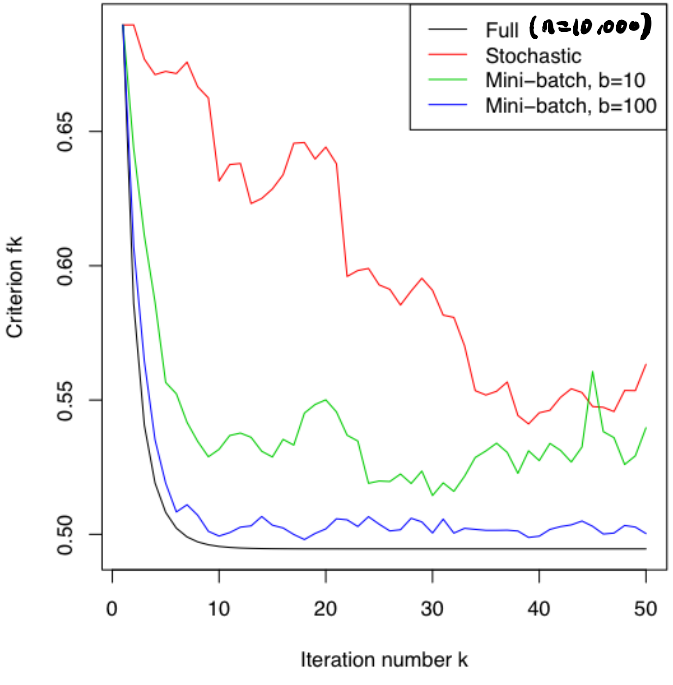
Mini-batch gradient descent:

$$w \leftarrow w - \eta \frac{1}{b} \sum_{i \in I_k} \nabla L_i(w)$$

- $I_k \subseteq \{1, \cdots, m\}$ is a random subset from the whole dataset. $|I_k| = b << m$
- $E\left[\frac{1}{b} \sum_{i \in I_k} \nabla L_i(w)\right] = \nabla L(w)$
  - unbiased
  - reduce variance

SGD Convergence

UC San Diego



- Logistic regression
- $\hat{y} = \sigma(w^T x + b)$, $x \in R^{20}$
- $n = 10,000$ data points

☆ SGD/mini-batch gradient descent is the dominant algorithm in training neural networks!

Today's Lecture

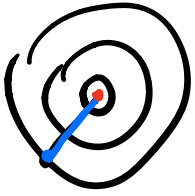
UC San Diego

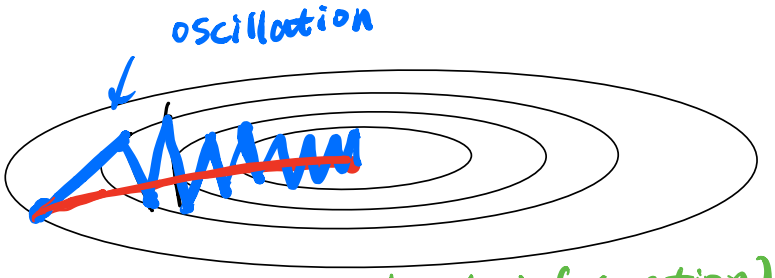
- Forward and Backward Computation across Multiple Examples
- **Optimization Algorithms**
- Regularization

Optimization Algorithms

Gradient Descent with Momentum

UC San Diego





η: learning rate

$$W := W - \eta V_{dW}$$

$$b := b - \eta V_{db}$$

Momentum (previous gradient information)

On iteration t:

- Compute dW, db on the current mini-batch
- $V_{dW} = \beta V_{dW} + (1-\beta) dW$

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

momentum

☆ Usually trains faster than standard gradient descent

Exponential averaging:

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

$\beta = 0.9 \leftarrow$ hyperparameter

$V_{10} = 0.9 V_9 + 0.1 g_{10}$

$V_9 = 0.9 V_8 + 0.1 g_9$

$V_8 = 0.9 V_7 + 0.1 g_8$

$V_{10} = 0.1 g_{10} + 0.9 (0.9 V_8 + 0.1 g_9)$

$= 0.1 g_{10} + 0.09 g_9 + 0.81 V_8 \; (0.1 g_8 + 0.9 V_7)$

$= 0.1 g_{10} + 0.1 \times 0.9 \, g_9 + 0.1 \times (0.9)^2 g_8 + 0.1 \times (0.9)^3 g_7 + \cdots$

☆ momentum is an exponential moving average of past gradients

Exponential averaging:

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

← is it the "right" exponential average of past gradient?

$V_0 = 0$

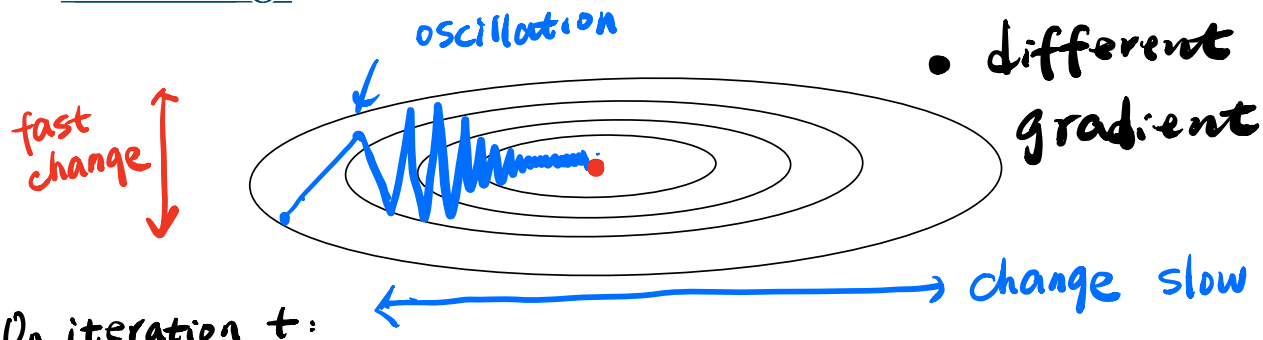
$V_1 = 0.9 V_0 + 0.1 g_1 = 0.1 g_1$

$V_2 = 0.81 V_0 + 0.1 g_2 + 0.09 g_1 = 0.1 g_2 + 0.09 g_1$

$V_t = \beta^t V_0 + (1-\beta) g_t + (1-\beta)\beta g_{t-1} + (1-\beta)\beta^2 g_{t-2} + \cdots$

Sum of coff: $1-\beta^t$ $\Rightarrow$ $V_t^{\text{corrected}} = \frac{V_t}{1-\beta^t}$

RMSprop (Root Mean Squared Propagation)



- different gradient scale

On iteration t:

- Compute dw, db on the current mini-batch
- $S_{dw} = \beta_2 S_{dw} + (1-\beta_2) dw^2$
- $S_{db} = \beta_2 S_{db} + (1-\beta_2) db^2$

Adagrad is very similar:
$S_{dw} = S_{dw} + (dw)^2$

- $w \leftarrow w - \eta \frac{dw}{\sqrt{S_{dw}+\varepsilon}}$, $b \leftarrow b - \eta \frac{db}{\sqrt{S_{db}+\varepsilon}}$, ε is a small constant (10^{-8})

Adam Optimizer

UC San Diego

☆ Combines the best properties of "momentum" and "RMSprop"

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

momentum term

root mean squared term

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"

$$S_{dw} = \beta_2 S_{dw} + (1-\beta_2)\,dw^2$$
$$S_{db} = \beta_2 S_{db} + (1-\beta_2)\,db^2$$

} "RMSprop"

Adam Optimizer

UC San Diego

$$w \leftarrow w - \eta \frac{V_{dw}}{\sqrt{S_{dw}} + \varepsilon}, \quad b \leftarrow b - \eta \frac{V_{db}}{\sqrt{S_{db}} + \varepsilon}$$

+ bias correction

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

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

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

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

η : needs to be tune

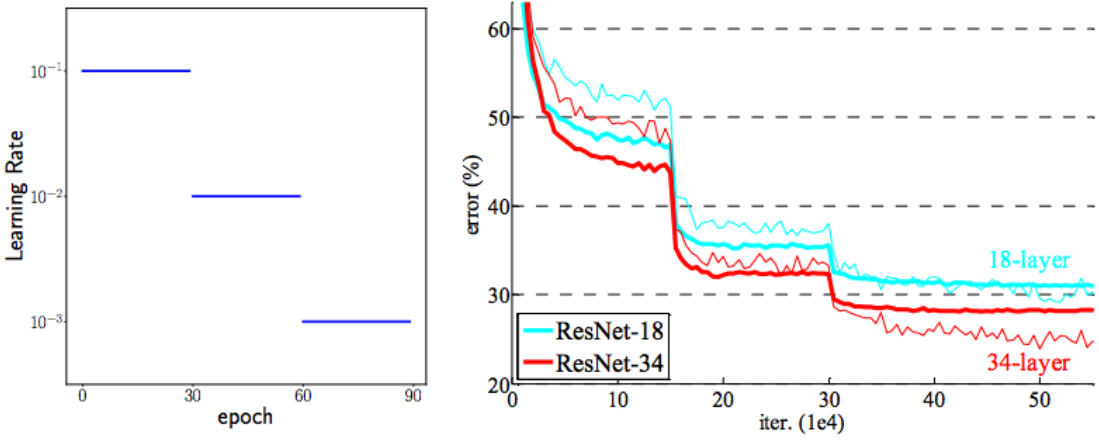
$\beta_1 : 0.9 \; (dw)$

$\beta_2 : 0.999 \; (dw^2)$

$\varepsilon : 10^{-8}$

Learning Rate Decay

UC San Diego



(a) Learning rate decay strategy

(b) Figure taken from He et al. (2016)

Figure 1: Training error in (b) is shown by thin curves, while test error in (b) by bold curves.

[1] You, Kaichao, Mingsheng Long, Jianmin Wang, and Michael I. Jordan. "How does learning rate decay help modern neural networks?." *arXiv preprint arXiv:1908.01878* (2019).

- Staircase (reduce by certain factor every few epochs)
- Exponentral decaying: $\eta = \eta_0 * \text{decay}^{\text{epoch \#}}$
- Epoch number based:
  - $\eta = \frac{1}{1 + \text{decay} * \text{epoch \#}} \cdot \eta_0$
  - $\eta = \eta_0 \frac{c}{\sqrt{\text{epoch-num}}}$
- Mini-batch number based: $\eta = \eta_0 \frac{c}{\sqrt{t}}$
- Mannal decay

Problem of Local Optima

Landscape Conjecture

UC San Diego

- For many years, most practitioners believed that local minima were a common problem in training deep neural networks
- Today, that doesn't seem to be the case – remain an active research area
- Researchers suspect that: for large-size neural networks, most local minima have a low cost function value and are equivalent
- Thus, finding the true global minimum on the training set (as opposed to one of the many good local ones) is not very important and may lead to overfitting

Further Reading:

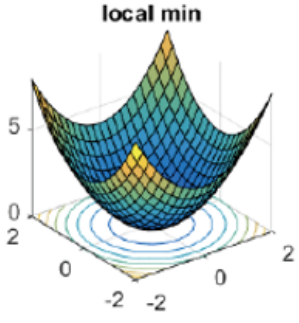
The Loss Surfaces of Multilayer Networks, https://arxiv.org/pdf/1412.0233.pdf

https://www.deeplearningbook.org/contents/optimization.html

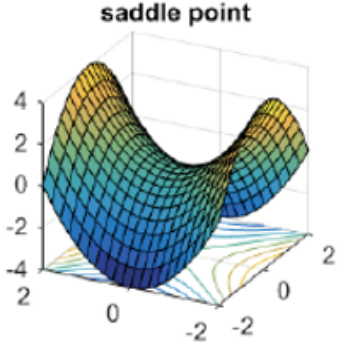
Problem of Saddle Points

UC San Diego

- For high-dimensional nonconvex optimization, local minima are in fact rare compared to another kind of point with zero gradient: saddle points



Hessian: positive definite

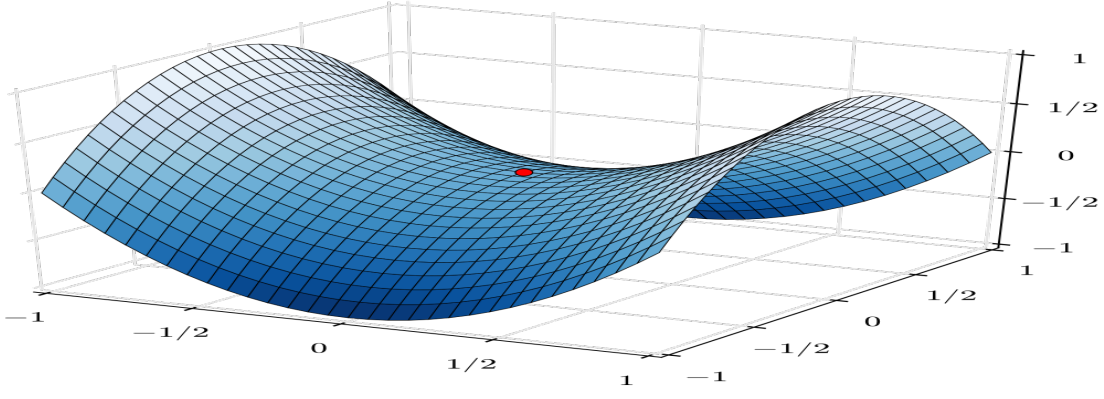


Hessian has both positive/negative eigenvalues

Further Reading:
How to Escape Saddle Points Efficiently https://arxiv.org/abs/1703.00887
https://www.deeplearningbook.org/contents/optimization.html

Problem of Plateaus

UC San Diego

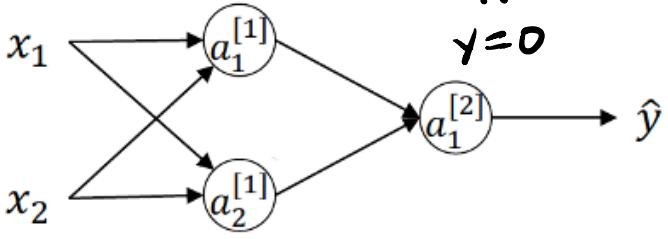


- Plateaus can slow down learning

Random Initialization

UC San Diego

$x \in \mathbb{R}^2$ $z^{[1]} = W^{[1]}x + b^{[1]}$



$z^{[1]} = W^{[1]}x + b^{[1]} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

$a^{[1]} = \text{sigmoid}(z^{[1]}) = \begin{bmatrix} \frac{1}{2} \\ \frac{1}{2} \end{bmatrix}$

$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]} = 0$

$a^{[2]} = \text{sigmoid}(z^{[2]}) = \frac{1}{2}$

What if we initialize all weights to 0?

$W^{[1]} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$, $b^{[1]} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

$W^{[2]} = [0 \;\; 0]$, $b^{[2]} = 0$

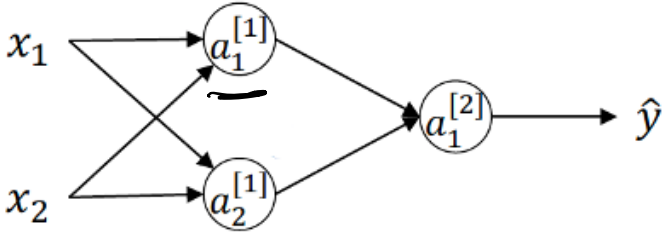
$dz^{[2]} = a^{[2]} - y = \frac{1}{2}$

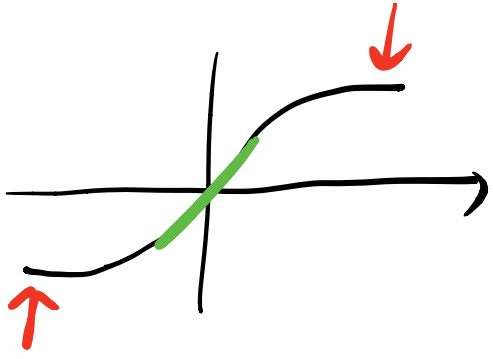
$dW^{[2]} = dz^{[2]} \cdot a^{[1]T}$

$= \frac{1}{2} \cdot [\frac{1}{2} \;\; \frac{1}{2}] = [\frac{1}{4} \;\; \frac{1}{4}]$

Same for all weights!

Random Initialization





- If weights are initialized too large or too small network won't learn well
- Xavier initialization, $\left(\pm \frac{\sqrt{6}}{\sqrt{n^{[l]}+n^{[l-1]}}}\right)$; Kaiming initializ

Today's Lecture

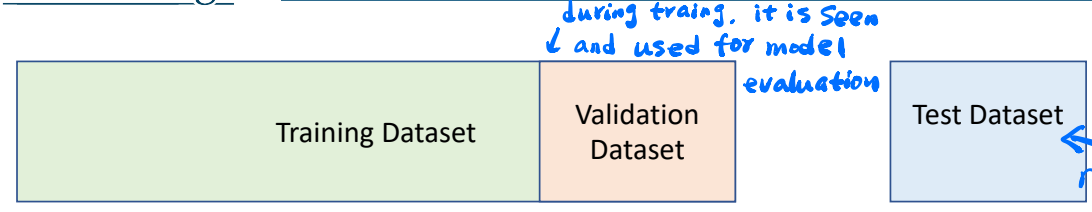
UC San Diego

- Forward and Backward Computation across Multiple Examples
- Optimization Algorithms
- **Regularization**

Regularization

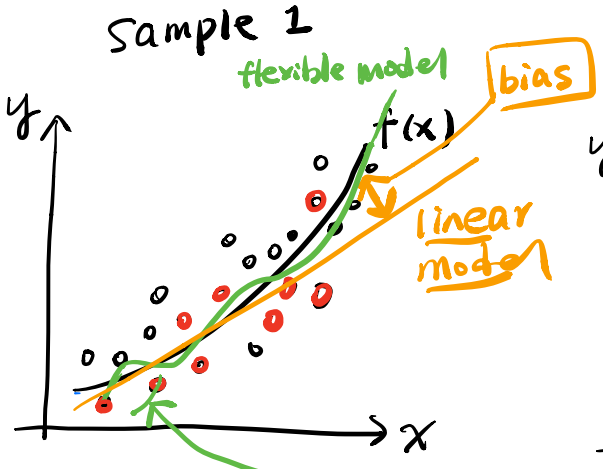
Train, Validation and Test Sets

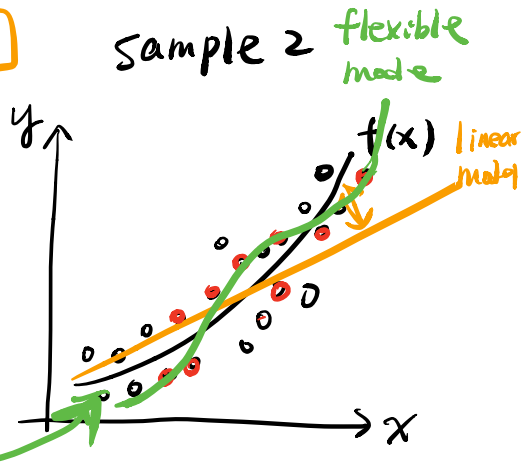
UC San Diego



[Classic: 70/15/15, 60/20/20] [Big data: 10,000 validation, test, remaining-traing]

- Training data: dataset we use to train the model (e.g. learning weights and biases in a NN model)
- Validation data (also called development data): dataset we use to evaluate model during traing & model hyperparameter tuning
- Test data: dataset to provide evaluation of the final model fit





Variance

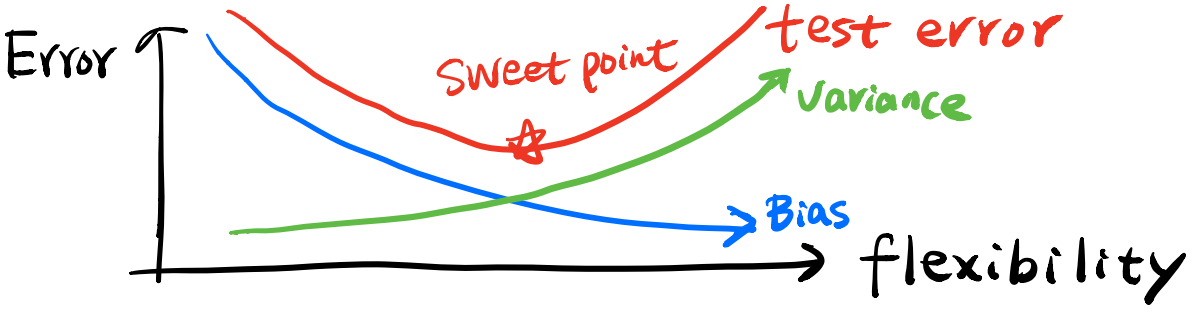
$y = f(x) + \varepsilon$

ε: zero-mean random noise

Bias and Variance Tradeoff

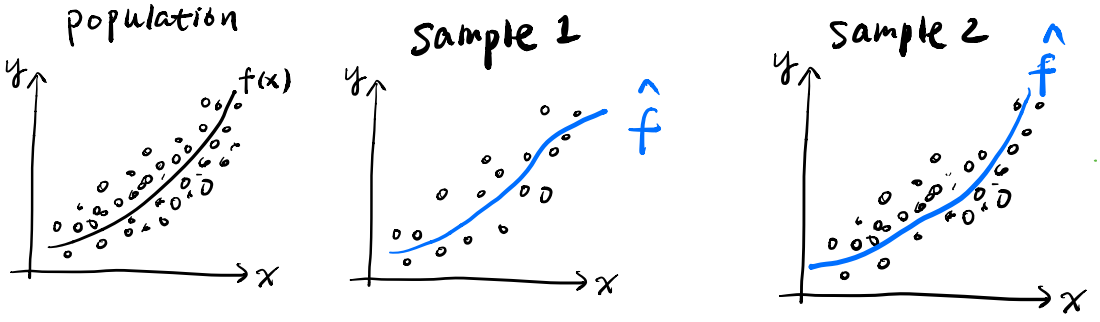
UC San Diego

- **Variance** refers to the amount by which $\hat{f}$ would change if we estimated it using a different training data set. In general, more flexible statistical methods have higher variance.
- **Bias** refers to the error that is introduced by approximating the data (which may arise in a real-life problem with very complicated relationship), by a much simpler model.
- As a general rule, as we use more flexible methods, the variance will increase and the bias will decrease.



Bias and Variance Tradeoff

UC San Diego



- For a given datapoint (x, y) underlying relationship as $y = f(x) + \epsilon$, ϵ is a zero mean random noise. The expected test MSE of a learned model $\hat{f}$ can always be decomposed into the sum of three fundamental quantities:

$$E\left(y - \hat{f}(x)\right)^2 = E\left(y - E[\hat{f}(x)] + E[\hat{f}(x)] - \hat{f}(x)\right)^2 = E\left(\left(E[\hat{f}(x)] - \hat{f}(x)\right)^2\right) + E\left(\left(f(x) + \epsilon - E[\hat{f}(x)]\right)^2\right)$$

$$= E\left(\left(E[\hat{f}(x)] - \hat{f}(x)\right)^2\right) + E\left(\left(f(x) - E[\hat{f}(x)]\right)^2\right) + Var(\epsilon^2)$$

irreducible

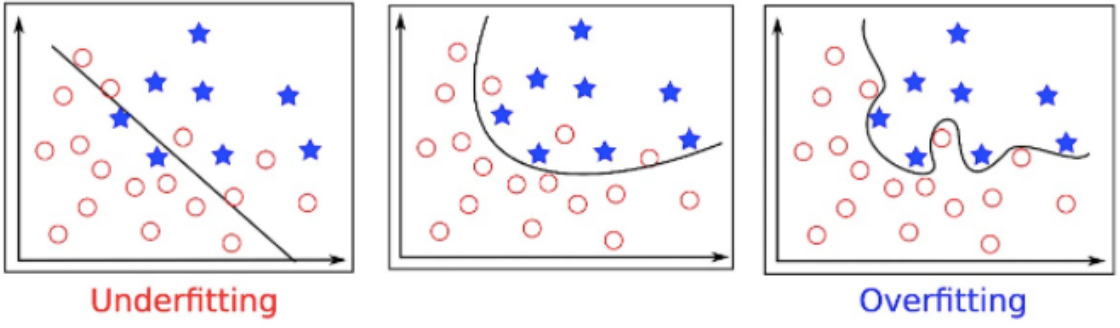
variance

bias

- In order to minimize the expected test error, we need to select a statistical learning method that simultaneously achieves *low variance* and *low bias*.

Bias and Variance

UC San Diego



High bias

High variance

UC San Diego

	Error	↓ High bias	
Traing dataset	1%	14%	1%
Validation data	15%	15%	1.3%

High variance
(overfitting)

↑
low bias
low variance
Good model!

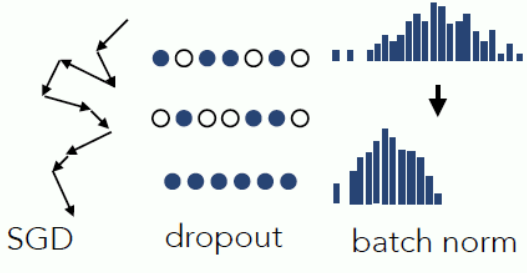
Regularization Methods in Deep Learning

UC San Diego

neural networks are amazingly flexible…
given enough parameters, they can perfectly fit random noise

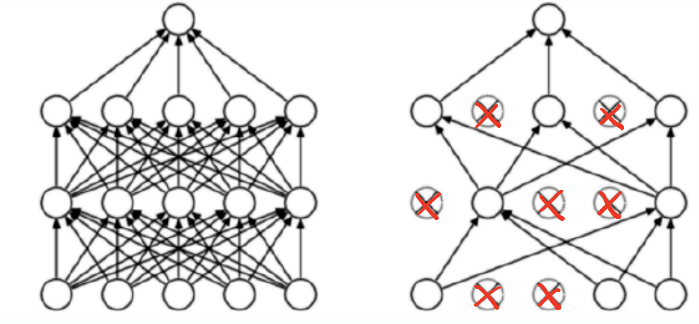
regularization combats **overfitting**

stochasticity (uncertainty)



Dropout

UC San Diego



standard neural net

neural net with dropout

0.5

Dropping out units (both hidden and observed) in a neural network

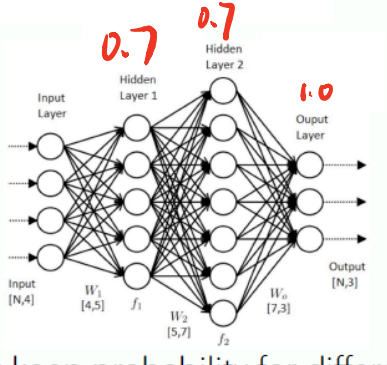
Keep with probability p, drop out with probability 1-p

Why Does it Work?

UC San Diego

Working with a reduced network at each iteration

Can't rely on one features, spread out the weights



Use varying keep probability for different layers

Wager, Stefan, Sida Wang, and Percy S. Liang. "Dropout training as adaptive regularization." *Advances in neural information processing systems*. 2013.

- Batch norm - normalizes each layer's activations according to statistics of the batch

$$a^{[l-1]} \xrightarrow{W^{[l]}, b^{[l]}} z^{[l]} \longrightarrow \tilde{z}^{[l]} \xrightarrow{g(\tilde{z}^{[l]})} a^{[l]}$$

$$\mu_B^{[l]} = \frac{1}{b}\sum_{i \in B} z^{[l](i)}$$

$$\sigma_B^{[l]2} = \frac{1}{b}\sum_{i \in B} \left(z^{[l](i)} - \mu_B^{[l]}\right)^2$$

(applied to each dim of $z^{[l]}$ independently)

- Normalize: $\dfrac{z^{[l]} - \mu_B^{[l]}}{\sqrt{\sigma_B^{[l]2} + \varepsilon}}$

- Scale & shift $\gamma * \dfrac{z^{[l]} - \mu_B^{[l]}}{\sqrt{\sigma_B^{[l]2} + \varepsilon}} + \beta$

Batch Normalization, Szegedy & Ioffe, 2015

- During test: we may not have a full batch to compute the μ_B, σ_B
- Use μ_pop, σ_pop (estimated μ, σ for the entire training dataset)

☆ Why batch norm have regularization effect?

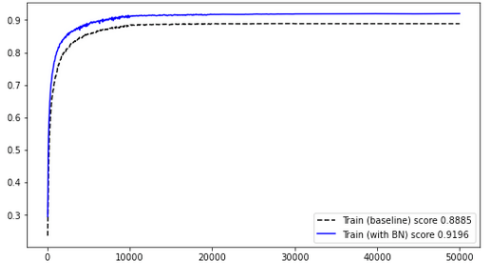
use μ, σ from mini-batch (noise)

Batch Normalization, Szegedy & Ioffe, 2015

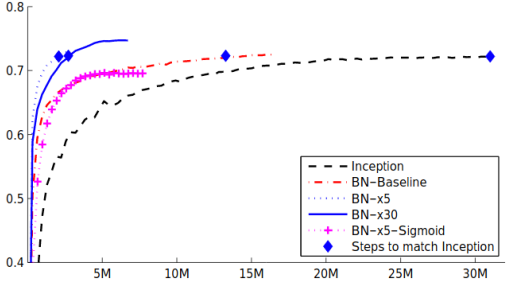
Batch Normalization Helps with Optimization

UC San Diego

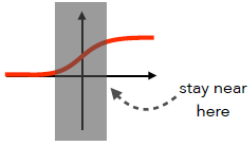
- Batch Normalization is not only a regularization tool but also helps with optimization (perhaps more important)

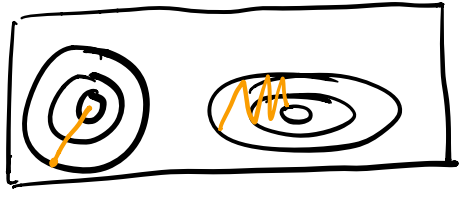


Training 3 fully connected layers of 100 neurons each, all activated by sigmoid function on MNIST



Inception Network on ImageNet





Batch Normalization, Szegedy & Ioffe, 2015

Regularization Methods in Deep Learning

UC San Diego

neural networks are amazingly flexible…
given enough parameters, they can perfectly fit random noise

regularization combats **overfitting**