

Week 1-2 Regularization

笔记本: DL 2 - Deep NN Hyperparameter Tunning, Regularization & Optimization

创建时间: 2021/1/9 09:25

更新时间: 2021/1/9 09:42

Logistic regression

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

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

$\lambda = \text{regularization parameter}$
lambda lambda

$J(w,b) = \frac{1}{m} \sum_{i=1}^m \ell(y_i, \hat{y}_i) + \frac{\lambda}{2m} \|w\|_2^2$

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

$L_1 \text{ regularization } \frac{\lambda}{2m} \sum_{j=1}^{n_x} |w_j| = \frac{\lambda}{2m} \|w\|_1$

$w \text{ will be sparse}$

~~$+\frac{\lambda}{2m} b^2$~~
omit

L2 regularization (L2 norm)

w will be sparse for L1

regularization

For NN: (Frobenius norm)

Neural network

$$J(w^{(1)}, b^{(1)}, \dots, w^{(L)}, b^{(L)}) = \underbrace{\frac{1}{m} \sum_{i=1}^m \ell(y^{(i)}, \hat{y}^{(i)})}_{\text{loss}} + \underbrace{\frac{\lambda}{2m} \sum_{l=1}^L \|w^{(l)}\|_F^2}_{\text{regularization}}$$

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

"Frobenius norm"

$$\|\cdot\|_2^2$$

$$\|\cdot\|_F^2$$

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

$$\rightarrow w^{(l)} := w^{(l)} - \alpha dw^{(l)}$$

"Weight decay"

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

$$= \underbrace{w^{(l)} - \frac{\alpha \lambda}{m} w^{(l)}}_{\text{weight decay}} - \alpha (\text{from backprop})$$

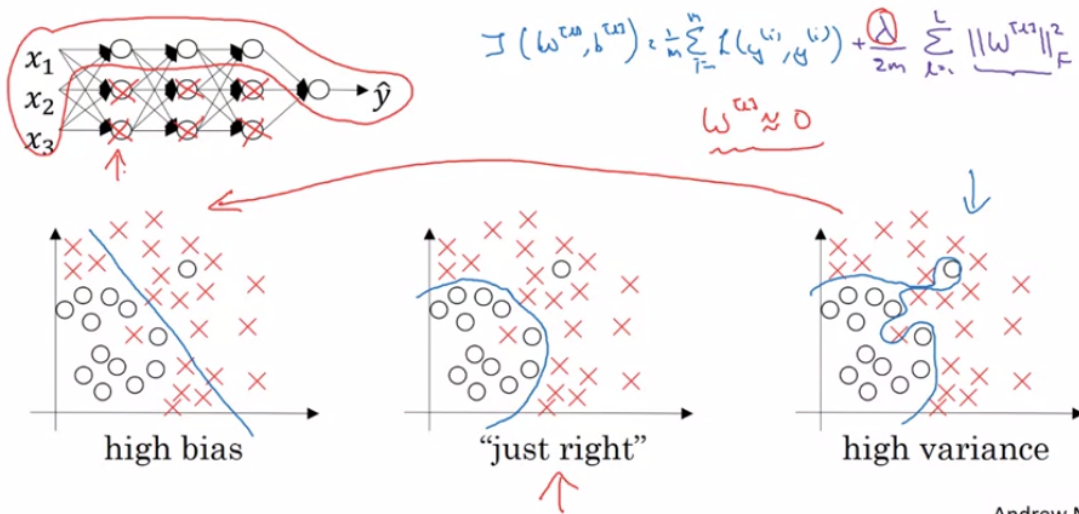
$$\frac{\partial J}{\partial w^{(l)}} = dw^{(l)}$$

Andrew Ng

(also called weight decay)

Why it works?

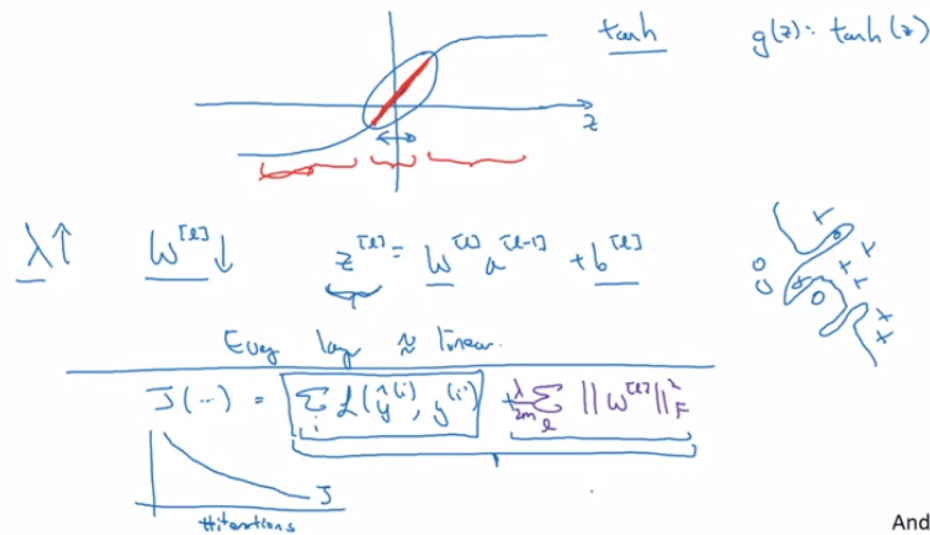
How does regularization prevent overfitting?



Andrew Ng

also, every layer approximately linear

How does regularization prevent overfitting?



Implementing dropout ("Inverted dropout")

Illustrate with layer $l=3$. $\text{keep-prob} = \frac{0.8}{x}$ 0.2

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

$a3 = \text{np.multiply}(a3, d3)$ $\# a3 \times = d3$

$\rightarrow a3 /= \text{keep-prob}$ \leftarrow

50 units. \rightarrow 10 units shut off

$z^{(4)} = w^{(4)} \cdot \underbrace{a^{(3)}}_{\substack{\text{reduced by } 20\% \\ /= 0.8}} + b^{(4)}$ Test

(Inverted dropout makes sure that $\text{Exp}(a)$ stays the same)

Making predictions at test time

$$a^{(6)} = X$$

No drop out.

$$\begin{aligned} z^{(1)} &= w^{(1)} a^{(0)} + b^{(1)} \\ a^{(1)} &= g^{(1)}(z^{(1)}) \\ z^{(2)} &= w^{(2)} a^{(1)} + b^{(2)} \\ a^{(2)} &= \dots \end{aligned}$$

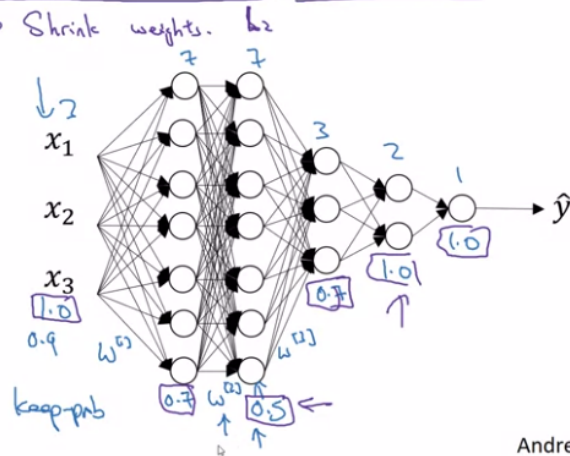
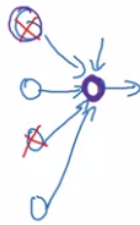
\downarrow
 \hat{y}

$/= \text{keep-prob}$

Why dropout works? (usually used in CV)

Why does drop-out work?

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



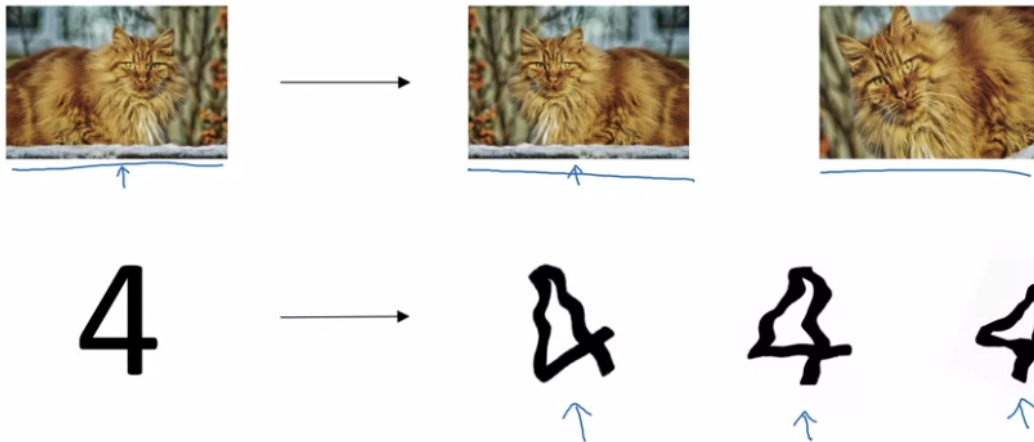
tune keep-prob for different layers

Downside: cost function is not well-defined

Other methods:

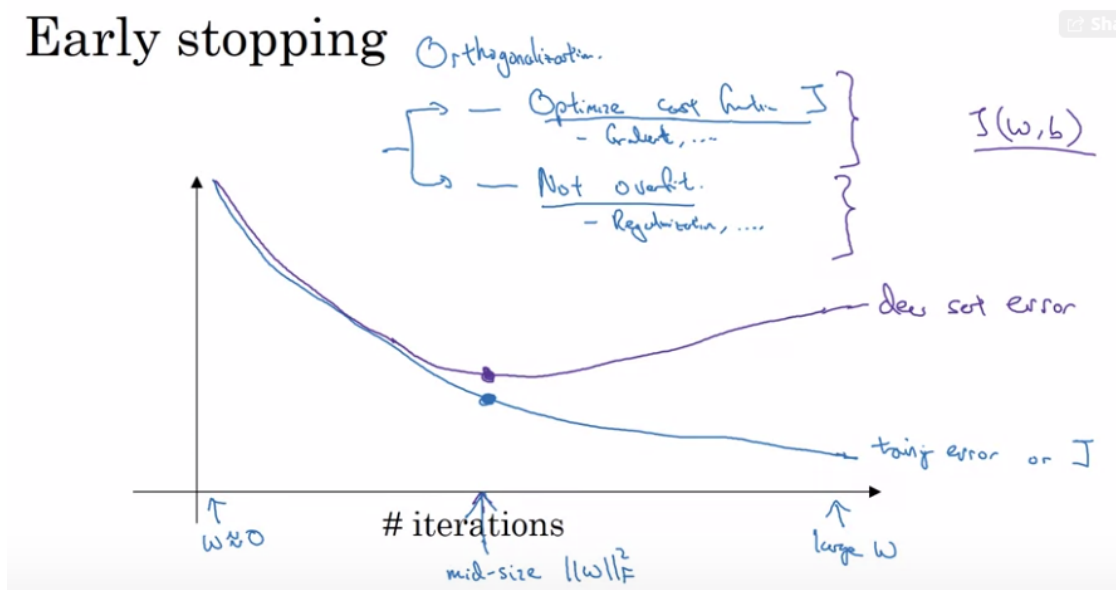
(1) Data augmentation

Data augmentation



(2) Early stopping

Early stopping



Downside: (Orthogonalization, we do one task at a time, but early stopping enquires we do both tasks at the same time)