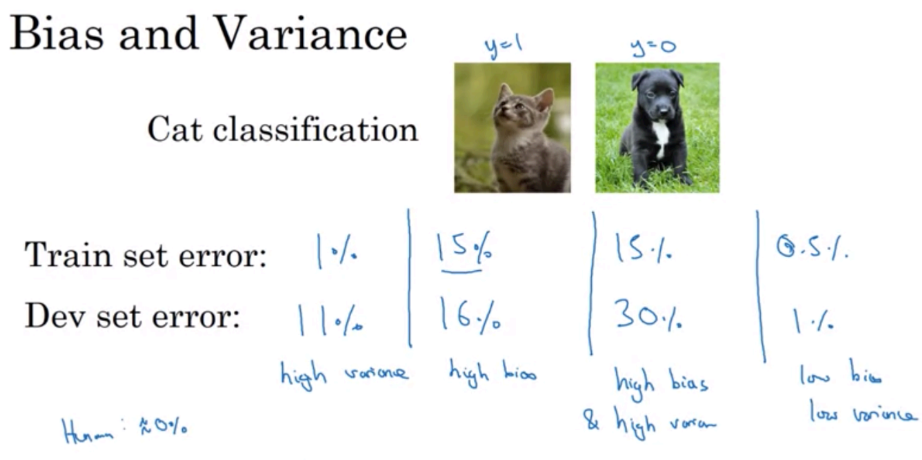
Train / Dev / Test sets

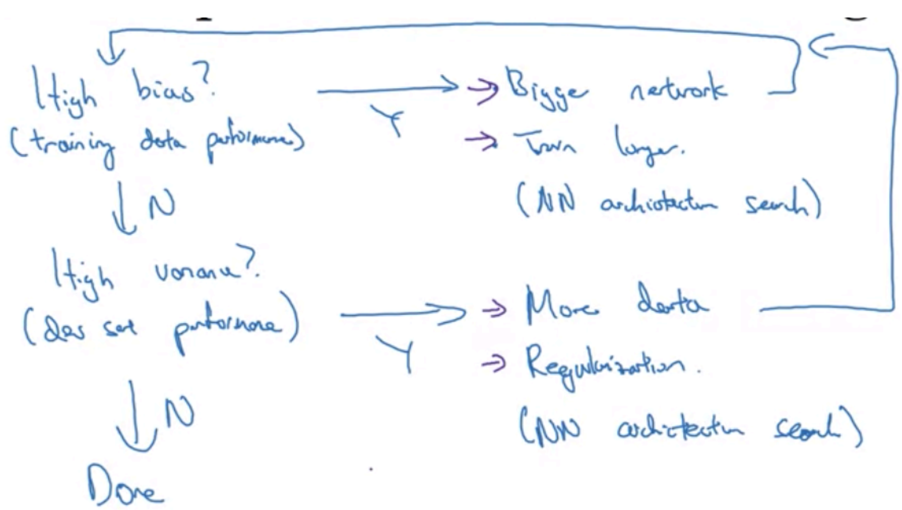
- Applied ML is highly iterative
- Make sure dev and test come from same distribution (this is hard, some people scrap website for more training data, but training distribution may be different)
- Okay not to have test set -> unbiased estimate of model performance (train/dev is okay,)

Bias / Variance



- Optimal error: bases error => usually 0%
- The difference between train set error and dev set error give you good estimate on variance

Basic Recipe for machine learning



- Note: Pre DL era, bias variance trade off, now you can use DL decrease either one without hurting the other.

Regularization -> prevent overfitting, or you can just use more data

Logistic regression

$$\min_{w,b} J(w,b) \qquad w \in \mathbb{R}^{n_x}, \; b \in \mathbb{R}$$

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

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

L1 regularization $\quad \frac{\lambda}{2m}\sum_{i=1}^{n_x} |w| = \frac{\lambda}{2m}\|w\|_1 \qquad$ w will be sparse

- Lambda - regularization parameter, often set using dev set.
- L2 norm used much more often

Neural network

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

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

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

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

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

"Weight decay"

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

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

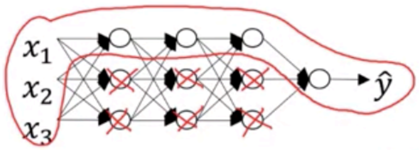
$$\left(1 - \frac{\alpha\lambda}{m}\right) W^{[l]}$$

Andrew Ng

Initialization

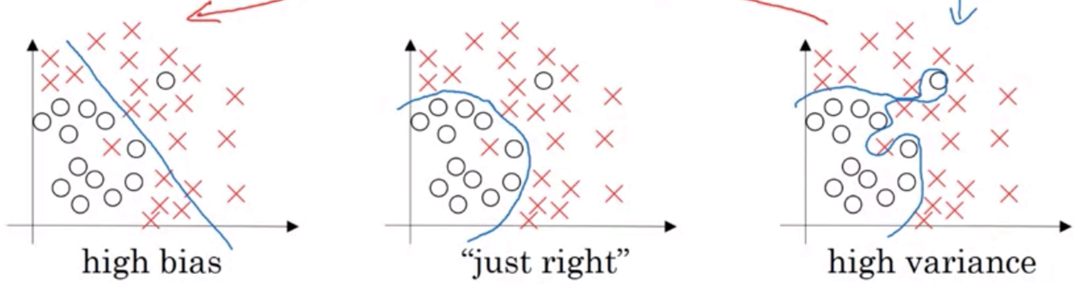
Finally, try "He Initialization"; this is named for the first author of He et al., 2015. (If you have heard of "Xavier initialization", this is similar except Xavier initialization uses a scaling factor for the weights $W^{[l]}$ of `sqrt(1./layers_dims[l-1])` where He initialization would use `sqrt(2./layers_dims[l-1])`.)

How does regularization prevent overfitting?



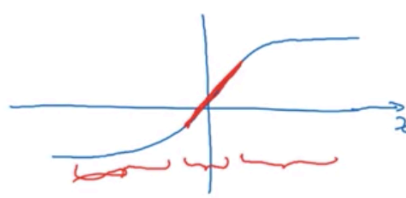
$$J(w^{[l]}, b^{[l]}) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m} \sum_{l=1}^{L} \|w^{[l]}\|_F^2$$

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



Why Regularization prevent overfitting

- Reduce variance, zero out the effect of some hidden unit



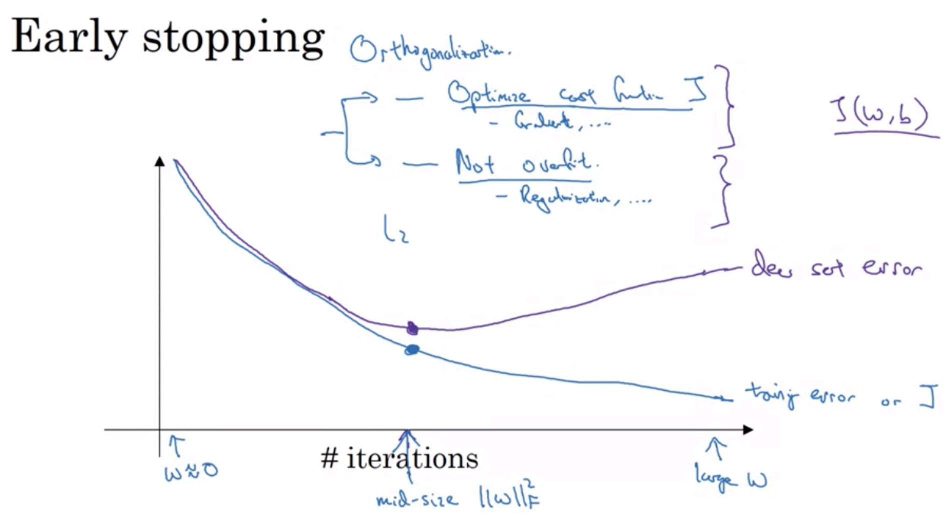
$\lambda \uparrow \quad w^{[l]} \downarrow$

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

```
d3 = np.random.rand(a3.shape[0], a3.shape[1]) < keep-prob
a3 = np.multiply(a3, d3)        # a3 *= d3.
a3 /= keep-prob
```

Another intuition

- Restrict the range of activation function (to almost linear)



Dropout

A3 => inverted dropout (remove scaling problem)

- Making prediction => no dropout

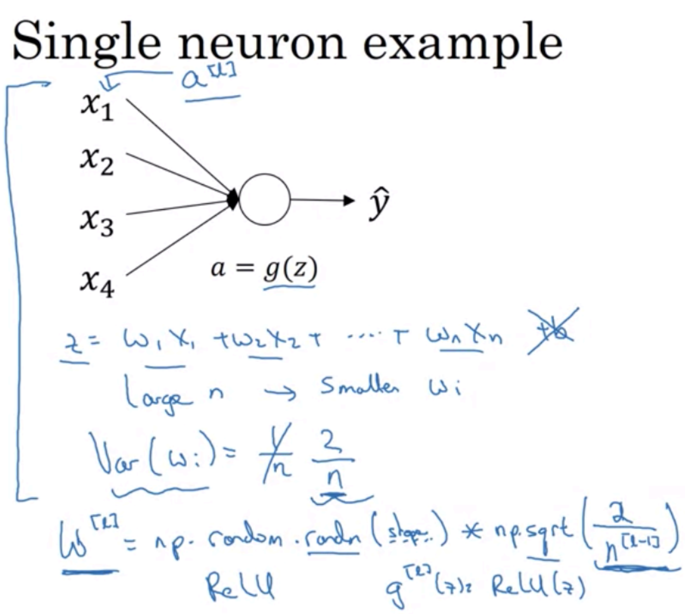
Intuition of Dropout

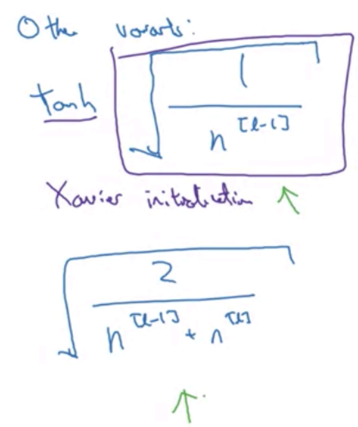
- Can't rely on any one feature, so have to spread out weights. ~> shrink weights
- Kinda similar to L2 norm but not exactly

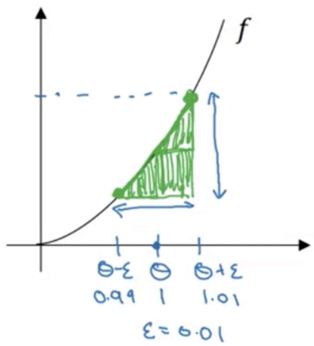
Other methods

- Augmentation
- Early stopping

(Do not need too tune L2 norm)







Normalizing input features
- Why?
  - Less elongated space
  - Gradient descent less oscillating

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

Weight
- Red
  gradient (relu example)
-

Initialization
uce vanishing

^^^^ tanh -> Xavier initialization

$$f'(\theta)=\lim_{\varepsilon\to 0}\frac{f(\theta+\varepsilon)-f(\theta-\varepsilon)}{2\varepsilon} \qquad O(\varepsilon^2) \quad 0.01 \quad 0.0001 \qquad \Bigg| \qquad \frac{f(\theta+\varepsilon)-f(\theta)}{\varepsilon} \qquad \text{error: } O(\varepsilon) \quad 0.01$$

Andrew Ng

Numer
ical

Approx of Gradients

- Big green angle gives better approximation (2-sided difference)
- Twice as slow, but worth it for accuracy

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

Concatenate

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

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

Concatenate

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

Definiti
on of

limit: with relate to limit in numerical approximation (think about CS370)

Why 2 sided (on the left) is much more accurate

Gradient checking (Grad check)

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

for each i:

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

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

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

Check

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

10^{-5}

10^{-3} – worry.

$\varepsilon = 10^{-7}$

Gradient Checking -> debugging gradient

Goal?

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.

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

  $d\theta$ = 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

How to check? Compute the Euclidean distance of the 2