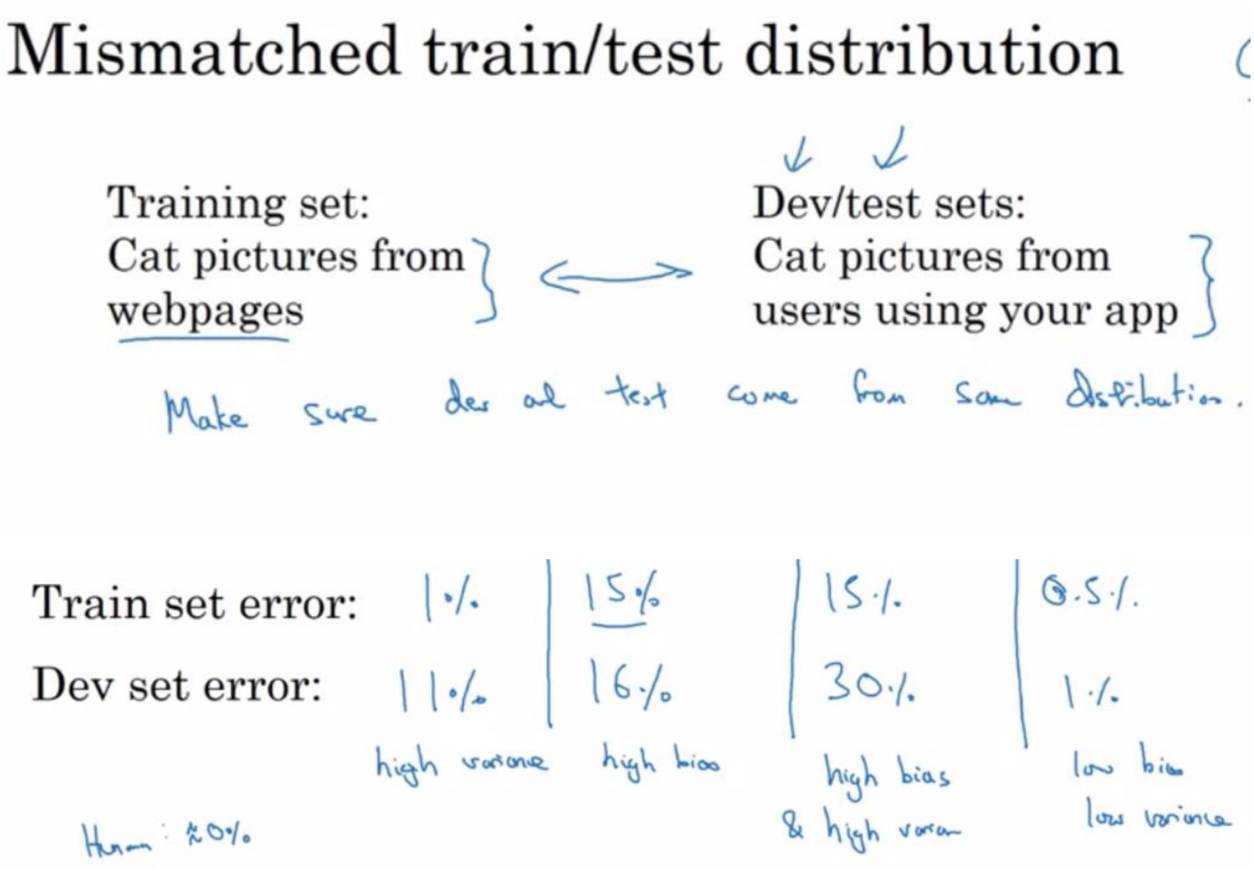
Hyperparameters Tuning

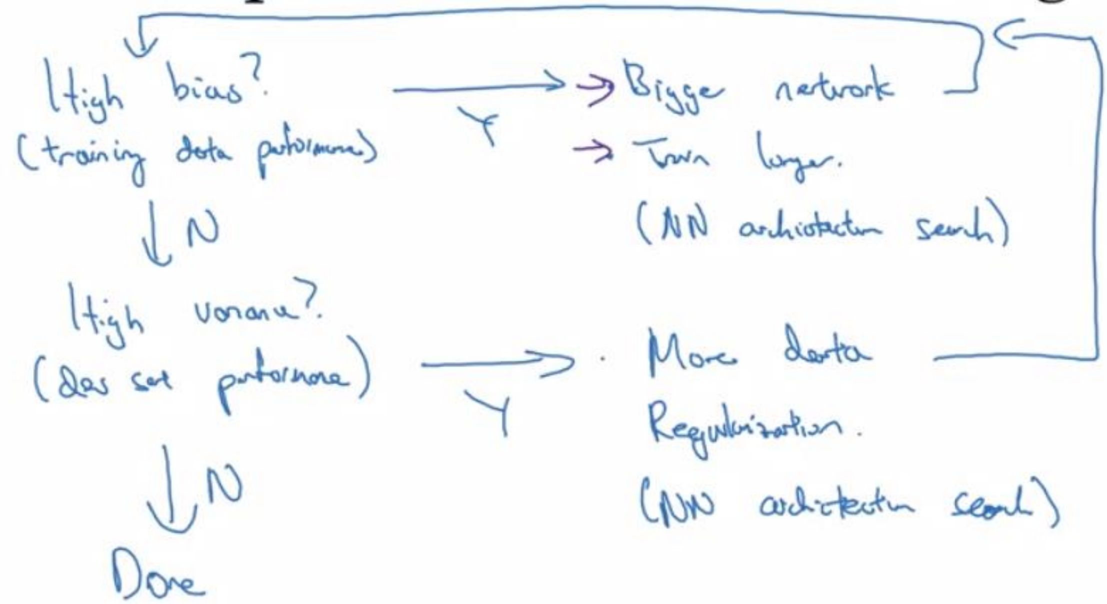
Week 1

Applied ML is a highly iterative process.

The gathered data is split intro training, development(cross-validation) and test set. In the previous era, the split ratio was about 60-20-20. But now, in the Big Data era, the dev and test set require to have a much smaller ratio, much of the data being used in the training set.



Basic recipe for machine learning



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

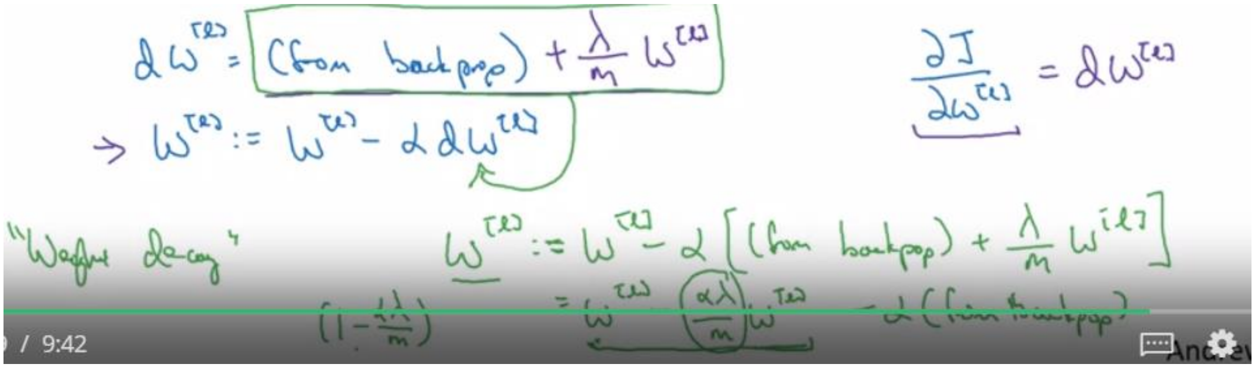
L1 regularization $\frac{\lambda}{2m} \sum_{i=1}^{n_x} |w| = \frac{\lambda}{2m} \|w\|_1$ w will be sparse

Lambda = Regularization Parameter (given by Dev Set)

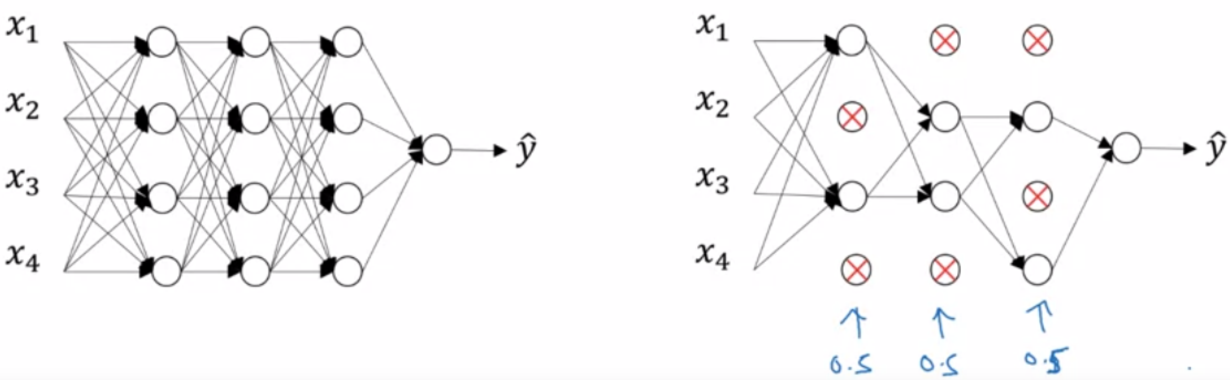
Neural network

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

Frobenius Norm of a Matrix



Dropout regularization



Implementing dropout ("Inverted dropout")

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

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

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

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

50 units. $\rightsquigarrow$ 10 units shut off

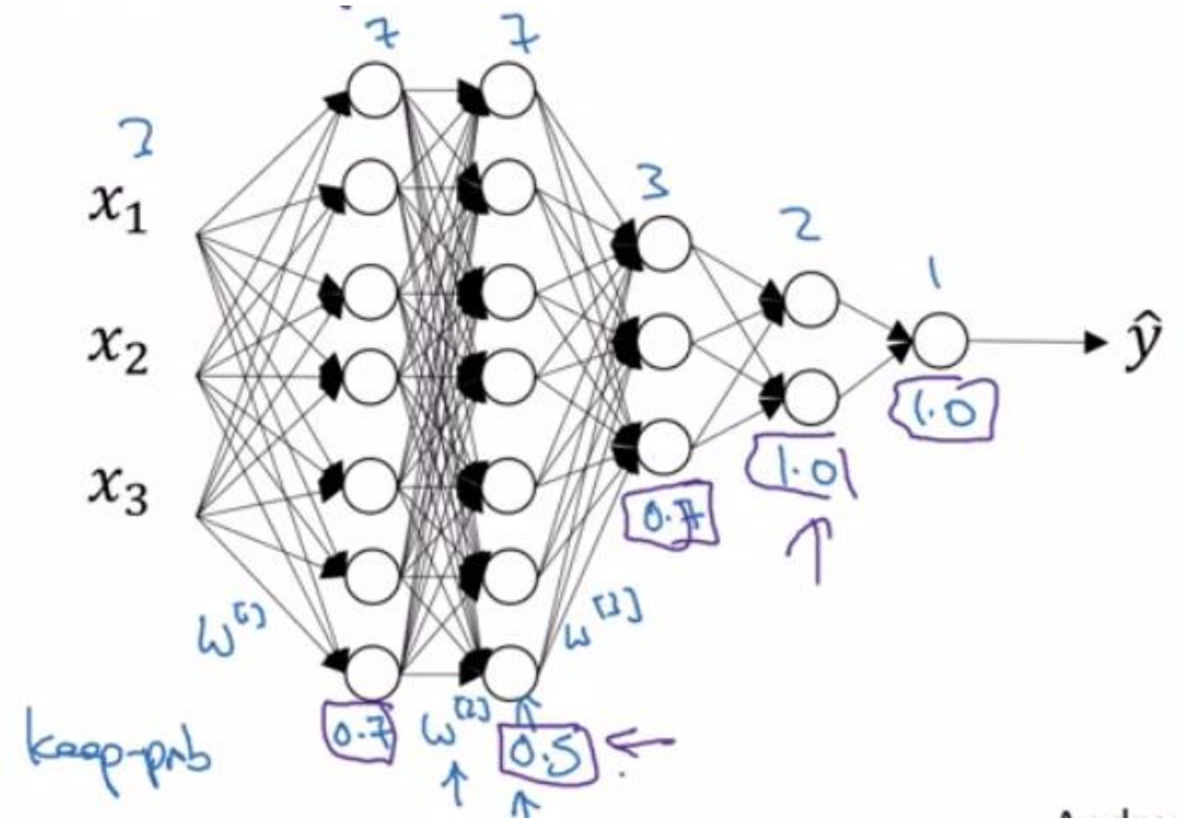
$$z^{[4]} = W^{[4]} \cdot a^{[3]} + b^{[4]}$$

reduced by 20%.

/= 0.8

Why does drop-out work?

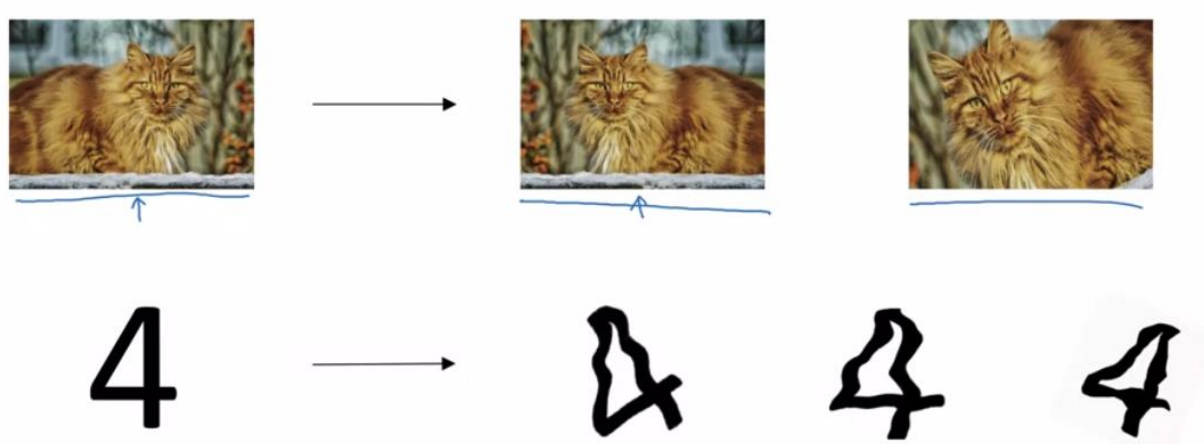
Intuition: Can't rely on any one feature, so have to spread out weights. $\rightsquigarrow$ Shrink weights. L2

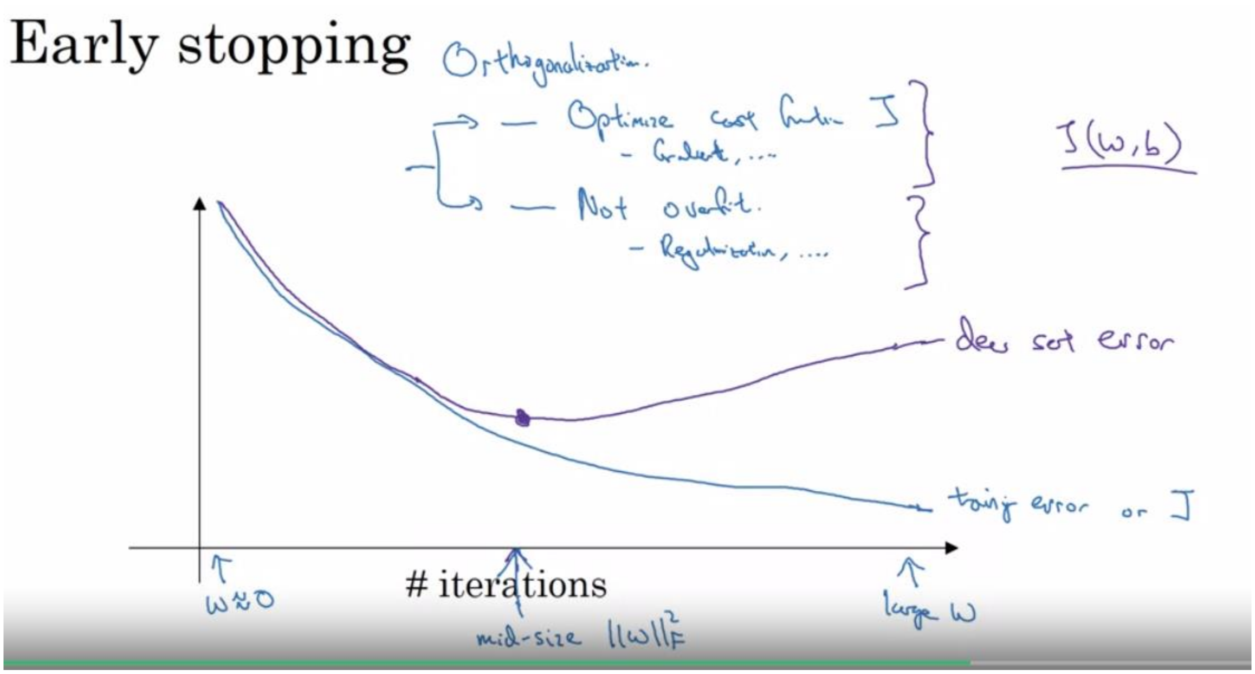


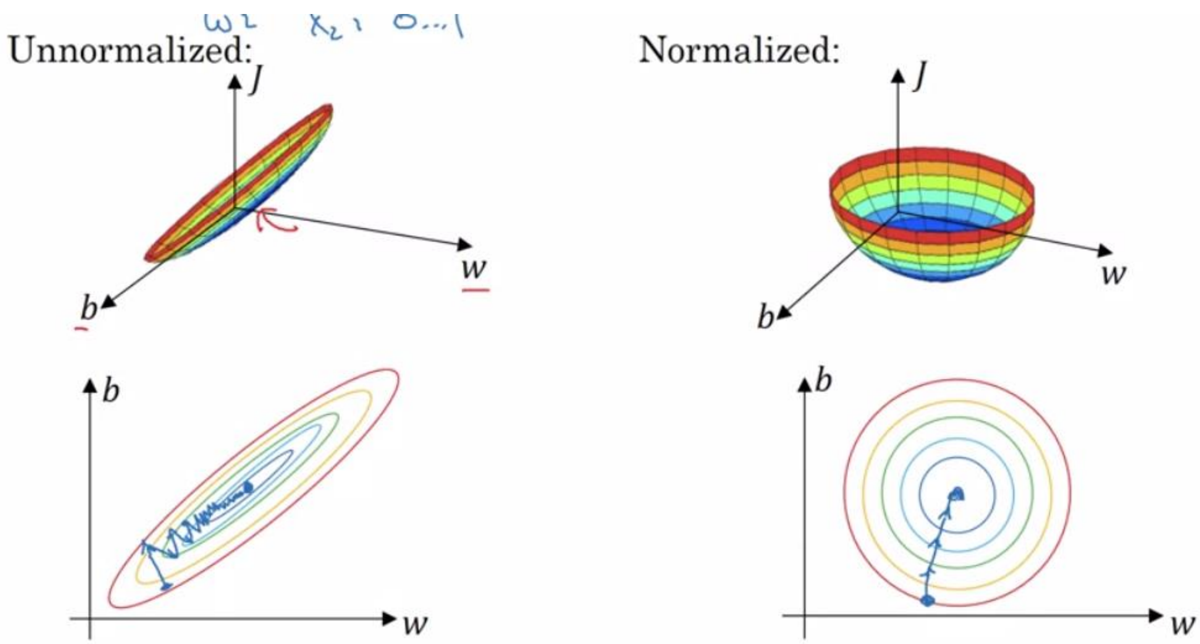
This technique is usually used in Computer Vision.

One disadvantage of the Dropout Regularization is that now the Cost Function is not "well-defined", and plotting J over Iterations may not always result in a smooth decreasing graph.

Data augmentation







Vanishing/Exploding Gradients Problem

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

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

Other variants:

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

Xavier initialization

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

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

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

$$\varepsilon = 10^{-7}$$

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])`.)

He initialization works well for networks with RELU activations.

With dropout, your neurons thus become less sensitive to the activation of one other specific neuron, because that other neuron might be shut down at any time.

- A **common mistake** when using dropout is to use it both in training and testing. You should use dropout (randomly eliminate nodes) only in training.
- Deep learning frameworks like tensorflow, PaddlePaddle, keras or caffe come with a dropout layer implementation. Don't stress - you will soon learn some of these frameworks.

Note that regularization hurts training set performance! This is because it limits the ability of the network to overfit to the training set. But since it ultimately gives better test accuracy, it is helping your system.