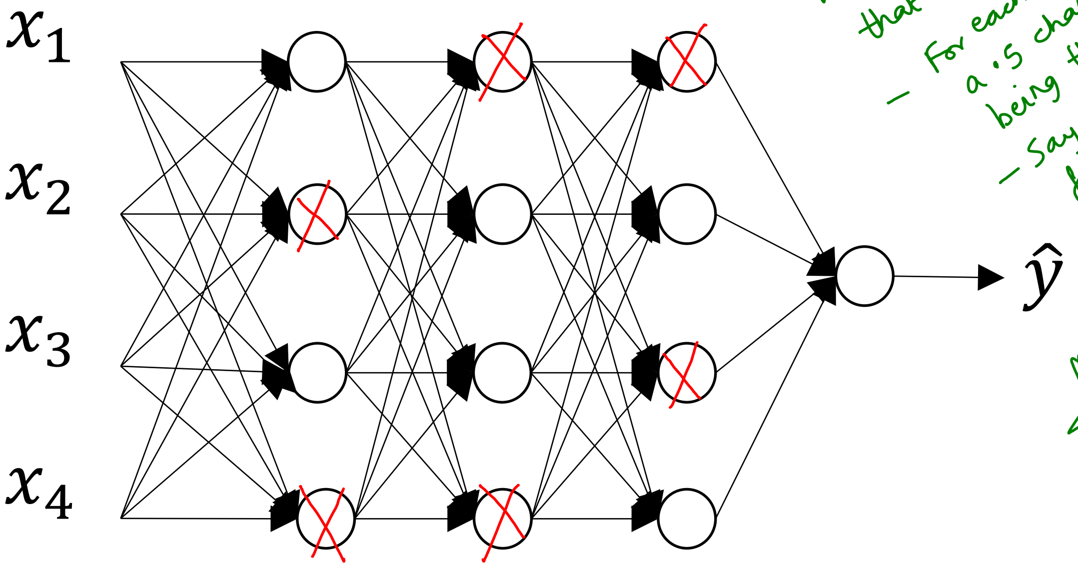
deeplearning.ai

Regularizing your neural network

Dropout regularization

Dropout regularization



Say this NN is overfitting → Say we toss a coin for each node

- If it comes as "heads", we remove that node
- For each node, there is a .5 chance of it not being there
- Say we get heads for 6 nodes marked in Red
- For that node, all its incoming/outgoing weights are also removed

☆ - This was only 1 simulation of dropout, do this At every iteration of the weight update

⇒ Your NN keeps getting smaller & smaller at every epoch

⇒ Regularization at each epoch

→ This way we have achieved good perf. w/ lesser # of nodes

Andrew Ng

Implementing dropout ("Inverted dropout")

For a single layer, $L = 3$ (say)

d3 = np.random.rand(a3.shape[0], a3.shape[1]) < keepProb

dropout vector for layer 3

Rows

Activations at layer 3

columns

keepProb = prob of keeping that node in the layer

Say np.Random Generates

[0.4, 0.5, 0.9, 0.7, 0.95] < 0.8 → keepProb

↳ a3's dim was (5×1)

= [1, 1, 0, 1, 0]

ie, 3rd & 5th elements are eliminated

After this, we do

a3 = np.multiply(a3, d3)

↳ only 1st, 2nd, 4th values are Retained

a3 = a3/keepProb } Inverted Dropout

Why? → Say "a" had (50×1) dim

If keepProb = .8 ⇒ 20% of "a" is gone ⇒ We have effectively made "a" (40×1)

$z^{[4]} = \underbrace{w^{[4]} \cdot a^{[3]}}_{\text{expected Value has } \downarrow} + b^{[4]}$

∴ to Bump up expected value of a

we do a = a/keepProb

= a/.8

This way a's expected value will still evaluate to be similar to what it was if there was no dropout

(But dim(a) ↓)

Andrew Ng

Making predictions at test time

$a^{[0]} = X$

No drop out at test time → When you make a prediction during test time, you dont want predictions to be Random

$z^{[1]} = w^{[1]} \cdot a^{[0]} + b^{[1]}$

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

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

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

$\vdots$

L times to get $a^{[L]}$ or $\hat{y}$

Andrew Ng

deeplearning.ai

Regularizing your neural network

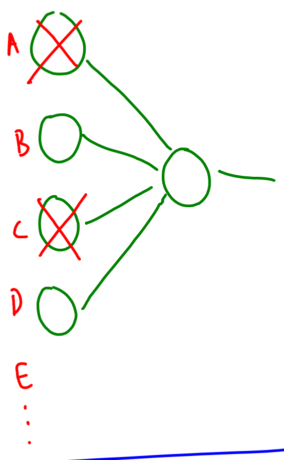
Understanding dropout

Why does drop-out work?

Keep Prob varies by layer
$\Rightarrow W^{[1]} = 7 \times 3$ (keepProb = .5)
$W^{[2]} = 7 \times 7$ (keepProb = .3)
$W^{[3]} = 3 \times 7$ (keepProb = .5)
$W^{[4]} = 2 \times 3$ (keepProb = .8)
$W^{[5]} = 1 \times 2$ (keepProb = .9)

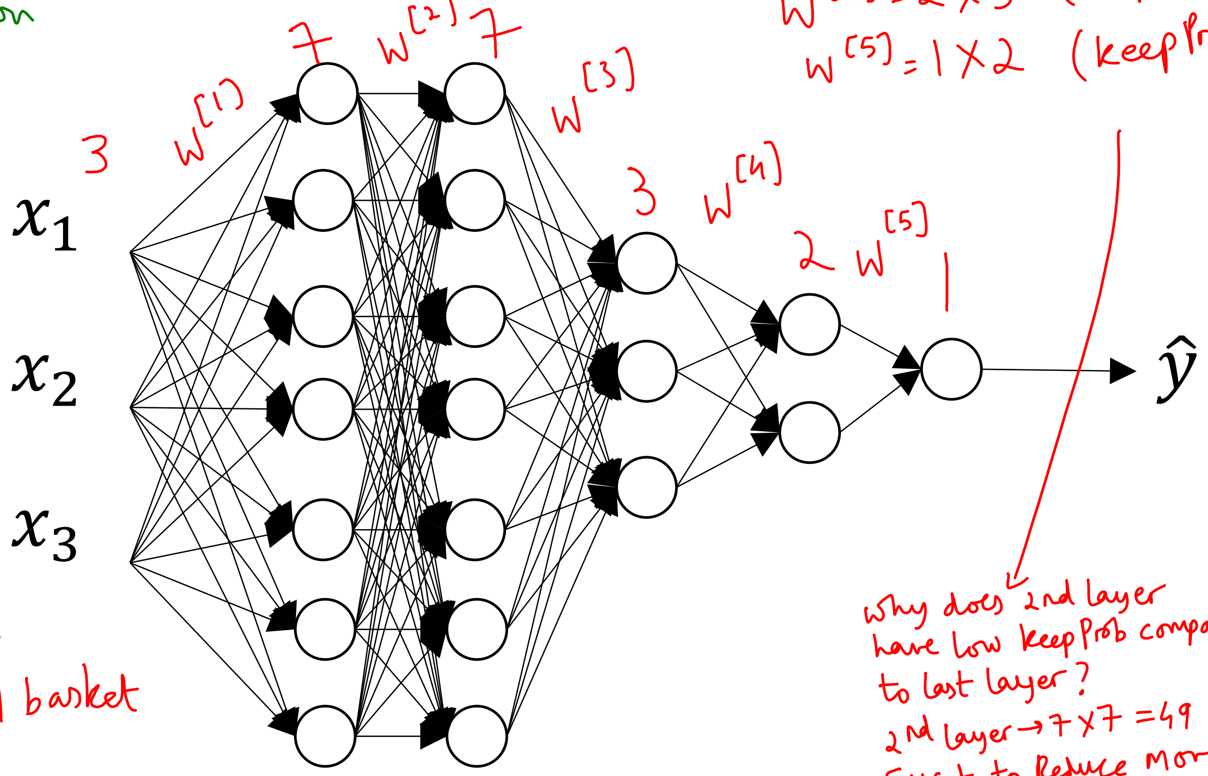
Intuition: Can't rely on any one feature, so have to spread out weights.

- Dropout has similar effect to L2 Regularization
- spread out weights $\Rightarrow$ weights $\downarrow$



In each iteration, some nodes are removed
$\Rightarrow$ In iteration 1, maybe A is removed
In iteration 2, C
3, F, X
4, K, P

$\Rightarrow$ The NN will try to spread out weights to all its nodes so that it doesn't put all eggs in 1 basket



Why does 2nd layer have low keep Prob compared to last layer?
2nd layer $\rightarrow 7 \times 7 = 49$ nodes
[want to Reduce more nodes]
last layer $\rightarrow 2 \times 1 = 2$ nodes

Dropout is used more in computer vision because there is less data (more overfitting). May not make sense to use in other fields

Downside: J is no longer well defined
$\Rightarrow$ wont necessarily be monotonically $\downarrow$
(J = cost) $\Rightarrow$ You wont see - J

Andrew Ng