Neural Nets

1: Neural Nets

Neural Nets

- Linear combination of features

  b + w1 * x1 + w2 * x2 + … + wp * xp
- Non-linear activation (e.g. **wx** -> [-1, 1])

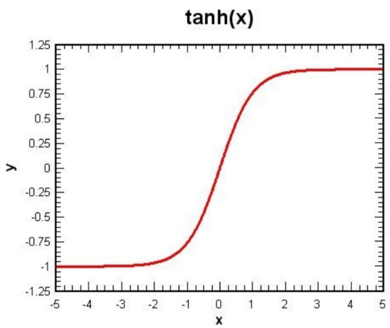
  f(b + w1 * x1 + w2 * x2 + … + wp * xp)
- Pass outputs on as inputs (multi-layer nets)
  - This gives non-linear decision boundaries
  - This gives a non-convex problem :(

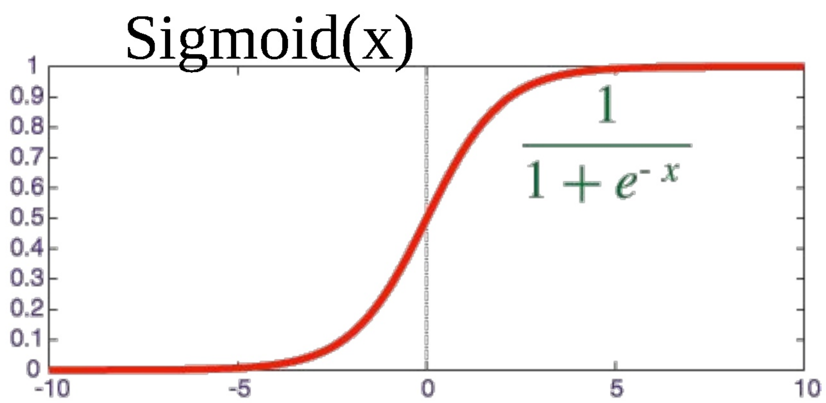
Neural Nets

- Linear combination of features

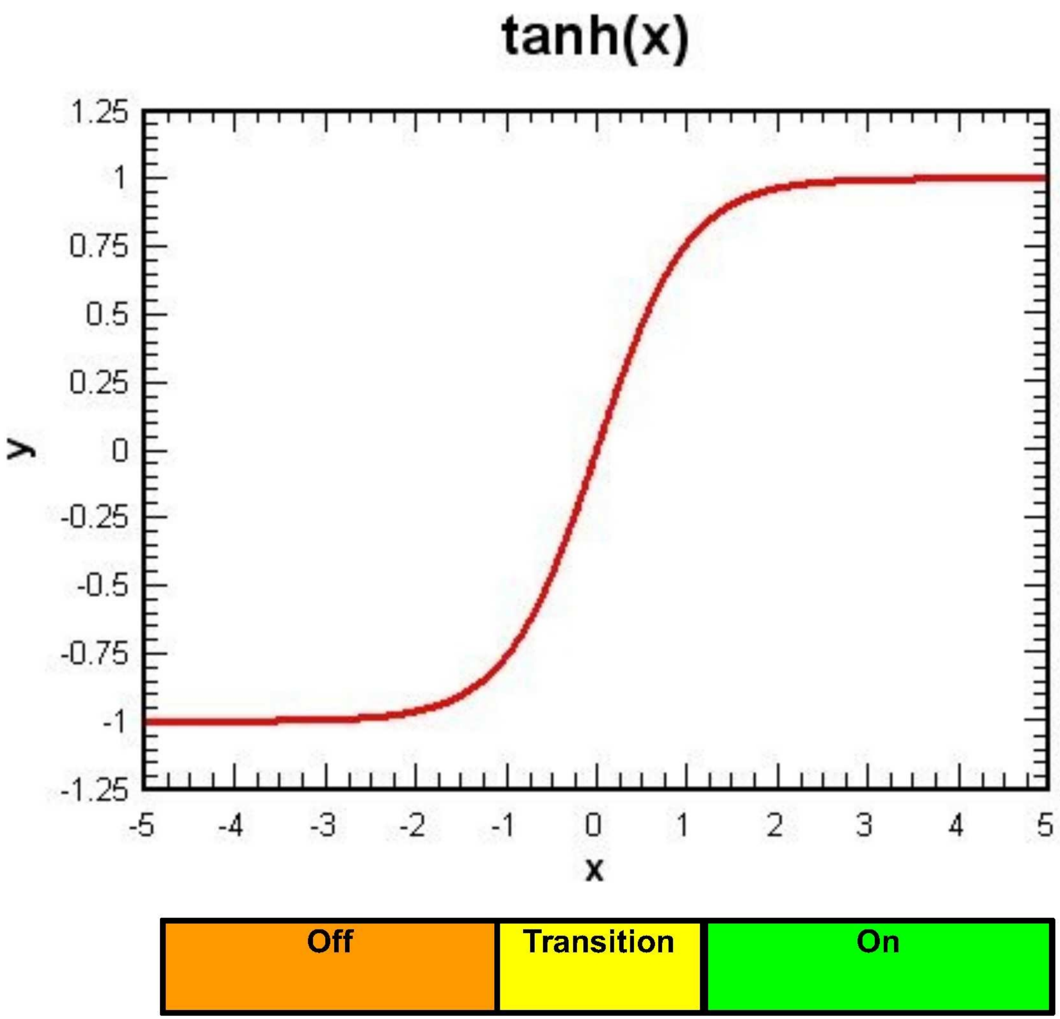
  b + w1 * x1 + w2 * x2 + … + wp * xp

- Non-linear activation

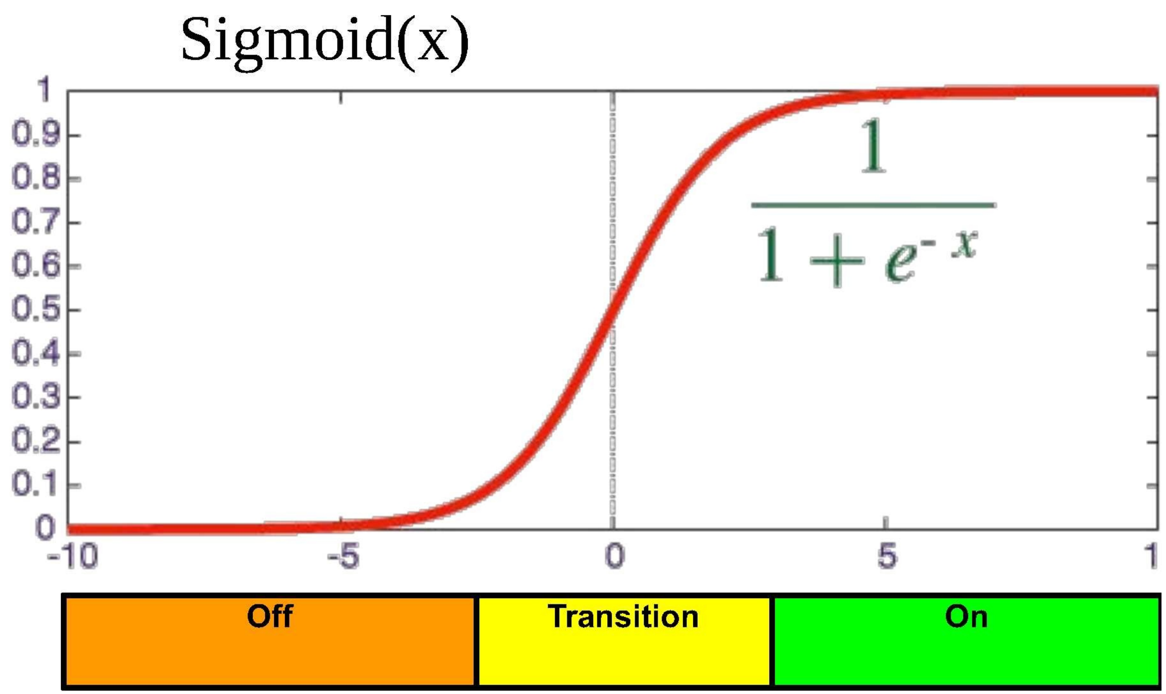




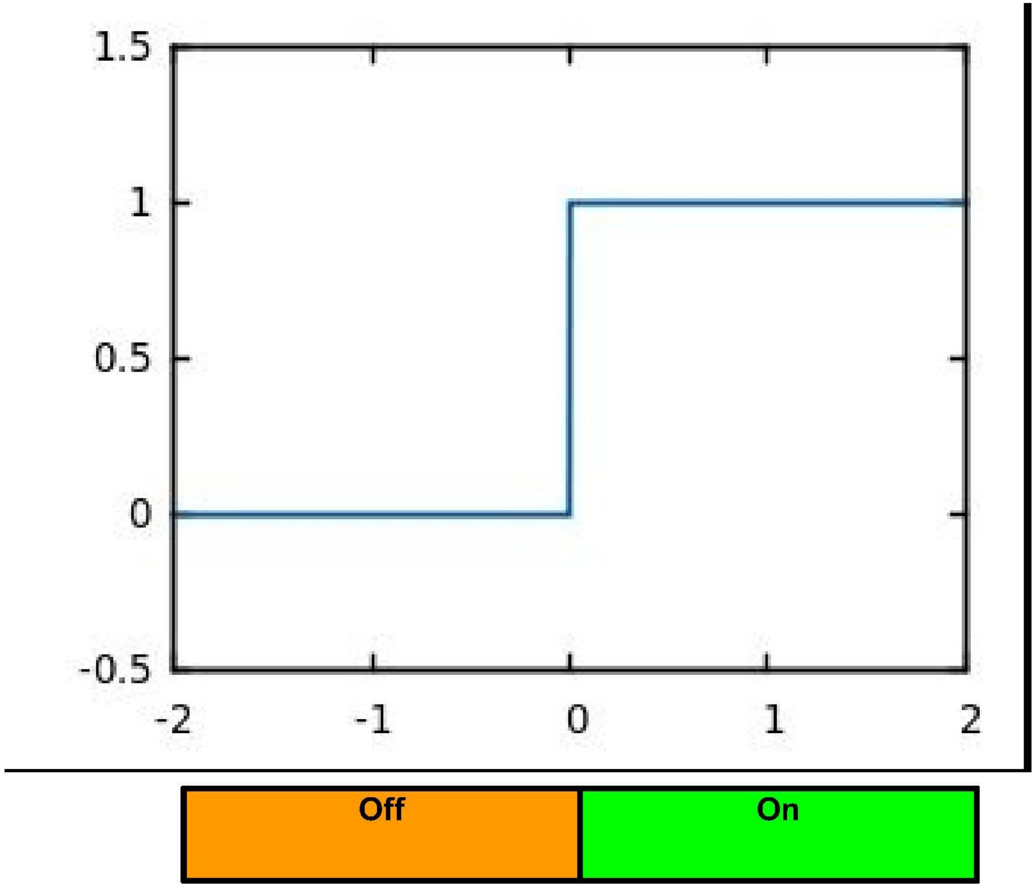
- Non-linear activation:
  - Once linear combination exceeds a cutoff dramatic change in behavior



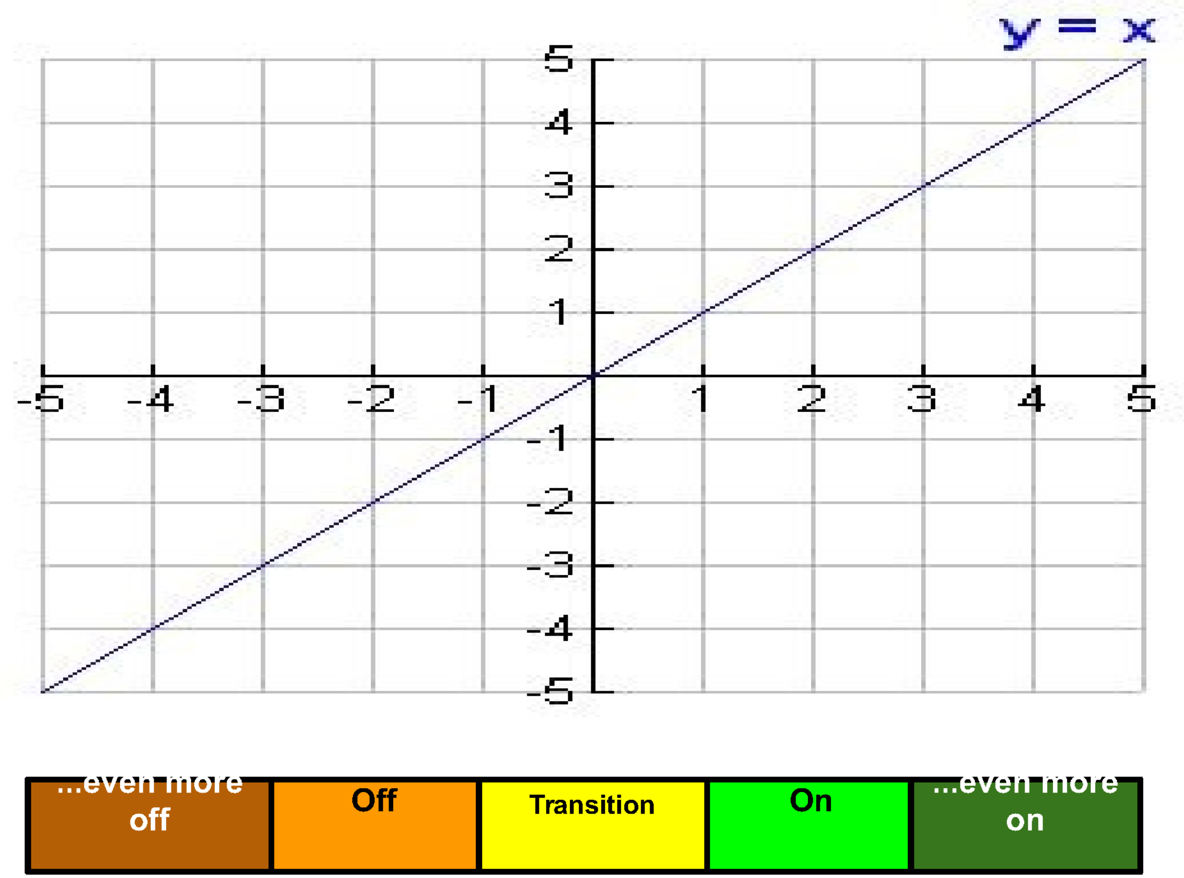
- Non-linear activation:
  - Once linear combination exceeds a cutoff dramatic change in behavior



Q: why use nonlinear activation instead of a *step activation* (no transition)?



Q: why use nonlinear activation instead of *linear activation*?



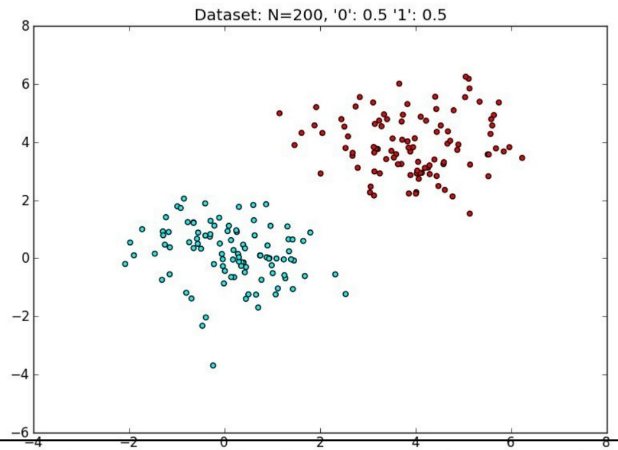
Example: Logistic Regression

Linear combination of features (log-odds), passed into sigmoid activation function to get *p*.

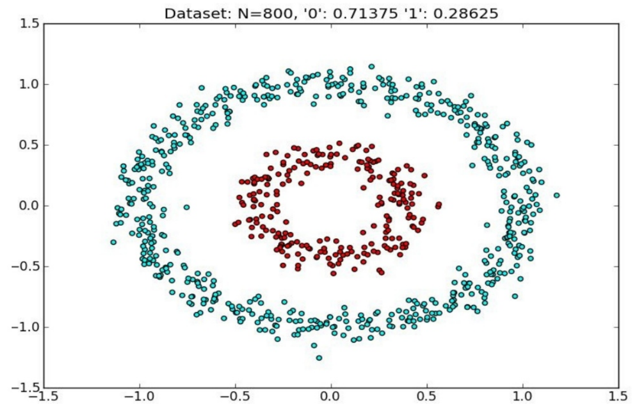
$$\ell odds = \log \frac{p}{1-p} = x_1\beta_1 + \ldots + x_m\beta_m$$
$$\implies p = \exp(\ell odds)/(1 + \exp(\ell odds)))$$

Layering for nonlinearity

- Single layer nets (e.g. logistic regression) can make linear decisions:



Linearly separable (you can draw a straight line to separate these two colors)



Not linearly separable (you can't draw a straight line to separate the two colors)

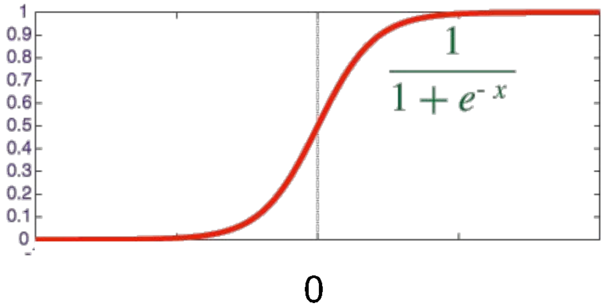
Layering for nonlinearity

- Multi-layer nets can approximate any function, i.e. can do any nonlinear separation.
  - Downside: the optimization problem is nonconvex
  - Downside: you may need *lots* of nodes / layers

Neural Net OR

Y = sigmoid(bias + X1w1 + X2w2)

bias = -15
(intercept)



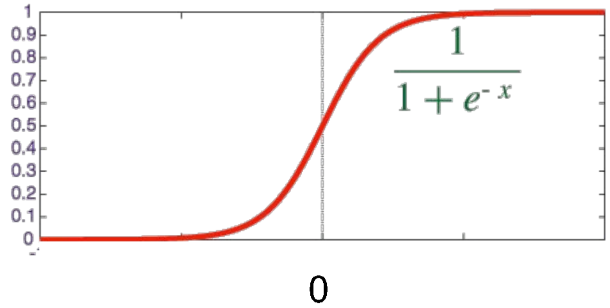
	X1	X2	Y
	0	0	0
	0	1	1
	1	0	1
	1	1	1
w	?	?	

Neural Net OR

Y = sigmoid(bias + X1w1 + X2w2)

bias = -15

(intercept)



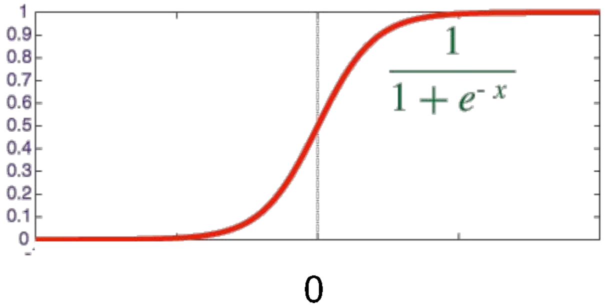
	X1	**X2**	**wx + bias**	**Y**
	0	0	-15	~0
	0	1	5	~1
	1	0	5	~1
	1	1	25	~1
w	+20	+20		

Either X being active enough to turn on Y

Neural Net AND

Y = sigmoid(bias + X1w1 + X2w2)

bias = -15
(intercept)

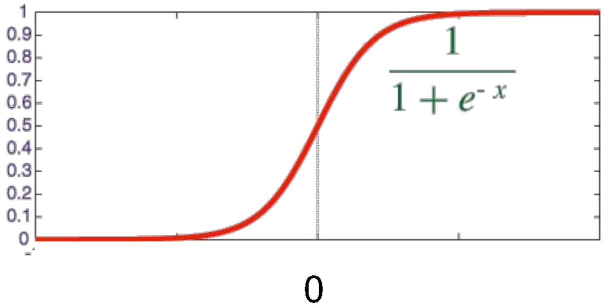


	X1	X2	Y
	0	0	0
	0	1	0
	1	0	0
	1	1	1
w	?	?	

Neural Net AND

Y = sigmoid(bias + w1X1 + w2X2)

bias = -15
(intercept)



	X1	X2	wx + bias	Y
	0	0	-15	~0
	0	1	-5	~0
	1	0	-5	~0
	1	1	5	~1
w	+10	+10		

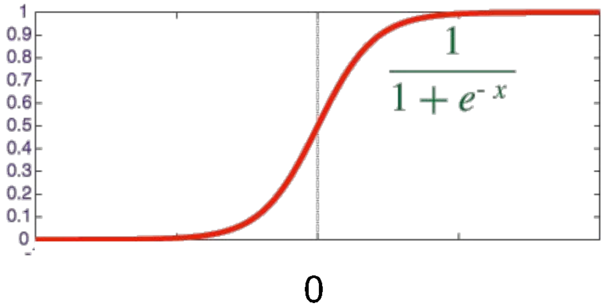
Need both X to be active to turn on Y

Neural Net XOR

can't do it with one layer

bias = -15

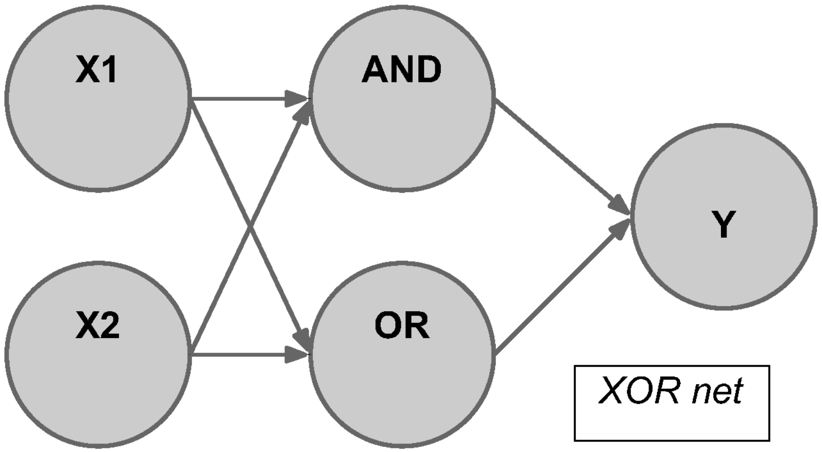
(intercept)



	X1	X2	Y
	0	0	0
	0	1	1
	1	0	1
	1	1	0
w	?	?	

Neural Net XOR

insert layer that does AND, OR (we know how to make AND and OR operations!)



W

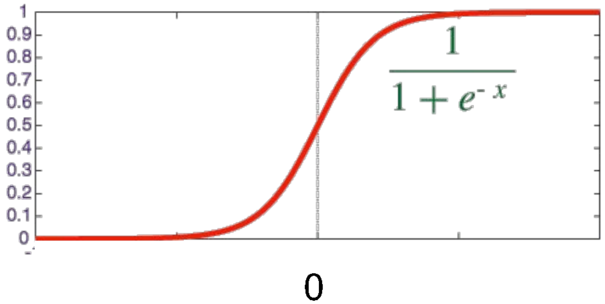
X1	X2	X1& X2	X1| X2	Y
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0
?	?	?	?	

Neural Net XOR

assume we have outputs of AND and OR layer...

Y = sigmoid(bias + w1A1+ w2A2)

bias = -15
(intercept)



	A1:= X1&X2	A2:= X1|X2	Y
	0	0	0
	0	1	1
	0	1	1
	1	1	0
w	?	?	

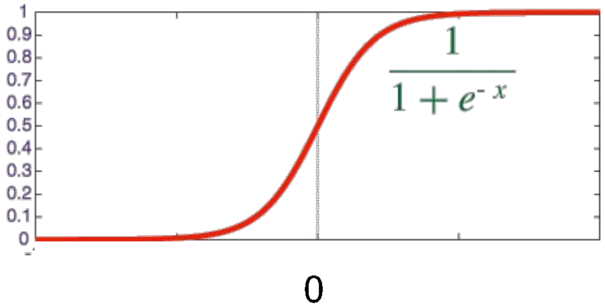
Neural Net XOR

assume we have outputs of AND and OR layer...

Y = sigmoid(bias + w1A1+ w2A2)

bias = -15

(intercept)



	A1:= X1&X2	**A2:= X1|X2**	**bias wA+**	**Y**
	0	0	-15	~0
	0	1	5	~1
	0	1	5	~1
	1	1	-5	~0
w	-10	+20		

The OR alone can turn Y on, but the AND disables it.

Neural net computation

- In practice: many layers, many nodes, very non-convex
  - Lots of calculations
    - Many nodes
    - Try lots of starting values
  - Parallel computing (most nodes don't interact with each other)
    - e.g. for XOR, could compute AND, OR in parallel

Backpropagation briefly

- Perceptron:
  - Find error, update parameters so error might get fixed
- Back-propagation:
  - Neural nets fit using gradient descent
  - BP also finds an error, at the output layer, then it sends it backwards through the neural net