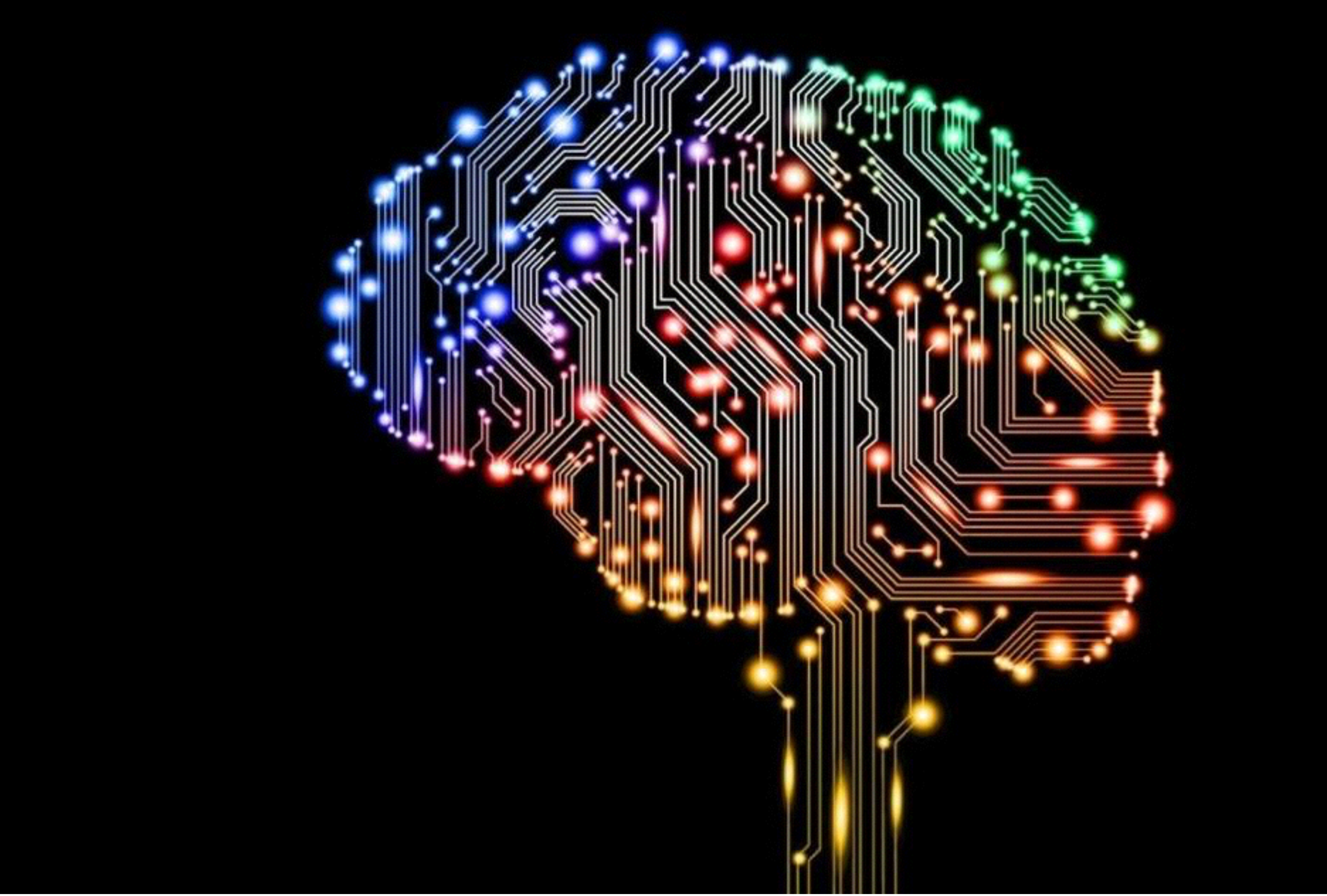
Deconstructing Data Science

David Bamman, UC Berkeley

Info 290
Lecture 16: Neural networks

Mar 16, 2017



https://www.forbes.com/sites/kevinmurnane/2016/04/01/what-is-deep-learning-and-how-is-it-useful

Neural network libraries

TensorFlow

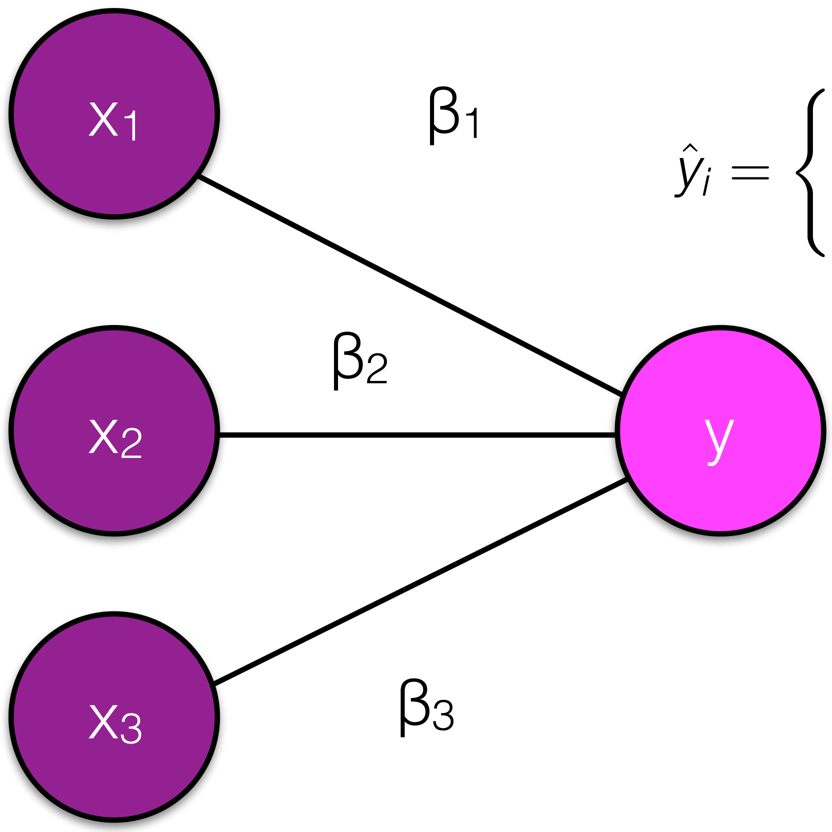
theano

The perceptron, again

$$\hat{y}_i = \begin{cases} 1 & \text{if } \sum_i^F x_i \beta_i \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

	x	β
not	1	-0.5
bad	1	-1.7
movie	0	0.3

The perceptron, again

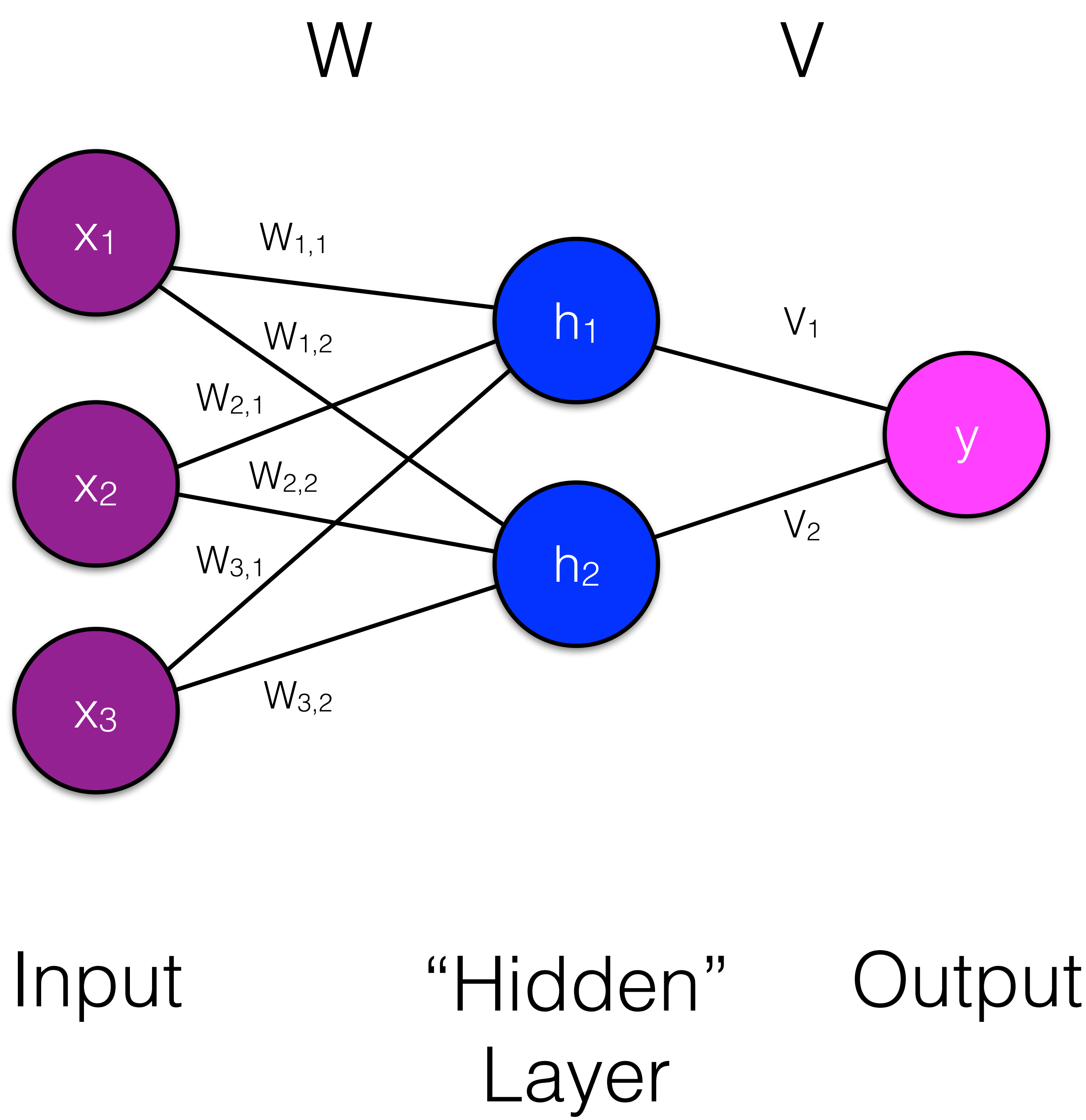


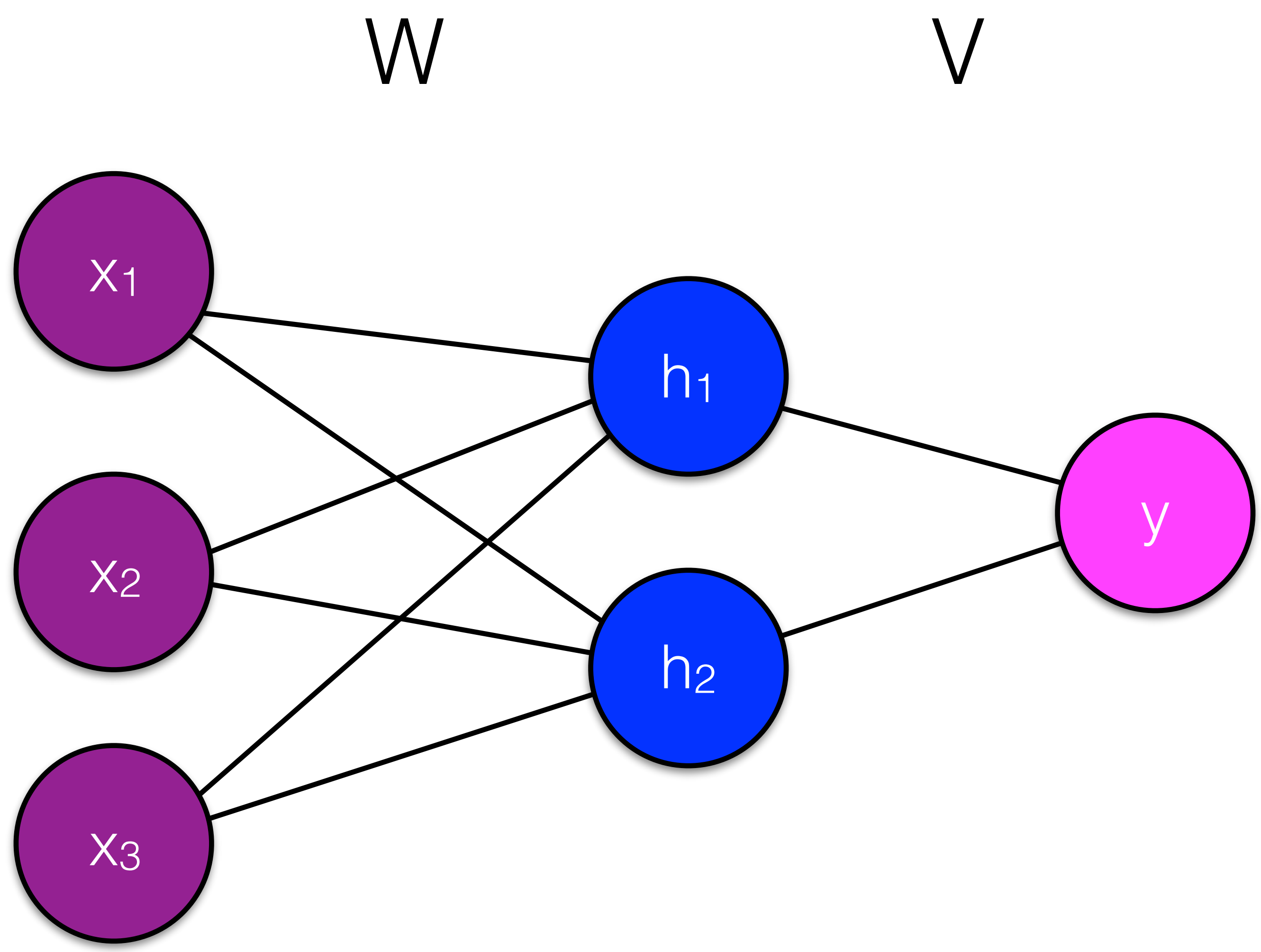
$$\hat{y}_i = \begin{cases} 1 & \text{if } \sum_i^F x_i\beta_i \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

	x	β
not	1	-0.5
bad	1	-1.7
movie	0	0.3

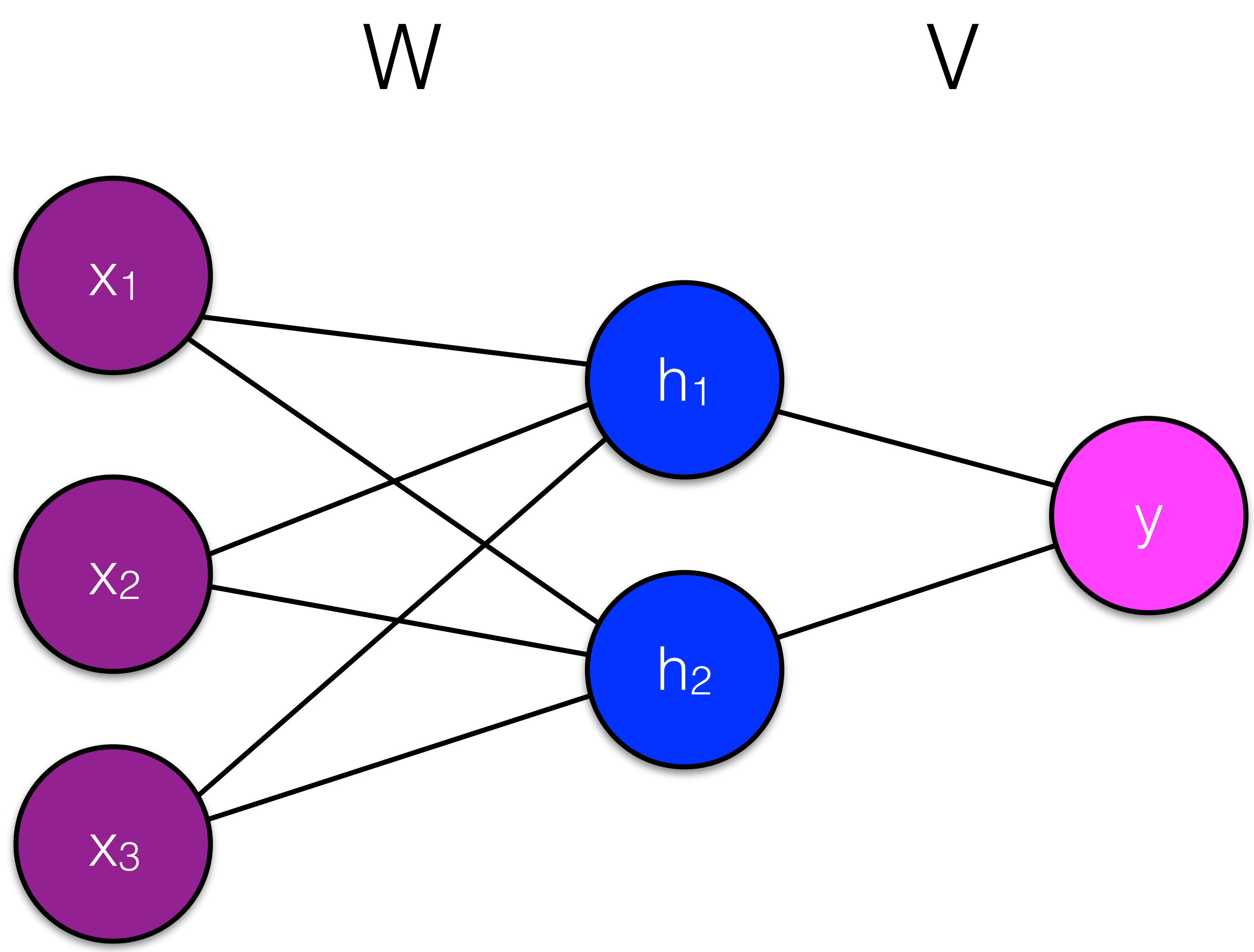
Neural networks

- Two core ideas:
    - Non-linear activation functions
    - Multiple layers


W
V
x1
x2
x3
W1,1
W1,2
W2,1
W2,2
W3,1
W3,2
h1
h2
V1
V2
y
Input
"Hidden" Layer
Output

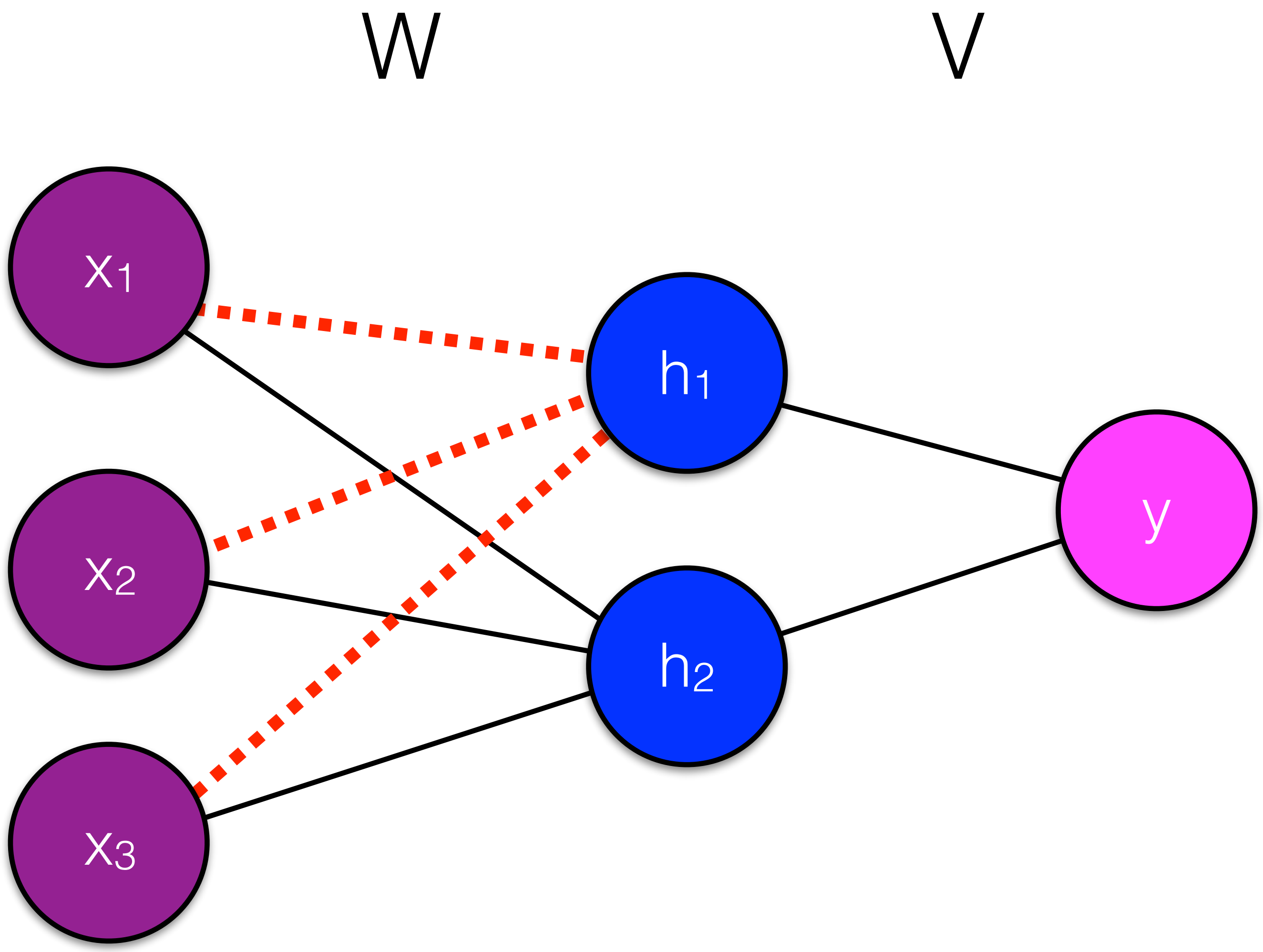


	x	W		V	y
not	1	-0.5	1.3	4.1	-1
bad	1	0.4	0.08	-0.9	
movie	0	1.7	3.1		



$$h_j = f\left(\sum_{i=1}^{F} x_i W_{i,j}\right)$$

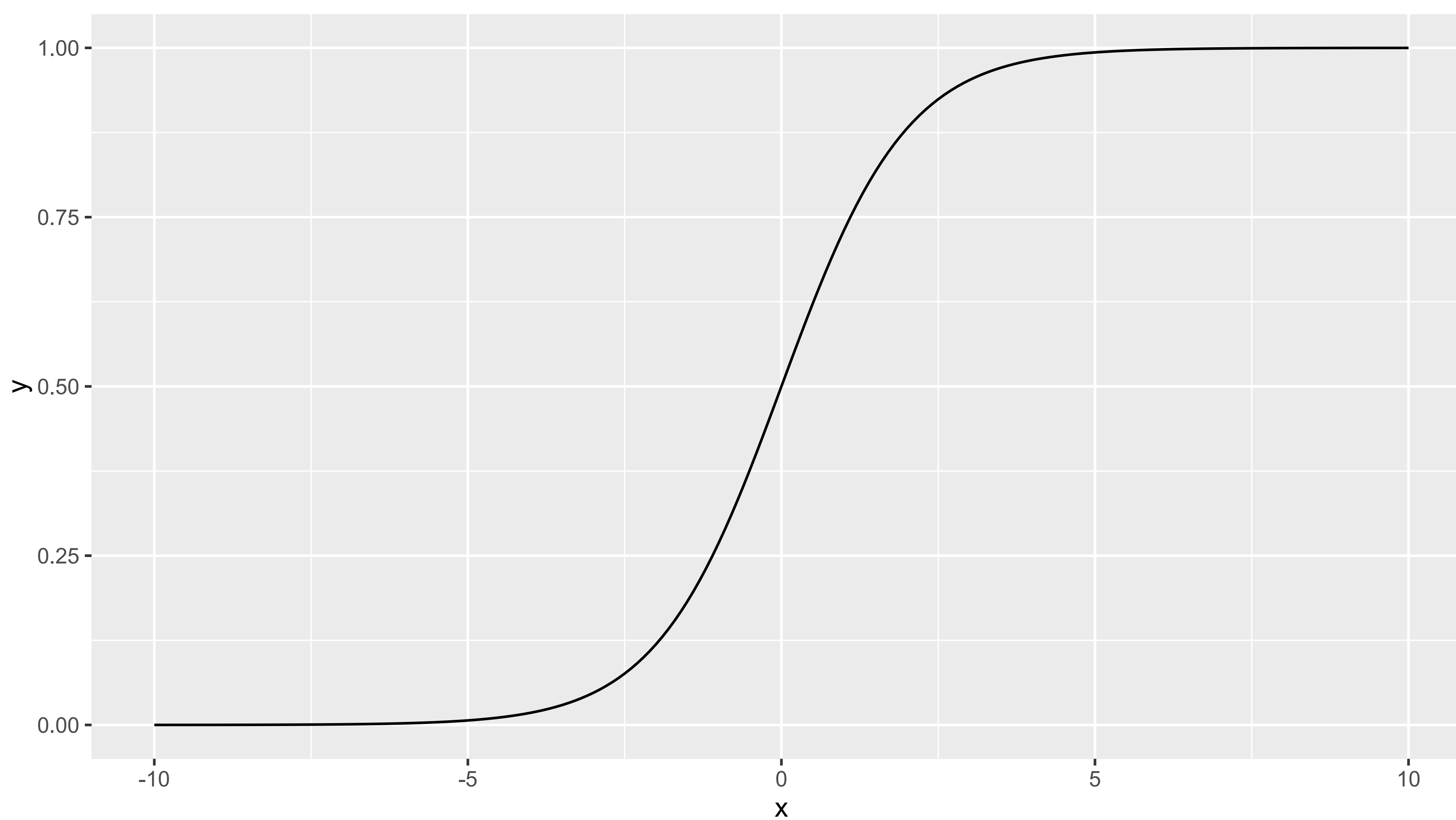
the hidden nodes are completely determined by the input and weights



$$h_1 = f\left(\sum_{i=1}^{F} x_i W_{i,1}\right)$$

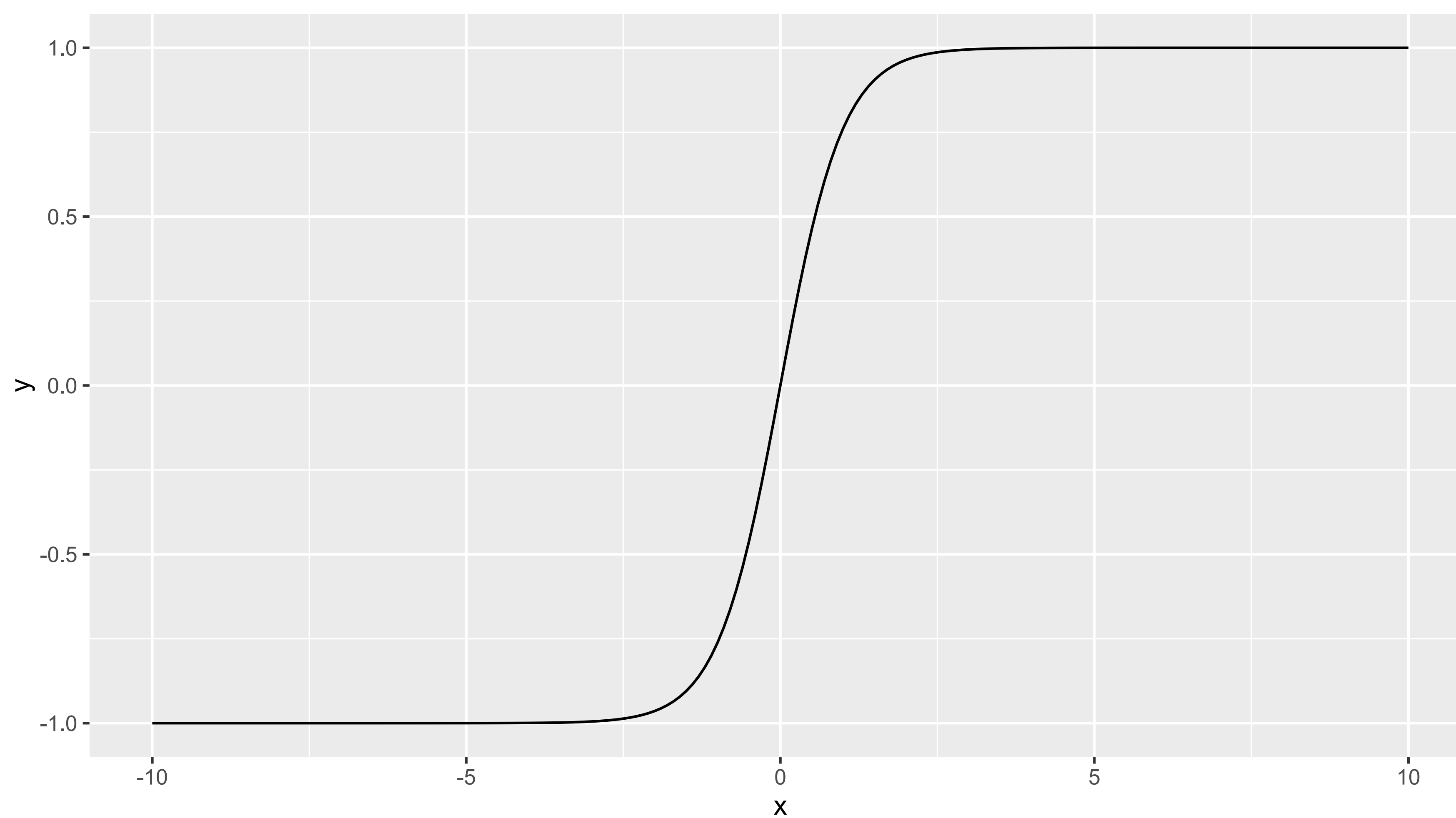
Activation functions

$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$



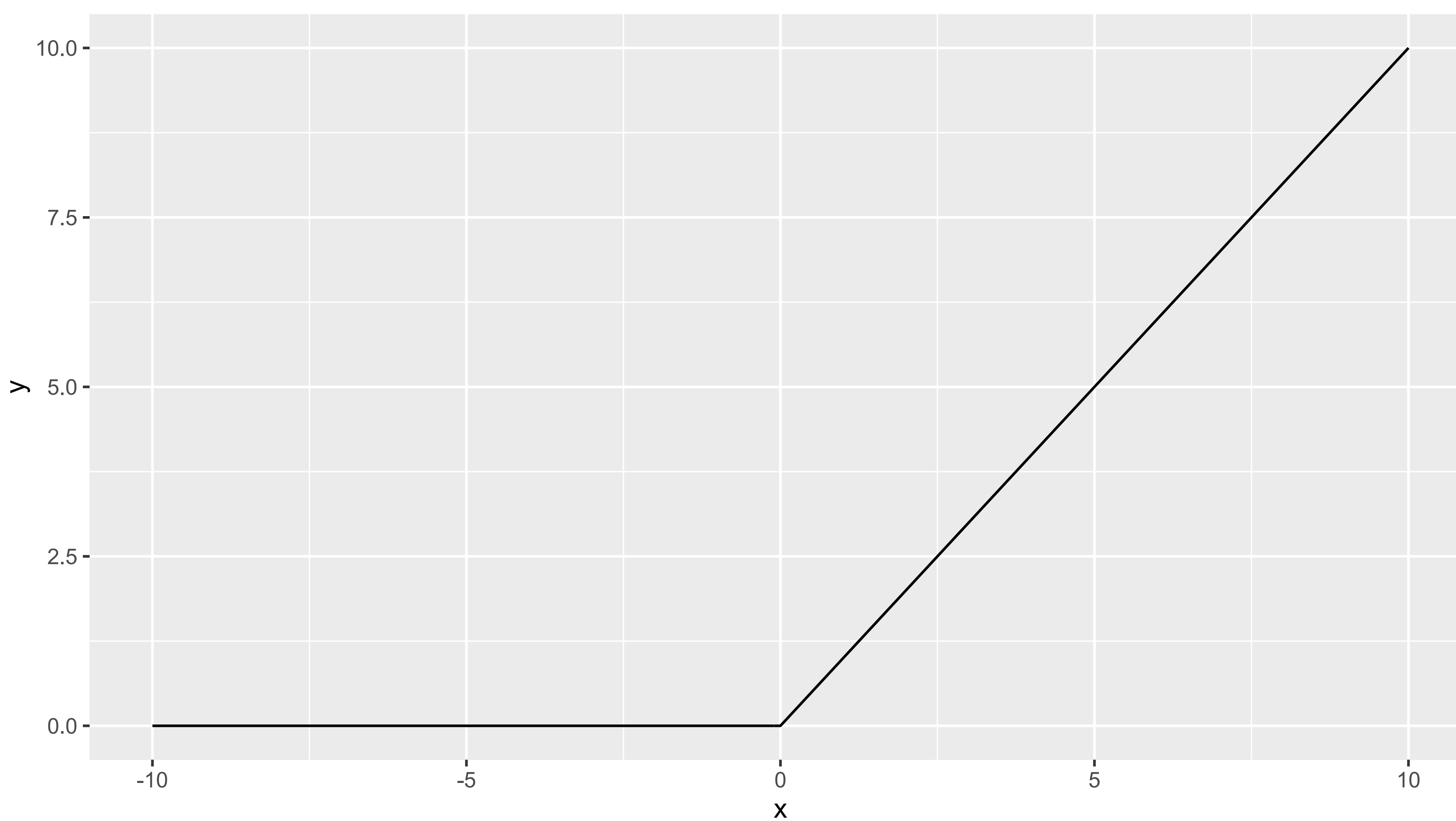
Activation functions

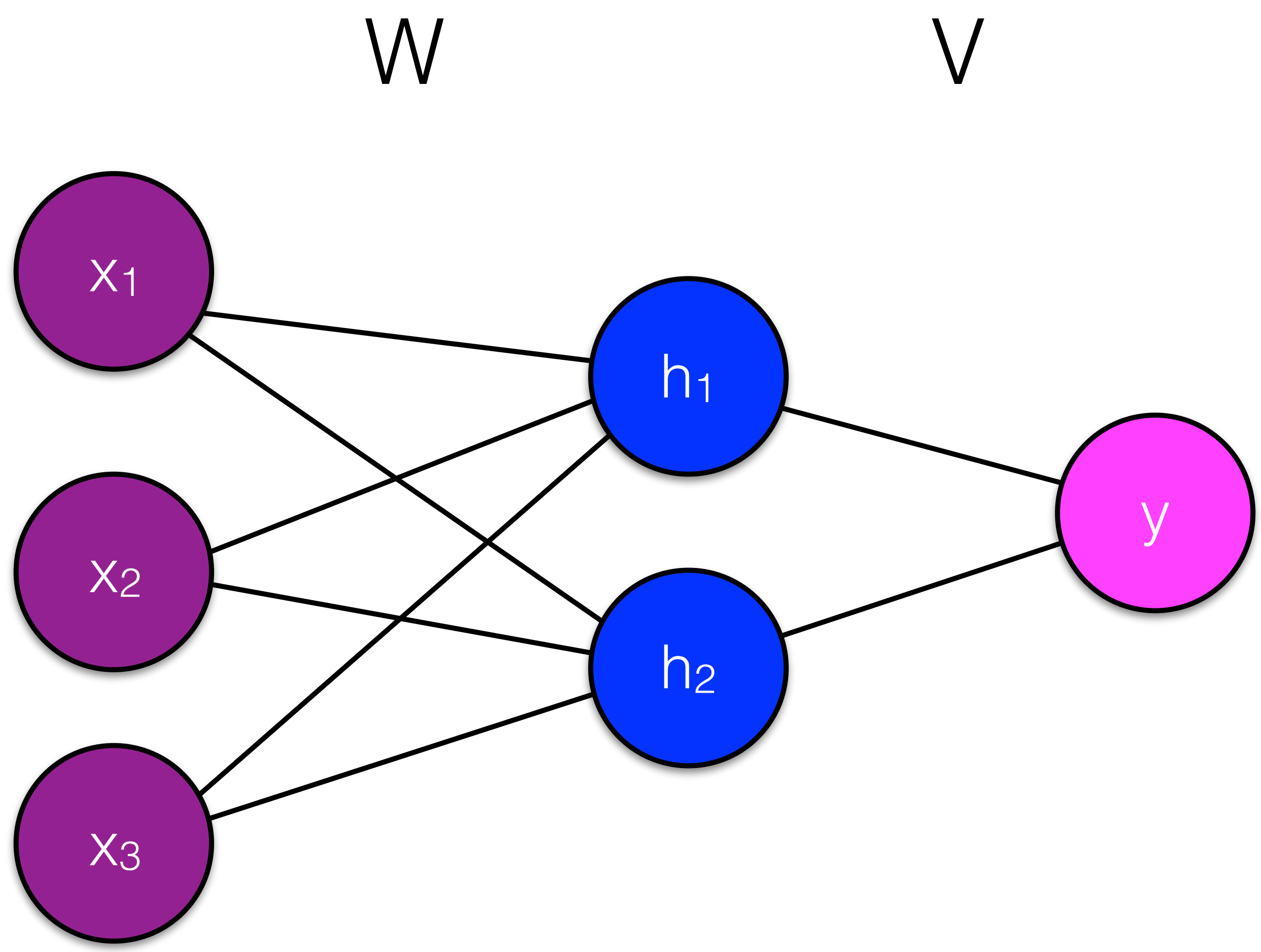
$$\tanh(z) = \frac{\exp(z) - \exp(-z)}{\exp(z) + \exp(-z)}$$



Activation functions

$$\mathit{rectifier}(z) = \max(0, z)$$

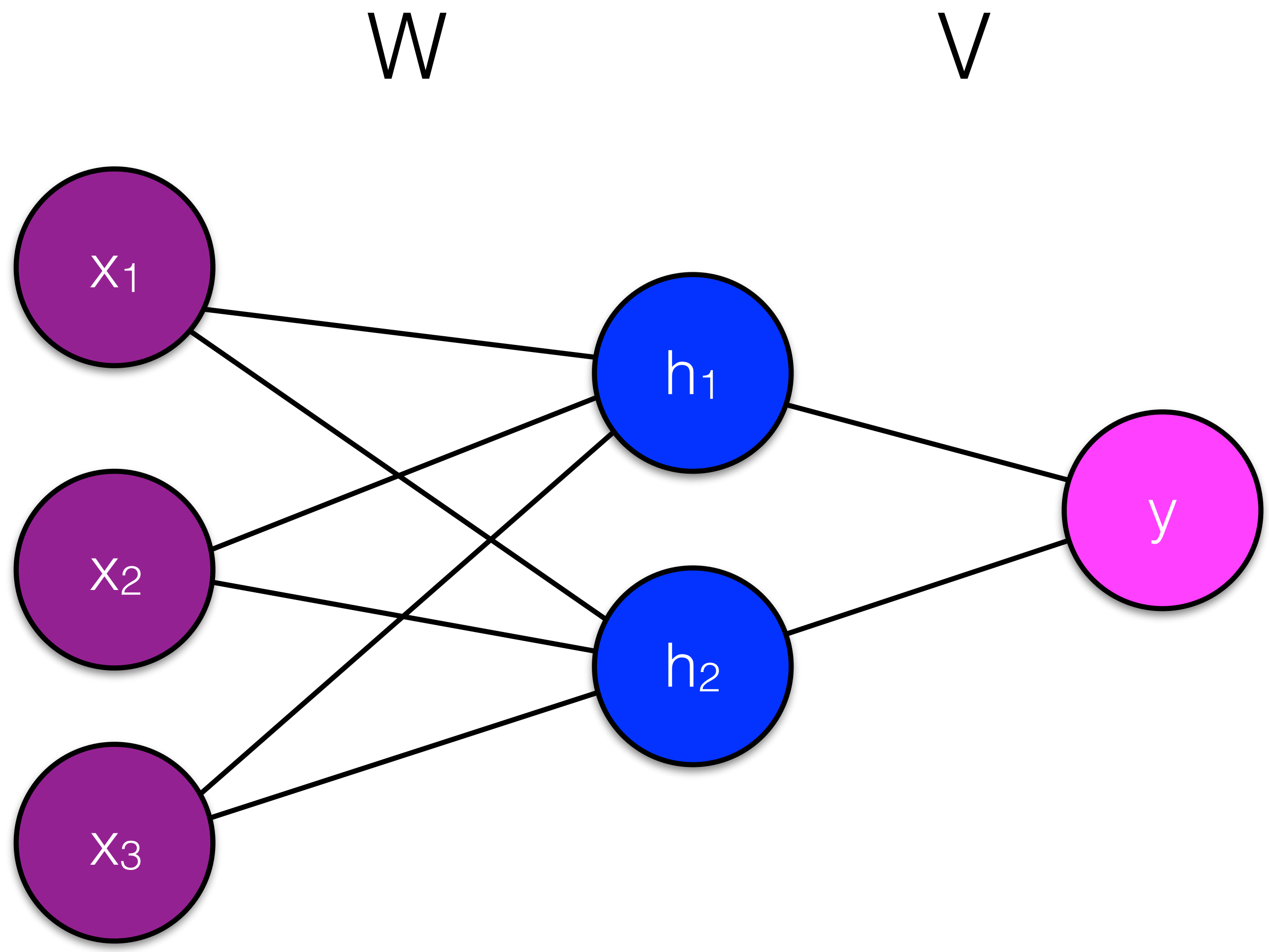




$$h_1 = \sigma\left(\sum_{i=1}^{F} x_i W_{i,1}\right)$$

$$h_2 = \sigma\left(\sum_{i=1}^{F} x_i W_{i,2}\right)$$

$$\hat{y} = V_1 h_1 + V_2 h_2$$



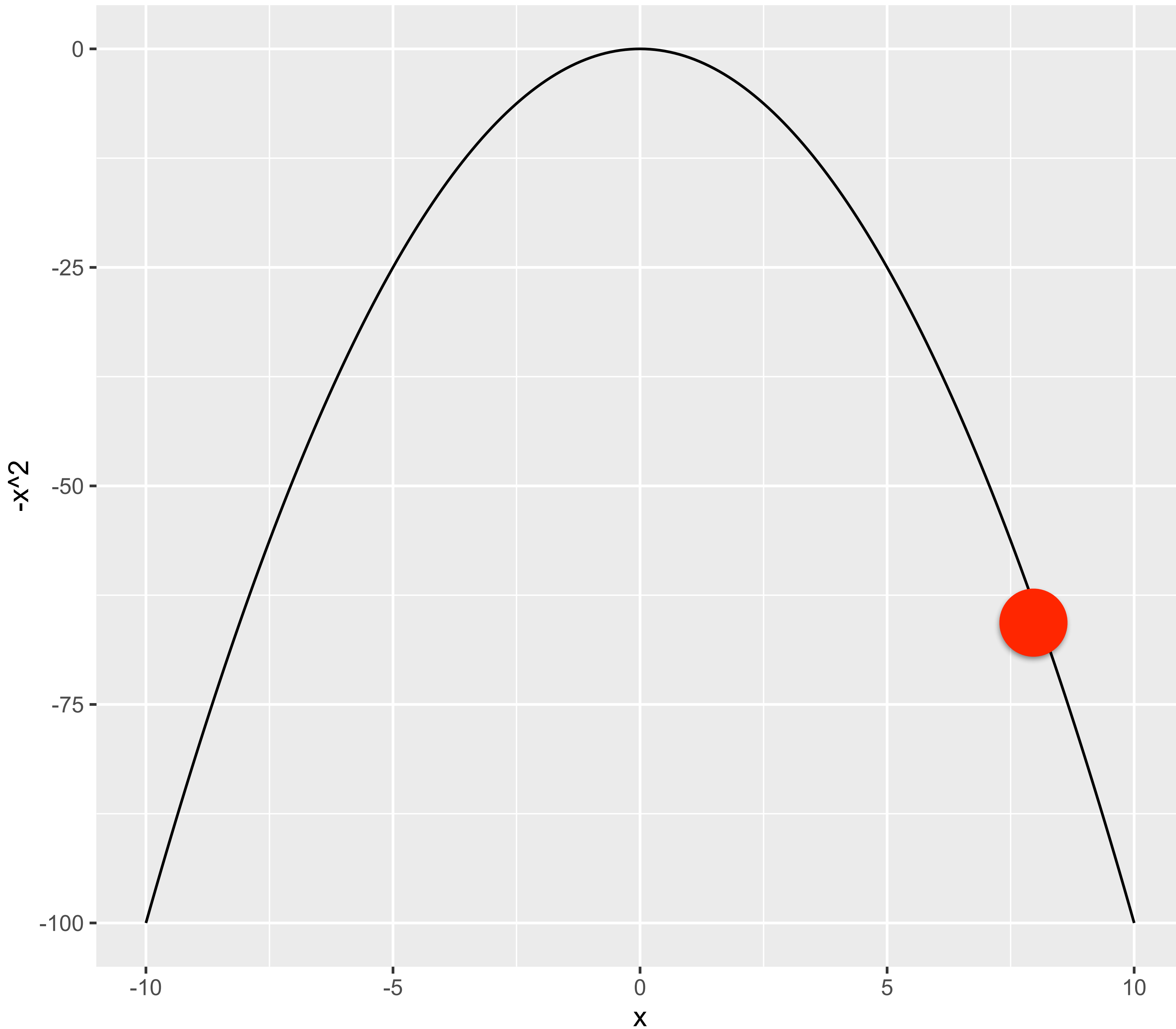
$$\hat{y} = V_1 \underbrace{\left(\sigma\left(\sum_{i=1}^{F} x_i W_{i,1}\right)\right)}_{h_1} + V_2 \underbrace{\left(\sigma\left(\sum_{i=1}^{F} x_i W_{i,2}\right)\right)}_{h_2}$$

we can express y as a function only of the input x and the weights W and V

$$\hat{y} = V_1 \underbrace{\left(\sigma\left(\sum_{i=1}^{F} x_i W_{i,1}\right)\right)}_{h_1} + V_2 \underbrace{\left(\sigma\left(\sum_{i=1}^{F} x_i W_{i,2}\right)\right)}_{h_2}$$

This is hairy, but differentiable

Backpropagation: Given training samples of <x,y> pairs, we can use stochastic gradient descent to find the values of W and V that minimize the loss.



$$x + \alpha(-2x)$$

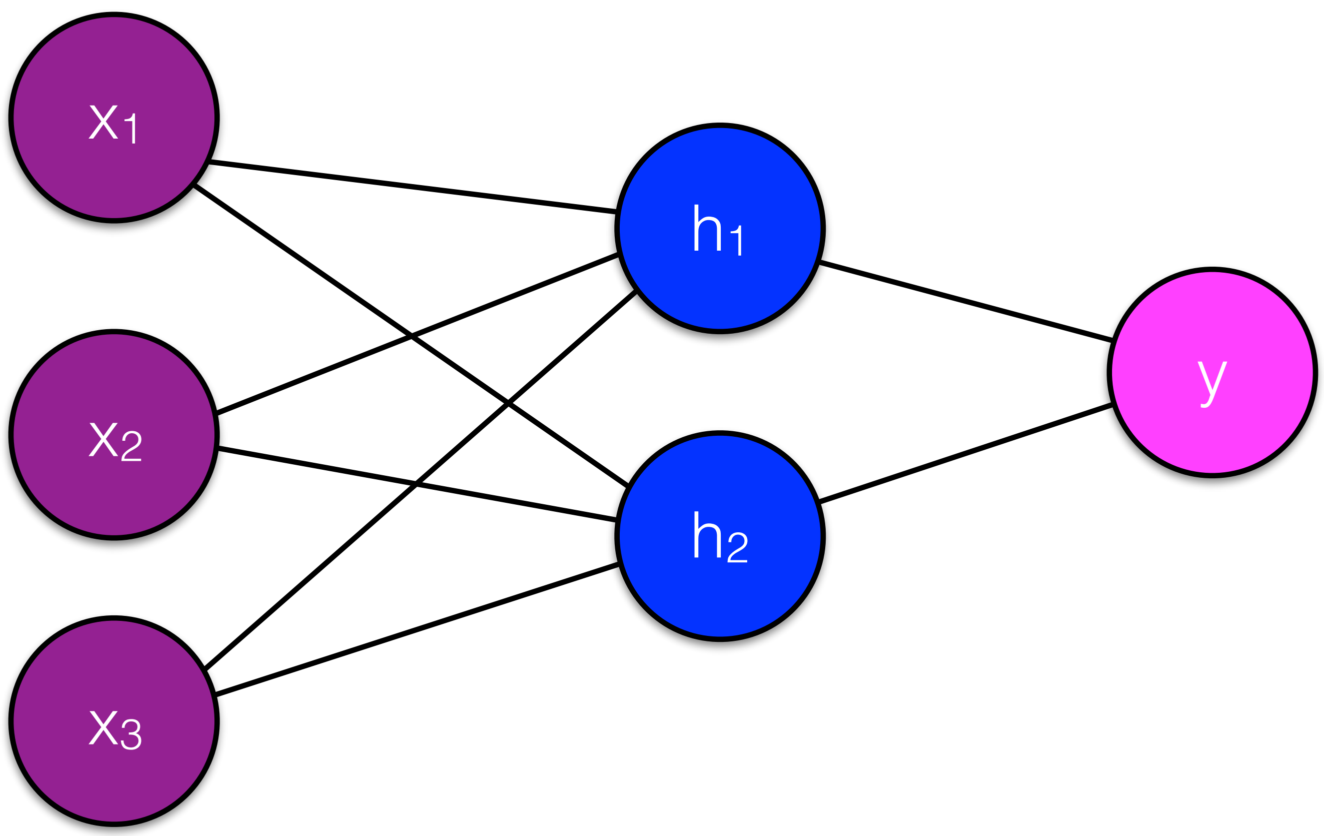
$[\alpha = 0.1]$

x	.1(-2x)
8.00	-1.60
6.40	-1.28
5.12	-1.02
4.10	-0.82
3.28	-0.66
2.62	-0.52
2.10	-0.42
1.68	-0.34
1.34	-0.27
1.07	-0.21
0.86	-0.17
0.69	-0.14

$$\frac{d}{dx} - x^2 = -2x$$

We can get to maximum value of this function by following the gradient

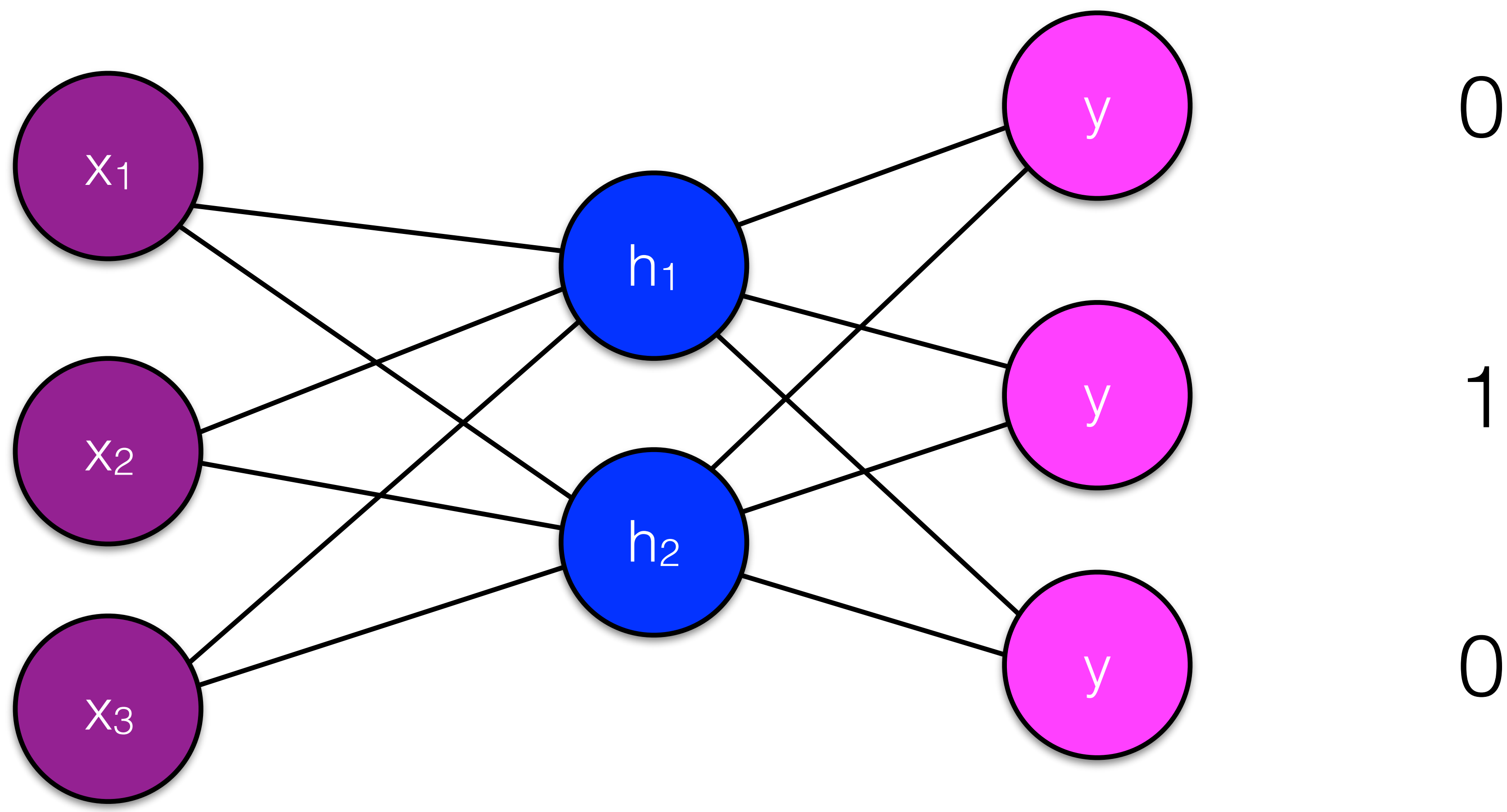
Neural network structures



1

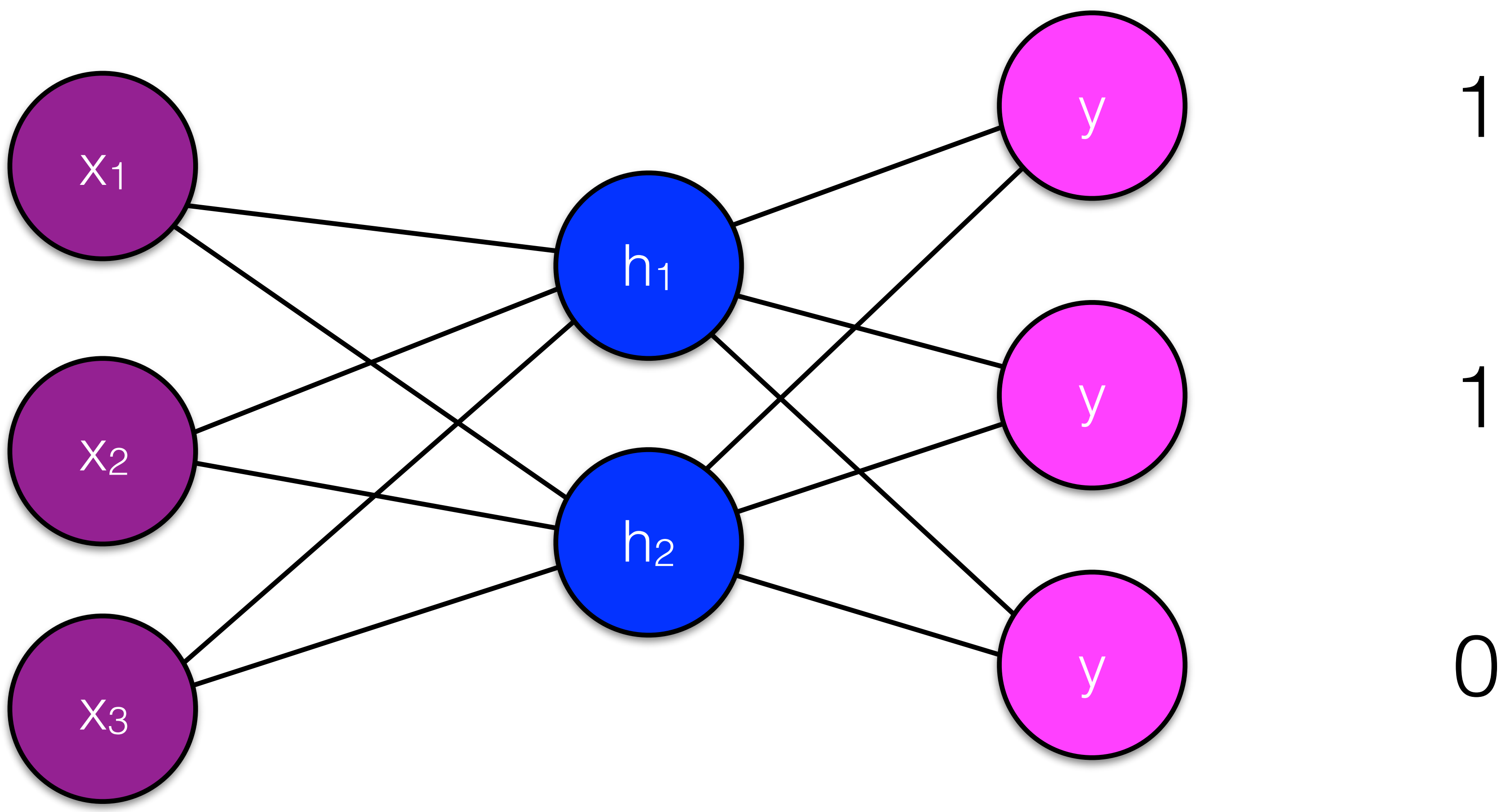
Output one real value

Neural network structures



Multiclass: output 3 values, only one = 1 in training data

Neural network structures



output 3 values, several = 1 in training data

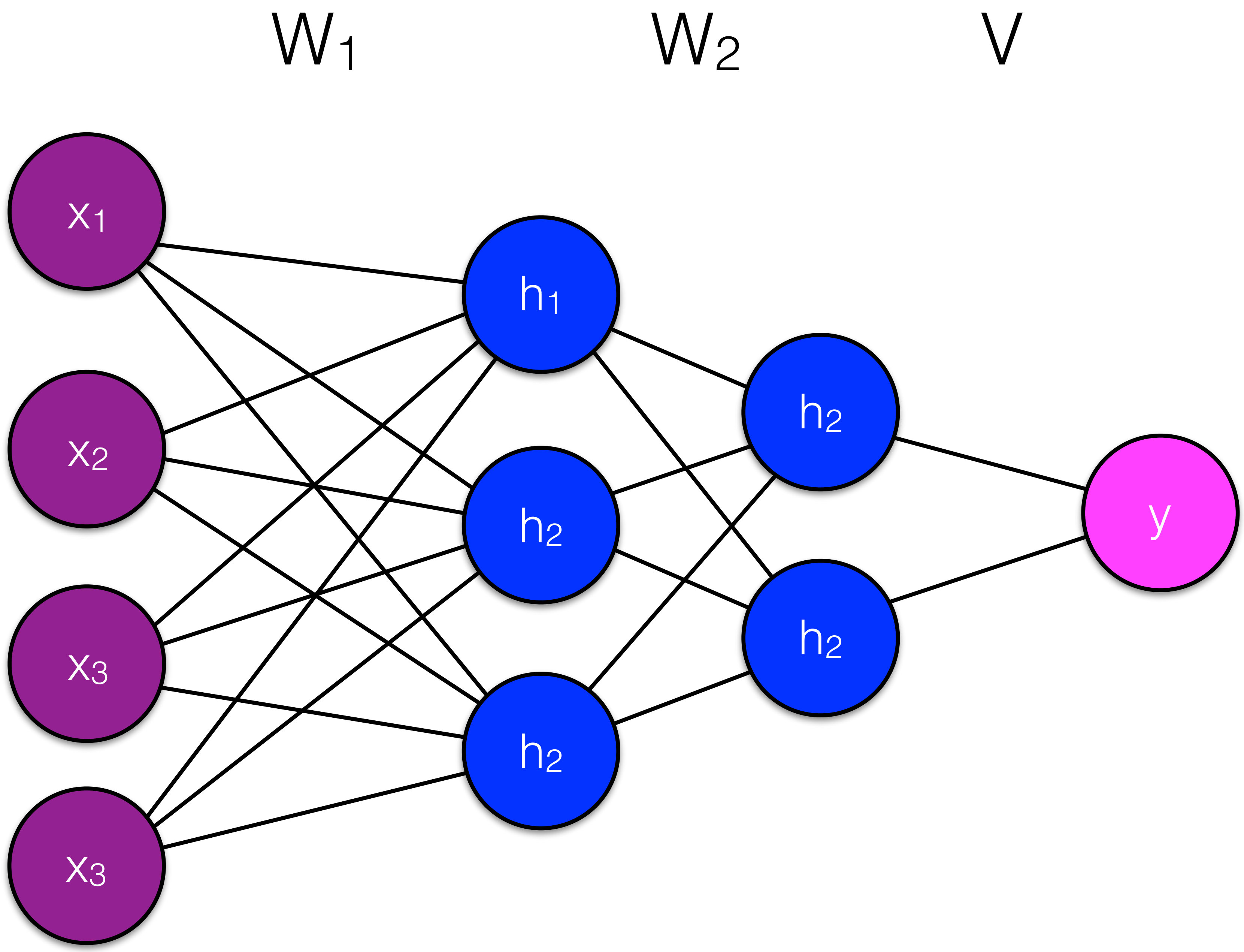
Regularization

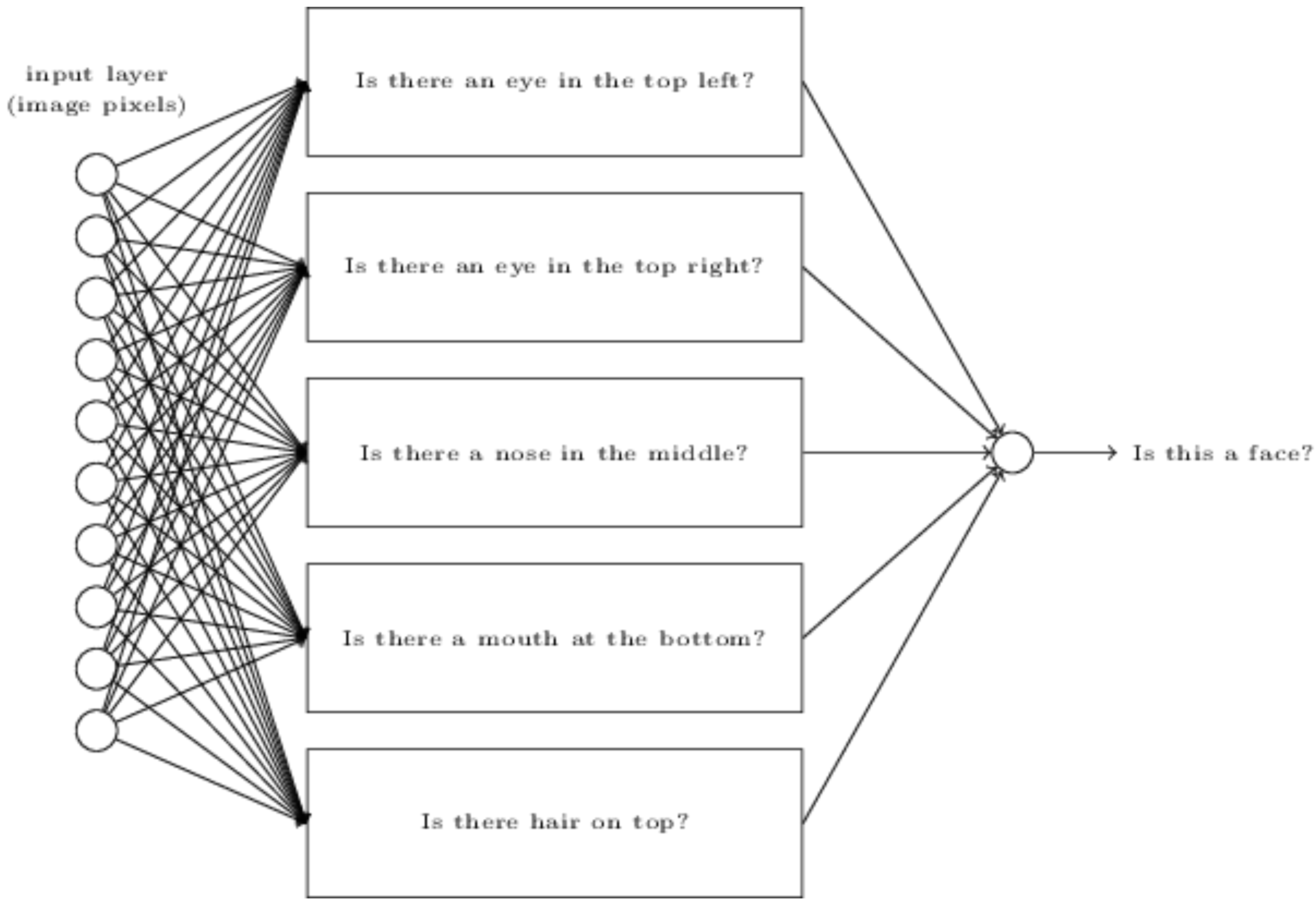
- Increasing the number of parameters = increasing the possibility for overfitting to training data

Regularization

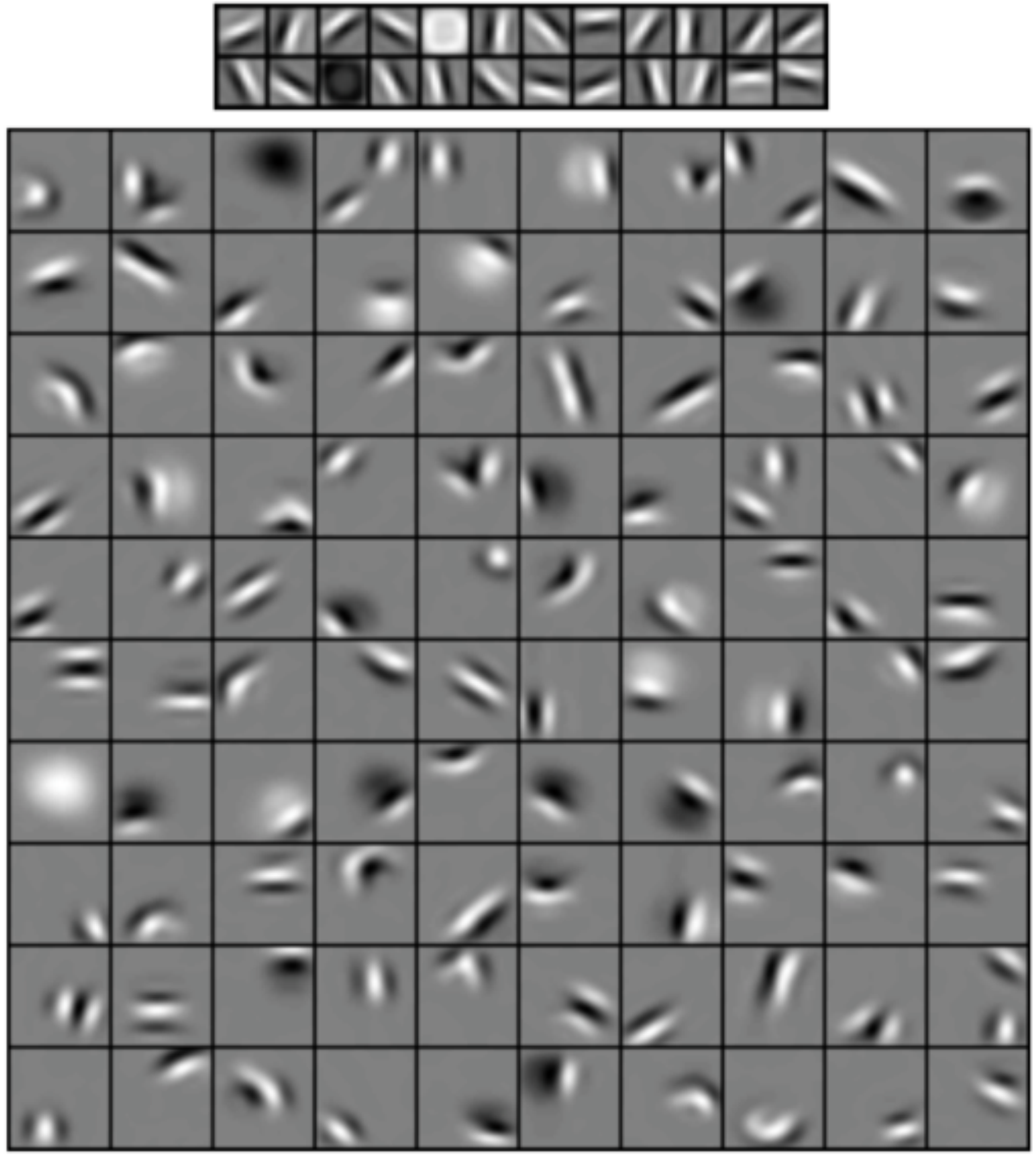
- L2 regularization: penalize W and V for being too large
- Dropout: when training on a <x,y> pair, randomly remove some node and weights.
- Early stopping: Stop backpropagation before the training error is too small.

Deeper networks





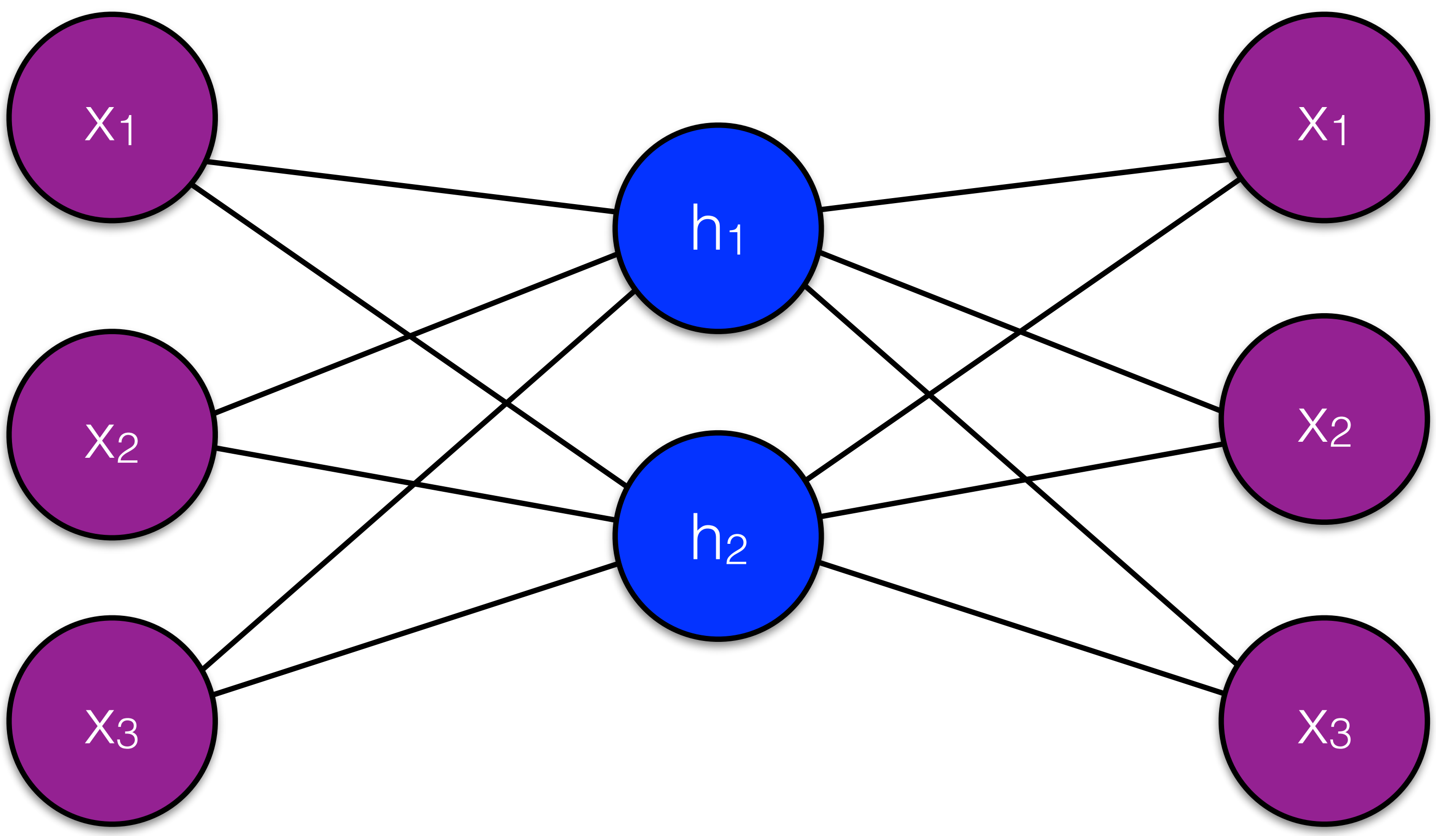
http://neuralnetworksanddeeplearning.com/chap1.html



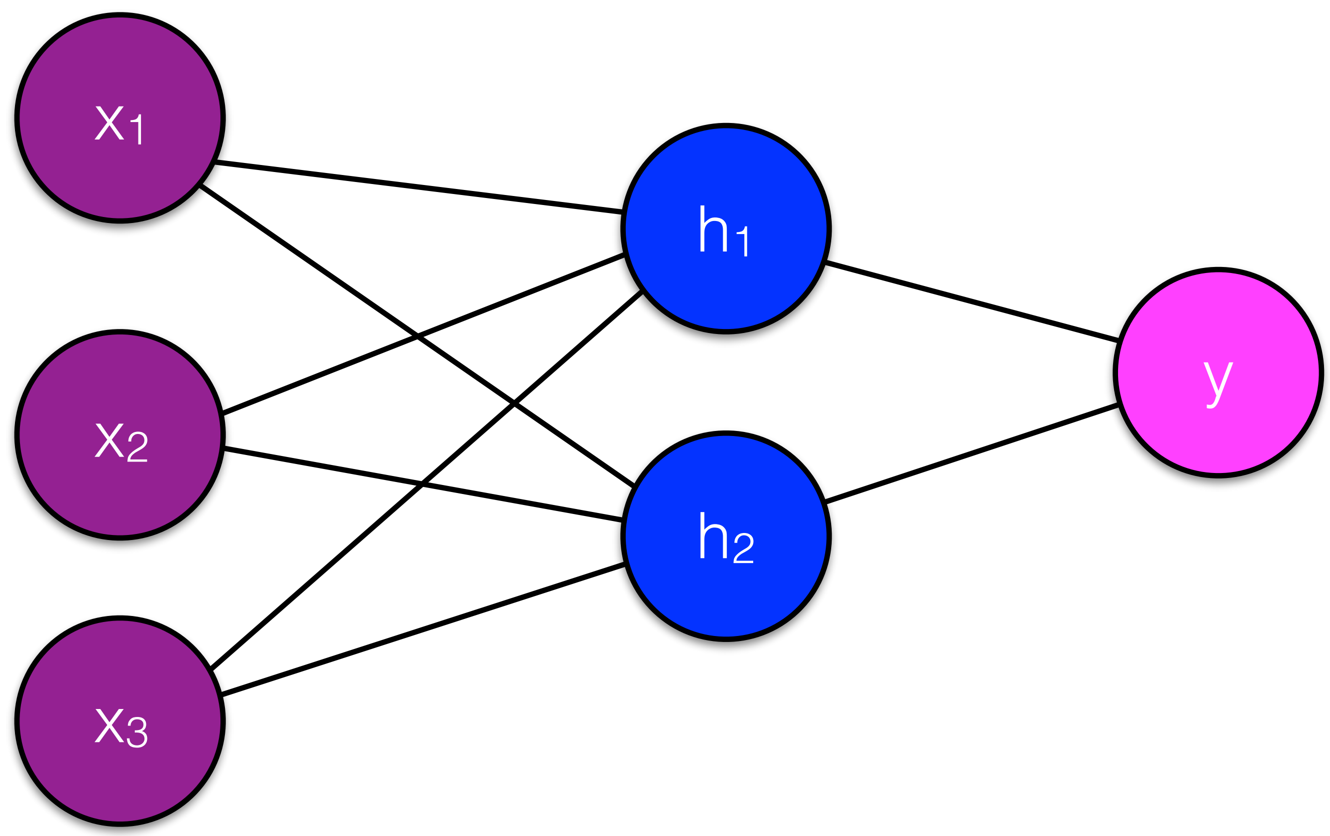
Higher order features learned for image recognition
Lee et al. 2009 (ICML)

Autoencoder

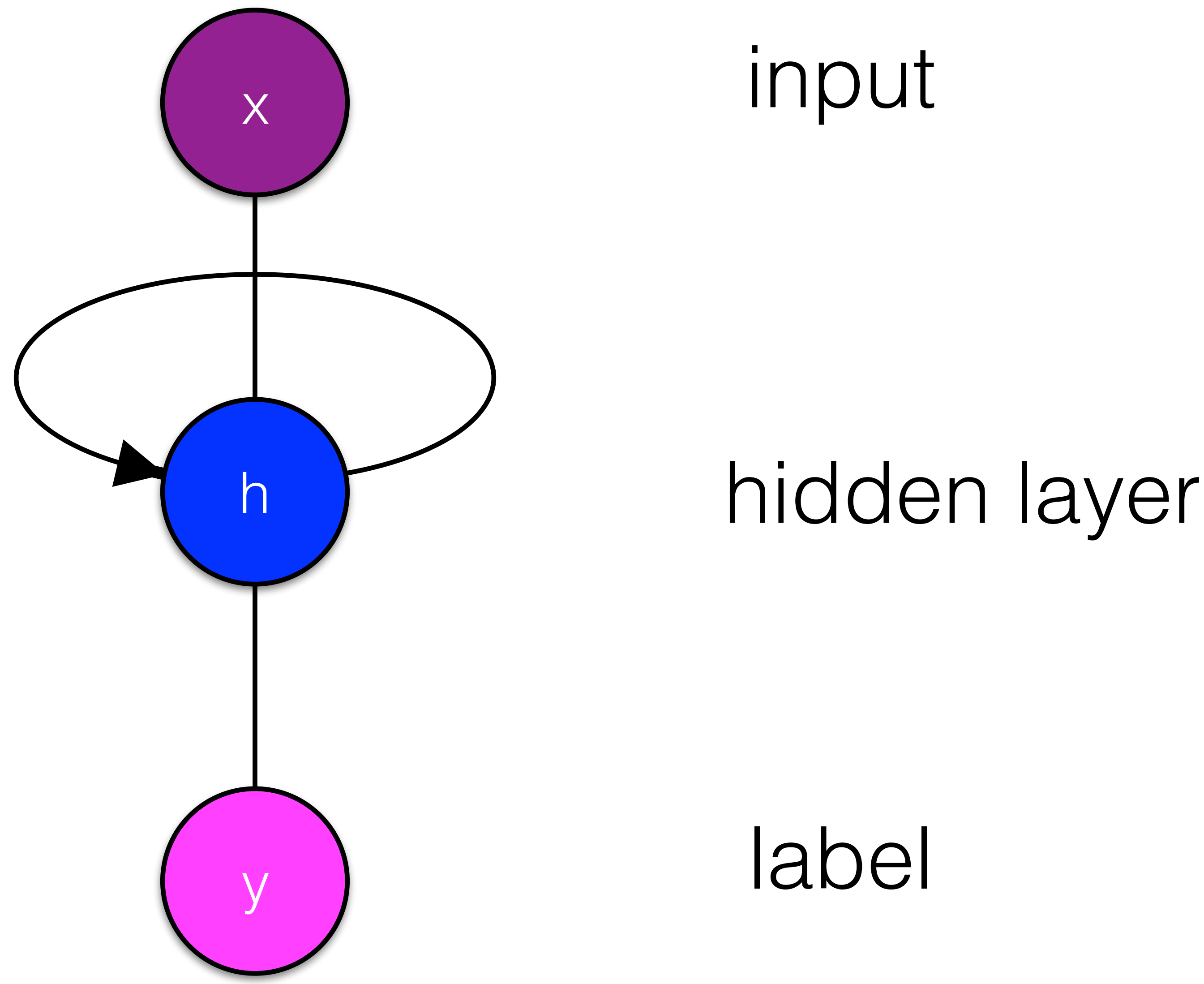
- Unsupervised neural network, where $y = x$
- Learns a low-dimensional representation of x



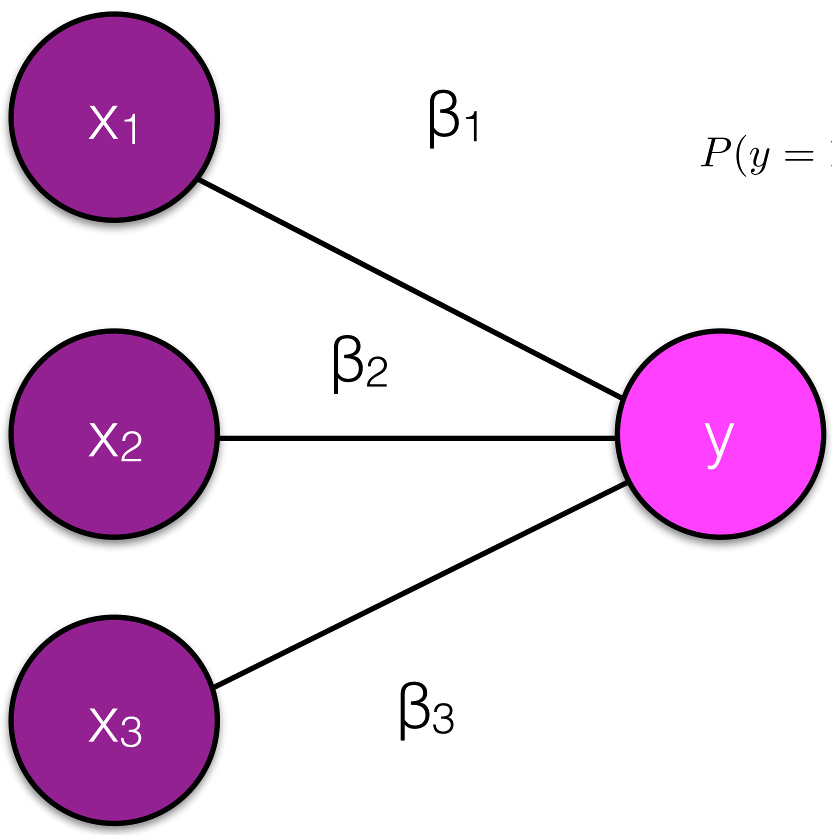
Feedforward networks



Recurrent networks



Interpretability

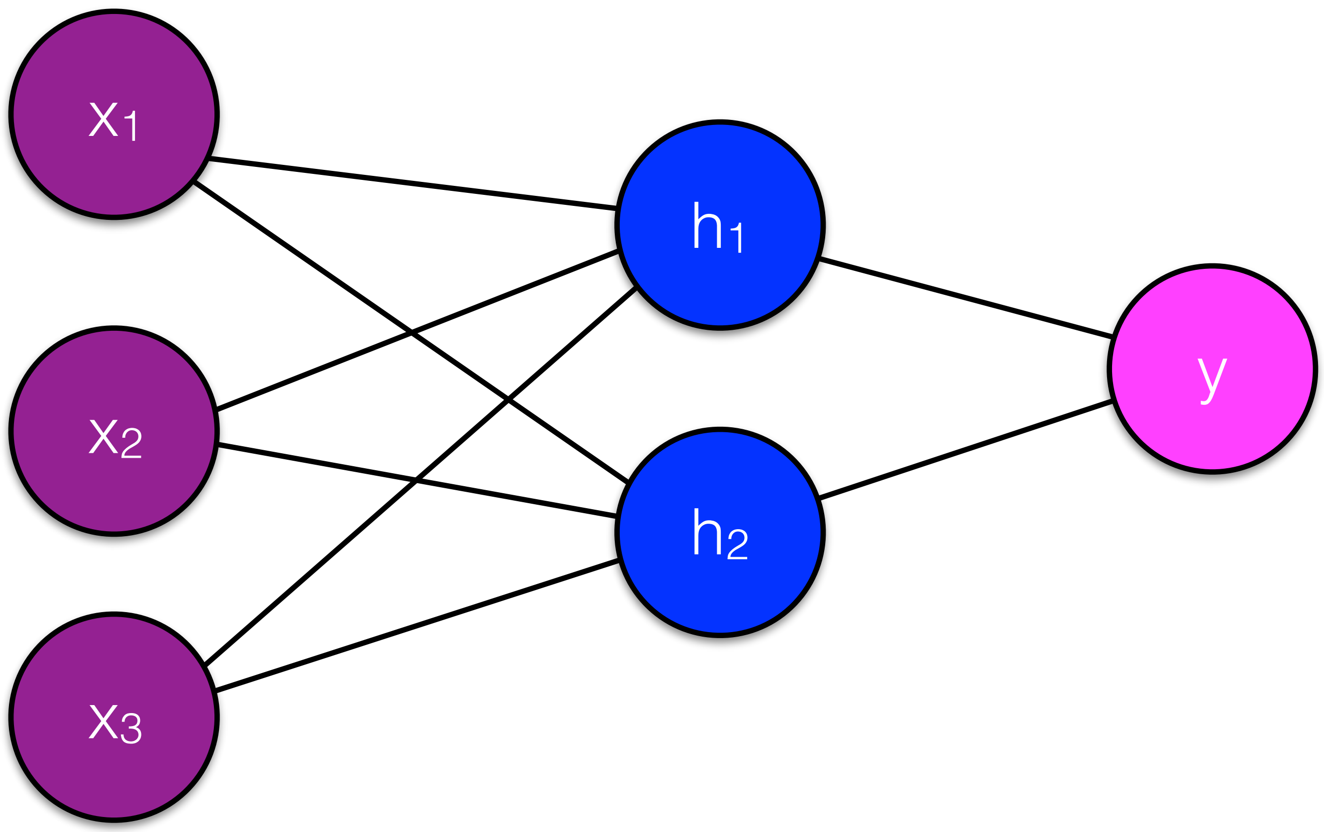


$$P(y = 1 \mid x, \beta) = \frac{\exp(x^\top \beta)}{1 + \exp(x^\top \beta)}$$

	x	β
not	1	-0.5
bad	1	-1.7
movie	0	0.3

With a single-layer linear model (logistic/linear regression, perceptron) there's an immediate relationship between x and y apparent in β

Interpretability



Non-linear activation functions induce dependencies between the inputs.