Deconstructing Data Science

David Bamman, UC Berkeley

Info 290
Lecture 16: Neural networks

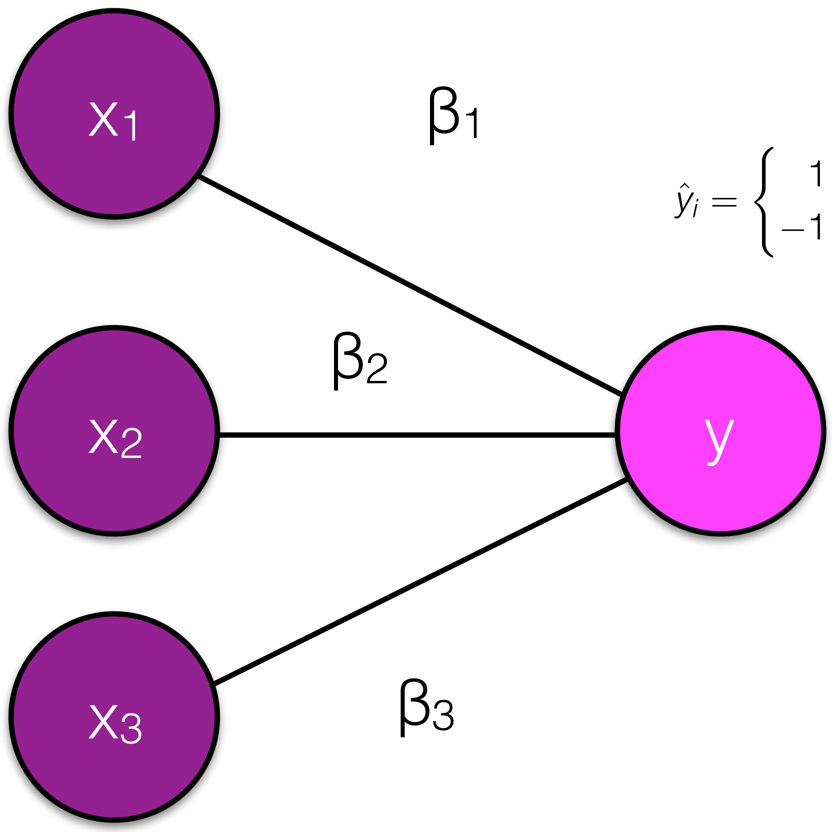
Mar 16, 2016

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	0.4
movie	0	1.7

The perceptron, again

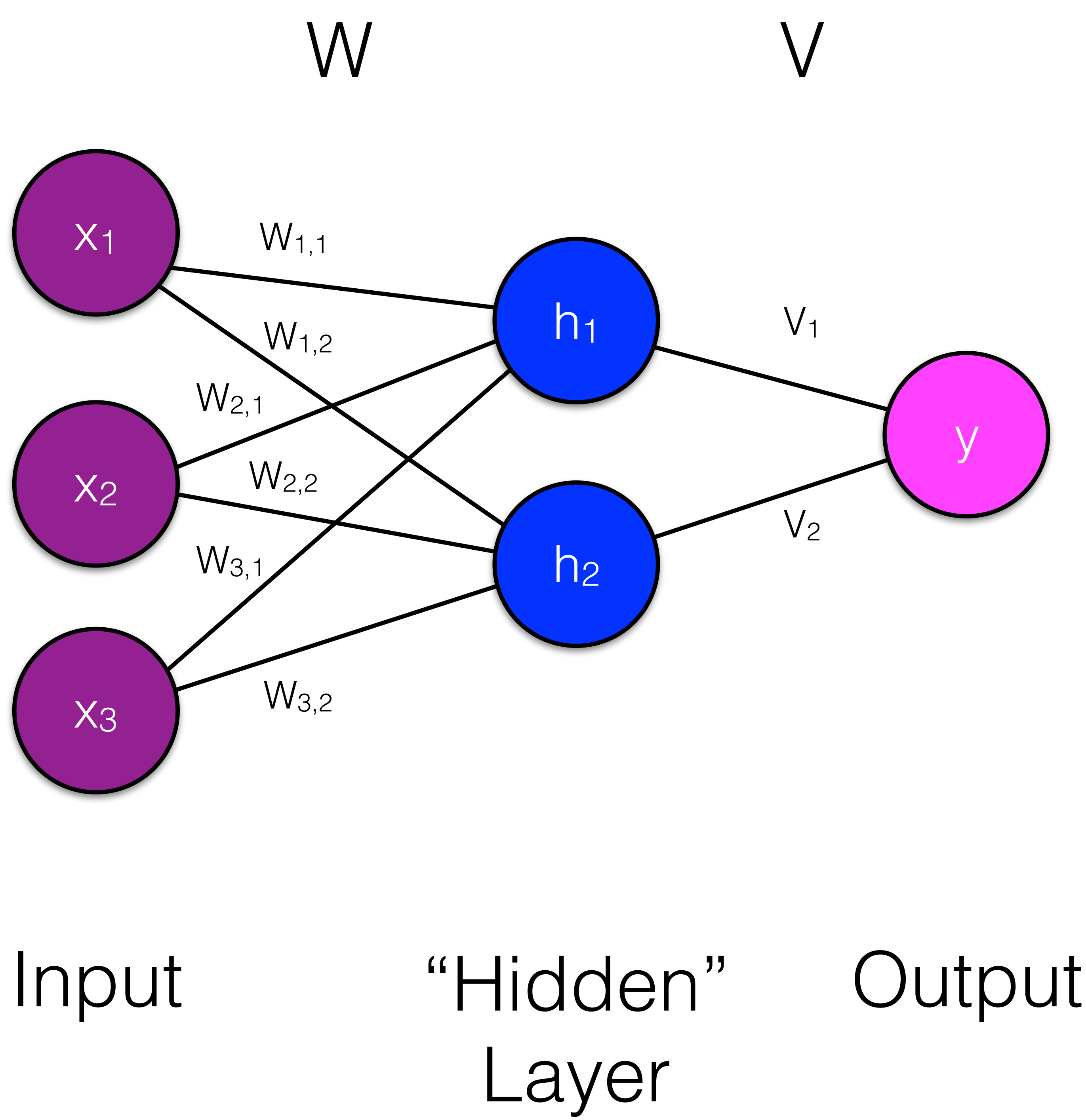


$$\hat{y}_i = \begin{cases} 1 & \text{if } \sum_i^F x_i \beta_i \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

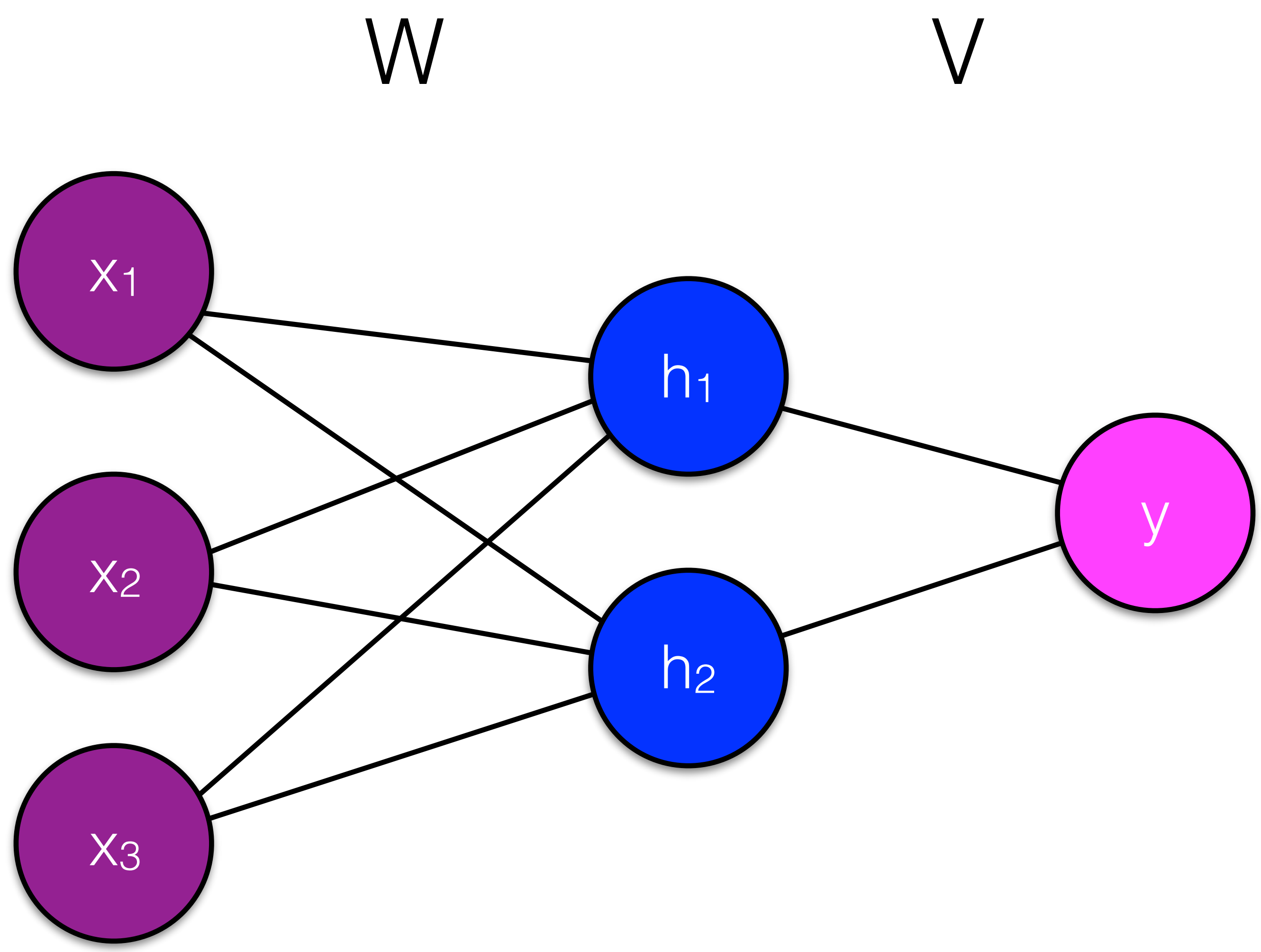
	x	β
not	1	-0.5
bad	1	0.4
movie	0	1.7

Neural networks

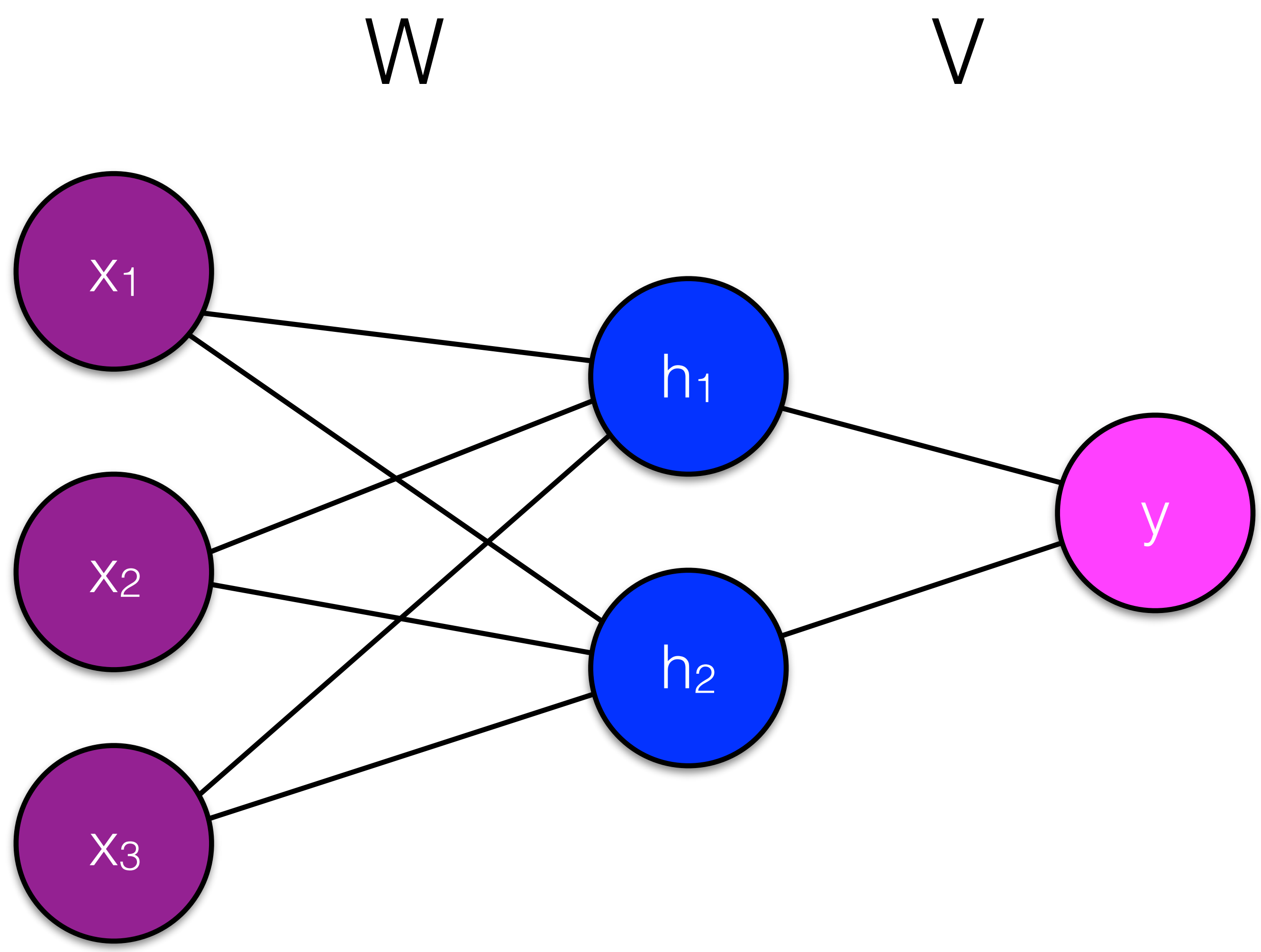
- Two core ideas:
    - Non-linear activation functions
    - Multiple layers



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

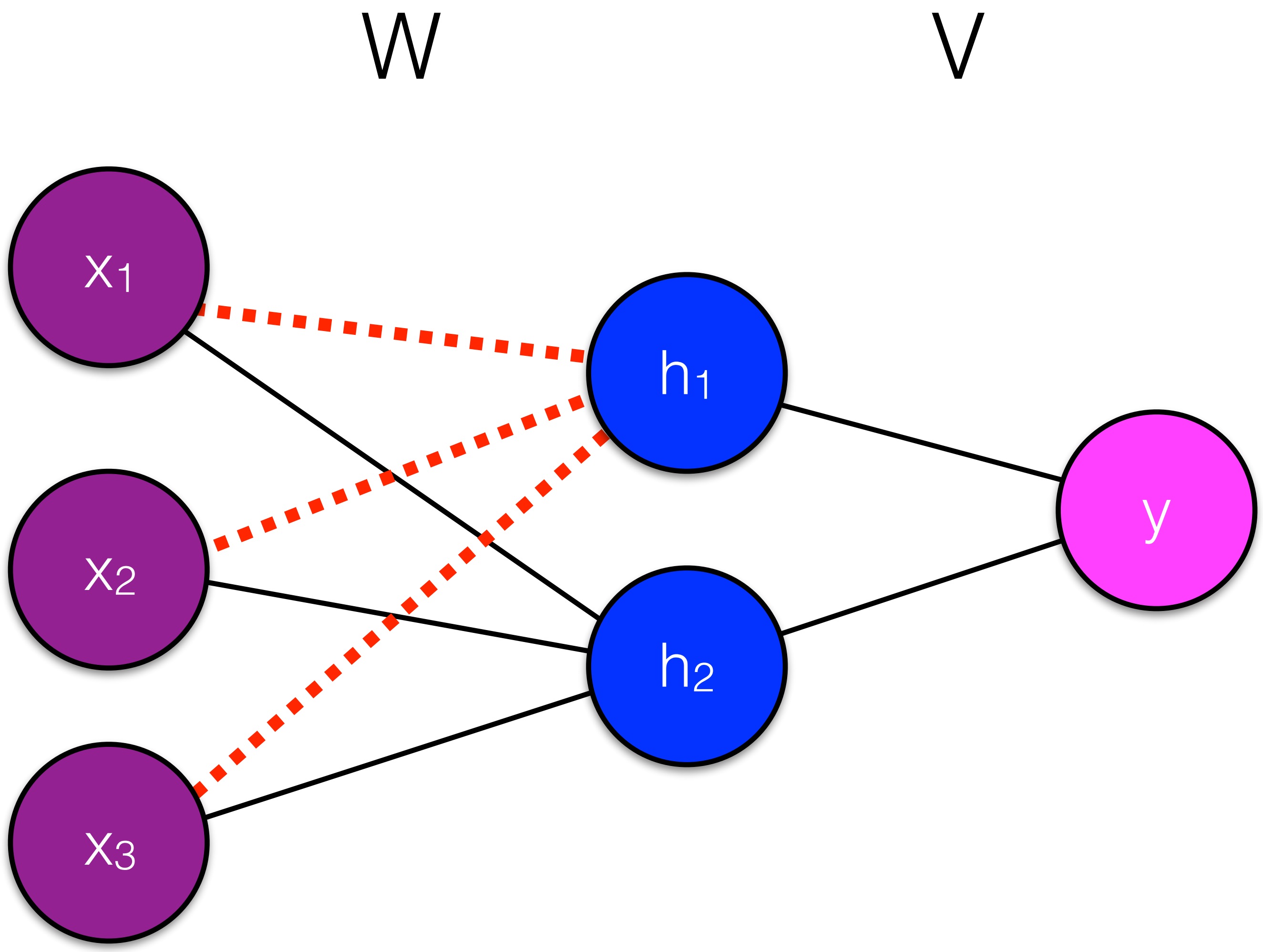


	x	W		V	y
not	1	-0.5	1.3	4.1	-1
bad	1	0.4	0.08	-0.9	
movie	0	1.7	3.1		



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

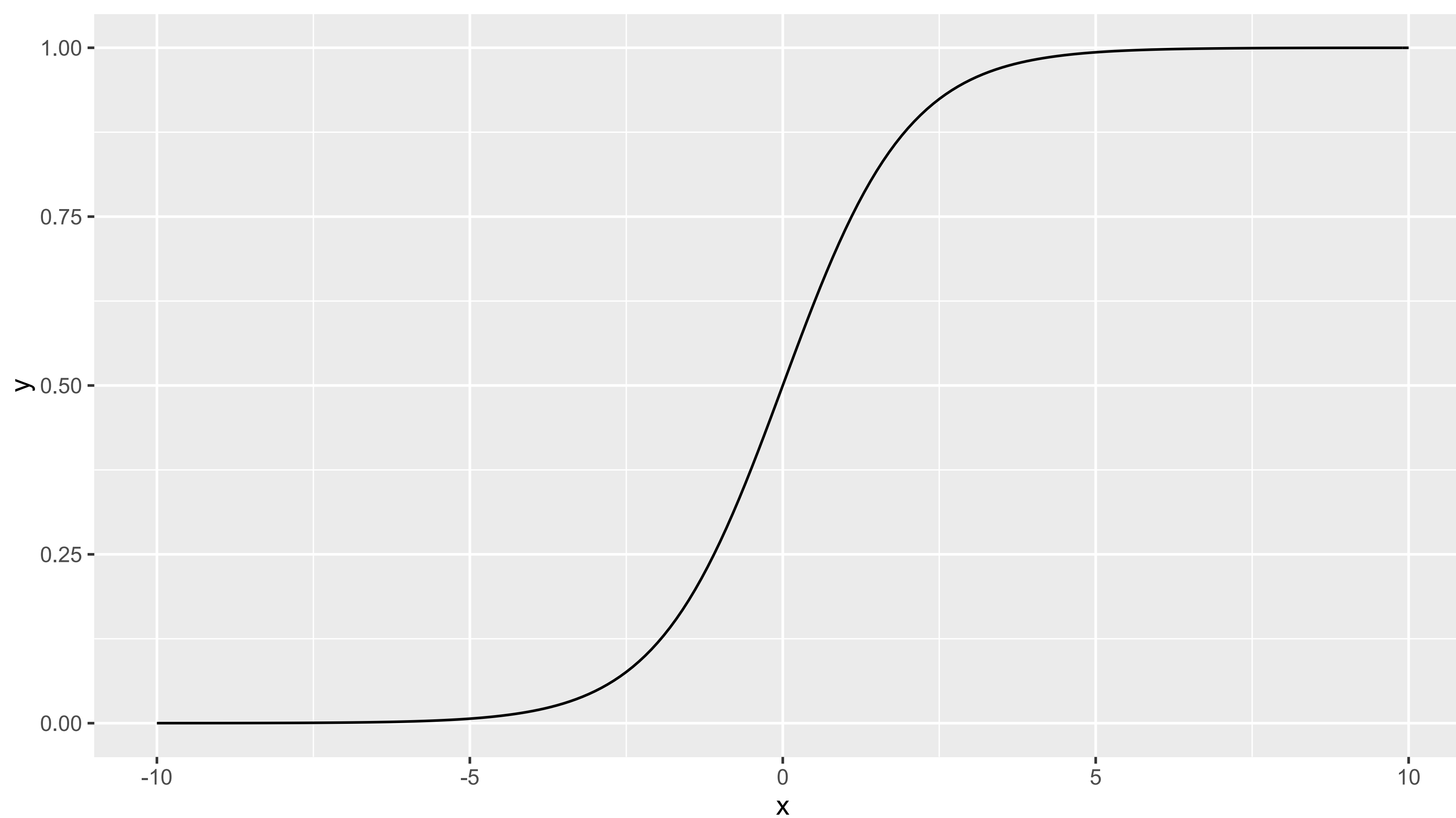
the hidden nodes are completed determined by the input and weights



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

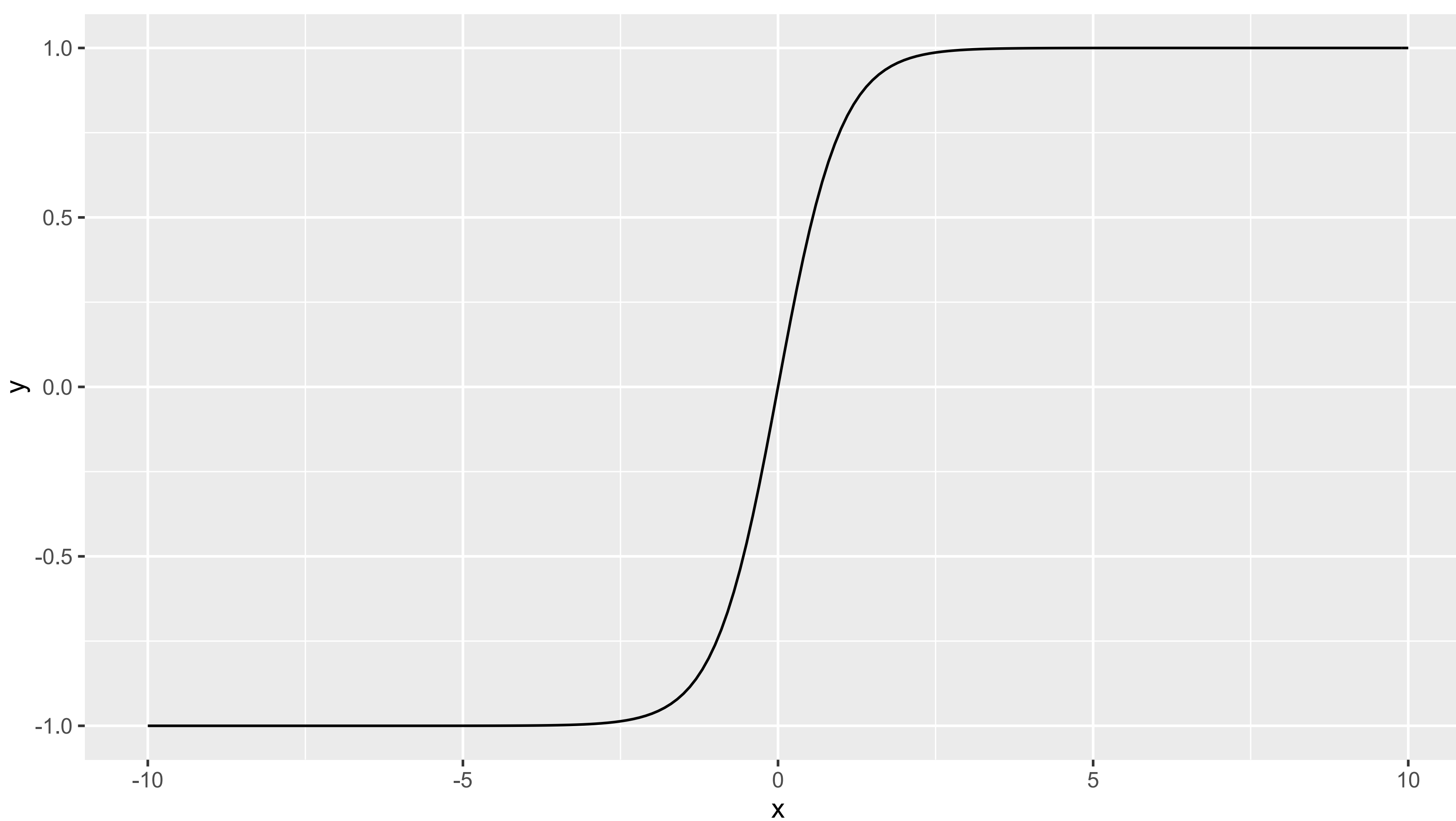
Activation functions

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



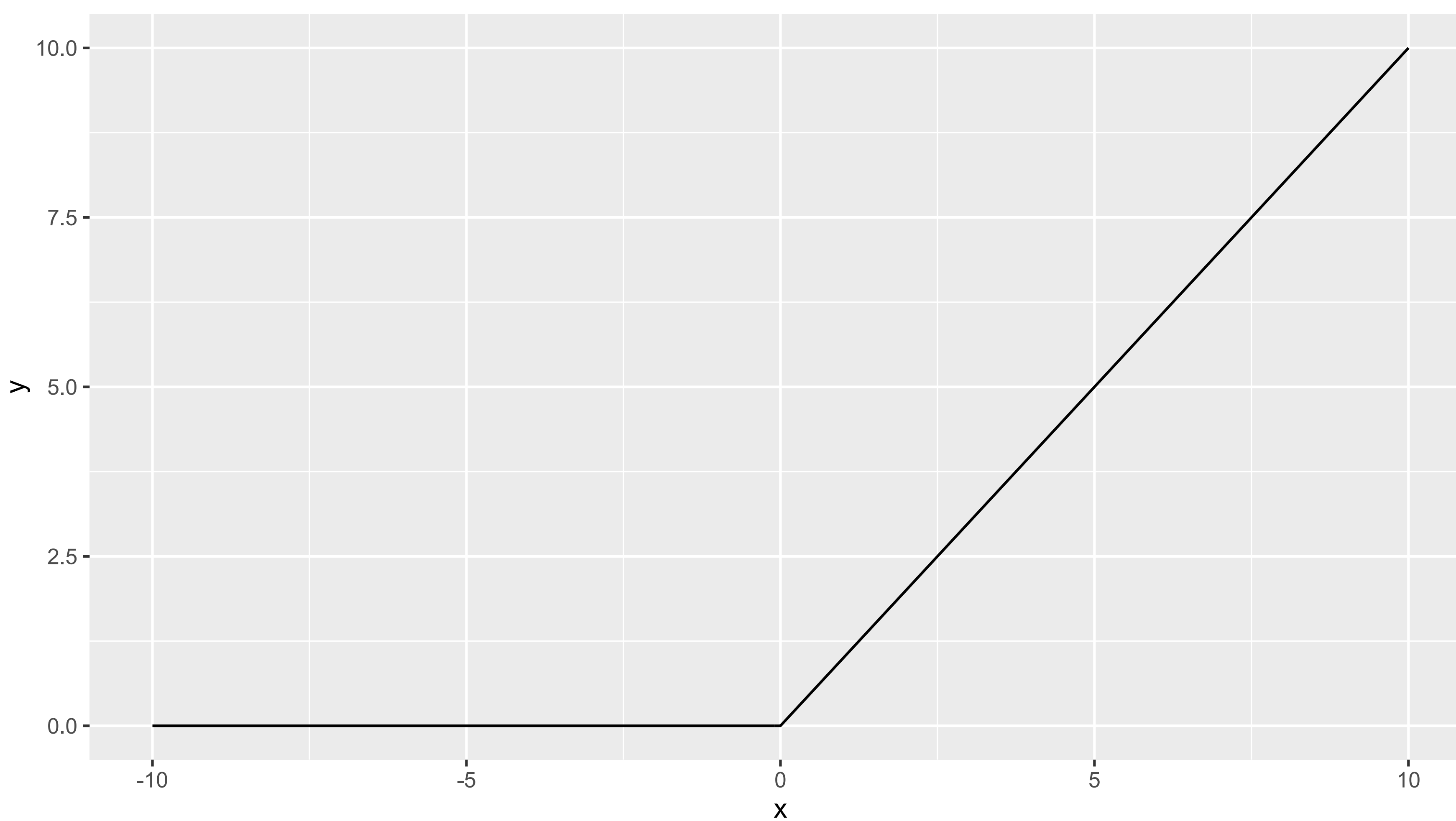
Activation functions

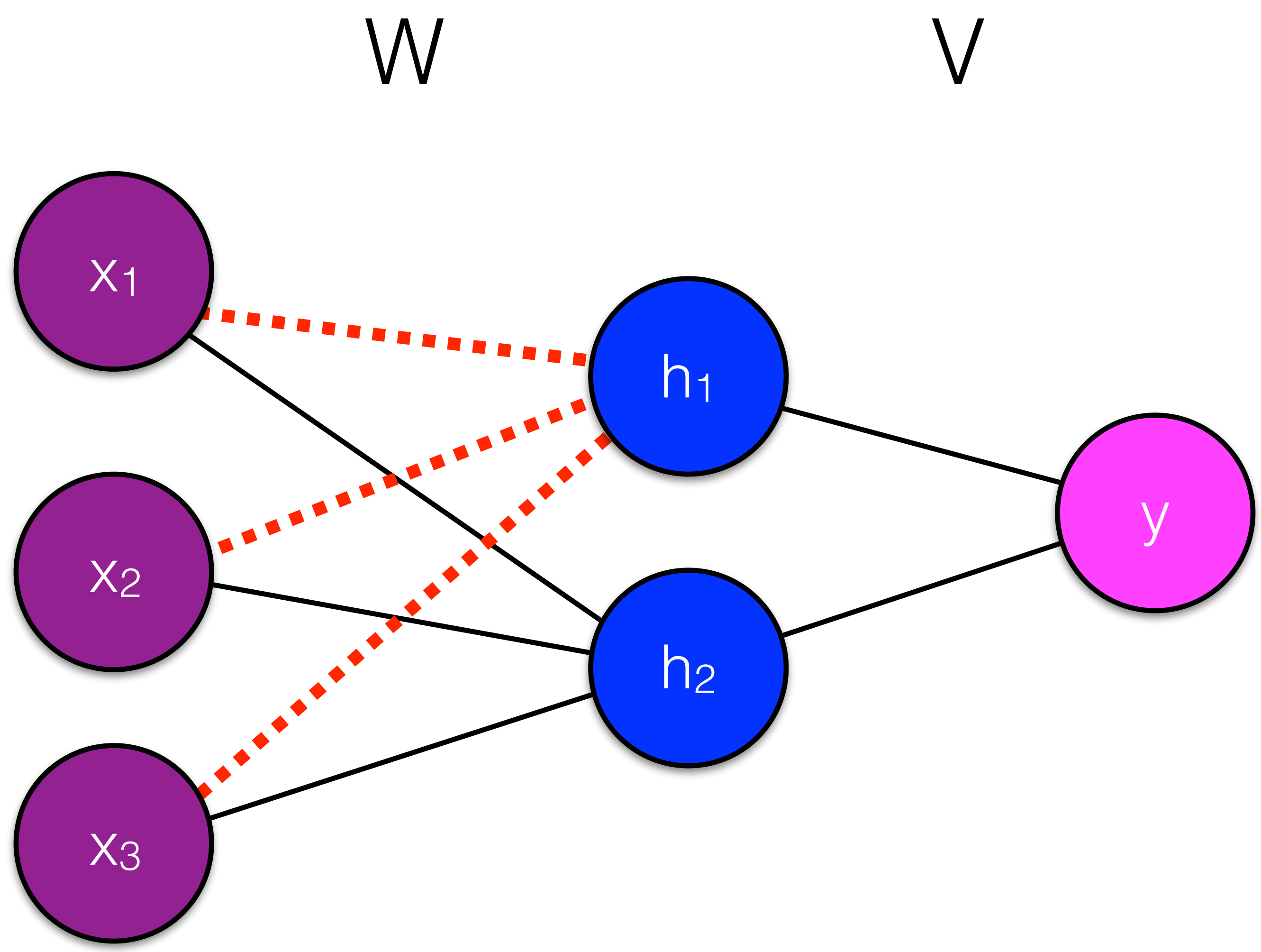
$$\tanh(z) = \frac{\exp(z) - \exp(-z)}{\exp(z) + \exp(-z)}$$



Activation functions

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

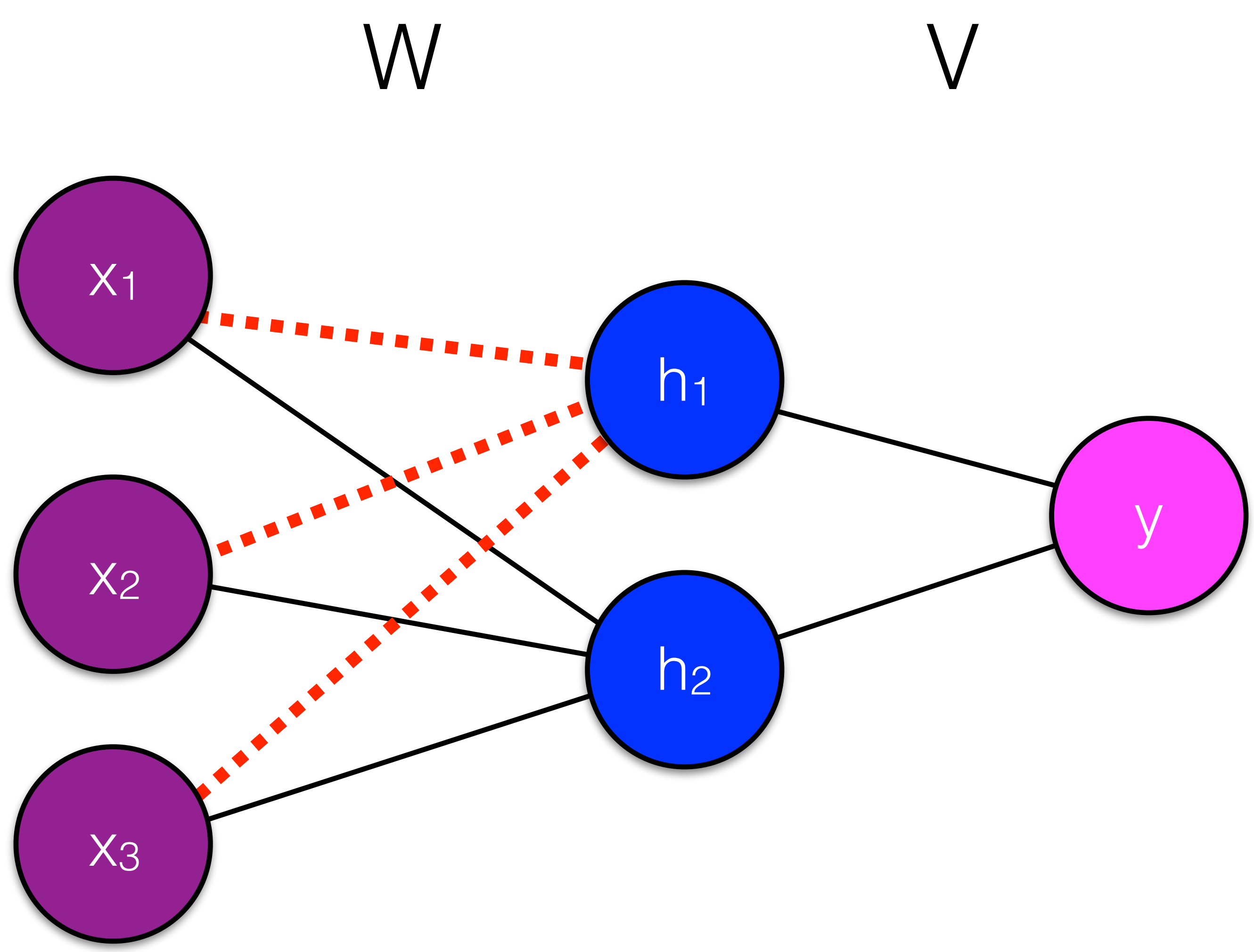




$$h_1 = \sigma\left(\sum_{i=1}^{F} x_i W_{i,1}\right)$$

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

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



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

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

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

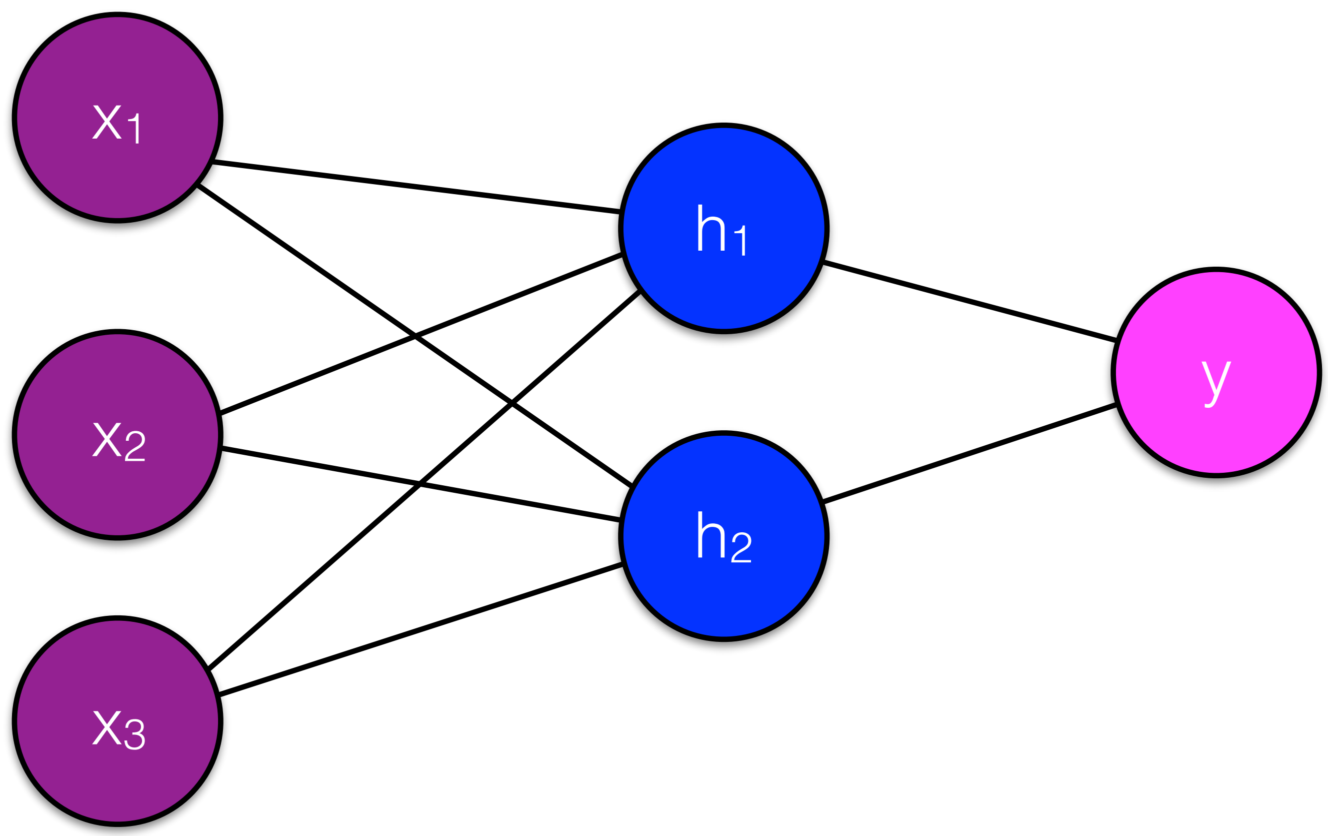
This is hairy, but differentiable

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

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.

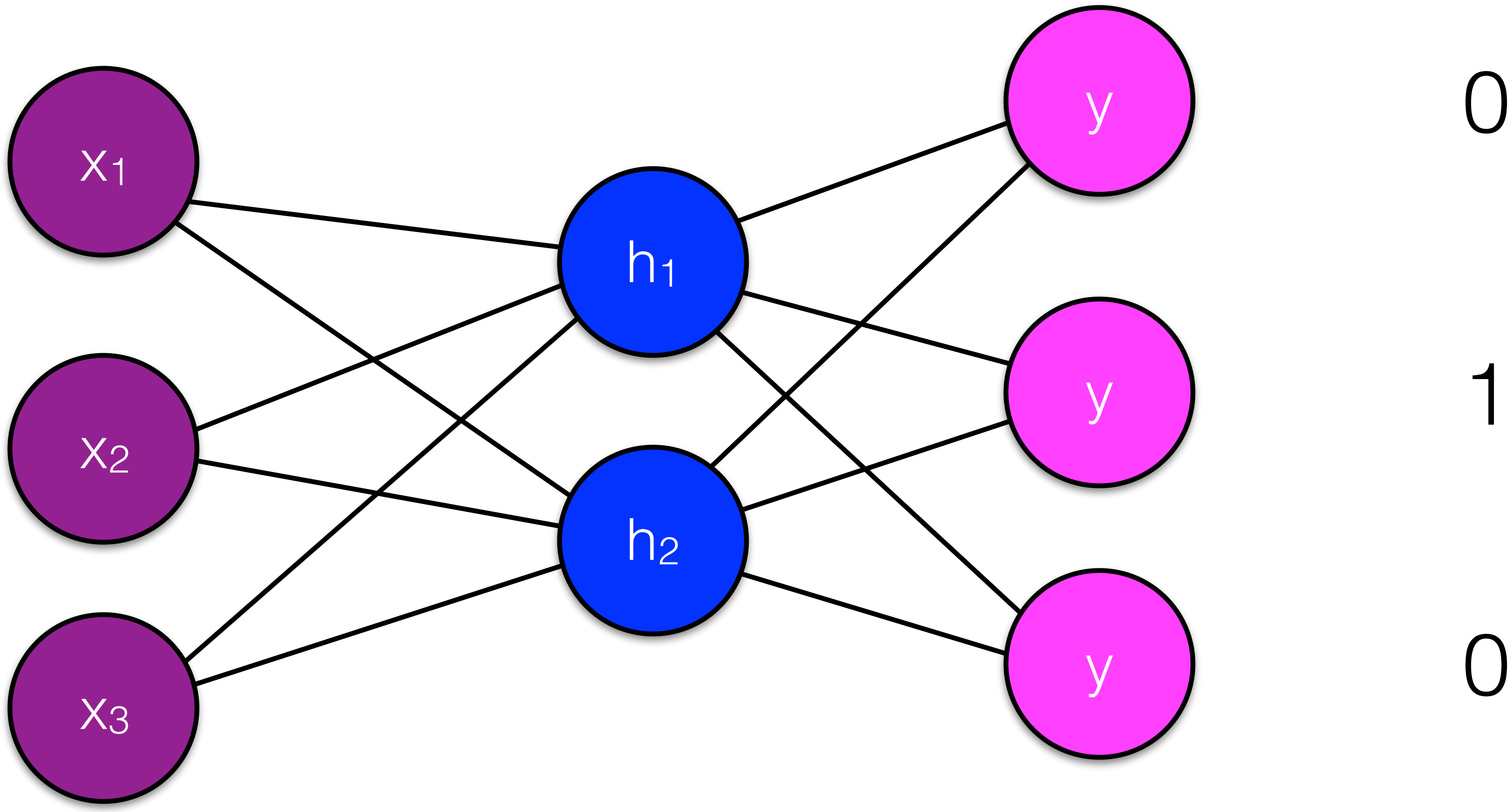
Neural network structures



1

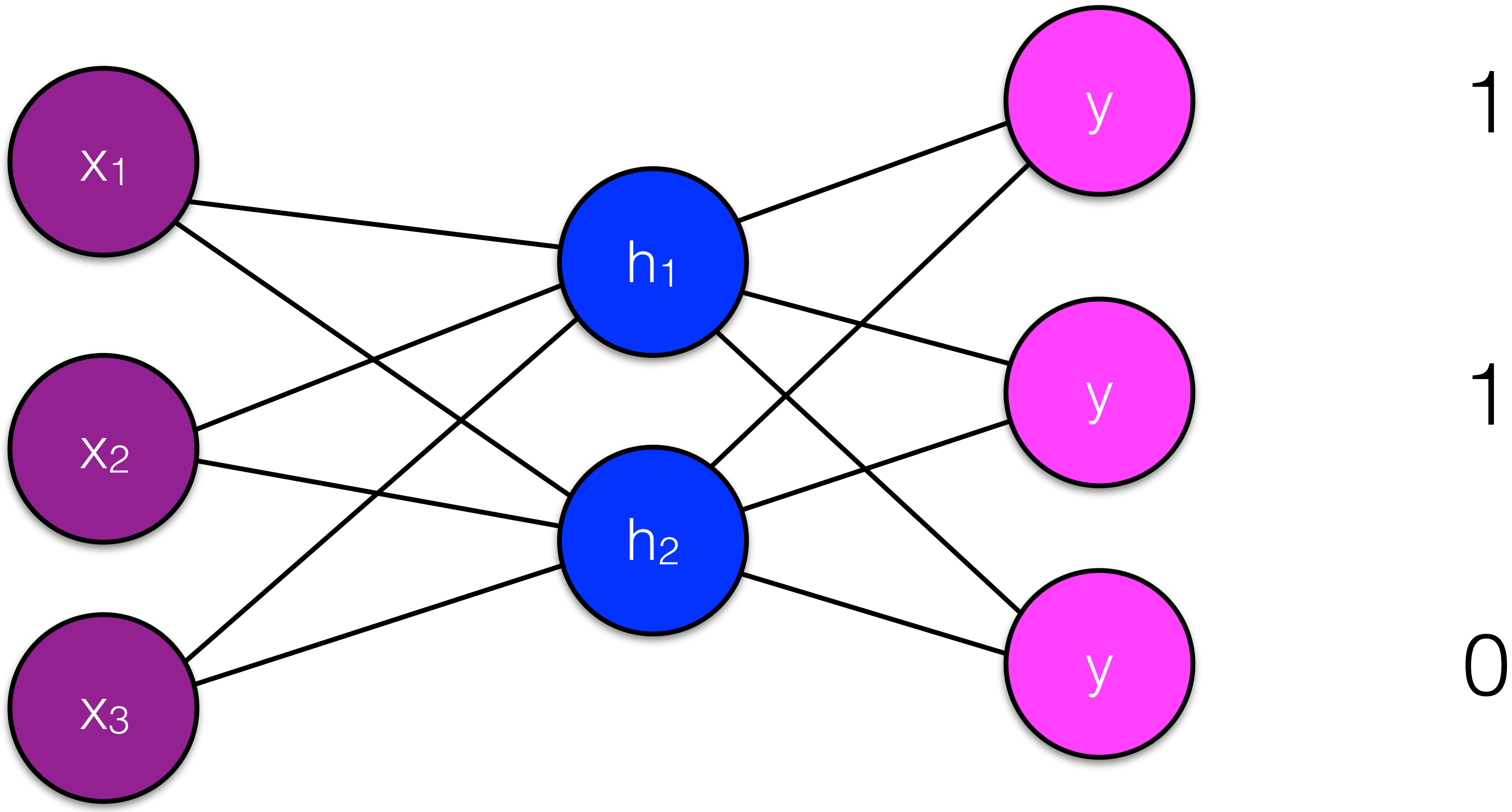
Output one real value

Neural network structures



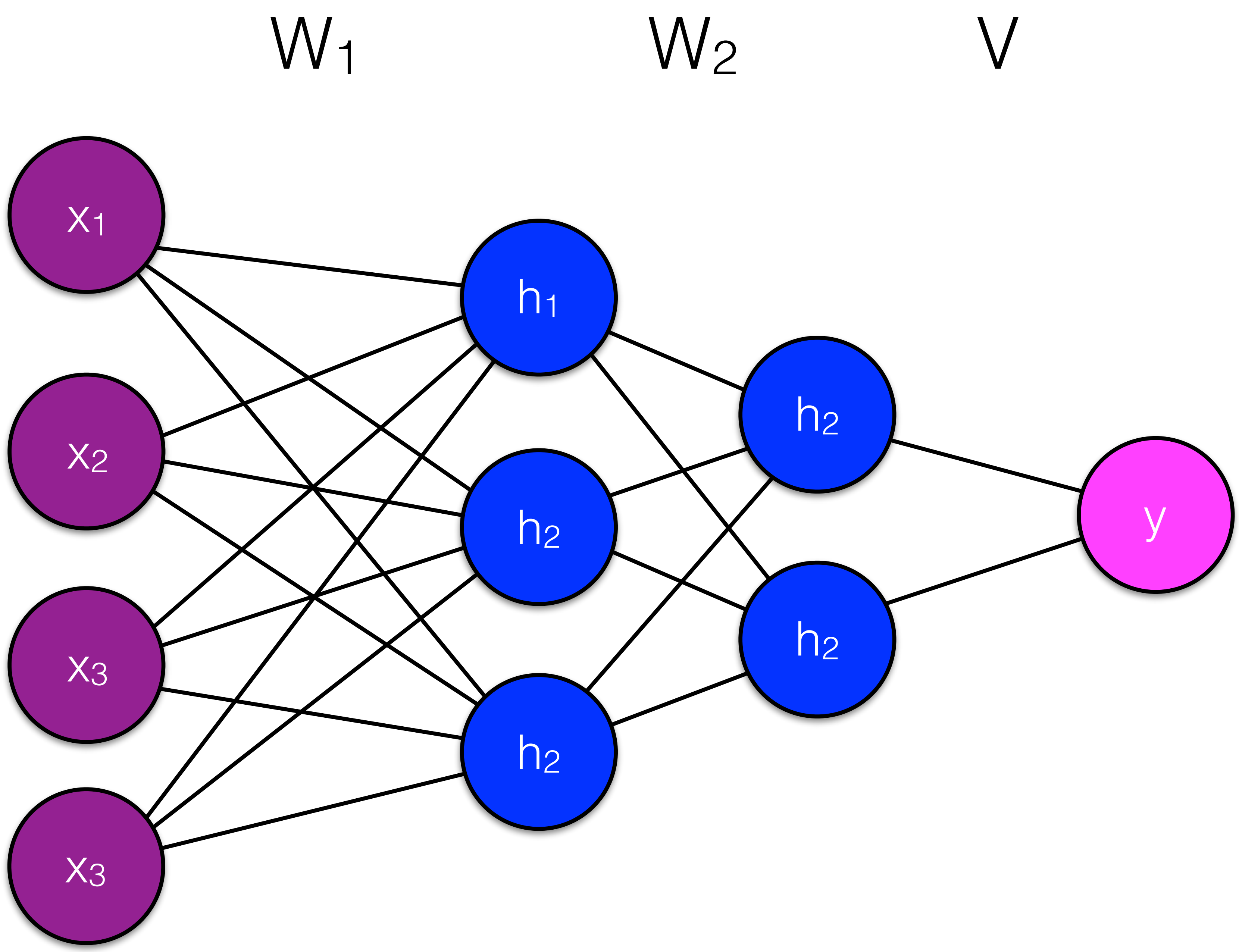
Multiclass: output 3 values, only one = 1 in training data

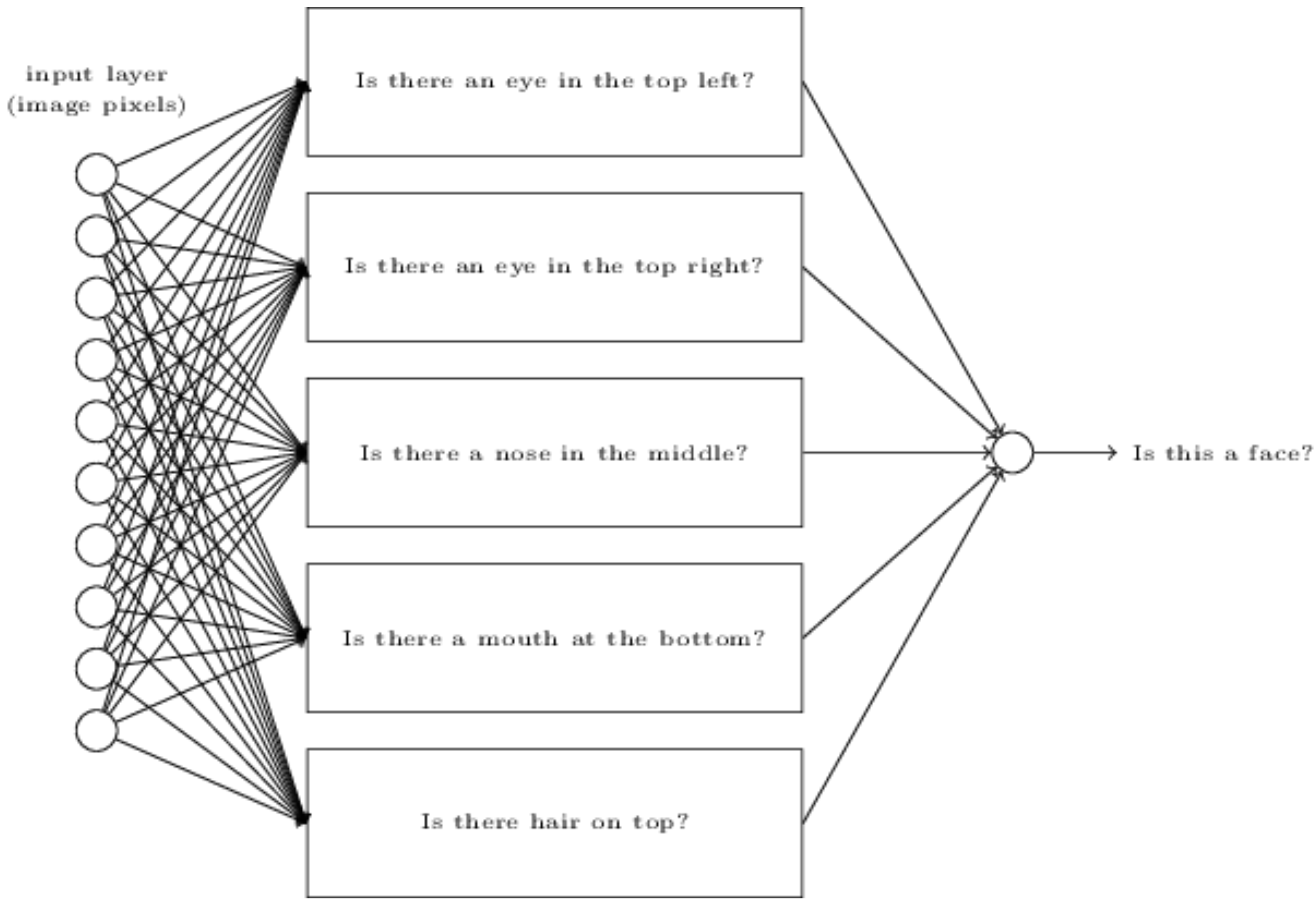
Neural network structures



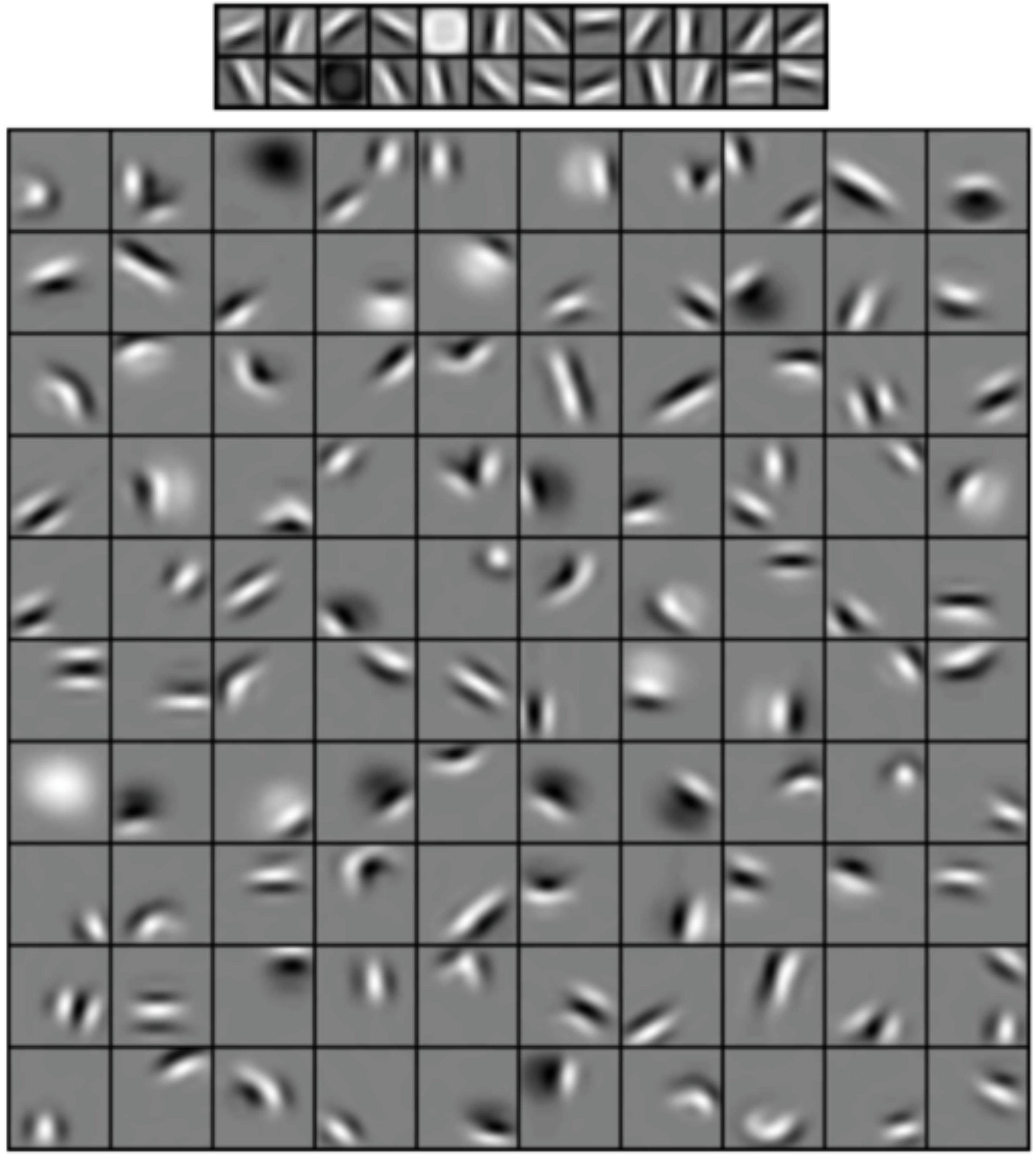
output 3 values, several = 1 in training data

Deeper networks





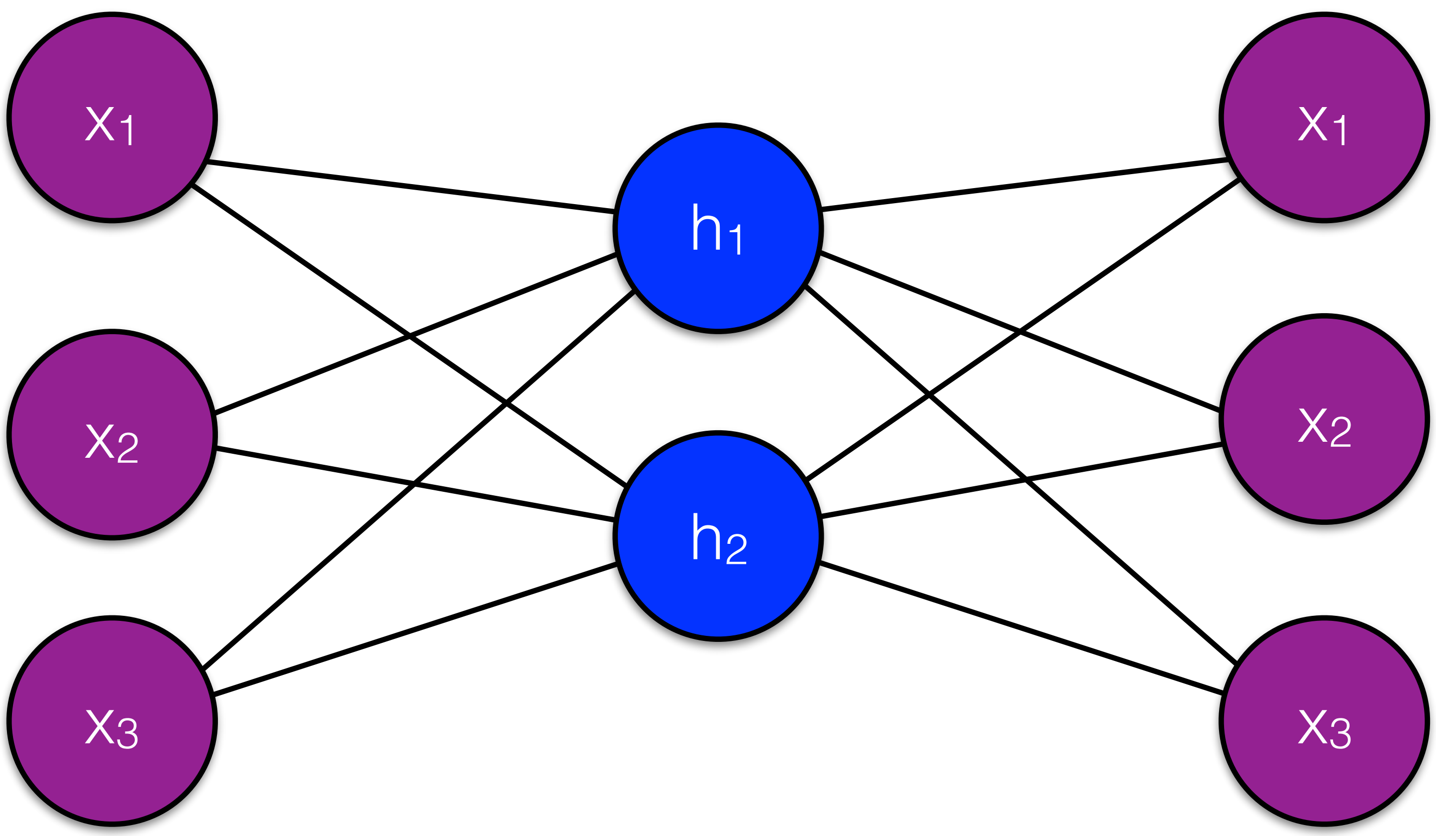
http://neuralnetworksanddeeplearning.com/chap1.html



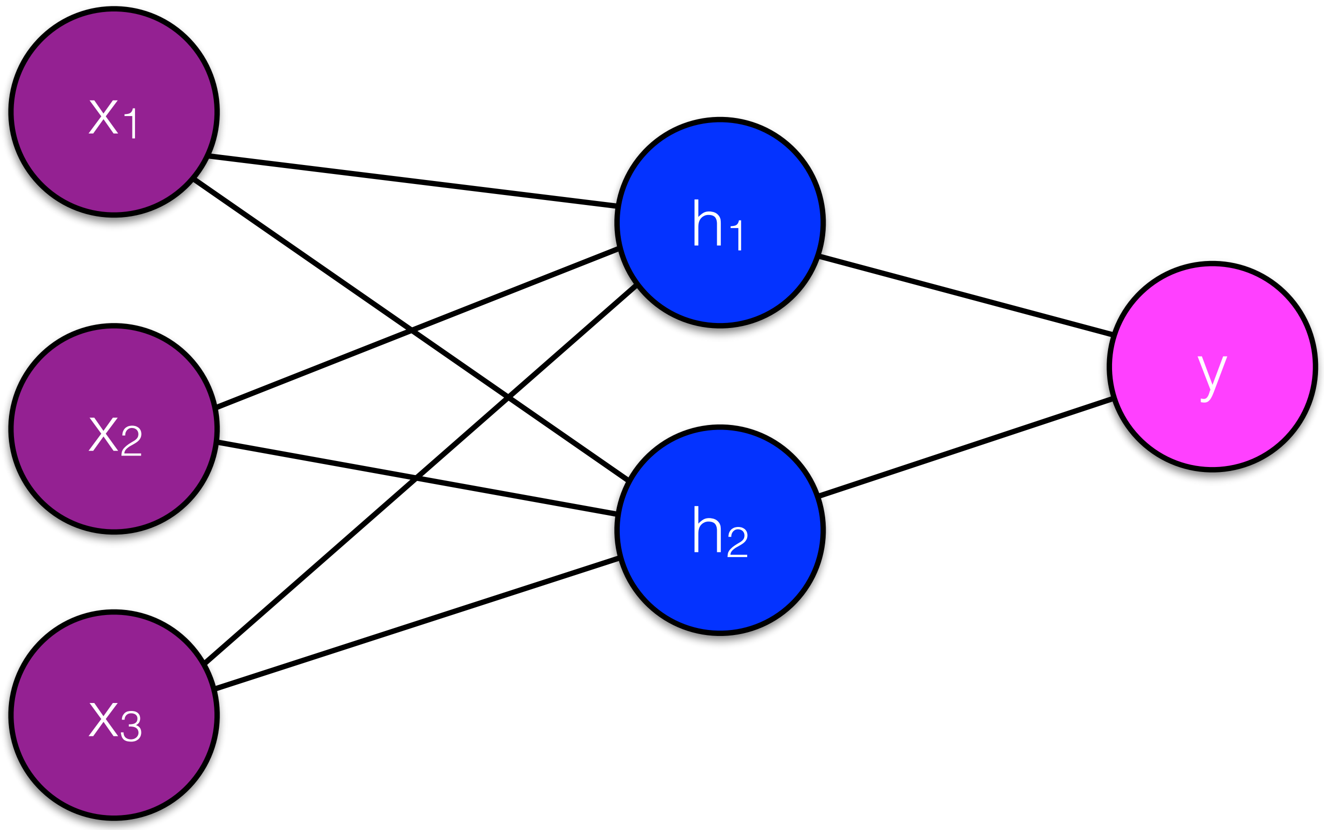
Higher order features learned for image recognition
Lee et al. 2009 (ICML)

Autoencoder

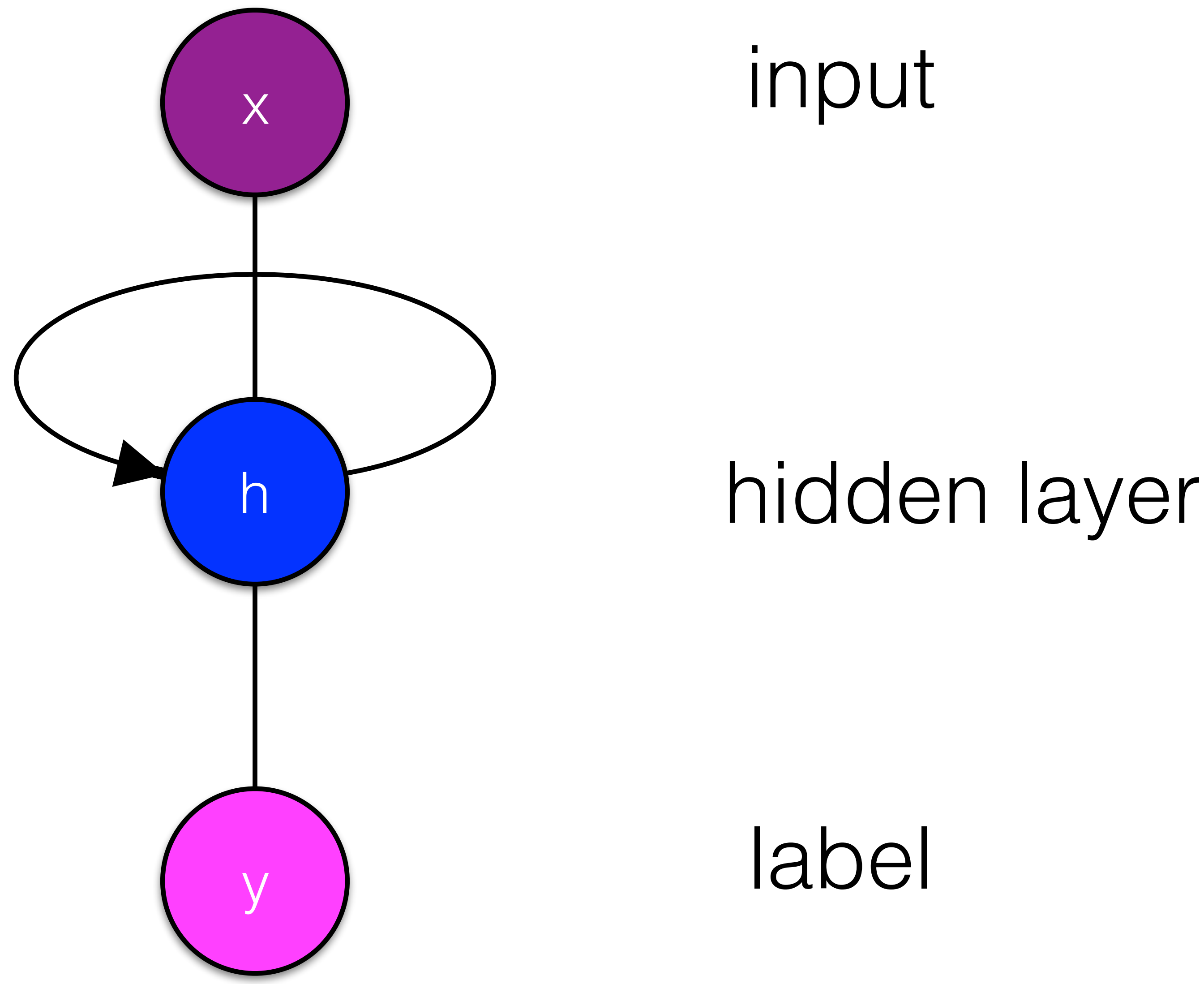
- Unsupervised neural network, where $y = x$
- Learns a low-dimensional representation of x



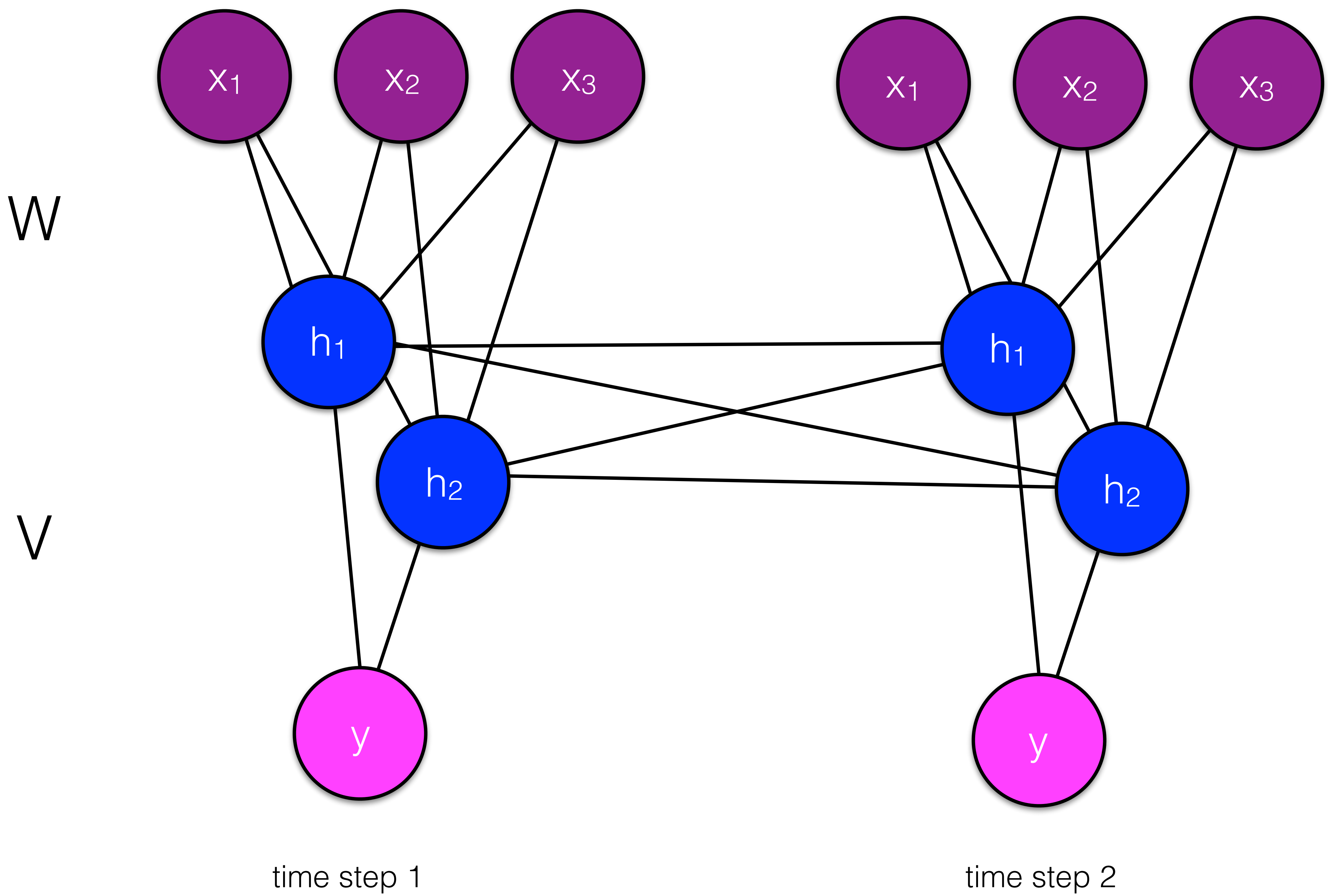
Feedforward networks



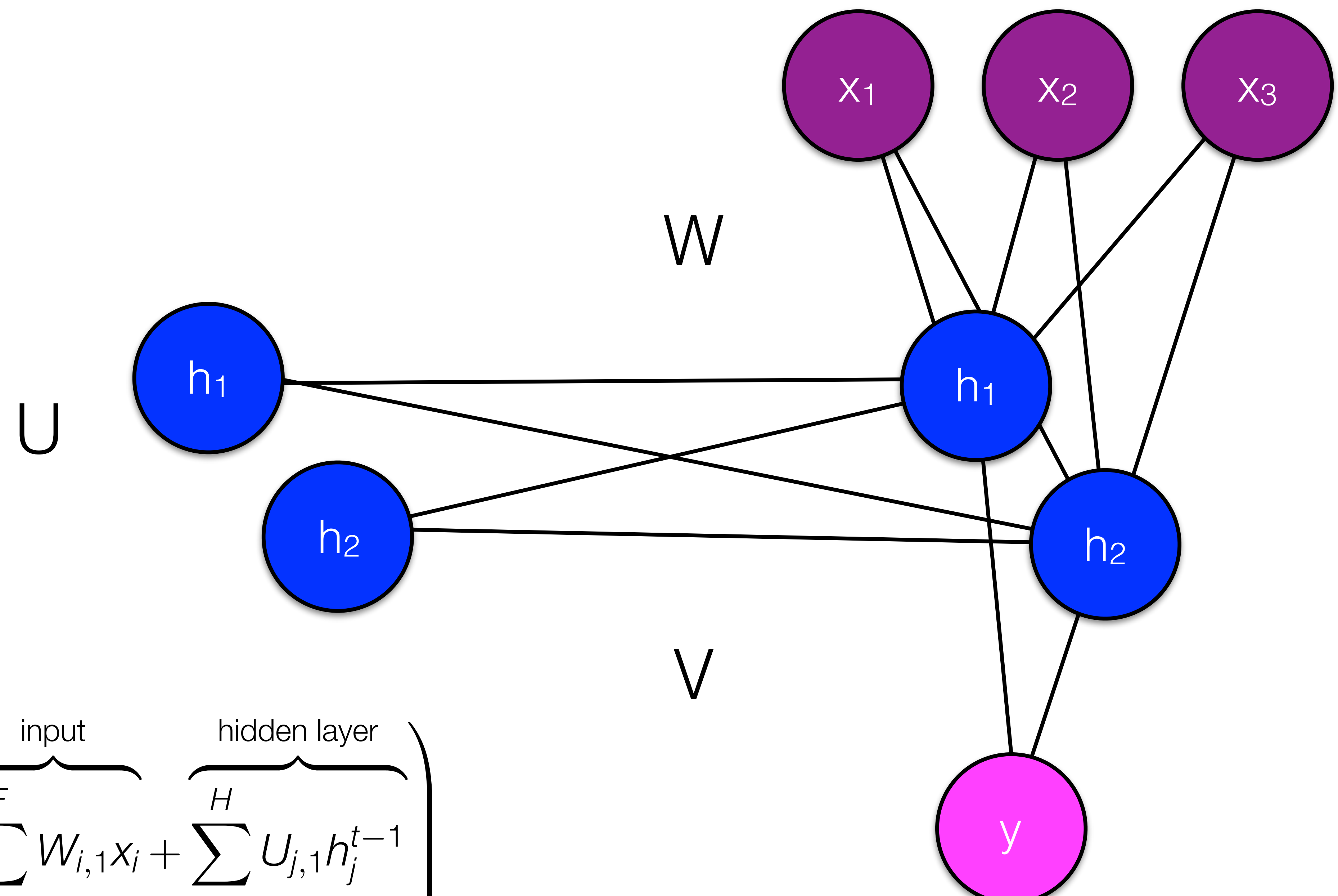
Recurrent networks



Recurrent networks



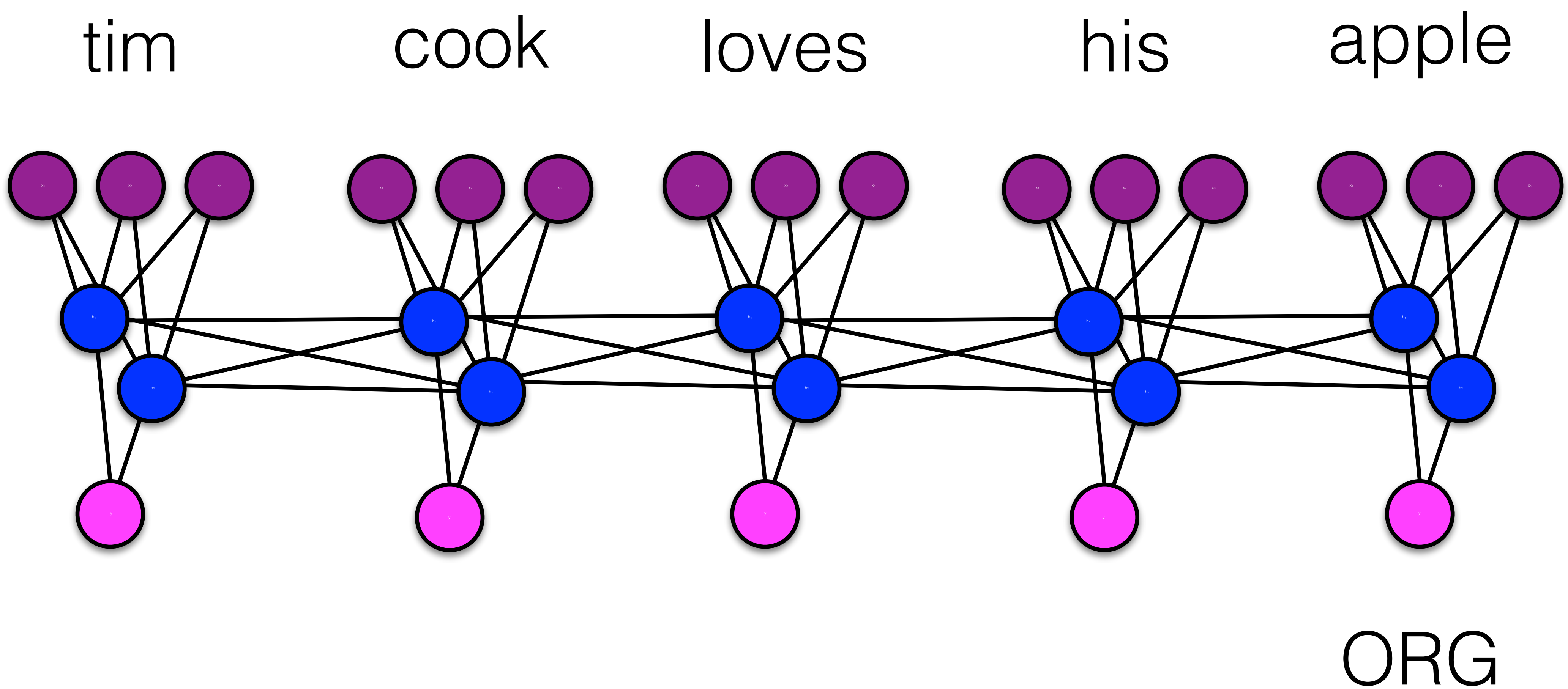
Recurrent networks



$$h_1 = \sigma\left(\overbrace{\sum_{i=1}^{F} W_{i,1} x_i}^{\text{input}} + \overbrace{\sum_{j=i}^{H} U_{j,1} h_j^{t-1}}^{\text{hidden layer}} \right)$$

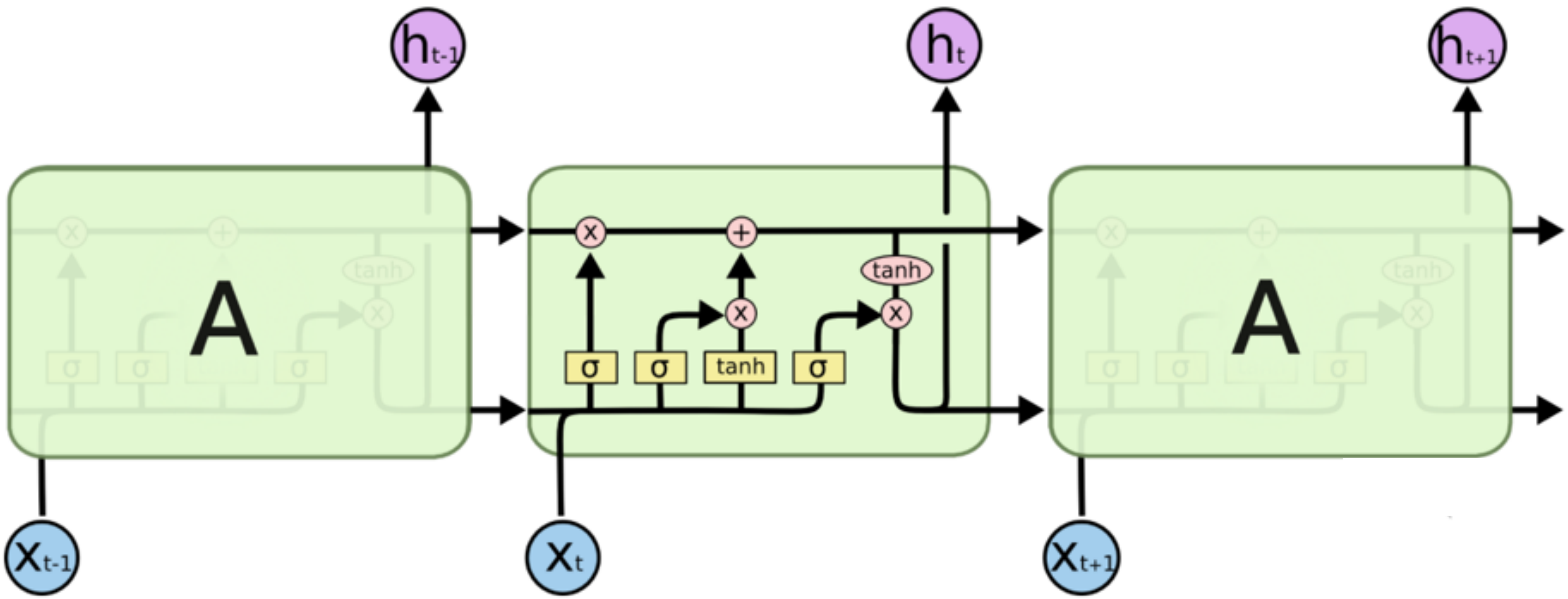
time step 2

Recurrent networks



RNNs often have a problem with long-distance dependencies.

LSTMs



http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Recurrent networks/LSTMs

task	x	y
language models	words in sequence	the next word in a sequence
part of speech tagging	words in sequence	part of speech
machine translation	words in sequence	translation