Natural Language Processing

Mehmet Can Yavuz, PhD
Adapted from Info 256 - David Bamman, UC Berkeley

History of NLP

- Foundational insights, 1940s/1950s
- Two camps (symbolic/stochastic), 1957-1970
- Four paradigms (stochastic, logic-based, NLU, discourse modeling), 1970-1983
- Empiricism and FSM (1983-1993)
- Field comes together (1994-1999)
- Machine learning (2000–today)

J&M 2008, ch 1

- Neural networks (~2014–today)

Neural networks in NLP

- Language modeling [Mikolov et al. 2010]
- Text classification [Kim 2014; Iyyer et al. 2015]
- Syntactic parsing [Chen and Manning 2014, Dyer et al. 2015, Andor et al. 2016]
- CCG super tagging [Lewis and Steedman 2014]
- Machine translation [Cho et al. 2014, Sustkever et al. 2014]
- Dialogue agents [Sordoni et al. 2015, Vinyals and Lee 2015, Ji et al. 2016]
- (for overview, see Goldberg 2017, 1.3.1)

Neural networks

- Discrete, high-dimensional representation of inputs (one-hot vectors) -> low-dimensional “distributed” representations.
- Non-linear interactions of input features
- Multiple layers to capture hierarchical structure

Neural network libraries

TensorFlow

theano

K

PYTORCH

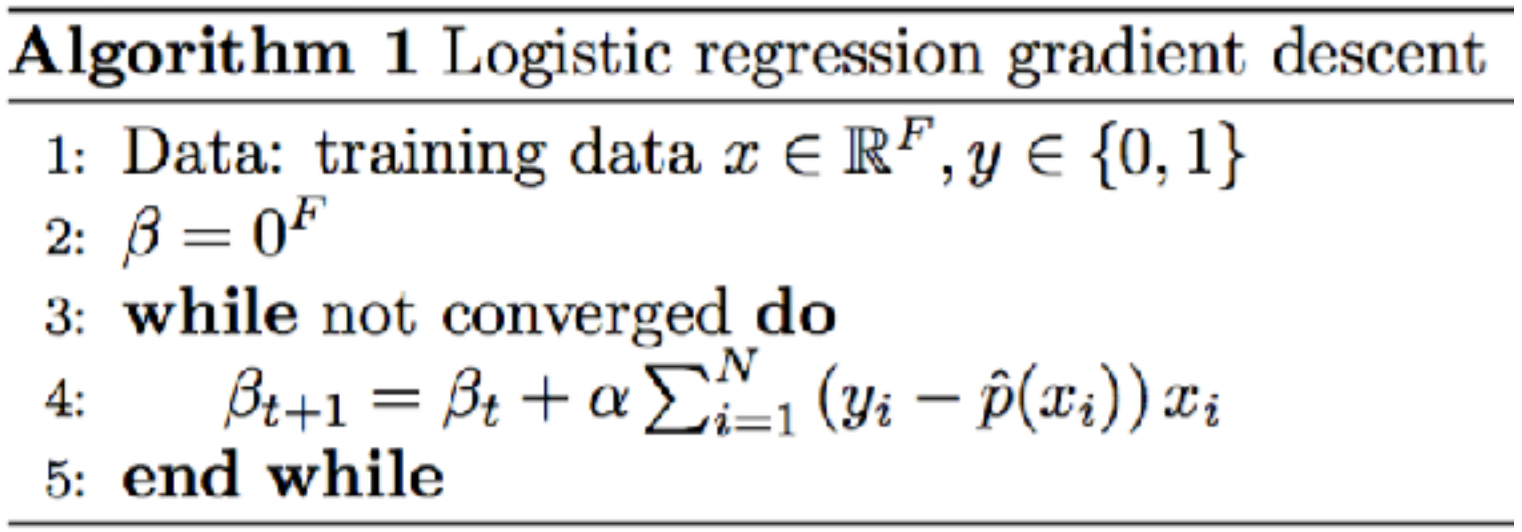
dy/net

Logistic regression

$$\hat{y} = \frac{1}{1 + \exp\left(-\sum_{i=1}^{F} x_i \beta_i\right)}$$

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

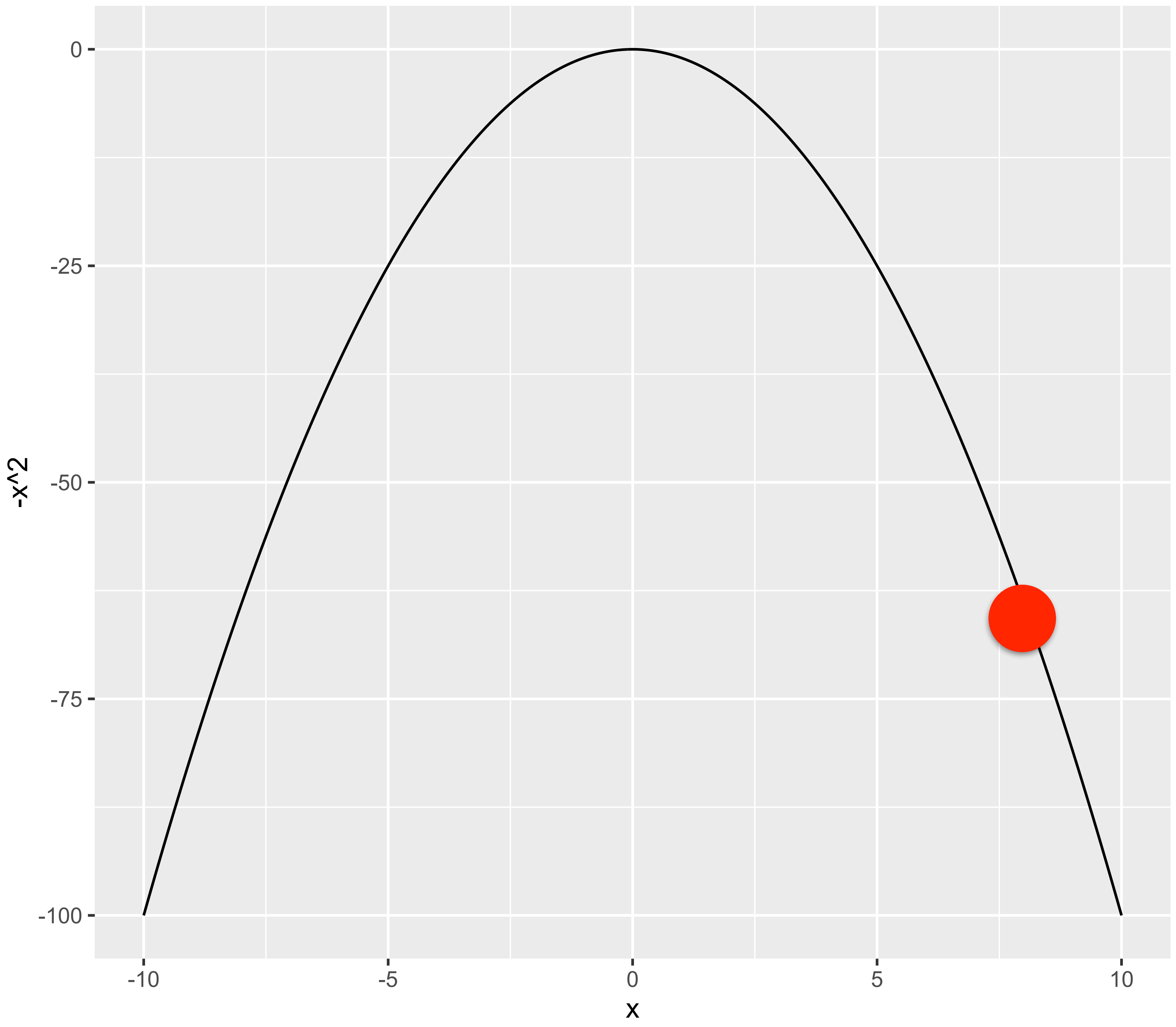
SGD



Algorithm 1 Logistic regression gradient descent

```
1: Data: training data $x \in \mathbb{R}^F, y \in \{0, 1\}$
2: $\beta = 0^F$
3: while not converged do
4:     $\beta_{t+1} = \beta_t + \alpha \sum_{i=1}^{N} (y_i - \hat{p}(x_i))\, x_i$
5: end while
```

Calculate the derivative of some loss function with respect to parameters we can change, update accordingly to make predictions on training data a little less wrong next time.



$x + \alpha(-2x)$

$[\alpha = 0.1]$

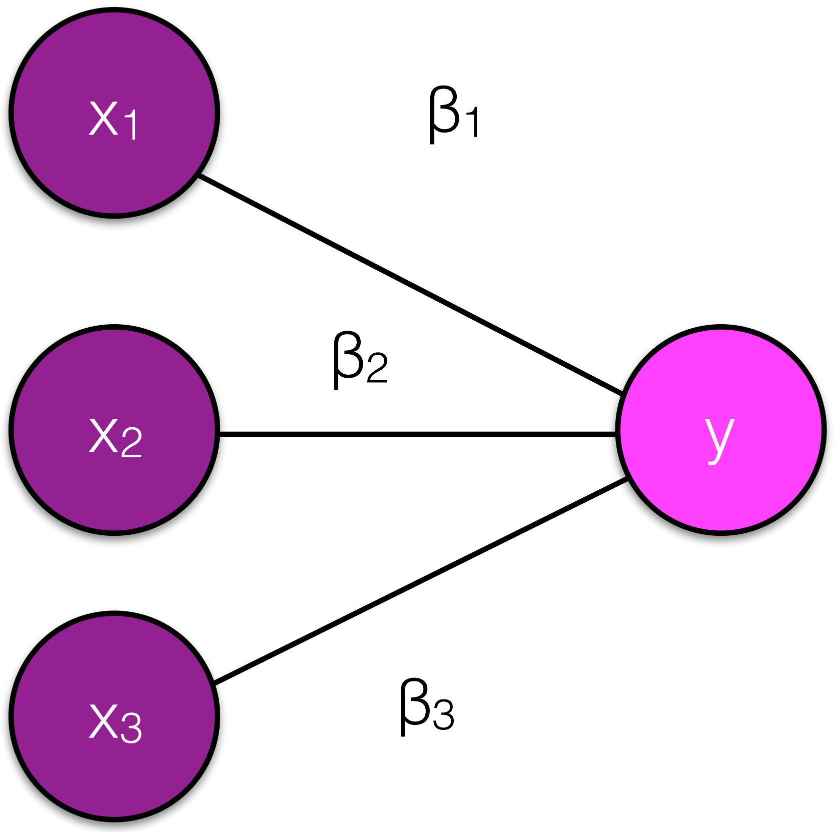
x	.1(-2x)
8.00	-1.6
6.40	-1.28
5.12	-1.02
4.10	-0.82
3.28	-0.66
2.62	-0.52
2.10	-0.42
1.68	-0.34
1.34	-0.27
1.07	-0.21
0.86	-0.17
0.69	-0.14

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

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

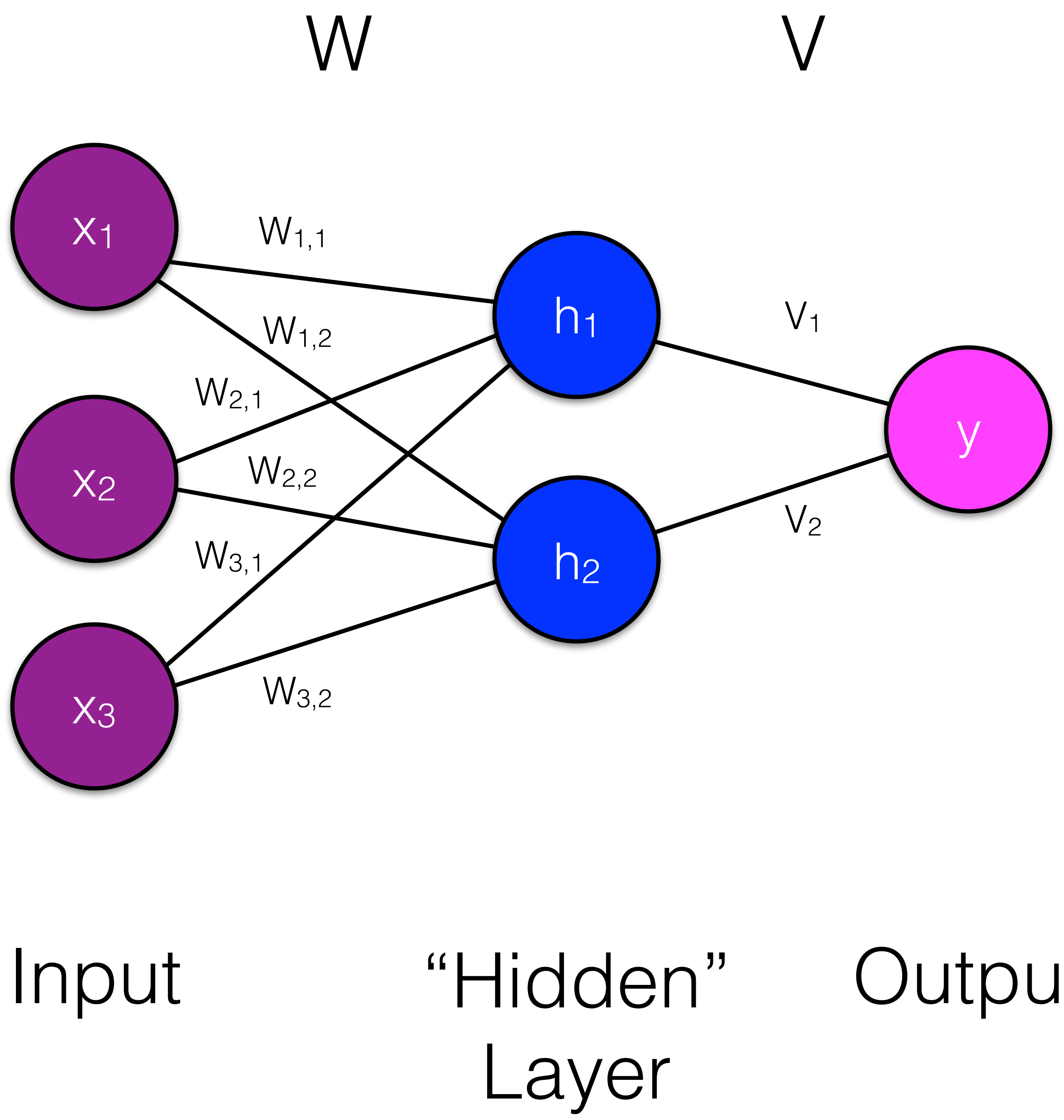
Logistic regression

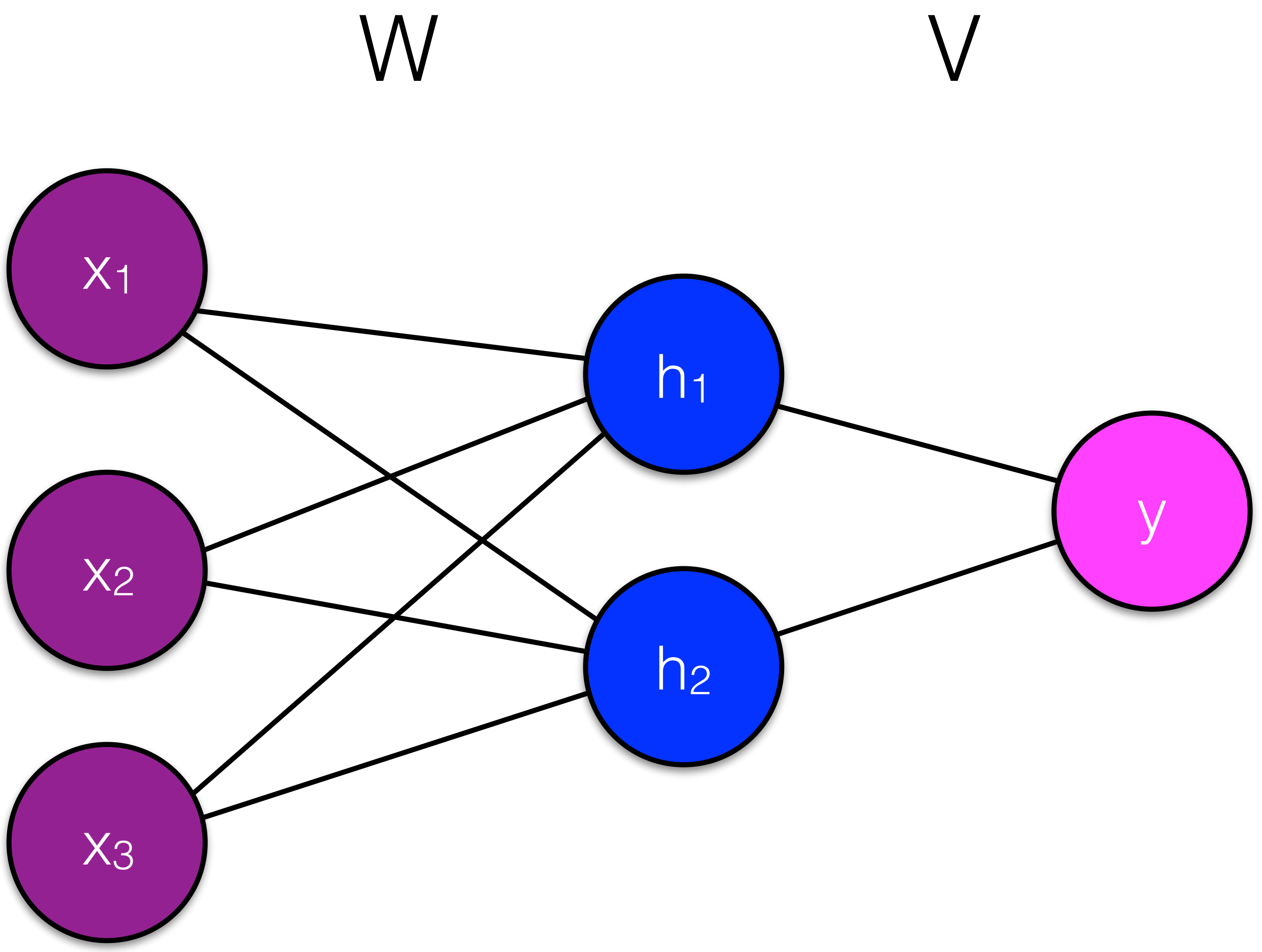
$$\hat{y} = \frac{1}{1 + \exp\left(-\sum_{i=1}^{F} x_i \beta_i\right)}$$



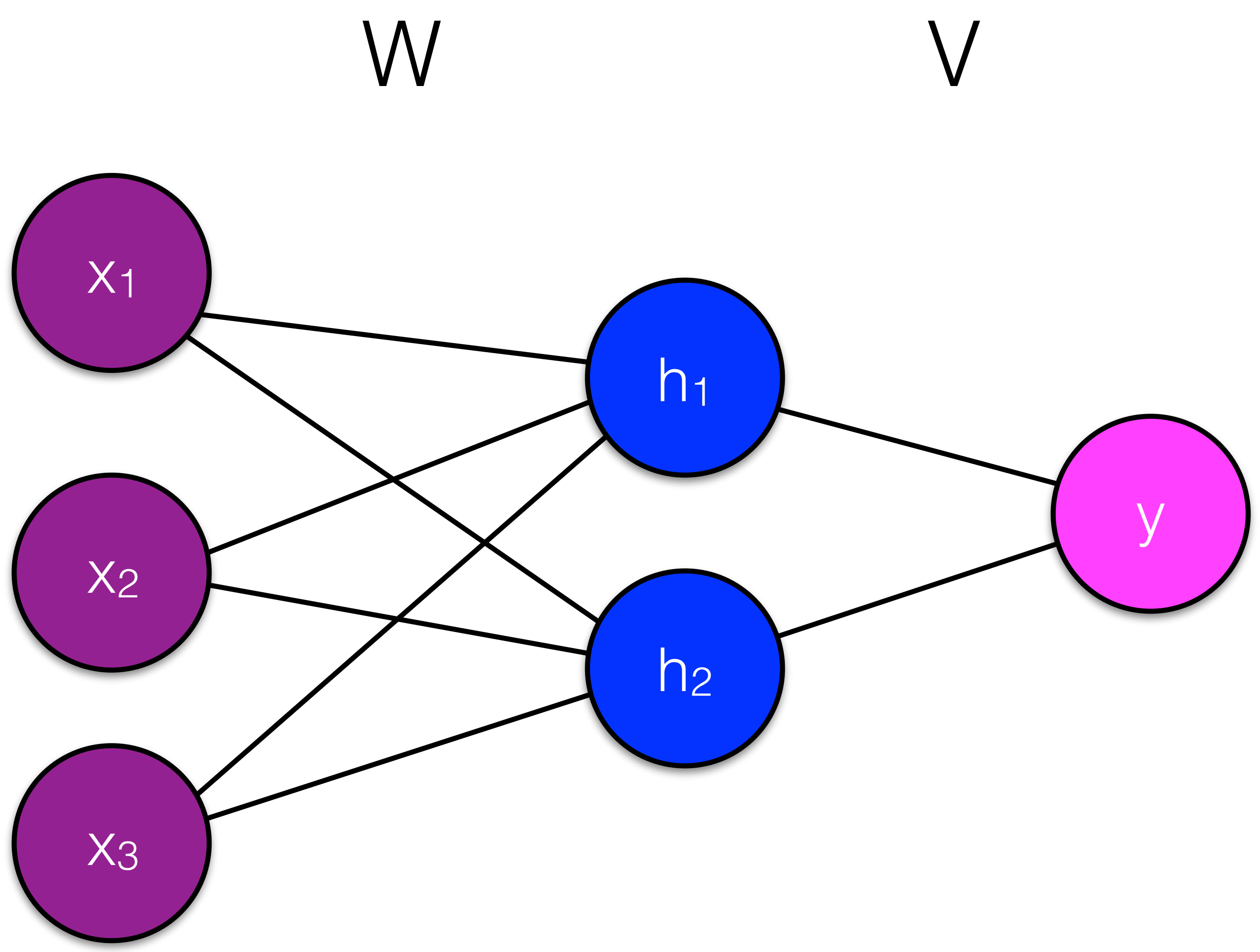
	x	β
not	1	-0.5
bad	1	-1.7
movie	0	0.3

*For simplicity, we're leaving out the bias term, but assume most layers have them as well.



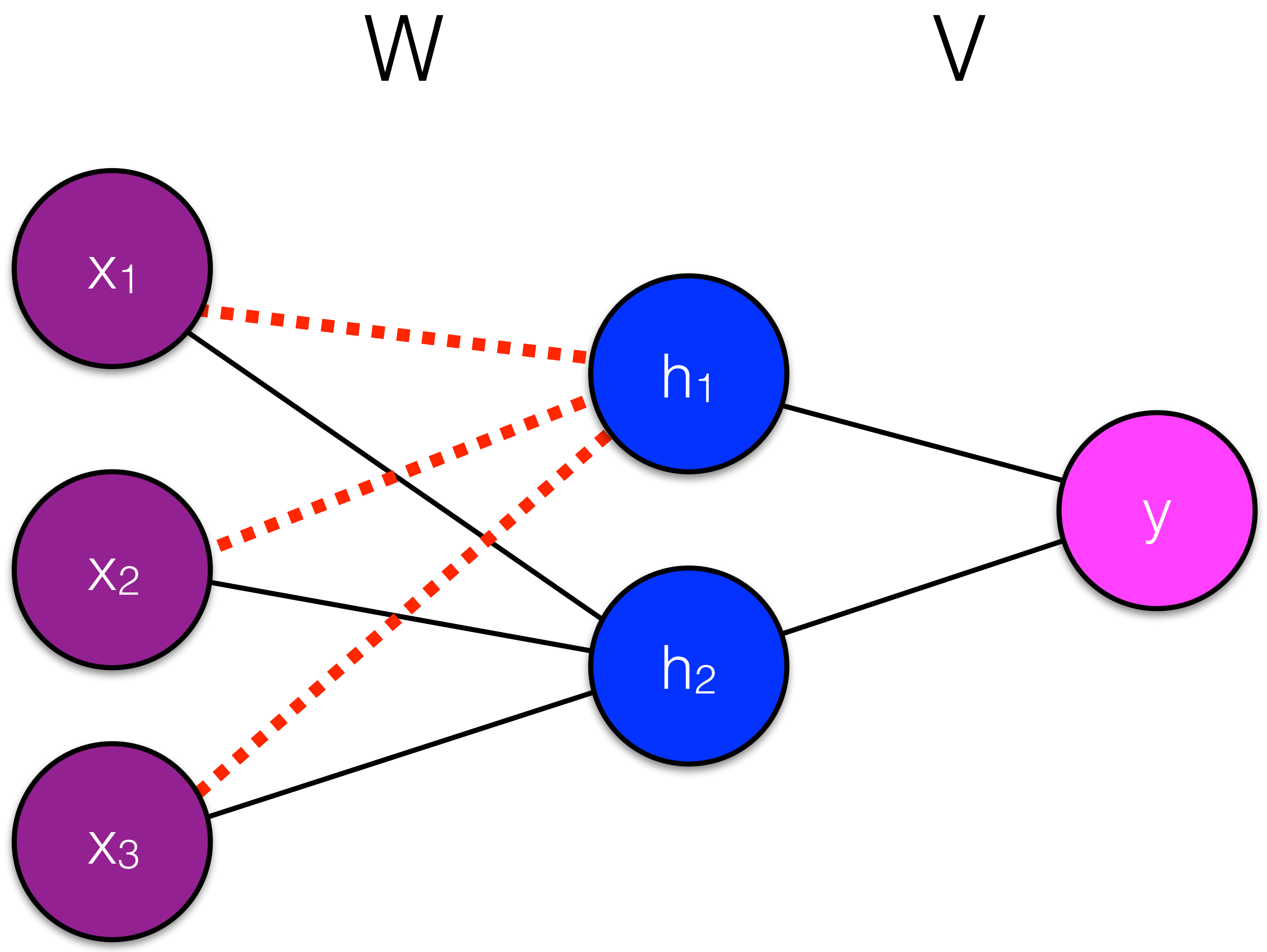


	x	W		V	y
not	1	-0.5	1.3	4.1	1
bad	1	0.4	0.08	-0.9	
movie	0	1.7	3.1		



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

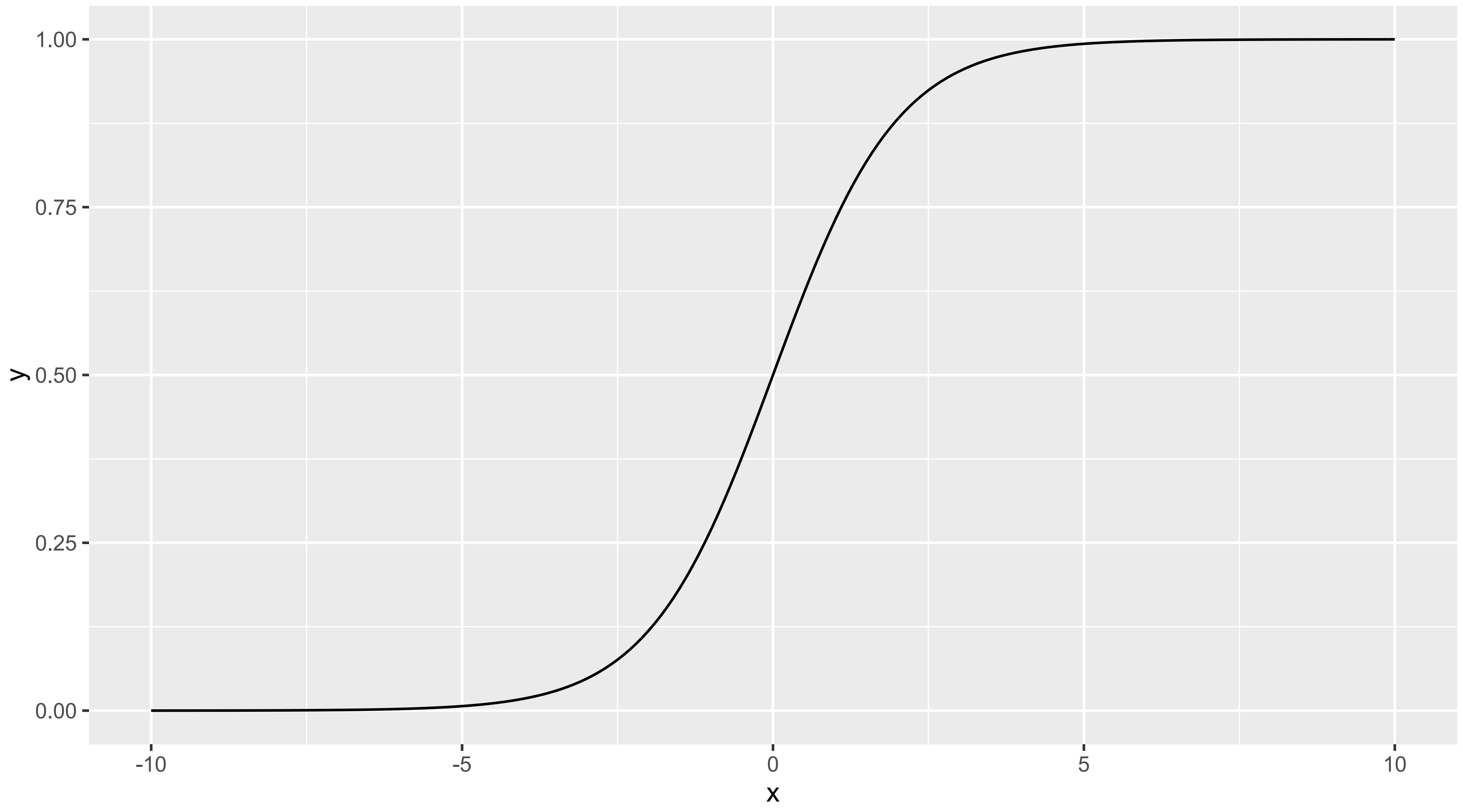
the hidden nodes are completely determined by the input and weights



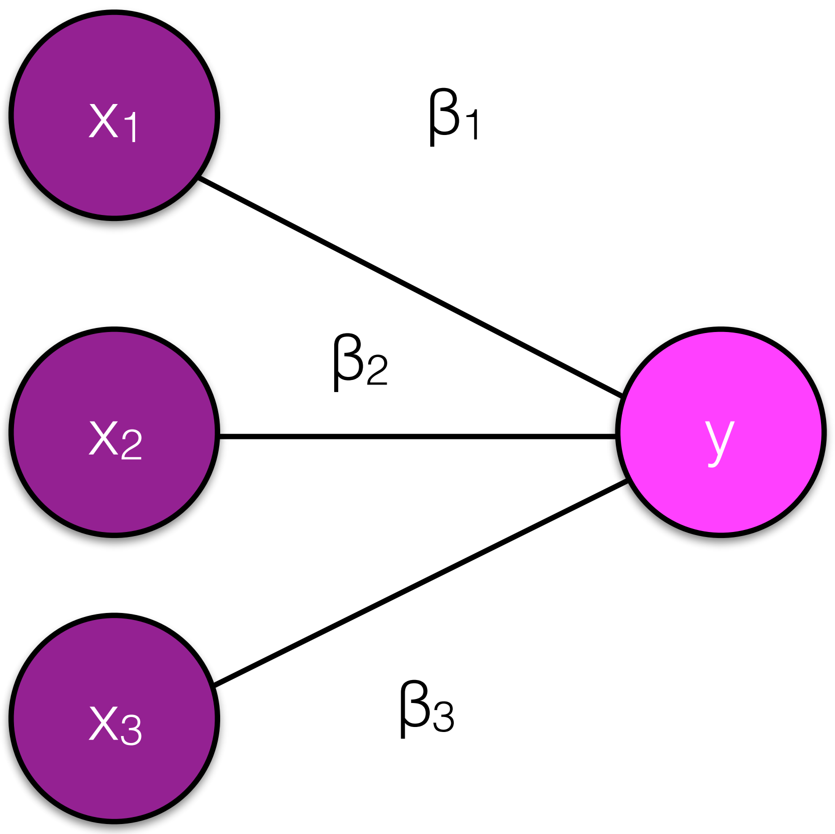
$$h_1 = f\left(\sum_{i=1}^{F} x_i W_{i,1}\right)$$

Activation functions

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



Logistic regression



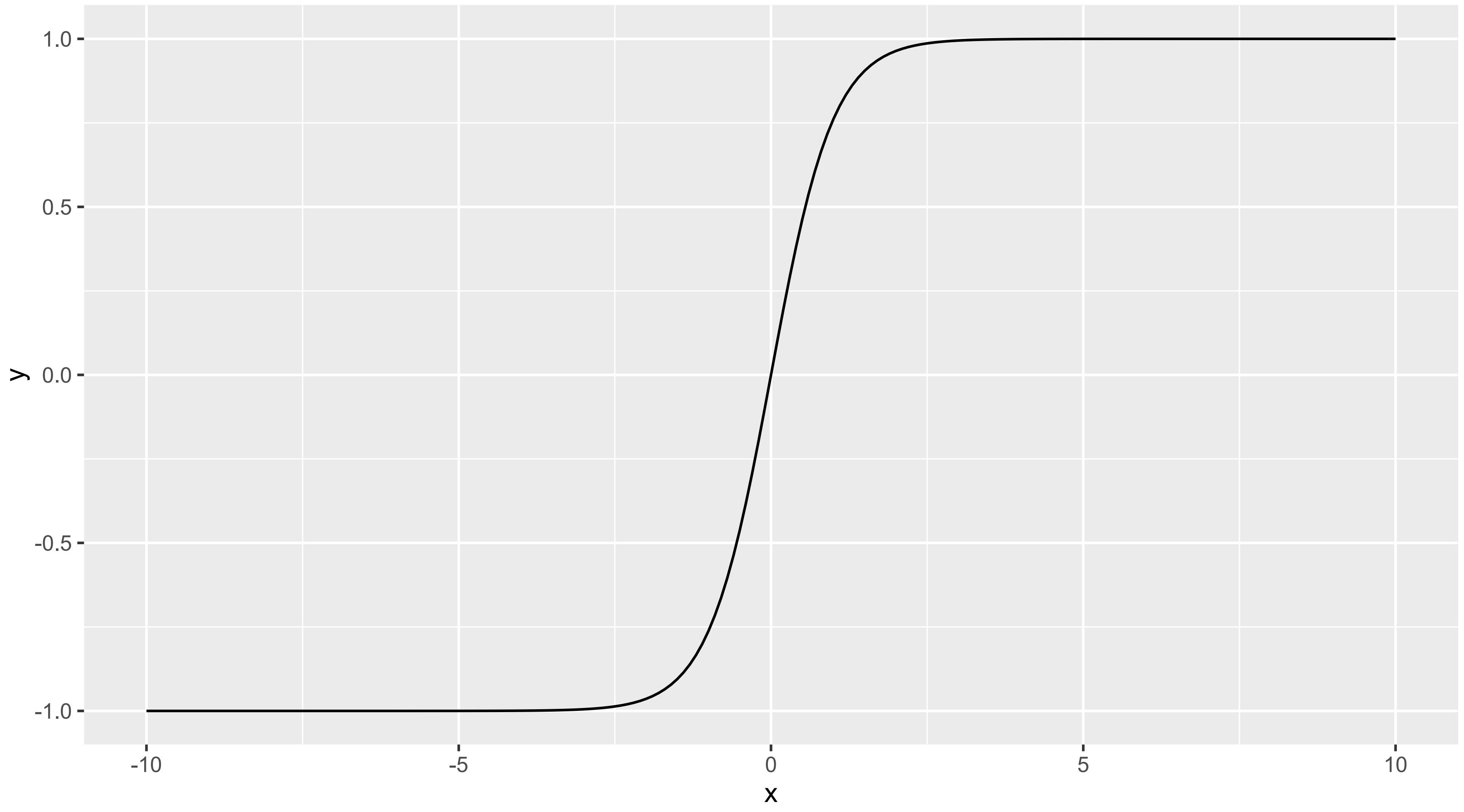
$$\hat{y} = \frac{1}{1 + \exp\left(-\sum_{i=1}^{F} x_i \beta_i\right)}$$

$$\hat{y} = \sigma\left(\sum_{i=1}^{F} x_i \beta_i\right)$$

We can think about logistic regression as a neural network with no hidden layers

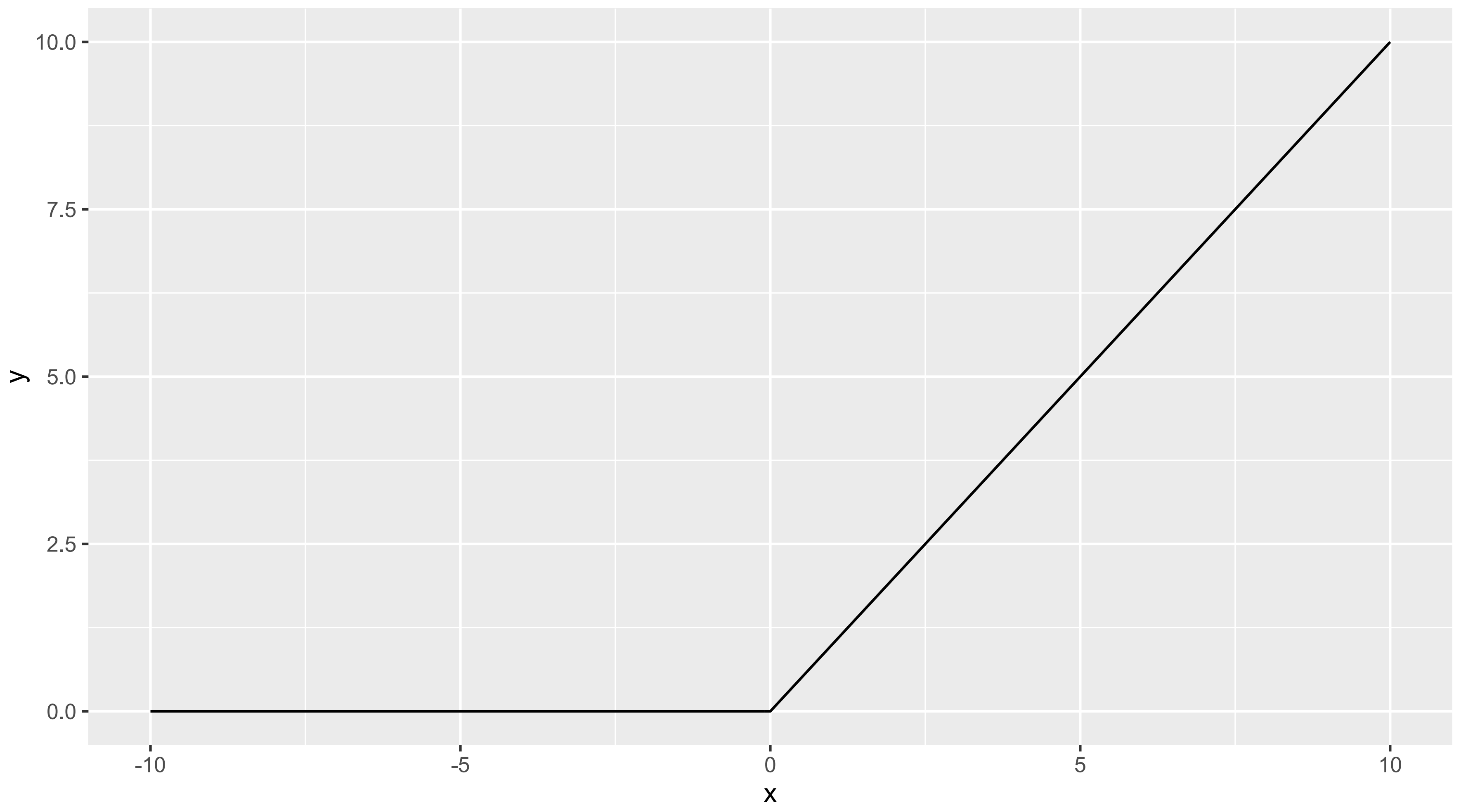
Activation functions

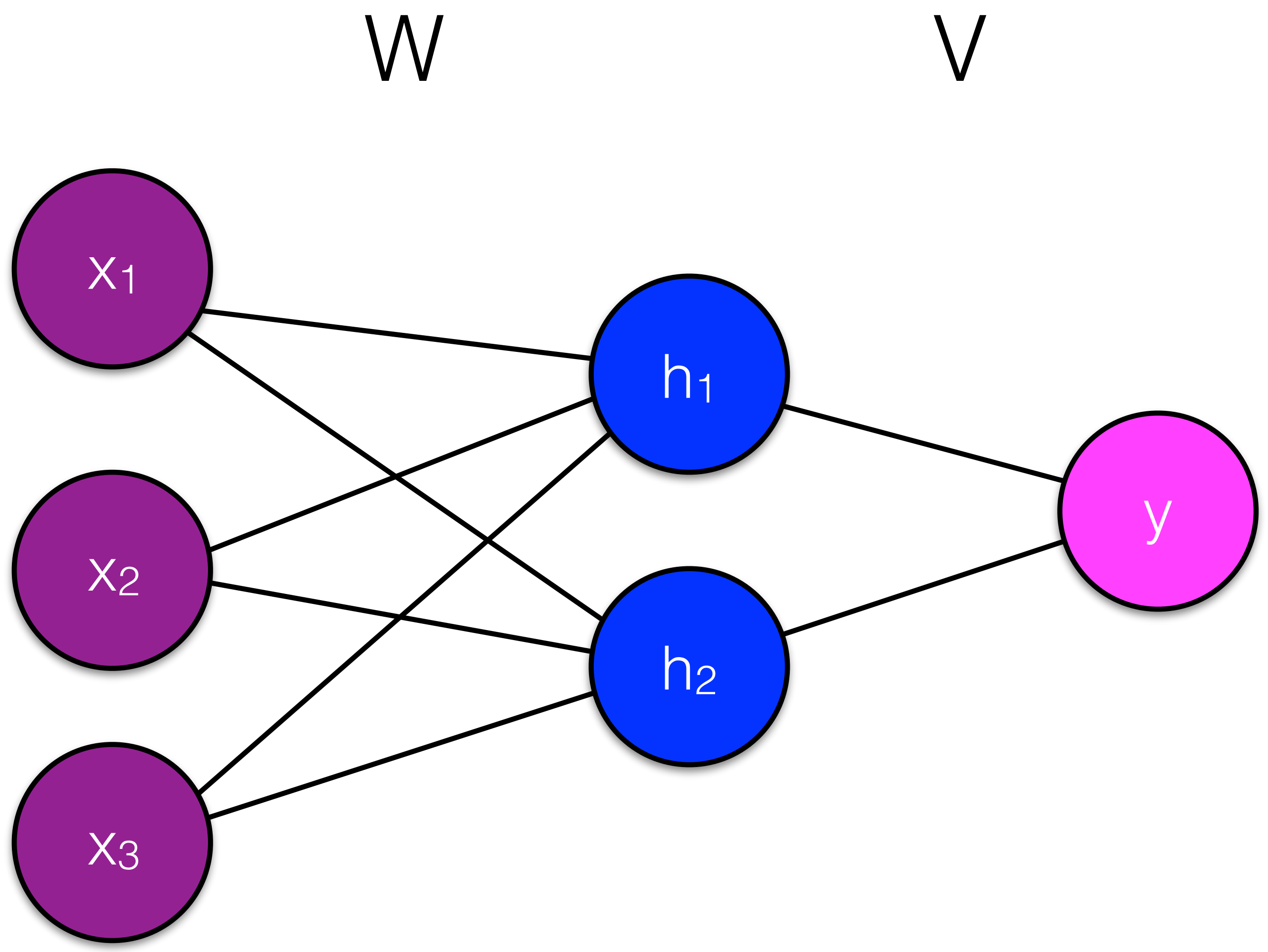
$$\tanh(z) = \frac{\exp(z) - \exp(-z)}{\exp(z) + \exp(-z)}$$



Activation functions

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

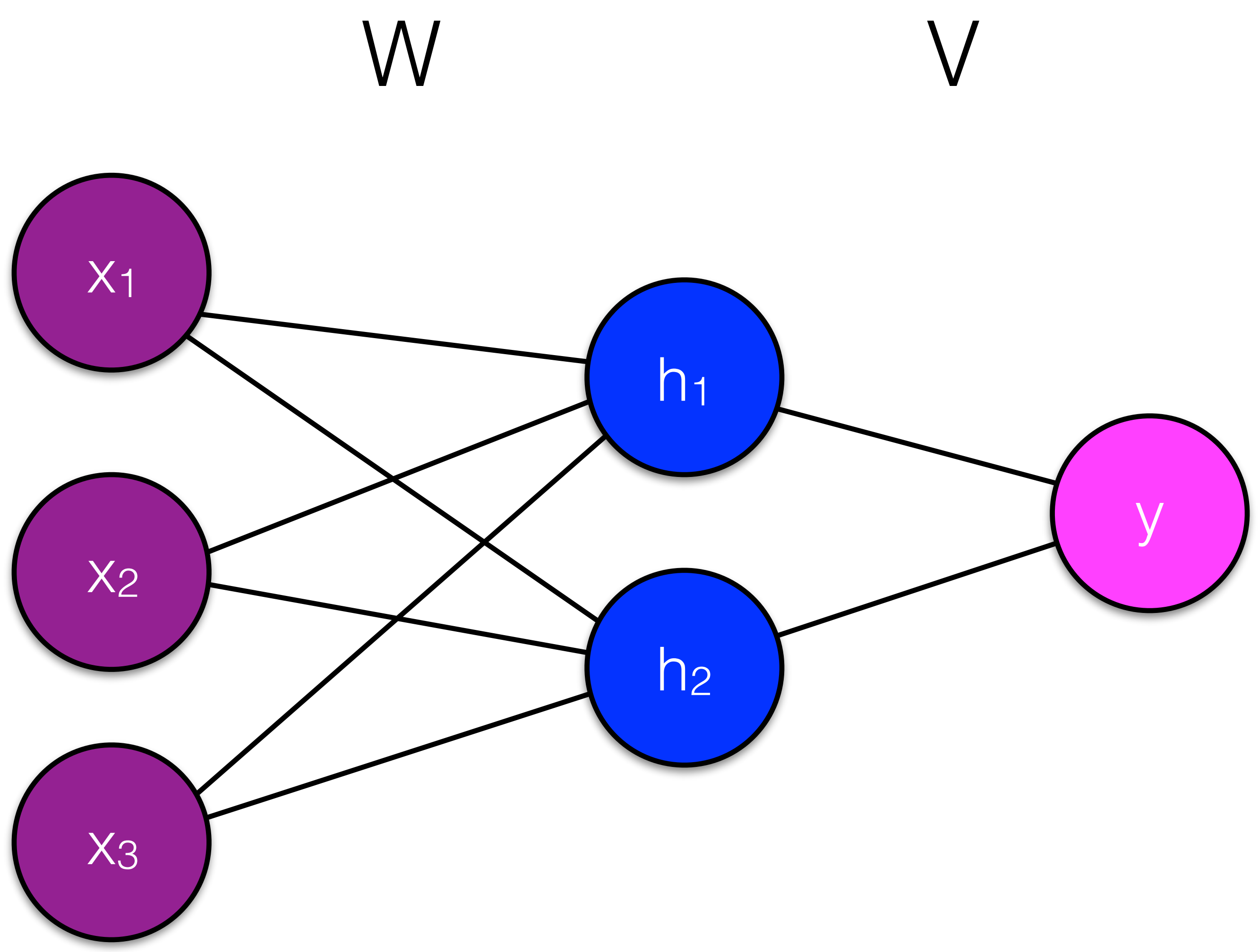




$$h_1 = \sigma\left(\sum_{i=1}^{F} x_i W_{i,1}\right)$$

$$\hat{y} = \sigma\left[V_1 h_1 + V_2 h_2\right]$$

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



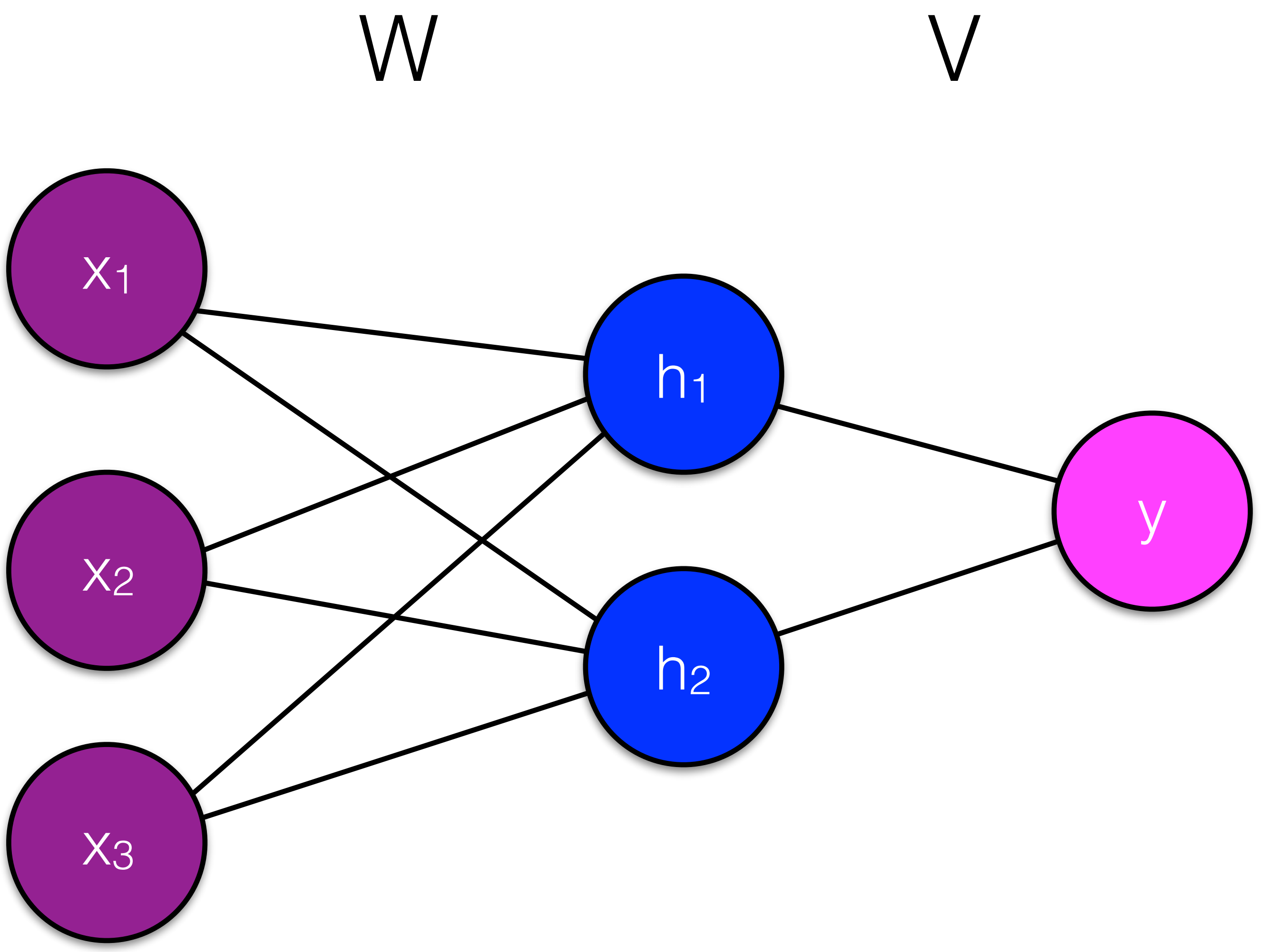
$$\hat{y} = \sigma\left[V_1\left(\sigma\left(\sum_{i}^{F} x_i W_{i,1}\right)\right) + V_2\left(\sigma\left(\sum_{i}^{F} x_i W_{i,2}\right)\right)\right]$$

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

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

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.



Neural networks are a series of functions chained together

$$\boxed{xW} \rightarrow \boxed{\sigma(xW)} \rightarrow \boxed{\sigma(xW)\,V} \rightarrow \boxed{\sigma(\sigma(xW)\,V)}$$

The loss is another function chained on top

$$\boxed{\log(\sigma(\sigma(xW)\,V))}$$

Chain rule

$$\frac{\partial}{\partial V} \log\left(\sigma\left(\sigma\left(xW\right)V\right)\right)$$

Let's take the likelihood for a single training example with label y =1; we want this value to be as high as possible

$$= \frac{\partial \log\left(\sigma\left(\sigma\left(xW\right)V\right)\right)}{\partial \sigma\left(\sigma\left(xW\right)V\right)} \frac{\partial \sigma\left(\sigma\left(xW\right)V\right)}{\partial \sigma\left(xW\right)V} \frac{\partial \sigma\left(xW\right)V}{\partial V}$$

$$= \overbrace{\frac{\partial \log\left(\sigma\left(hV\right)\right)}{\partial \sigma\left(hV\right)}}^{A} \overbrace{\frac{\partial \sigma\left(hV\right)}{\partial hV}}^{B} \overbrace{\frac{\partial hV}{\partial V}}^{C}$$

Chain rule

$$= \overbrace{\frac{\partial \log\left(\sigma\left(hV\right)\right)}{\partial \sigma\left(hV\right)}}^{A} \overbrace{\frac{\partial \sigma\left(hV\right)}{\partial hV}}^{B} \overbrace{\frac{\partial hV}{\partial V}}^{C}$$

$$= \overbrace{\frac{1}{\sigma\left(hV\right)}}^{A} \times \overbrace{\sigma\left(hV\right) \times \left(1 - \sigma\left(hV\right)\right)}^{B} \times \overbrace{h}^{C}$$

$$= (1 - \sigma\left(hV\right))h$$

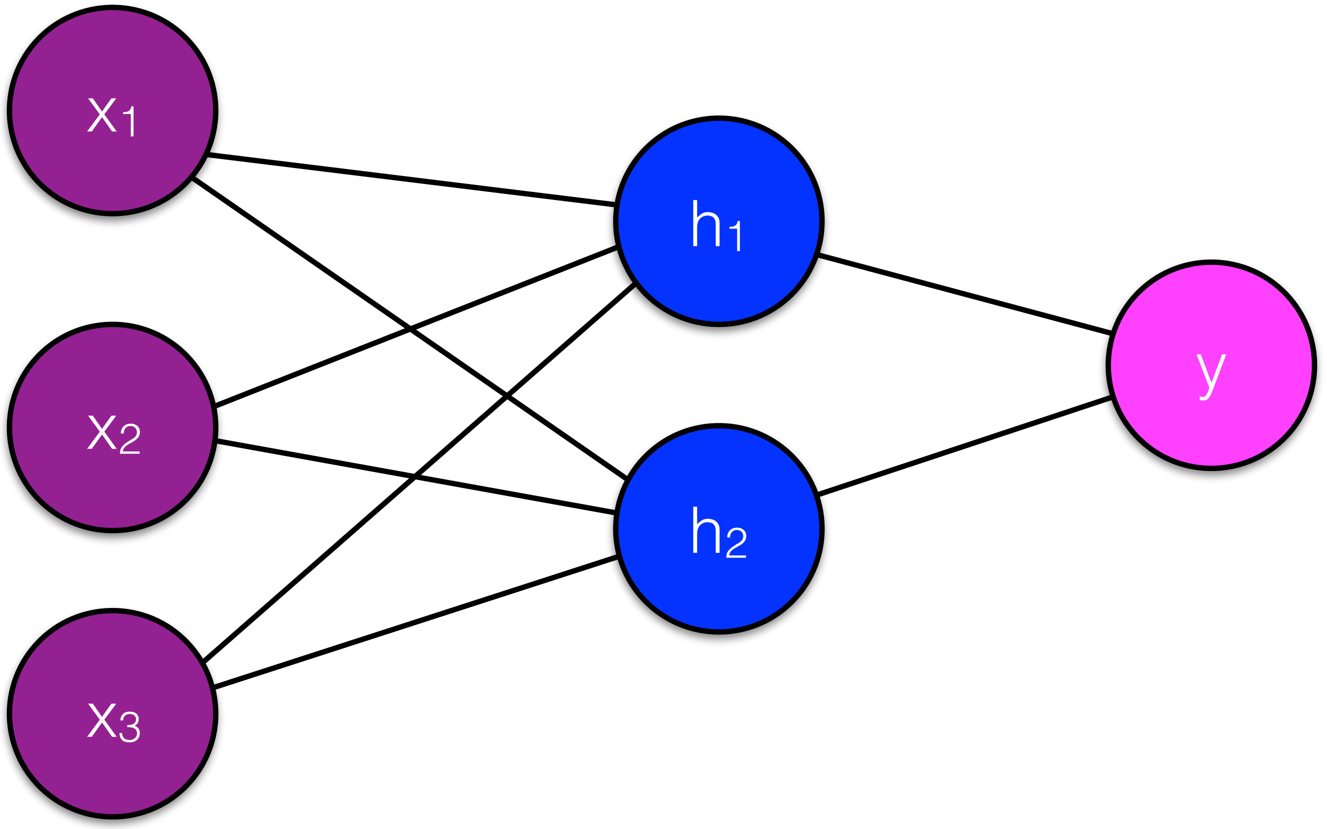
$$= (1 - \hat{y})h$$

Neural networks

- Tremendous flexibility on design choices (exchange feature engineering for model engineering)

- Articulate model structure and use the chain rule to derive parameter updates.

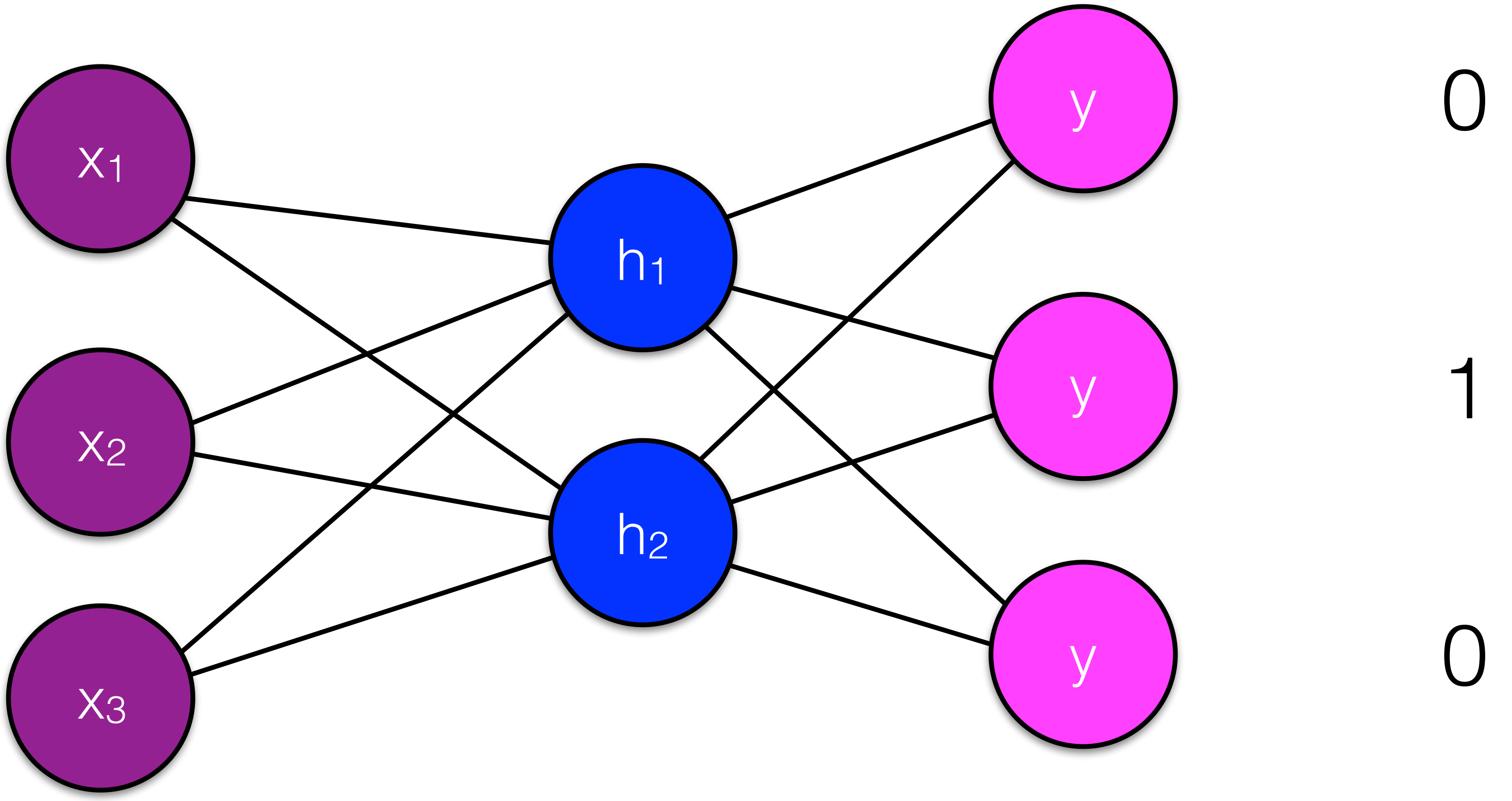
Neural network structures



1

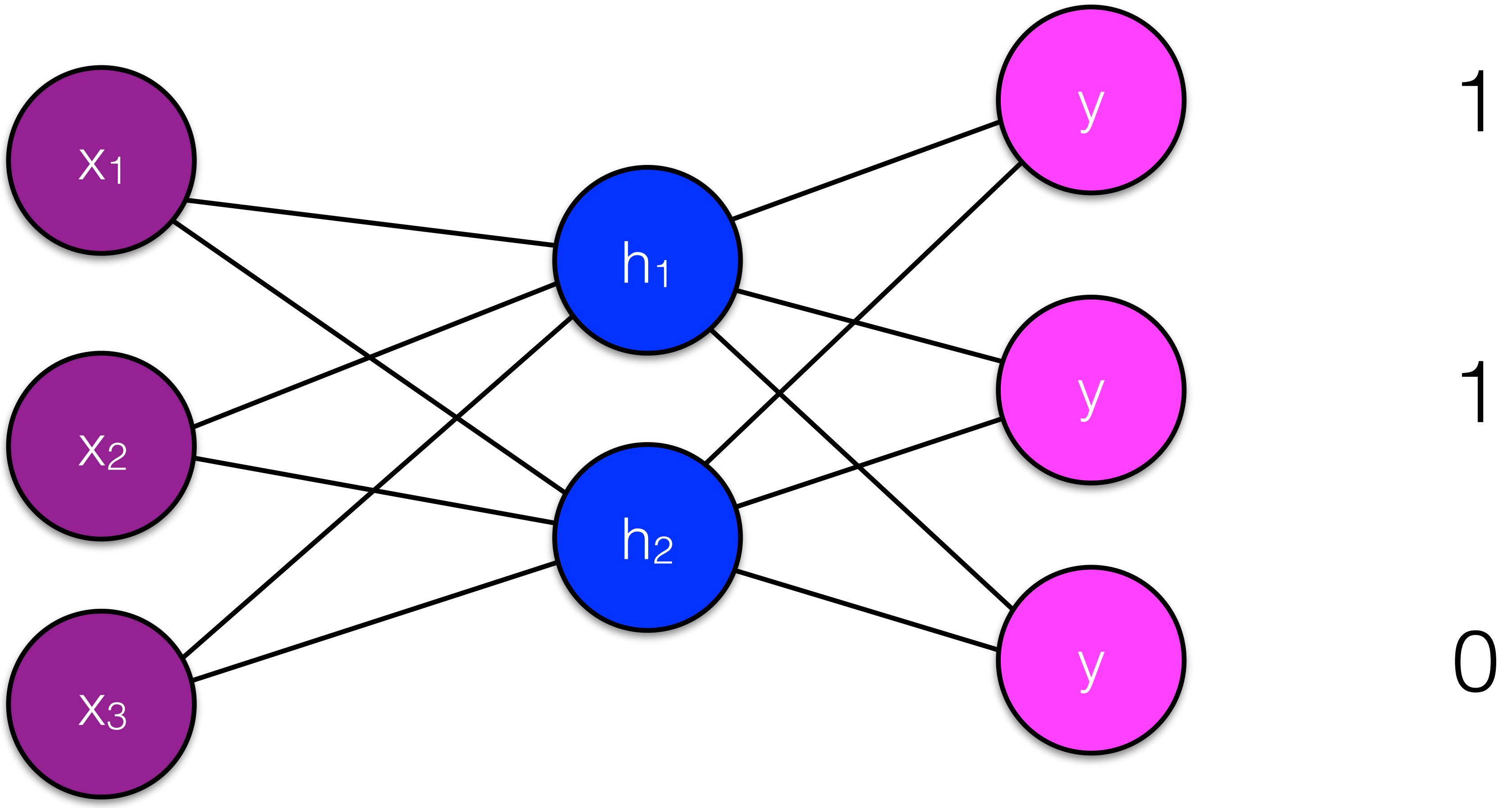
Output one real value

Neural network structures



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

Neural network structures



output 3 values, several = 1 in training data

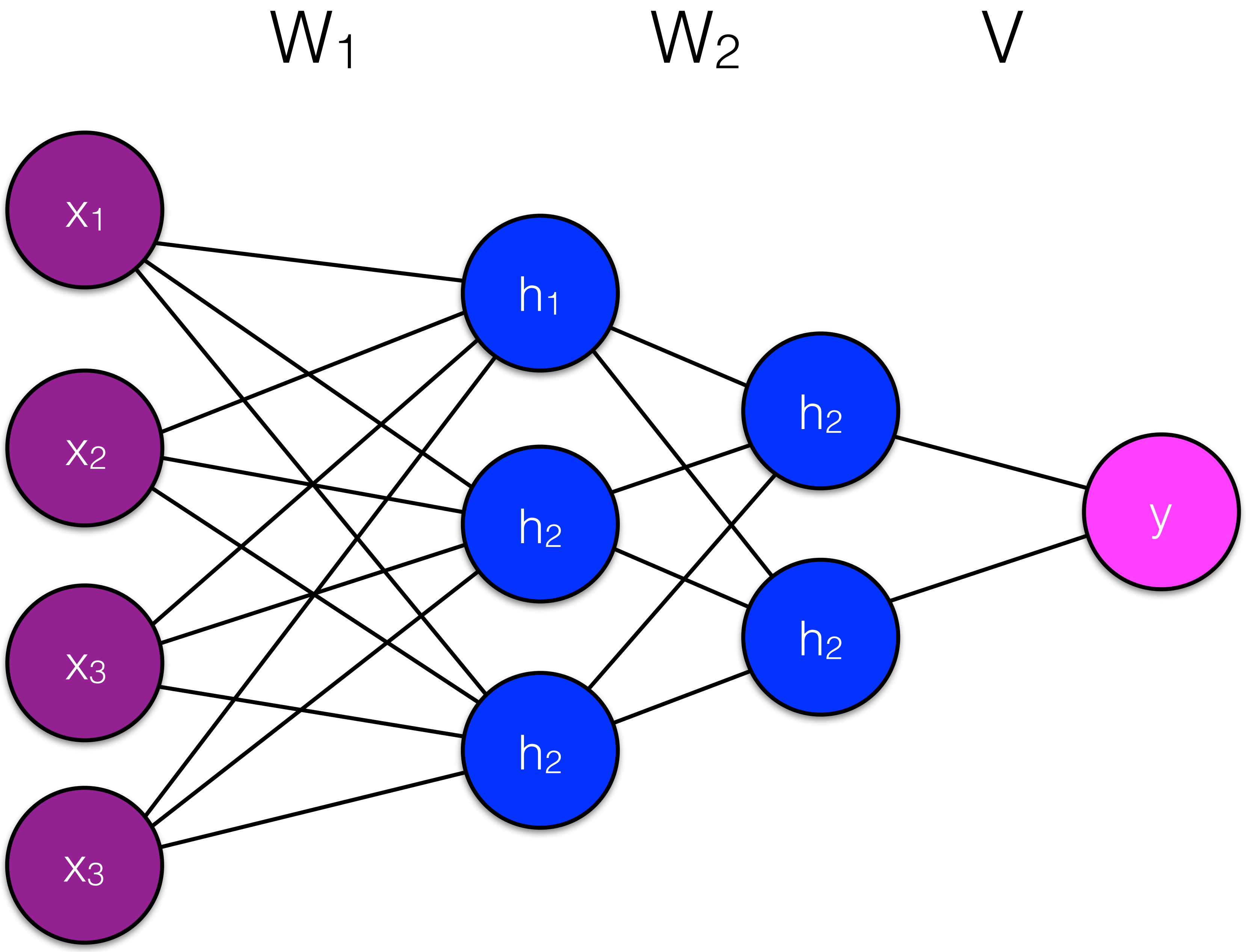
Regularization

- 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

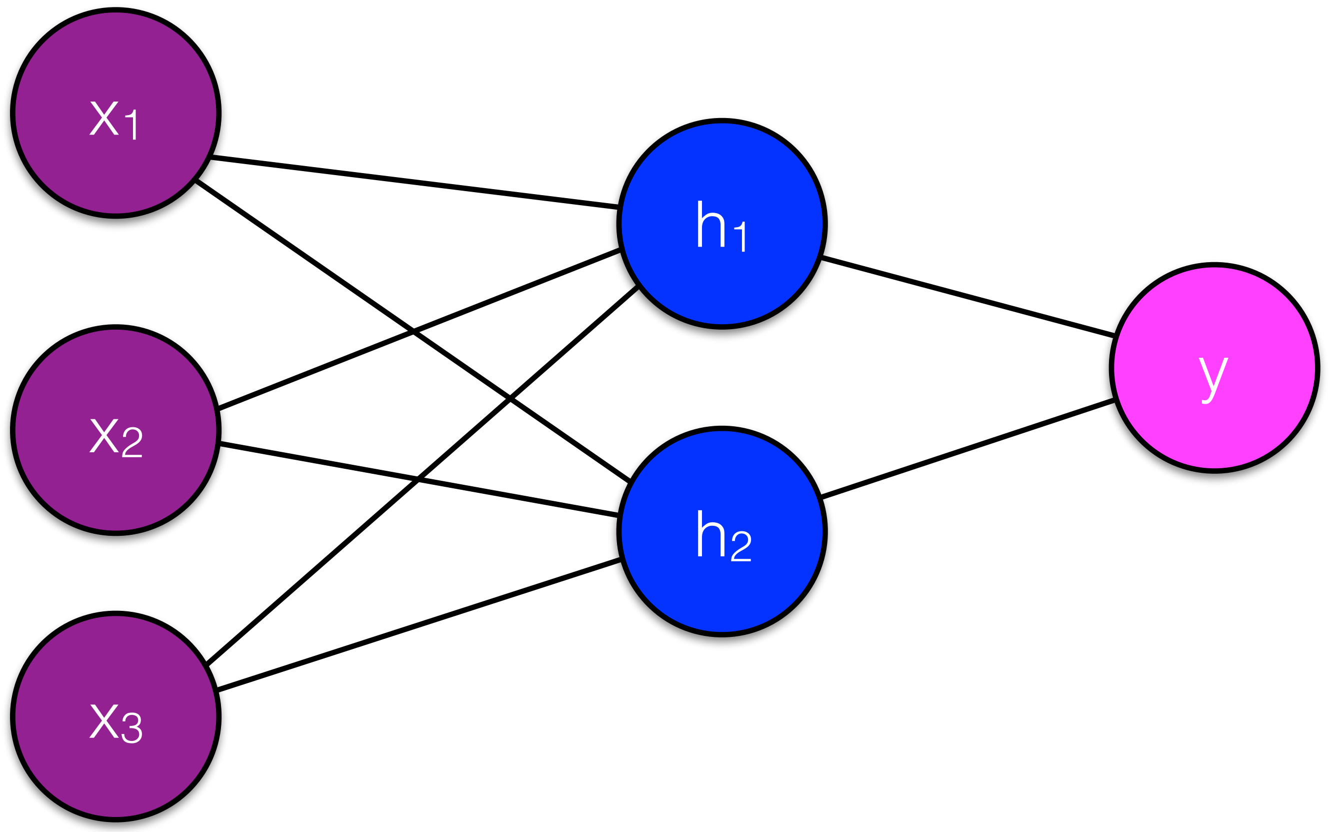


Keras

- We’ll be using keras to implement several neural architectures over the next few weeks
- Today: Sequential models

Sequential

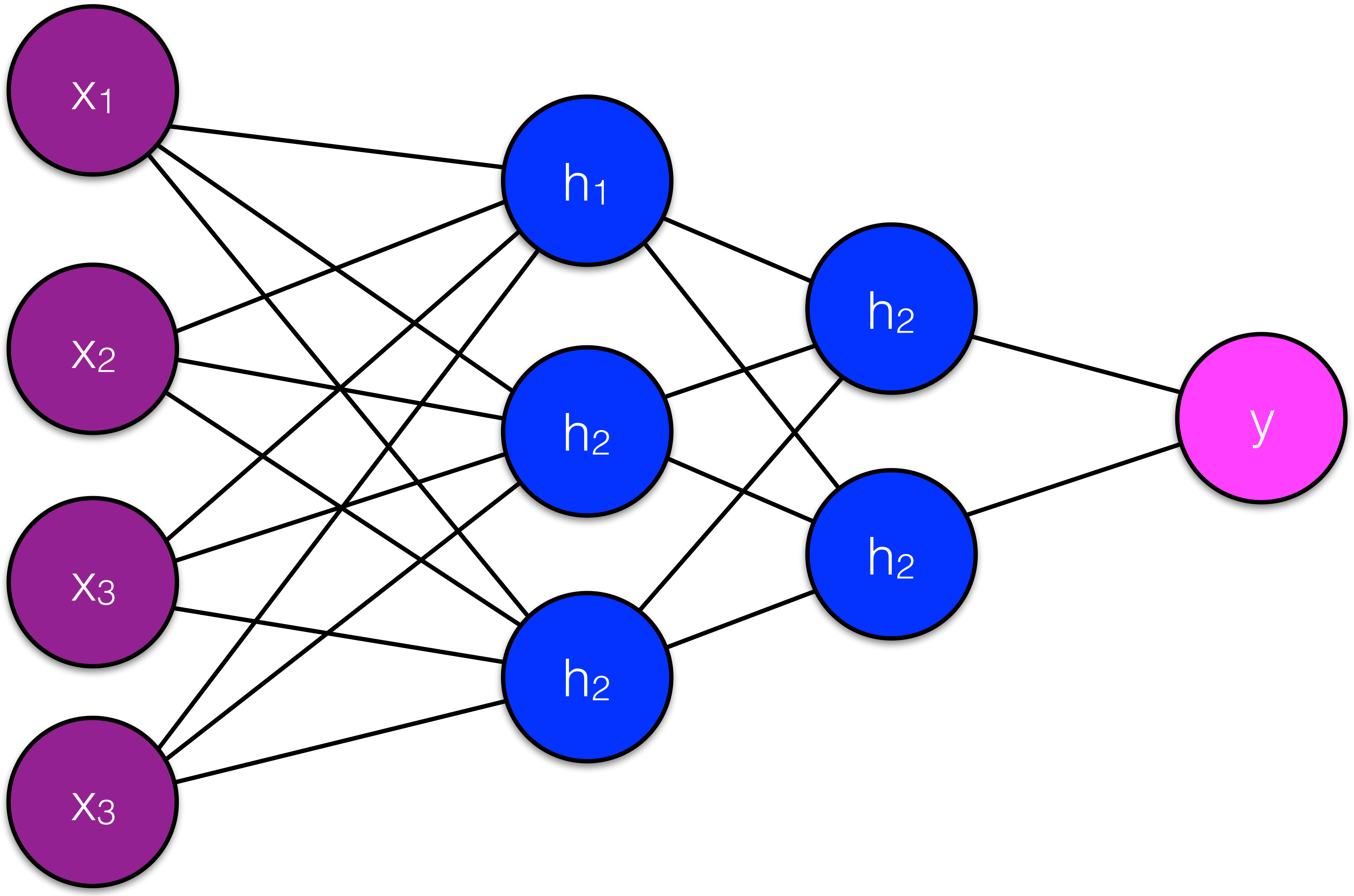
- Useful for models of limited complexity where the input to every layer is the output of the previous layer.



```
model=Sequential()

model.add(Dense(2,activation='relu',
input_shape=(3,)))

model.add(Dense(1,activation='sigmoid'))
```



```
model=Sequential()

model.add(Dense(3,activation='relu',
input_shape=(4,)))

model.add(Dense(2,activation='relu'))

model.add(Dense(1,activation='sigmoid'))
```