



# Natural Language Processing

Info 159/259

Lecture 4: Text classification 3 (Sept 4, 2018)

David Bamman, UC Berkeley

# Logistic regression

$$P(y = 1 \mid x, \beta) = \frac{1}{1 + \exp \left( - \sum_{i=1}^F x_i \beta_i \right)}$$

output space

$$\mathcal{Y} = \{0, 1\}$$

From last time

$X$  = feature vector

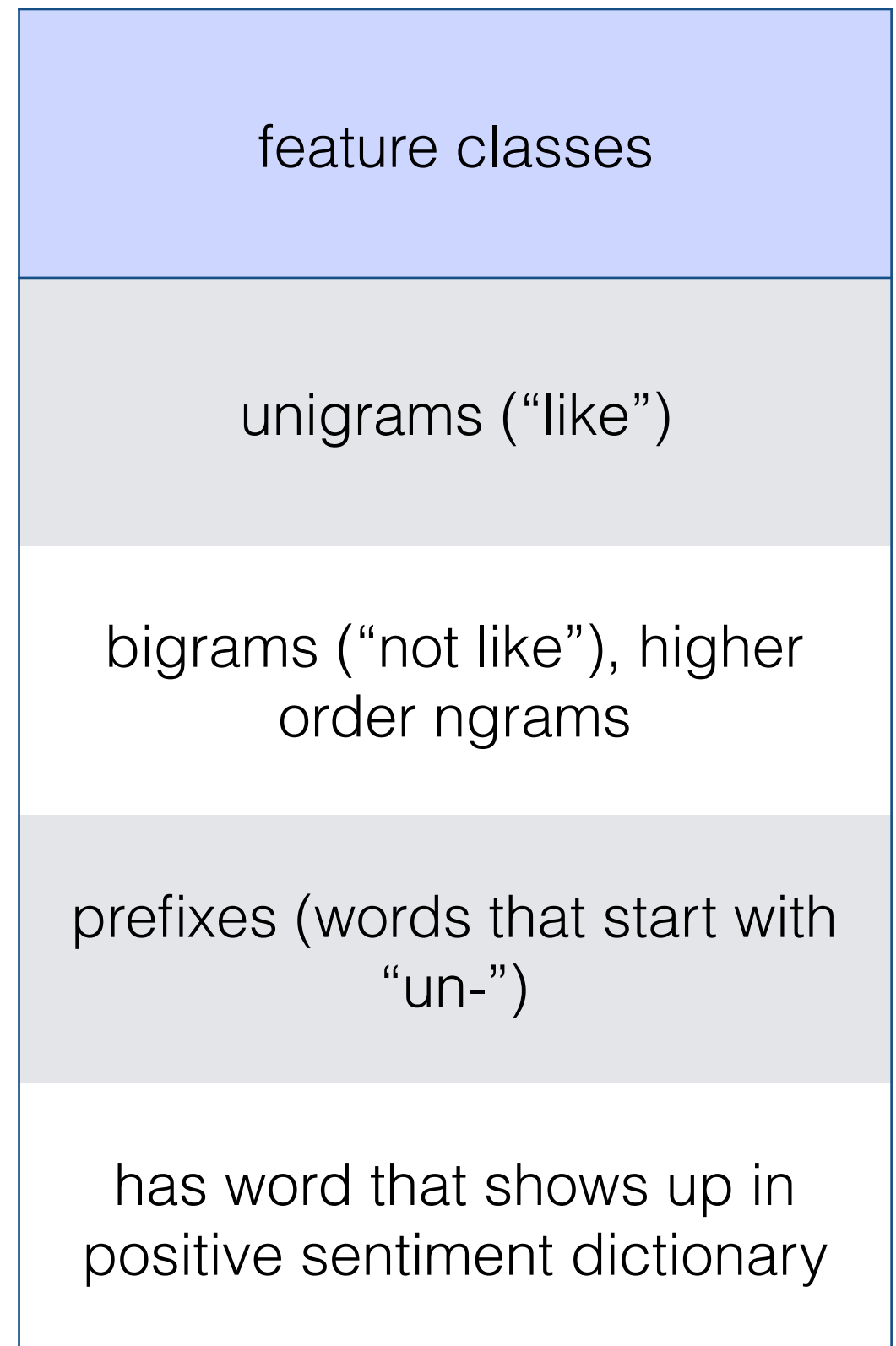
| Feature     | Value |
|-------------|-------|
| the         | 0     |
| and         | 0     |
| bravest     | 0     |
| love        | 0     |
| loved       | 0     |
| genius      | 0     |
| not         | 0     |
| fruit       | 1     |
| <i>BIAS</i> | 1     |

$\beta$  = coefficients

| Feature     | $\beta$ |
|-------------|---------|
| the         | 0.01    |
| and         | 0.03    |
| bravest     | 1.4     |
| love        | 3.1     |
| loved       | 1.2     |
| genius      | 0.5     |
| not         | -3.0    |
| fruit       | -0.8    |
| <i>BIAS</i> | -0.1    |

# Features

- Features are where you can encode your own **domain understanding** of the problem.



Many features that show up rarely may likely only appear (by chance) with one label

More generally, may appear so few times that the noise of randomness dominates

$\beta$  = coefficients

| Feature                                                | $\beta$ |
|--------------------------------------------------------|---------|
| like                                                   | 2.1     |
| did not like                                           | 1.4     |
| in_pos_dict_MPQA                                       | 1.7     |
| in_neg_dict_MPQA                                       | -2.1    |
| in_pos_dict_LIWC                                       | 1.4     |
| in_neg_dict_LIWC                                       | -3.1    |
| author=ebert                                           | -1.7    |
| author=ebert $\wedge$ dog<br>$\wedge$ starts with "in" | 30.1    |

# Feature selection

- We could threshold features by minimum count but that also throws away information
- We can take a probabilistic approach and encode a prior belief that all  $\beta$  should be 0 unless we have strong evidence otherwise

# L2 regularization

$$\ell(\beta) = \underbrace{\sum_{i=1}^N \log P(y_i | x_i, \beta)}_{\text{we want this to be high}} - \underbrace{\eta \sum_{j=1}^F \beta_j^2}_{\text{but we want this to be small}}$$

- We can do this by changing the function we're trying to optimize by adding a penalty for having values of  $\beta$  that are high
- This is equivalent to saying that each  $\beta$  element is drawn from a Normal distribution centered on 0.
- $\eta$  controls how much of a penalty to pay for coefficients that are far from 0 (optimize on development data)

## no L2 regularization

33.83 Won Bin

29.91 Alexander Beyer

24.78 Bloopers

23.01 Daniel Brühl

22.11 Ha Jeong-woo

20.49 Supernatural

18.91 Kristine DeBell

18.61 Eddie Murphy

18.33 Cher

18.18 Michael Douglas

## some L2 regularization

2.17 Eddie Murphy

1.98 Tom Cruise

1.70 Tyler Perry

1.70 Michael Douglas

1.66 Robert Redford

1.66 Julia Roberts

1.64 Dance

1.63 Schwarzenegger

1.63 Lee Tergesen

1.62 Cher

## high L2 regularization

0.41 Family Film

0.41 Thriller

0.36 Fantasy

0.32 Action

0.25 Buddy film

0.24 Adventure

0.20 Comp Animation

0.19 Animation

0.18 Science Fiction

0.18 Bruce Willis

# L1 regularization

$$\ell(\beta) = \underbrace{\sum_{i=1}^N \log P(y_i | x_i, \beta)}_{\text{we want this to be high}} - \underbrace{\eta \sum_{j=1}^F |\beta_j|}_{\text{but we want this to be small}}$$

- L1 regularization encourages coefficients to be **exactly** 0.
- $\eta$  again controls how much of a penalty to pay for coefficients that are far from 0 (optimize on development data)

# What do the coefficients mean?

$$P(y \mid x, \beta) = \frac{\exp(x_0\beta_0 + x_1\beta_1)}{1 + \exp(x_0\beta_0 + x_1\beta_1)}$$

$$P(y \mid x, \beta)(1 + \exp(x_0\beta_0 + x_1\beta_1)) = \exp(x_0\beta_0 + x_1\beta_1)$$

$$P(y \mid x, \beta) + P(y \mid x, \beta) \exp(x_0\beta_0 + x_1\beta_1) = \exp(x_0\beta_0 + x_1\beta_1)$$

$$P(y | x, \beta) + P(y | x, \beta) \exp(x_0 \beta_0 + x_1 \beta_1) = \exp(x_0 \beta_0 + x_1 \beta_1)$$

$$P(y | x, \beta) = \exp(x_0 \beta_0 + x_1 \beta_1) - P(y | x, \beta) \exp(x_0 \beta_0 + x_1 \beta_1)$$

$$P(y | x, \beta) = \exp(x_0 \beta_0 + x_1 \beta_1)(1 - P(y | x, \beta))$$

This is the odds of y  
occurring

$$\frac{P(y | x, \beta)}{1 - P(y | x, \beta)} = \exp(x_0 \beta_0 + x_1 \beta_1)$$

# Odds

- Ratio of an event occurring to its not taking place

$$\frac{P(x)}{1 - P(x)}$$

Green Bay Packers  
vs. SF 49ers

$$\frac{0.75}{0.25} = \frac{3}{1} = 3 : 1$$

probability of  
GB winning

odds for GB  
winning

$$P(y | x, \beta) + P(y | x, \beta) \exp(x_0 \beta_0 + x_1 \beta_1) = \exp(x_0 \beta_0 + x_1 \beta_1)$$

$$P(y | x, \beta) = \exp(x_0 \beta_0 + x_1 \beta_1) - P(y | x, \beta) \exp(x_0 \beta_0 + x_1 \beta_1)$$

$$P(y | x, \beta) = \exp(x_0 \beta_0 + x_1 \beta_1)(1 - P(y | x, \beta))$$

This is the odds of y occurring

$$\frac{P(y | x, \beta)}{1 - P(y | x, \beta)} = \exp(x_0 \beta_0 + x_1 \beta_1)$$

$$\frac{P(y | x, \beta)}{1 - P(y | x, \beta)} = \exp(x_0 \beta_0) \exp(x_1 \beta_1)$$

$$\frac{P(y \mid x, \beta)}{1 - P(y \mid x, \beta)} = \exp(x_0 \beta_0) \exp(x_1 \beta_1)$$

Let's increase the value of x by 1 (e.g., from 0 → 1)

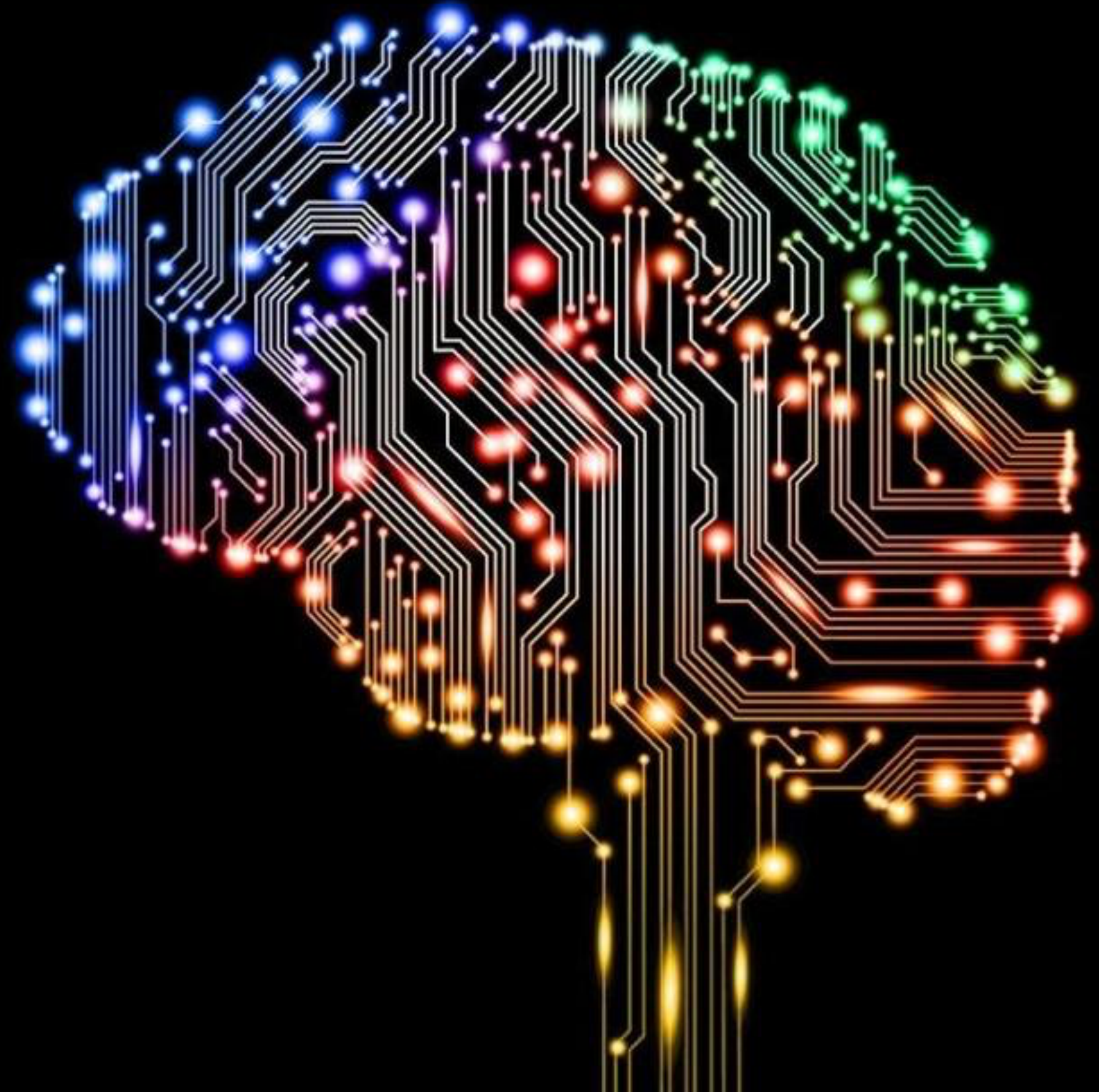
$$\exp(x_0 \beta_0) \exp((x_1 + 1) \beta_1)$$

$$\exp(x_0 \beta_0) \exp(x_1 \beta_1 + \beta_1)$$

$$\exp(x_0 \beta_0) \exp(x_1 \beta_1) \exp(\beta_1)$$

$\exp(\beta)$  represents the factor by which the **odds** change with a 1-unit increase in x

$$\frac{P(y \mid x, \beta)}{1 - P(y \mid x, \beta)} \exp(\beta_1)$$



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

# 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; Iyer 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, Sutskever et al. 2014]
- Dialogue agents [Sordani 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



theano



PYTORCH

py/net

# Logistic regression

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

|              | x | $\beta$ |
|--------------|---|---------|
| <i>not</i>   | 1 | -0.5    |
| <i>bad</i>   | 1 | -1.7    |
| <i>movie</i> | 0 | 0.3     |

# SGD

---

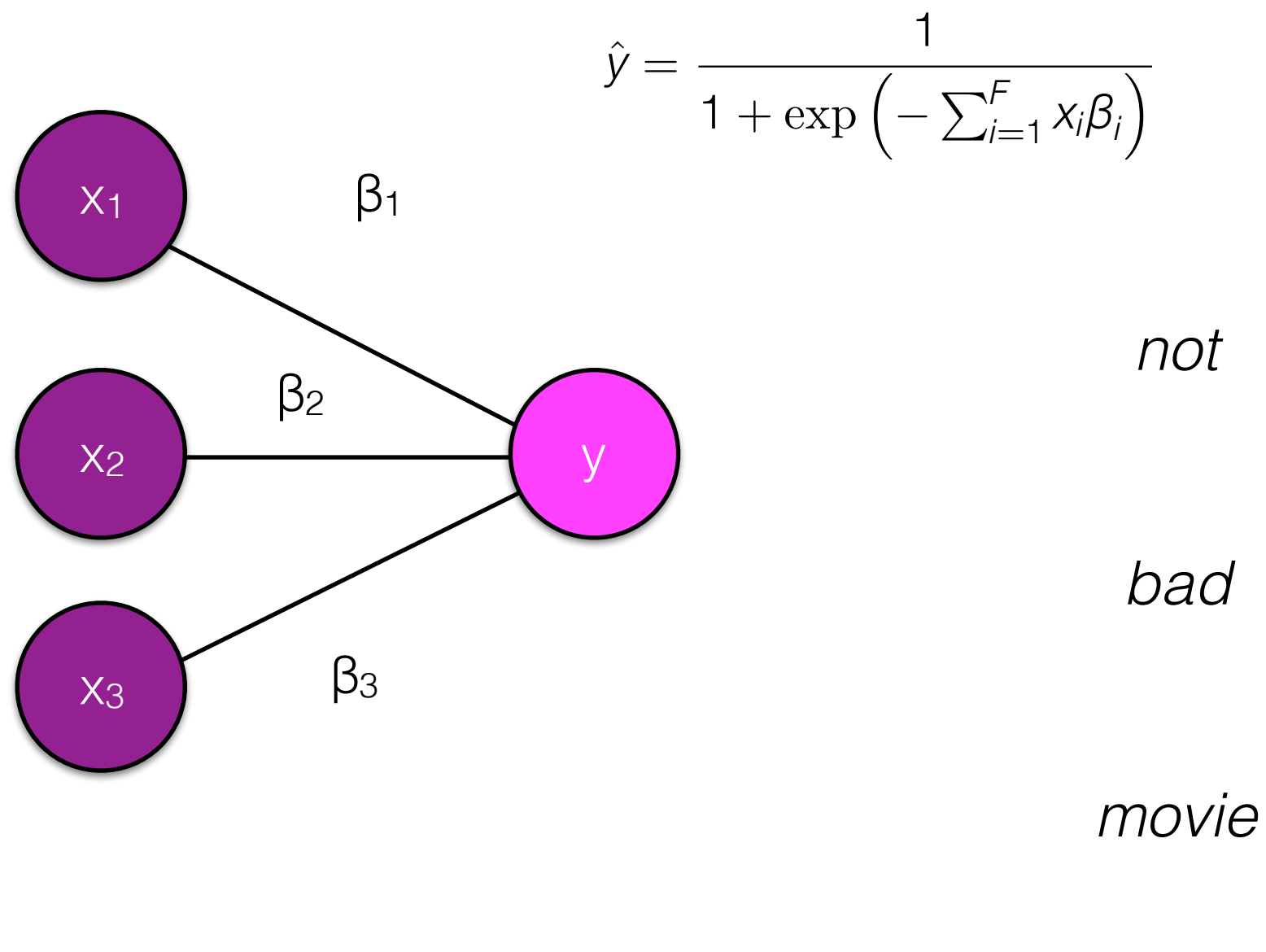
**Algorithm 1** Logistic regression gradient descent

---

- 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**
- 

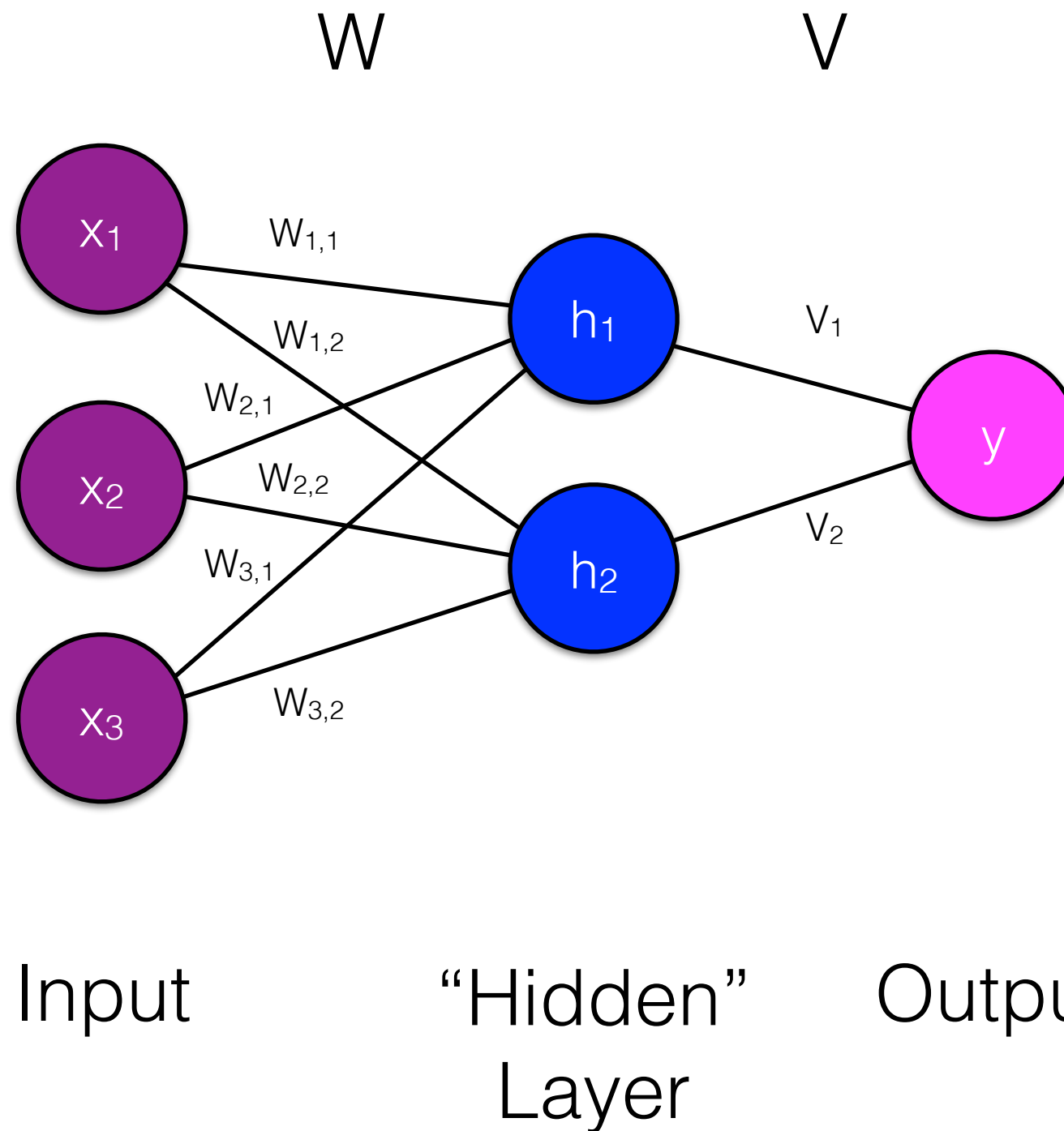
Calculate the derivative of some loss function with respect to parameters we can change, update accordingly to make predictions on training data **a little less wrong next time.**

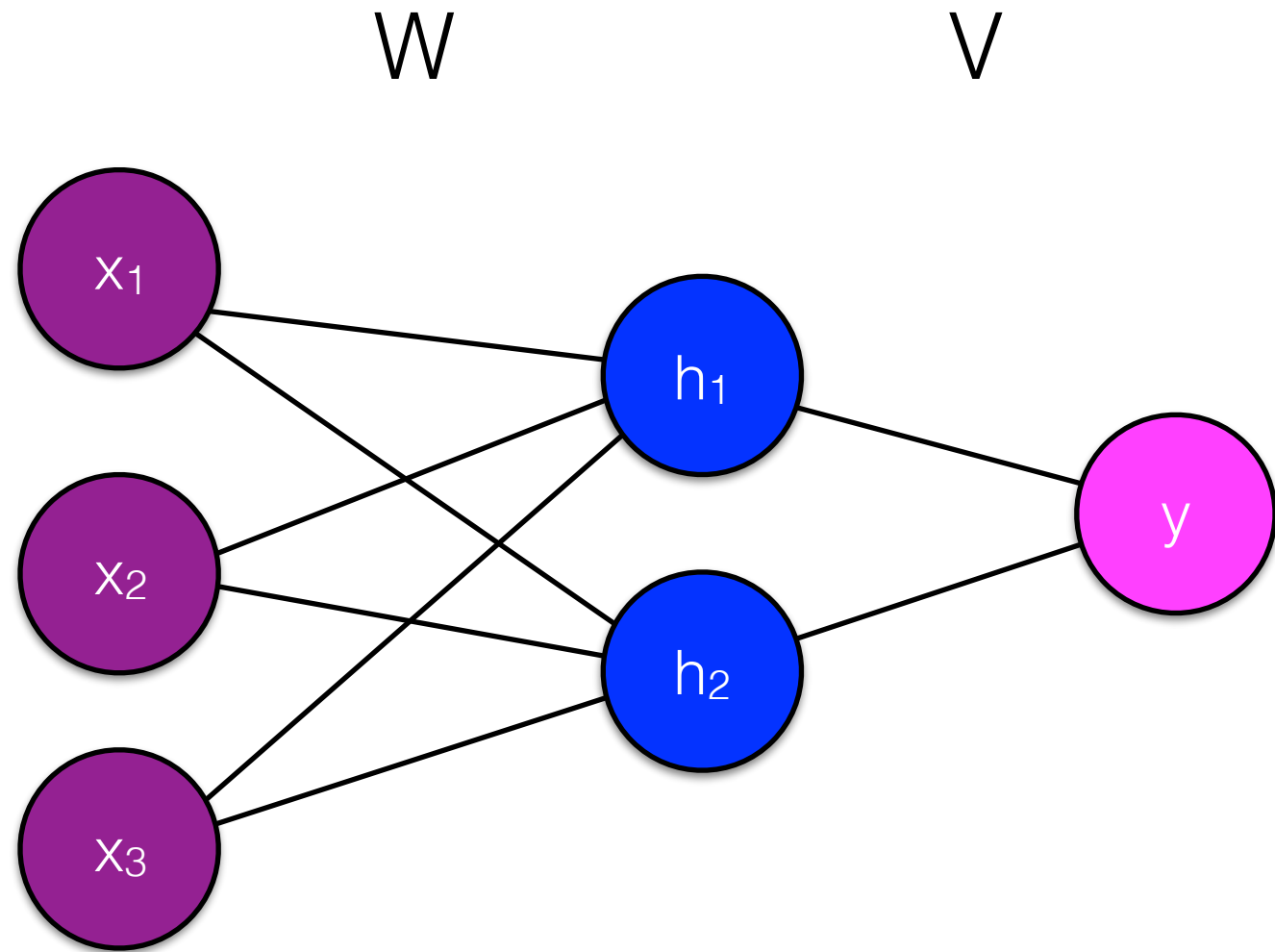
# Logistic regression



| x | $\beta$ |
|---|---------|
| 1 | -0.5    |
| 1 | -1.7    |
| 0 | 0.3     |

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





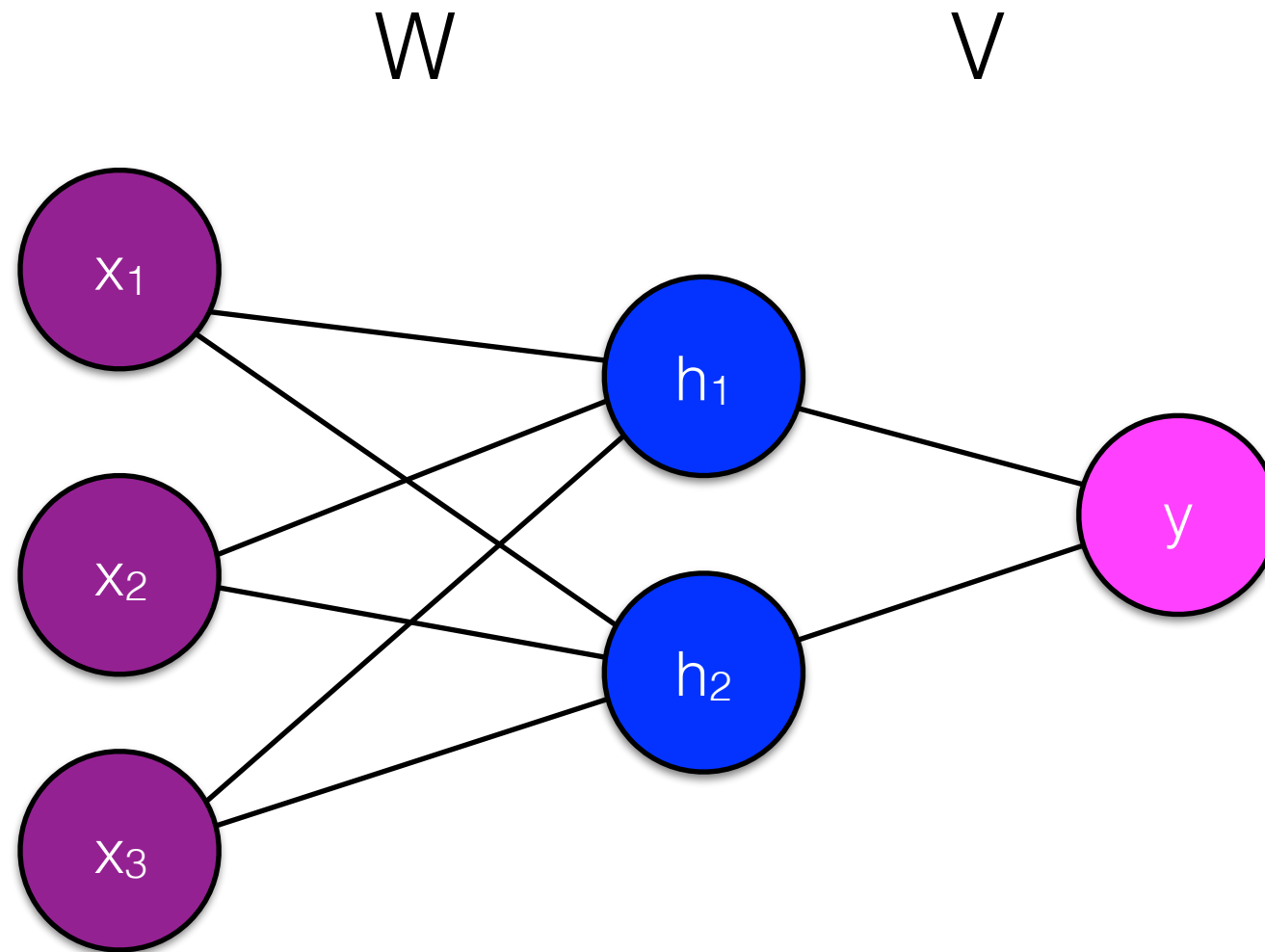
*not*  
*bad*  
*movie*

| x |
|---|
| 1 |
| 1 |
| 0 |

| W    |      |
|------|------|
| -0.5 | 1.3  |
| 0.4  | 0.08 |
| 1.7  | 3.1  |

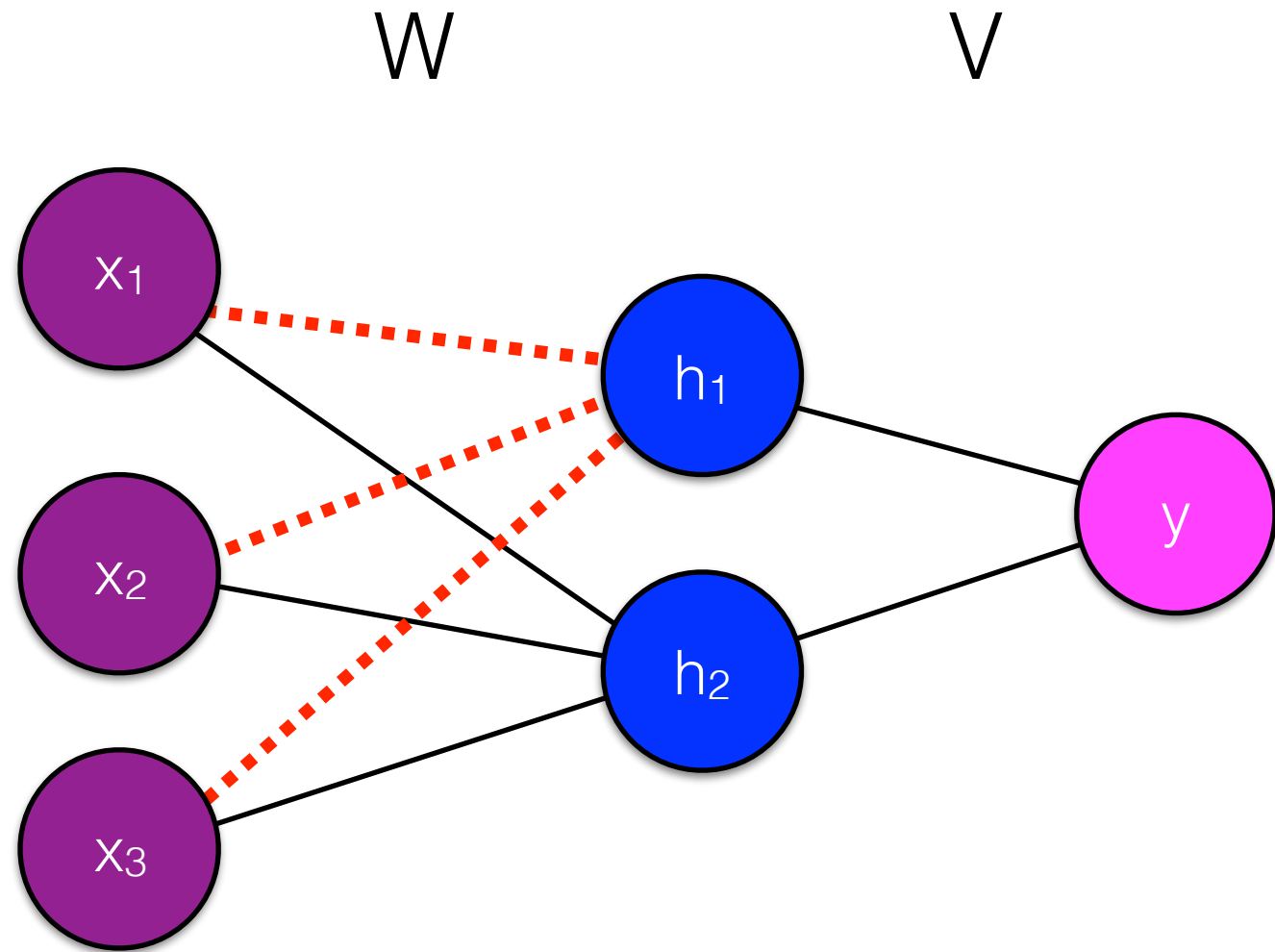
| V    |
|------|
| 4.1  |
| -0.9 |

| y |
|---|
| 1 |



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

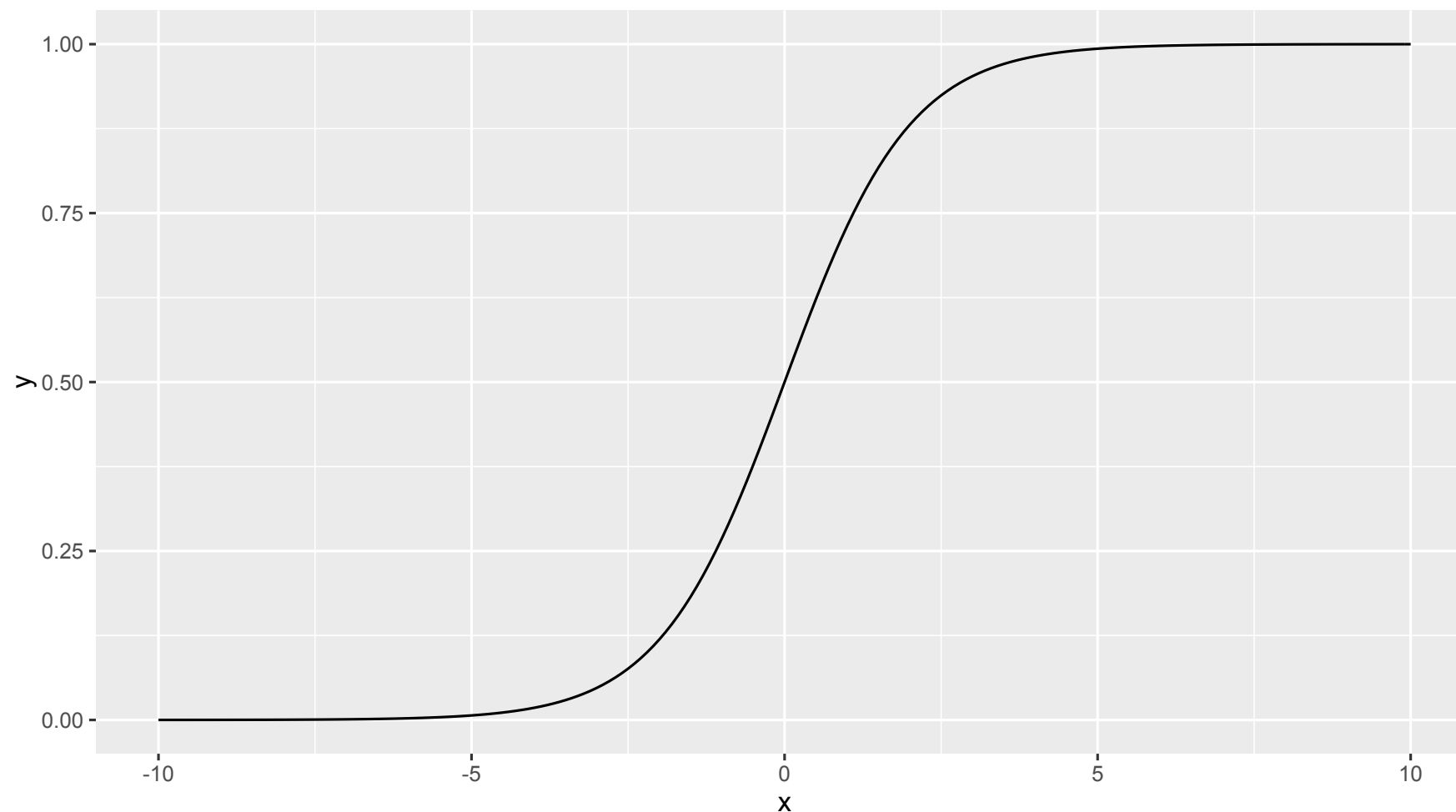
the hidden nodes are  
completely determined by the  
input and weights



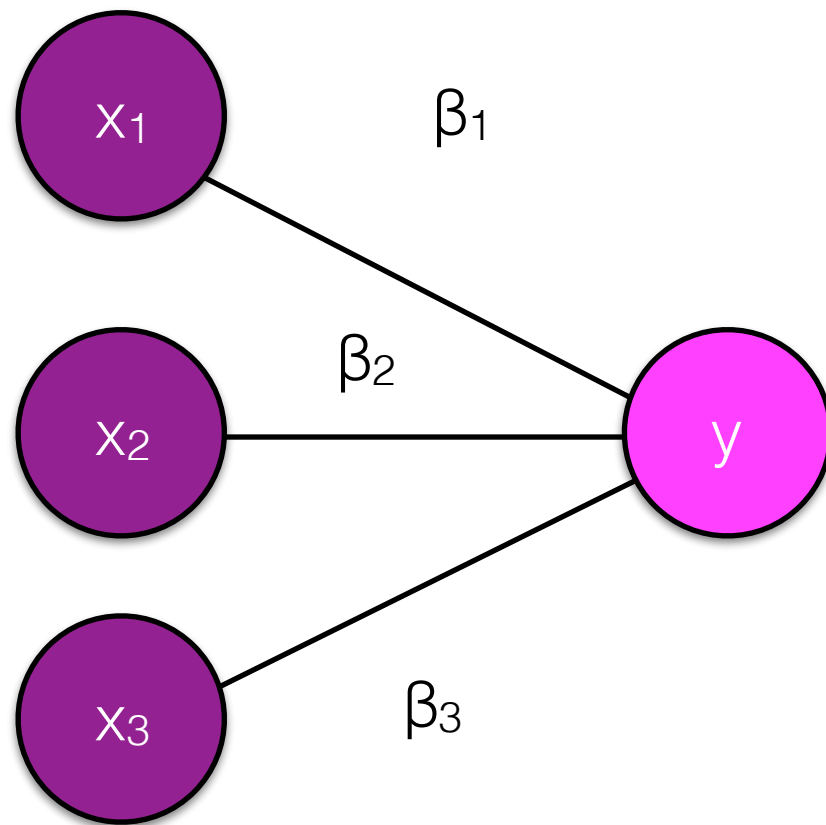
$$h_1 = f \left( \sum_{i=1}^F x_i W_{i,1} \right)$$

# Activation functions

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



# Logistic regression



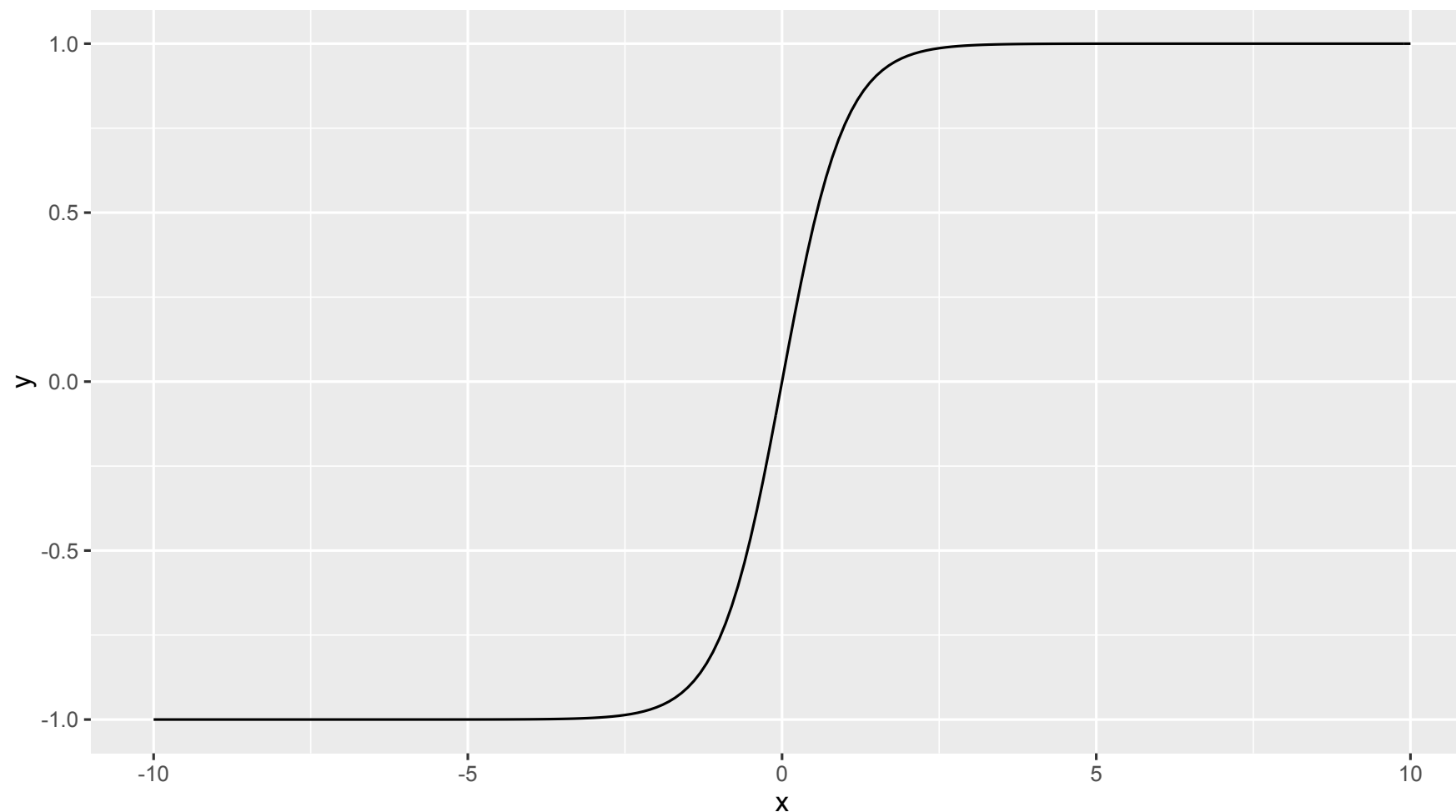
$$\hat{y} = \frac{1}{1 + \exp \left( - \sum_{i=1}^F x_i \beta_i \right)}$$

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

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

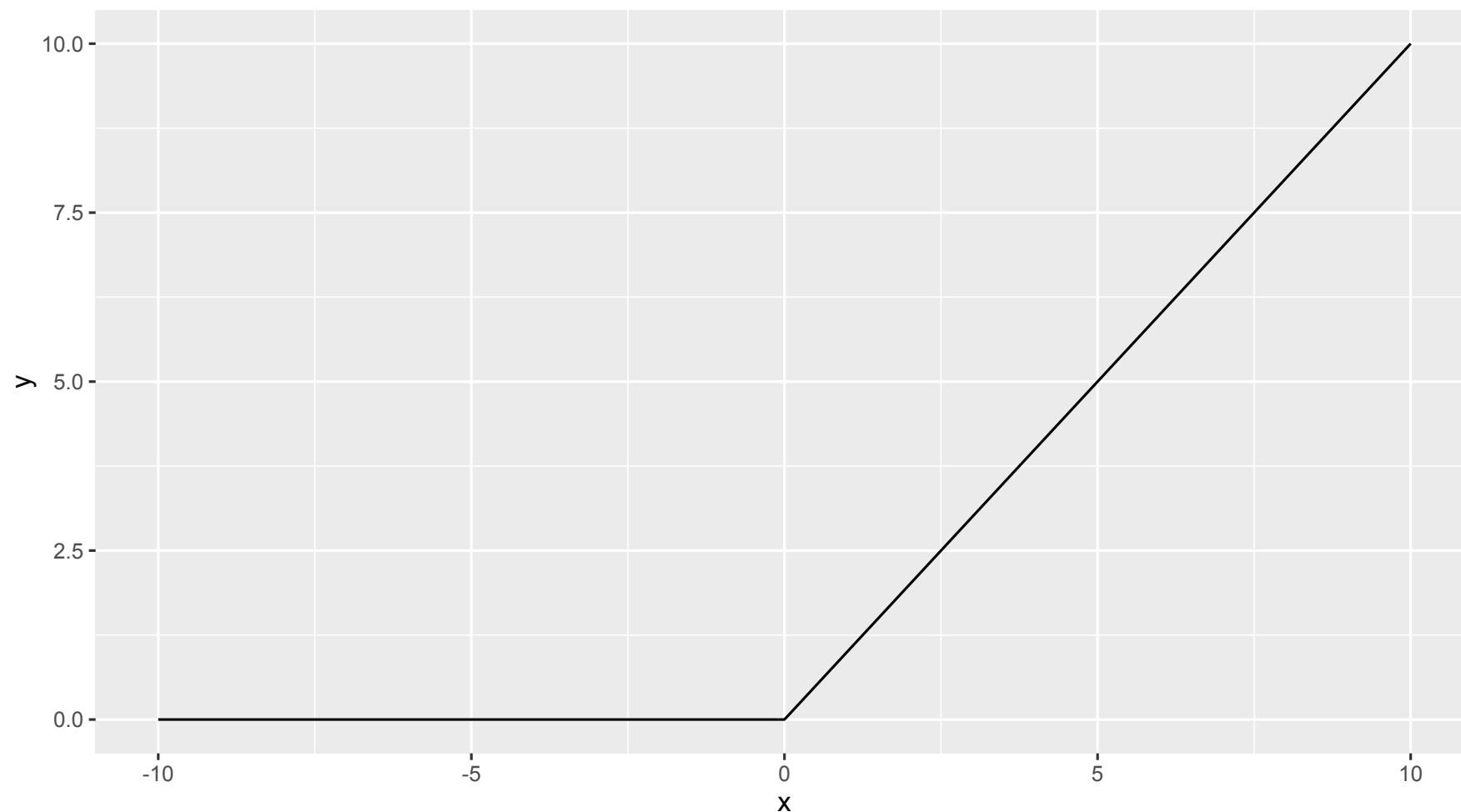
# Activation functions

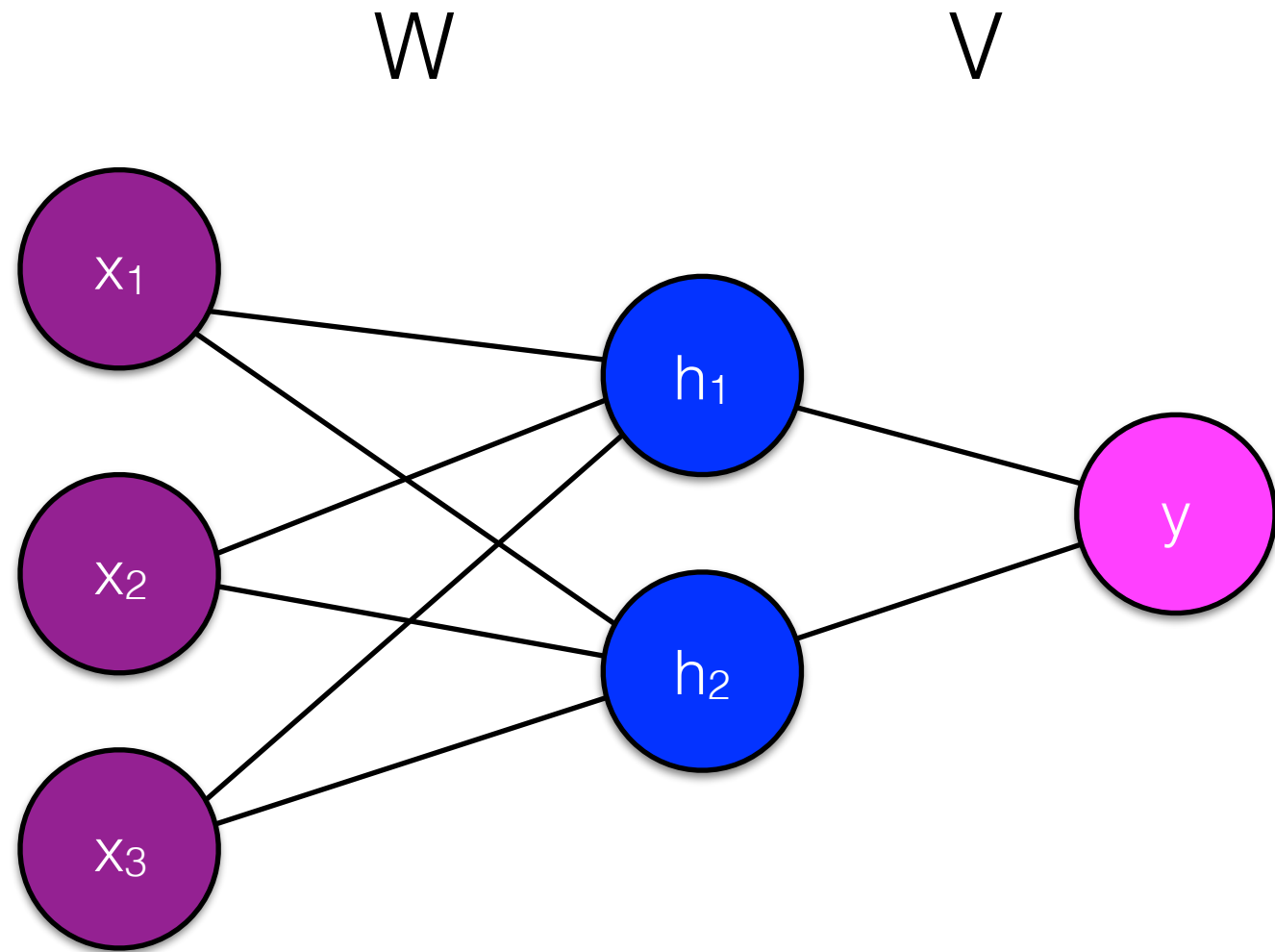
$$\tanh(z) = \frac{\exp(z) - \exp(-z)}{\exp(z) + \exp(-z)}$$



# Activation functions

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

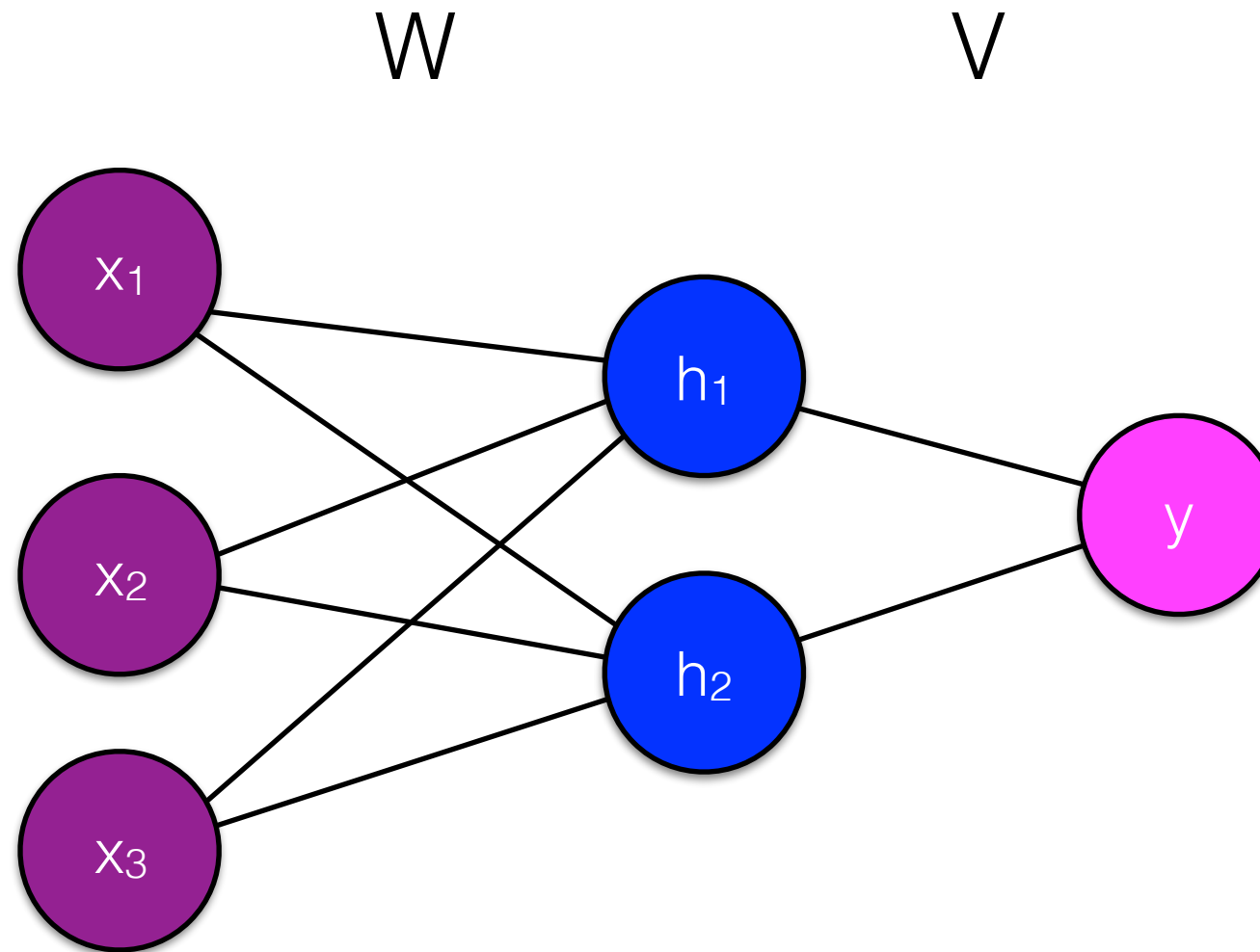




$$h_1 = \sigma \left( \sum_{i=1}^F x_i W_{i,1} \right)$$

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

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



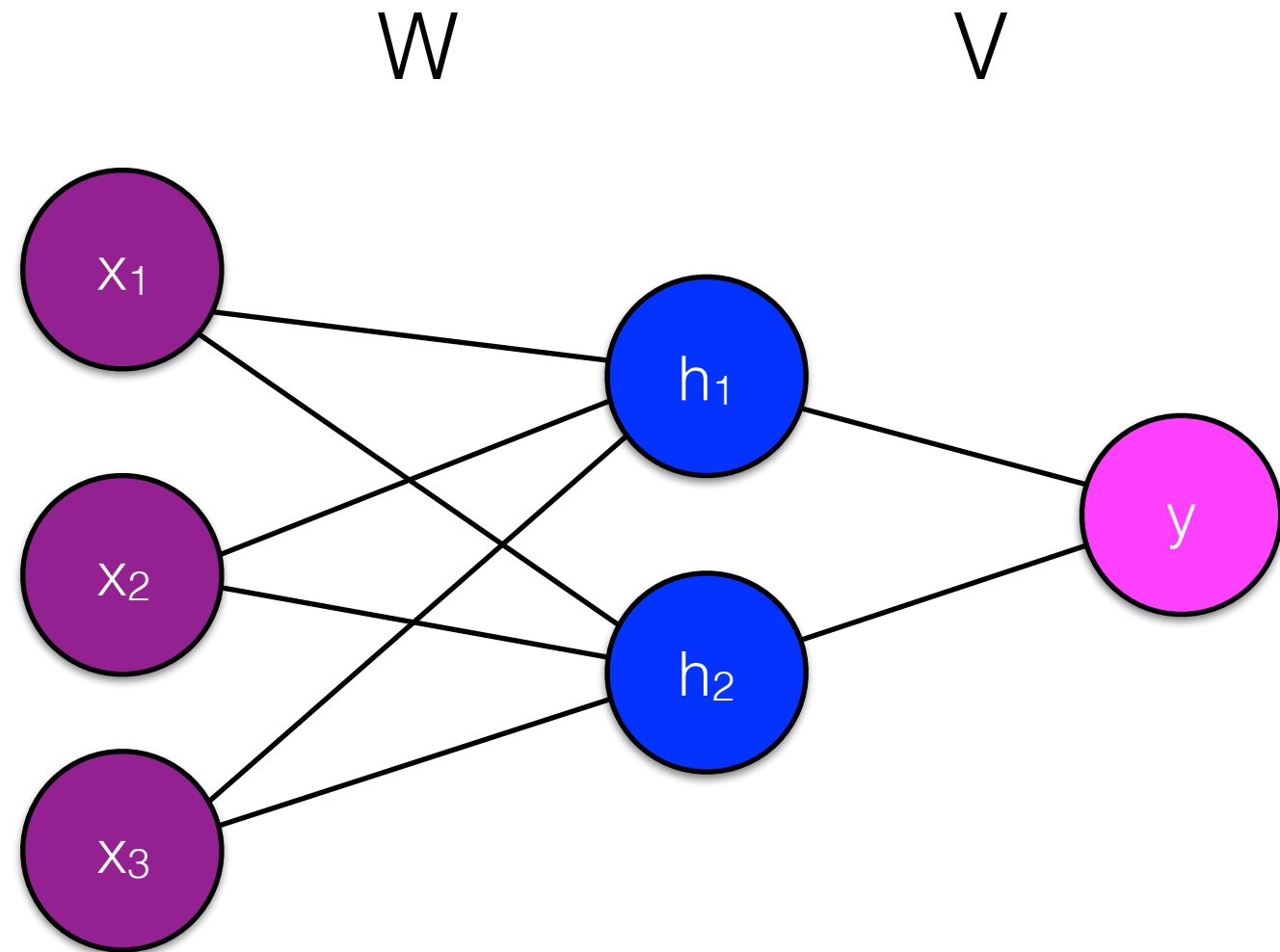
$$\hat{y} = \sigma \left[ V_1 \left( \sigma \left( \sum_i^F x_i W_{i,1} \right) \right) + V_2 \left( \sigma \left( \sum_i^F x_i W_{i,2} \right) \right) \right]$$

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  $\langle x, y \rangle$  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

$$xW \rightarrow \sigma(xW) \rightarrow \sigma(xW)V \rightarrow \sigma(\sigma(xW)V)$$

The loss is another function chained on top

$$\log(\sigma(\sigma(xW)V))$$

# Chain rule

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

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

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

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

# Chain rule

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

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

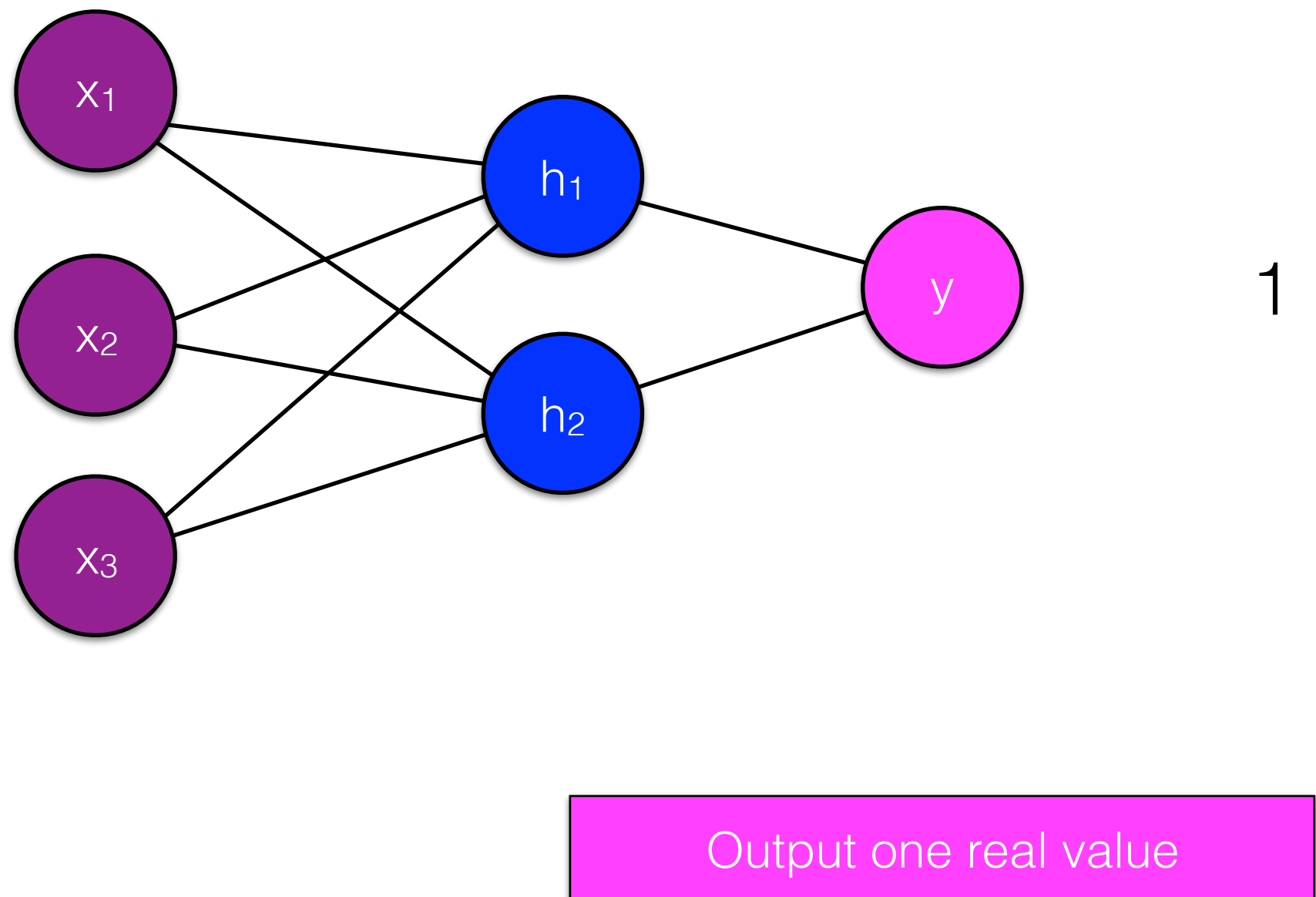
$$= (1 - \sigma(hV))h$$

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

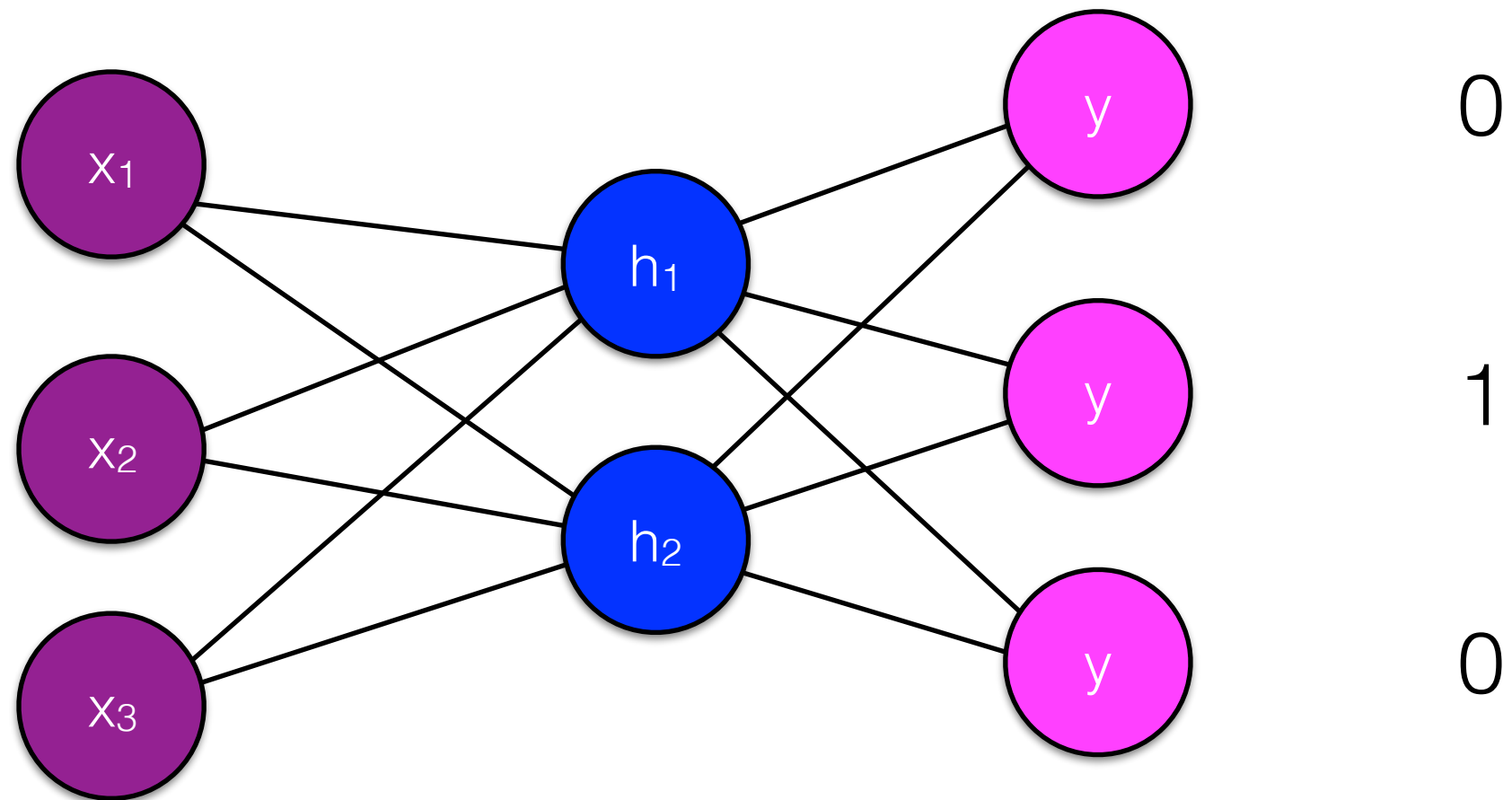
# Neural networks

- 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

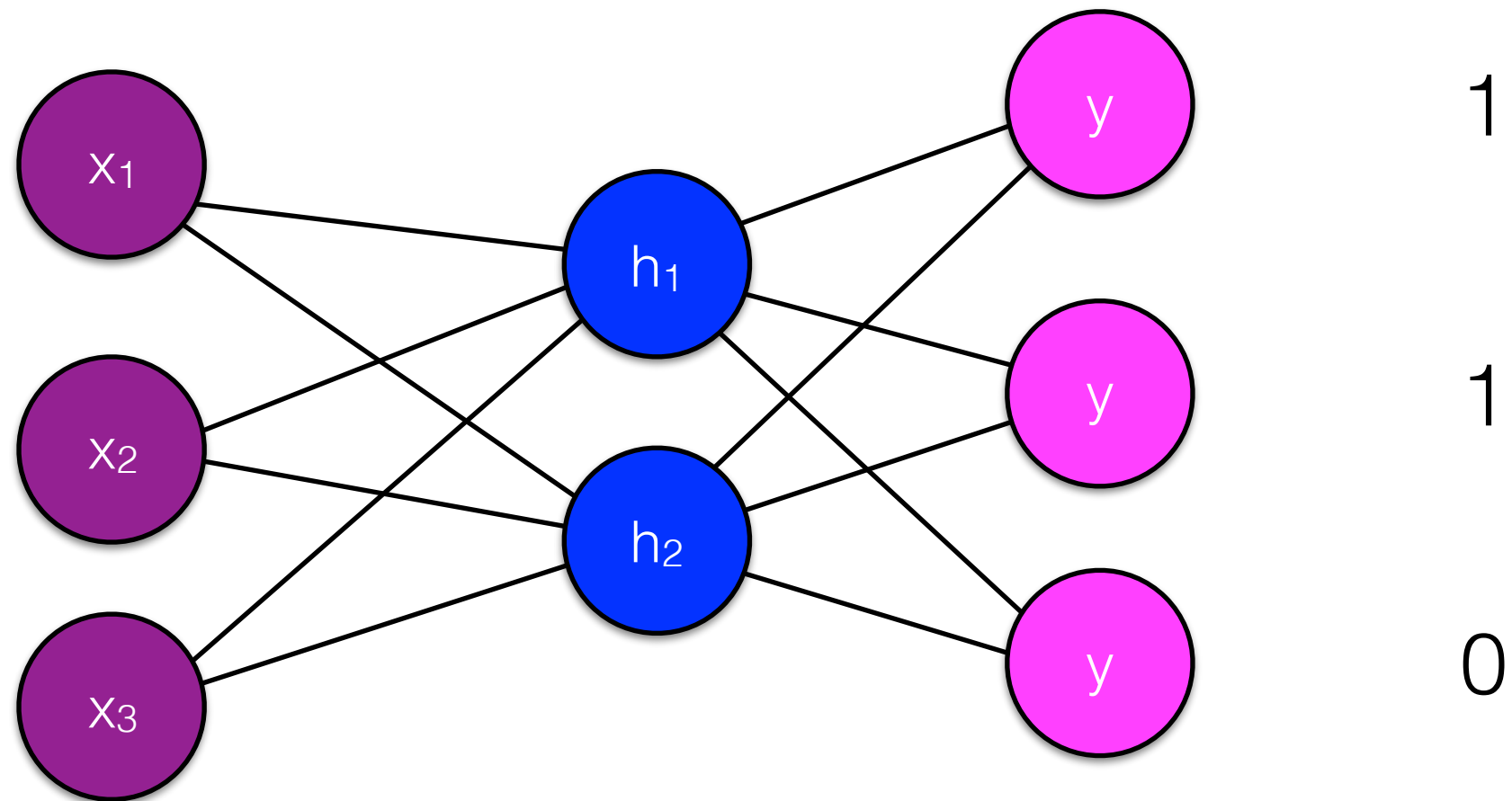


# Neural network structures



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

# Neural network structures



output 3 values, several = 1 in training data

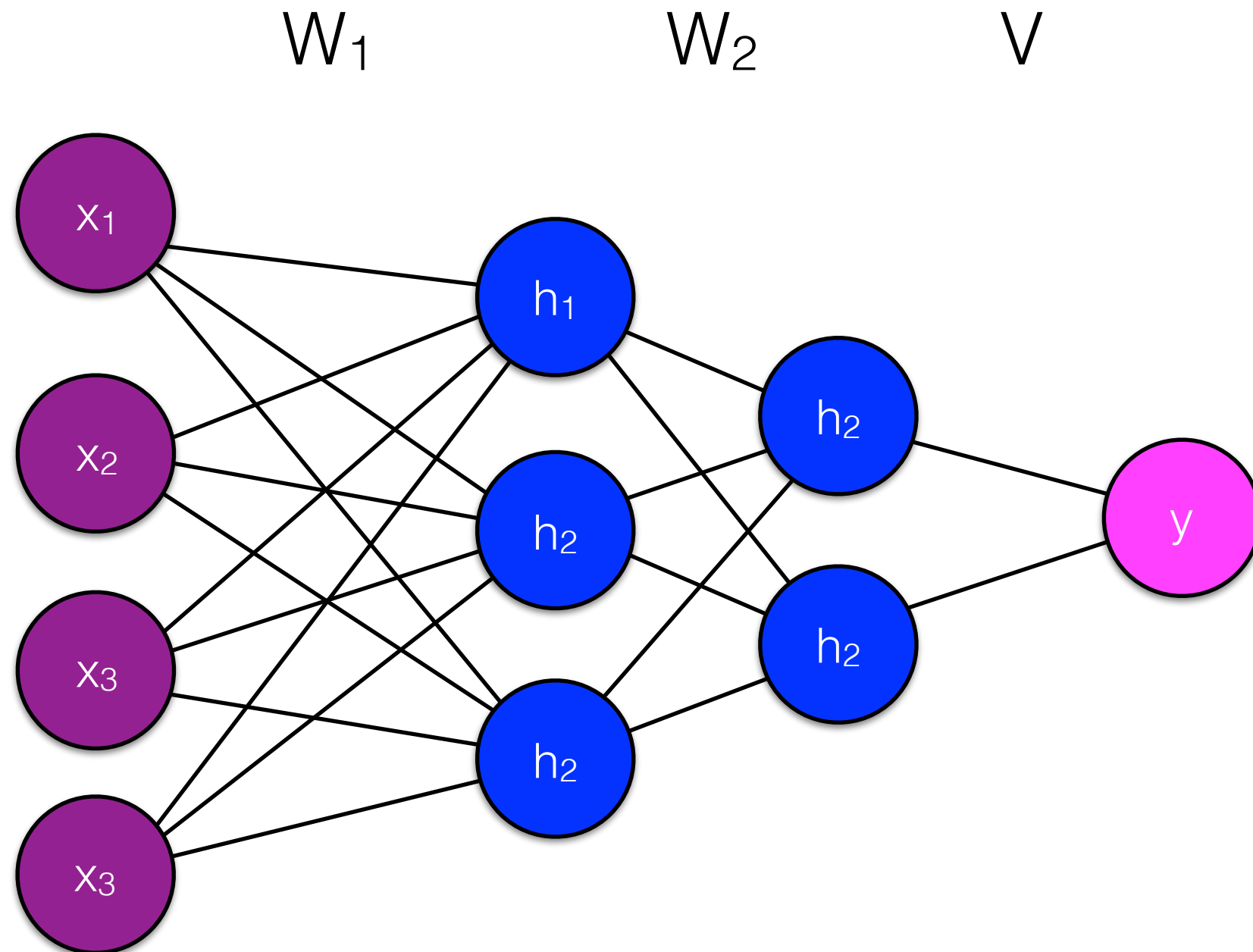
# Regularization

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

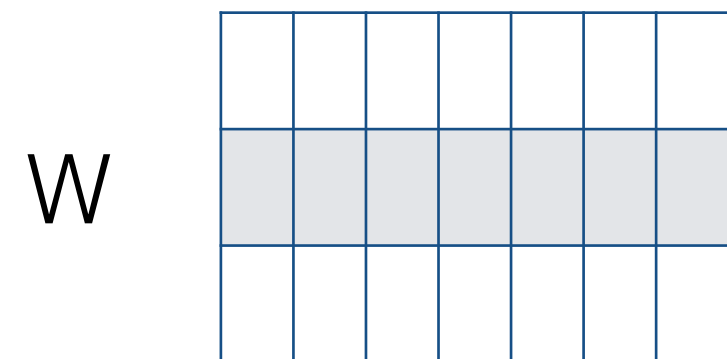
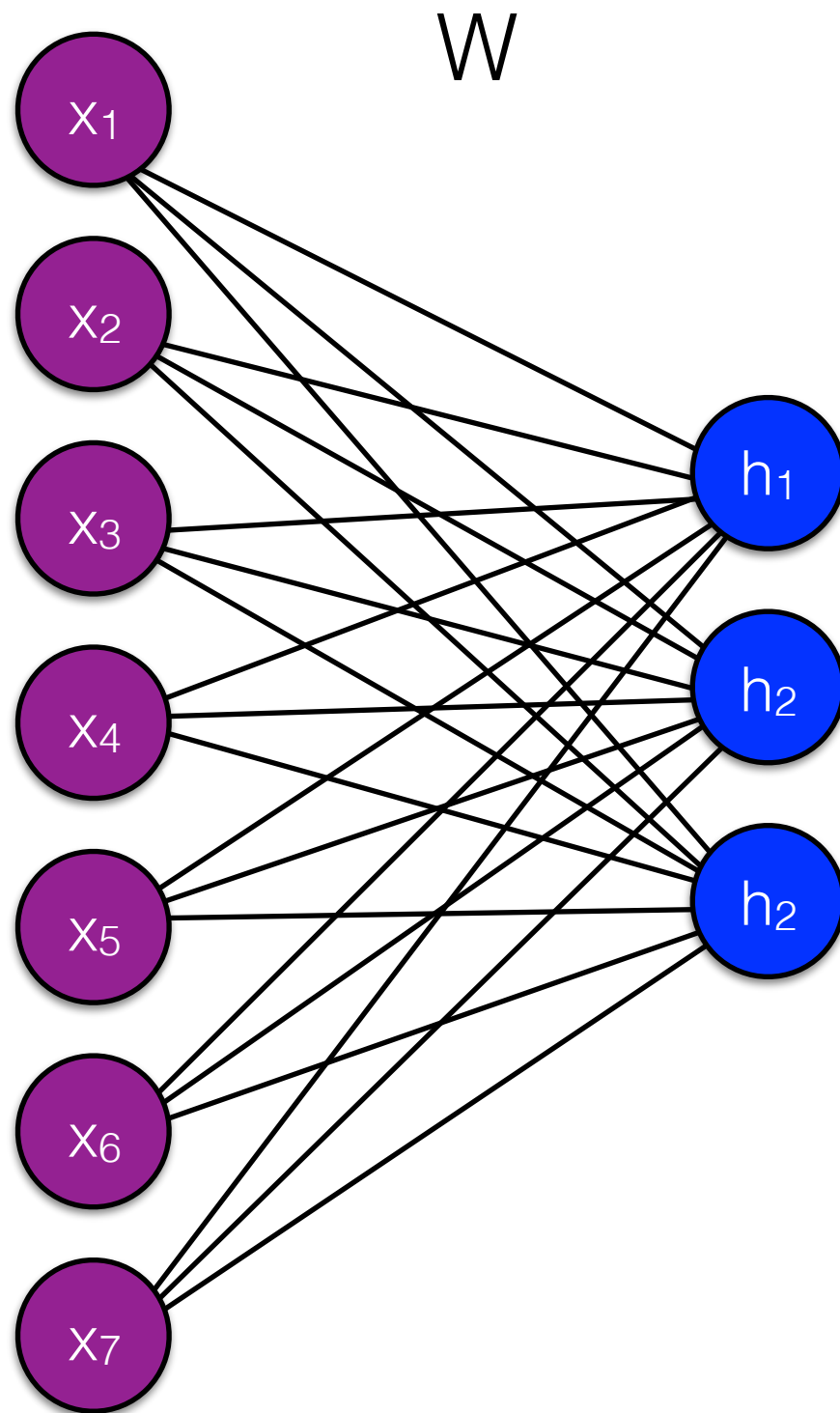
# Regularization

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

# Deeper networks



# Densely connected layer



$$h = \sigma(xW)$$

# Convolutional networks

- With convolution networks, the **same** operation is (i.e., the same set of parameters) is applied to **different** regions of the input

# 2D Convolution

|   |   |   |   |   |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 |

blurring



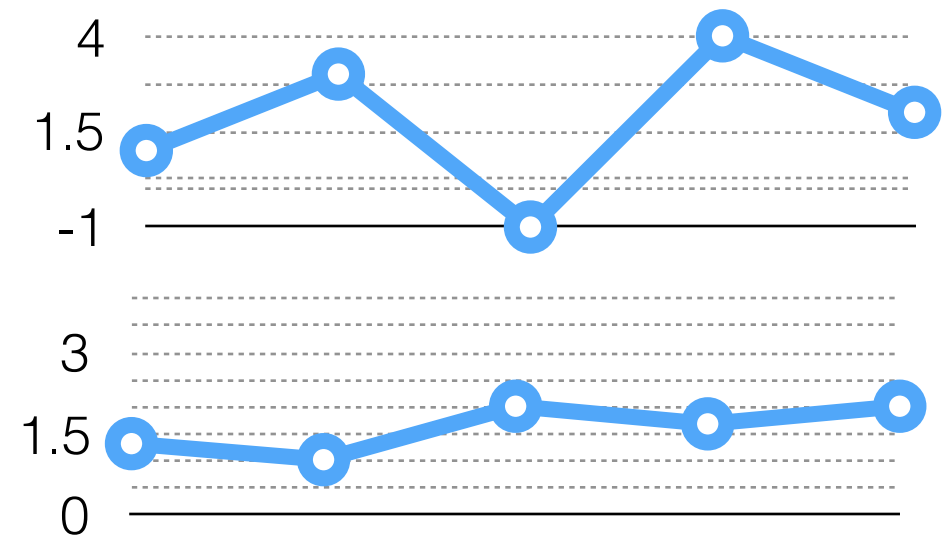
# 1D Convolution

convolution  $K$

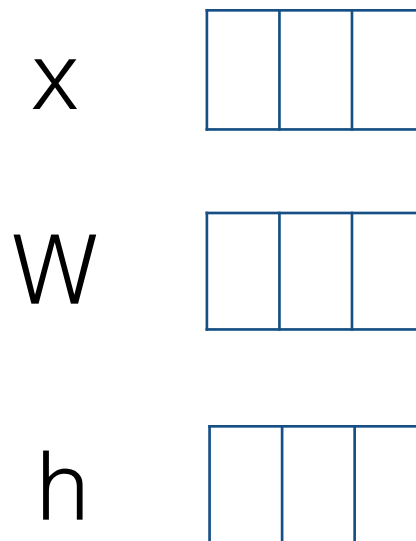
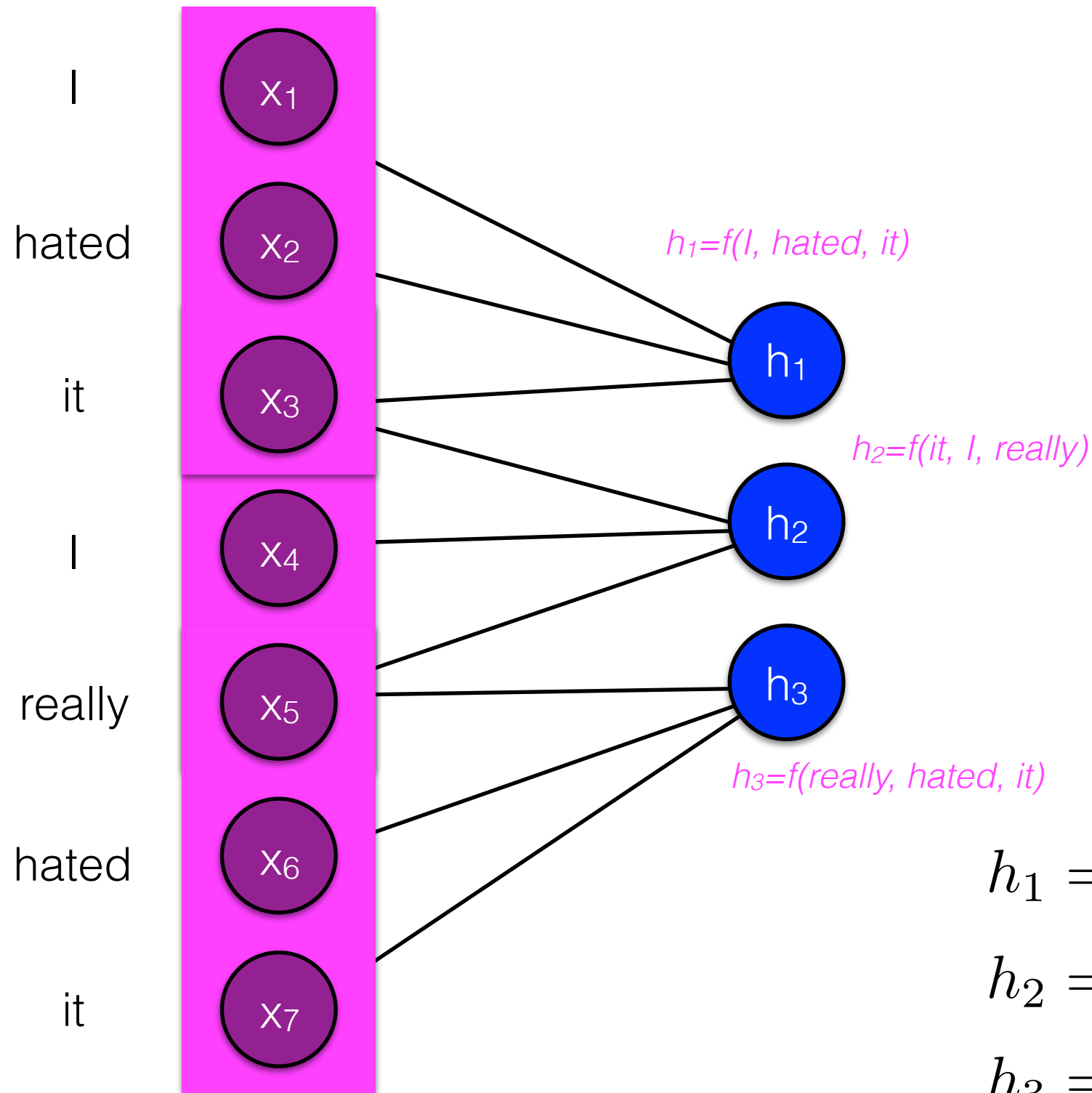
$\times$

|     |     |     |    |   |   |   |
|-----|-----|-----|----|---|---|---|
|     |     |     |    |   |   |   |
| 1/3 | 1/3 | 1/3 |    |   |   |   |
|     |     |     |    |   |   |   |
| 0   | 1   | 3   | -1 | 4 | 2 | 0 |

|       |   |   |       |   |
|-------|---|---|-------|---|
| 1 1/3 | 1 | 2 | 1 2/3 | 2 |
|-------|---|---|-------|---|



# Convolutional networks



$$h_1 = \sigma(x_1 W_1 + x_2 W_2 + x_3 W_3)$$

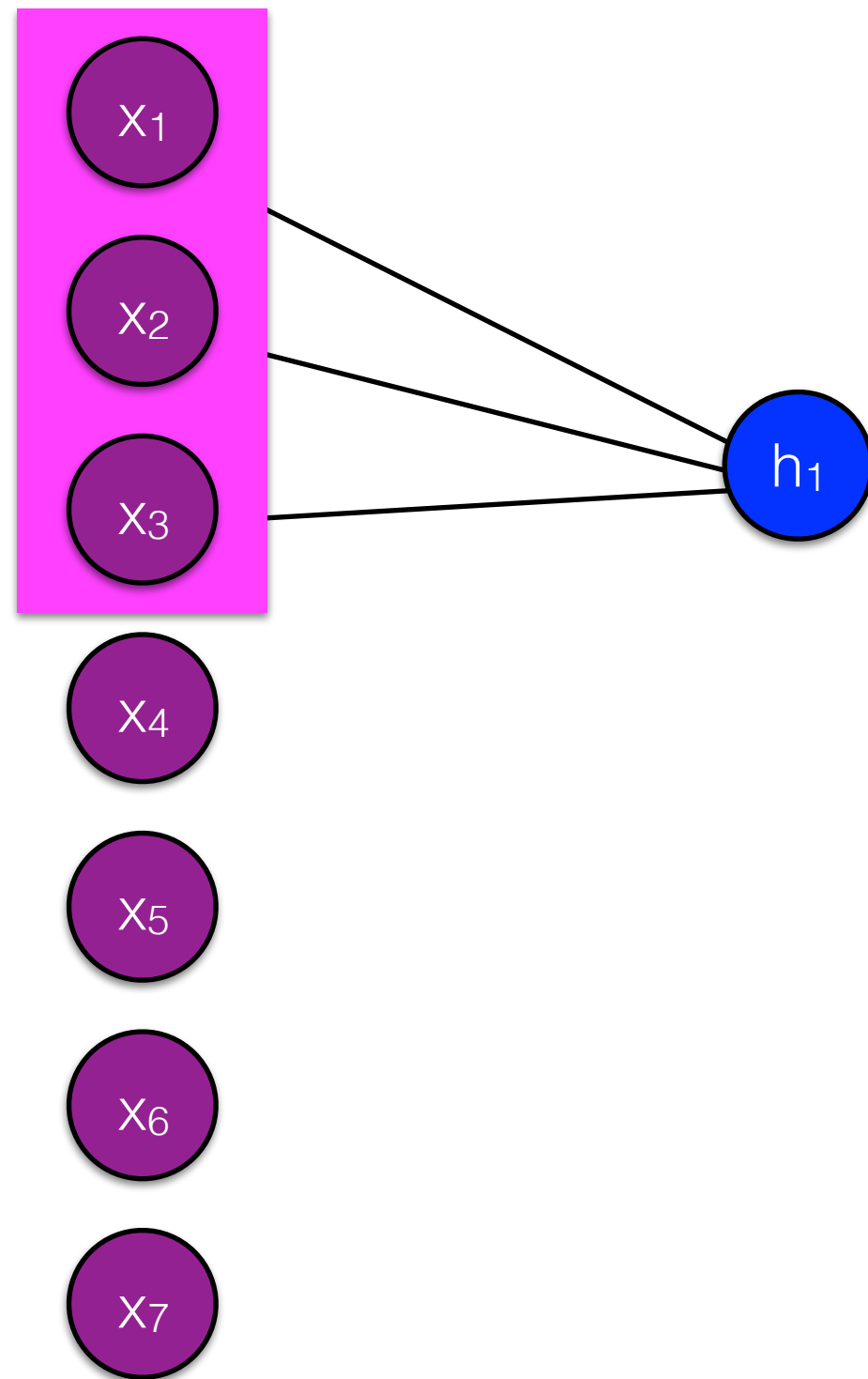
$$h_2 = \sigma(x_3 W_1 + x_4 W_2 + x_5 W_3)$$

$$h_3 = \sigma(x_5 W_1 + x_6 W_2 + x_7 W_3)$$

# Indicator vector

- Every token is a V-dimensional vector (size of the vocab) with a single 1 identifying the word
- We'll get to distributed representations of words in on 9/18

| vocab item | indicator |
|------------|-----------|
| a          | 0         |
| aa         | 0         |
| aal        | 0         |
| aalii      | 0         |
| aam        | 0         |
| aardvark   | 1         |
| aardwolf   | 0         |
| aba        | 0         |

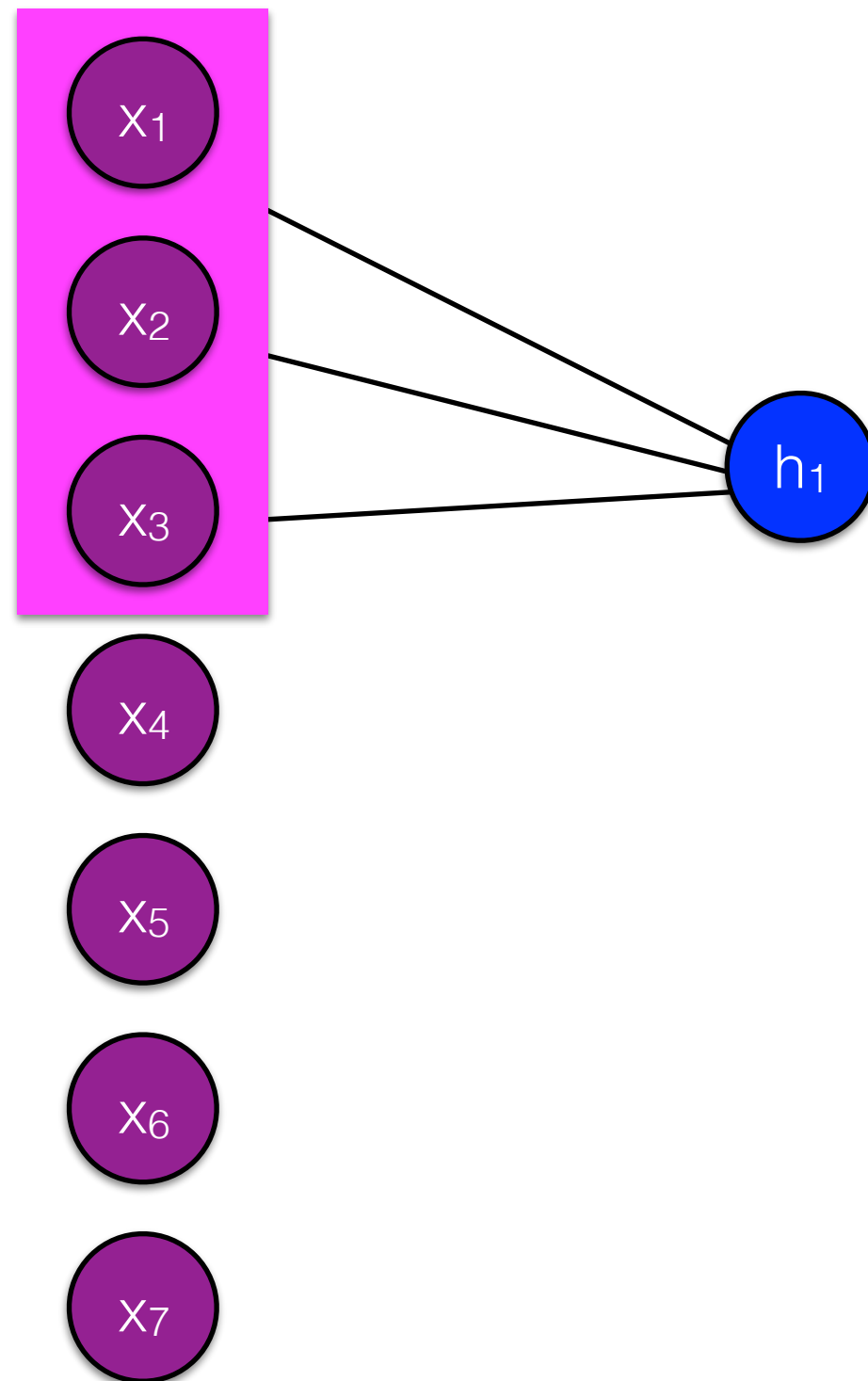


|       | X                                                                   |  | W                                                                                    |
|-------|---------------------------------------------------------------------|--|--------------------------------------------------------------------------------------|
|       | <div><div></div><div>1</div><div></div><div></div><div></div></div> |  | <div><div>3.1</div><div>-2.7</div><div>1.4</div><div>-0.7</div><div>-1.4</div></div> |
| $x_1$ |                                                                     |  | $w_1$                                                                                |
|       | <div><div></div><div></div><div></div><div>1</div><div></div></div> |  | <div><div>9.2</div><div>-3.1</div><div>-2.7</div><div>1.4</div><div>0.1</div></div>  |
| $x_2$ |                                                                     |  | $w_2$                                                                                |
|       | <div><div>1</div><div></div><div></div><div></div><div></div></div> |  | <div><div>0.3</div><div>-0.4</div><div>-2.4</div><div>-4.7</div><div>5.7</div></div> |
| $x_3$ |                                                                     |  | $w_3$                                                                                |

$$h_1 = \sigma(x_1 W_1 + x_2 W_2 + x_3 W_3)$$

$$h_2 = \sigma(x_3 W_1 + x_4 W_2 + x_5 W_3)$$

$$h_3 = \sigma(x_5 W_1 + x_6 W_2 + x_7 W_3)$$

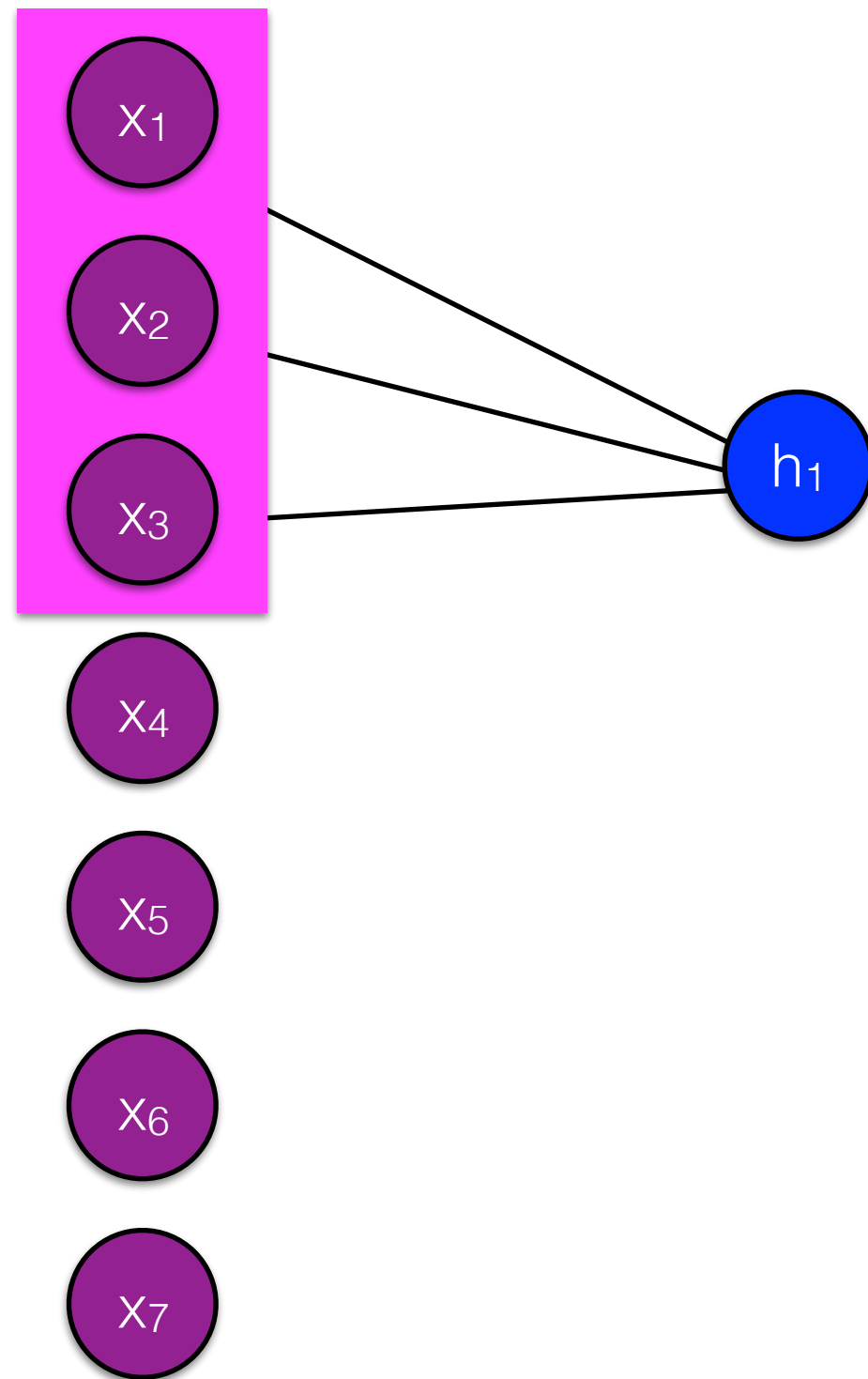


|       | X |       | W    |
|-------|---|-------|------|
| $x_1$ |   | $W_1$ | 3.1  |
|       | 1 |       | -2.7 |
|       |   |       | 1.4  |
|       |   |       | -0.7 |
| $x_2$ |   | $W_2$ | -1.4 |
|       |   |       | 9.2  |
|       |   |       | -3.1 |
|       |   |       | -2.7 |
| $x_3$ | 1 |       | 1.4  |
|       |   | $W_3$ | 0.1  |
|       |   |       | 0.3  |
|       |   |       | -0.4 |
|       |   |       | -2.4 |
|       |   |       | -4.7 |
|       |   |       | 5.7  |

For indicator vectors, we're just adding these numbers together

$$h_1 = \sigma(W_{1,x_1^{id}} + W_{2,x_2^{id}} + W_{3,x_3^{id}})$$

(Where  $x_n^{id}$  specifies the location of the 1 in the vector — i.e., the vocabulary id)

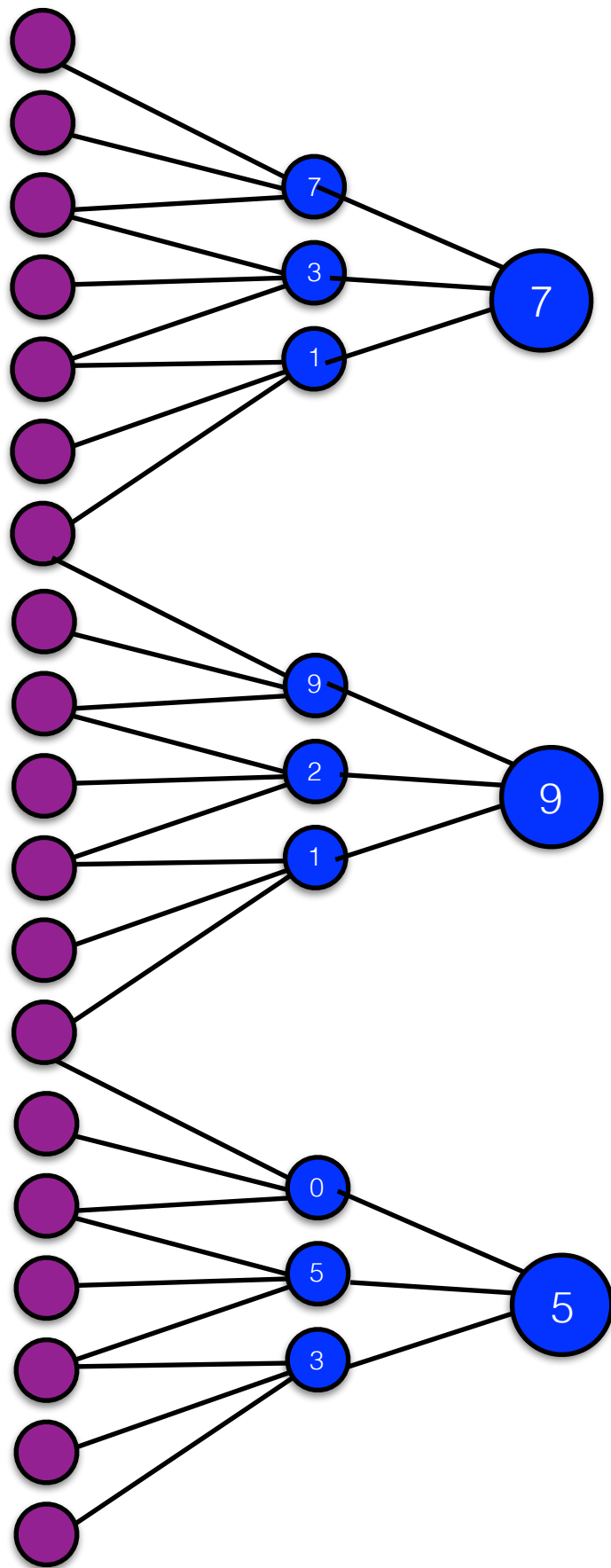


|       | X    |       | W    |
|-------|------|-------|------|
| $X_1$ | 0.4  | $W_1$ | 3.1  |
|       | 0.8  |       | -2.7 |
|       | 1.2  |       | 1.4  |
|       | -1.3 |       | -0.7 |
|       | 0.4  |       | -1.4 |
| $X_2$ | 0.2  | $W_2$ | 9.2  |
|       | -5.3 |       | -3.1 |
|       | -1.2 |       | -2.7 |
|       | 5.3  |       | 1.4  |
|       | 0.4  |       | 0.1  |
| $X_3$ | 2.6  | $W_3$ | 0.3  |
|       | 2.7  |       | -0.4 |
|       | -3.2 |       | -2.4 |
|       | 6.2  |       | -4.7 |
|       | 1.9  |       | 5.7  |

For dense input vectors (e.g., embeddings), full dot product

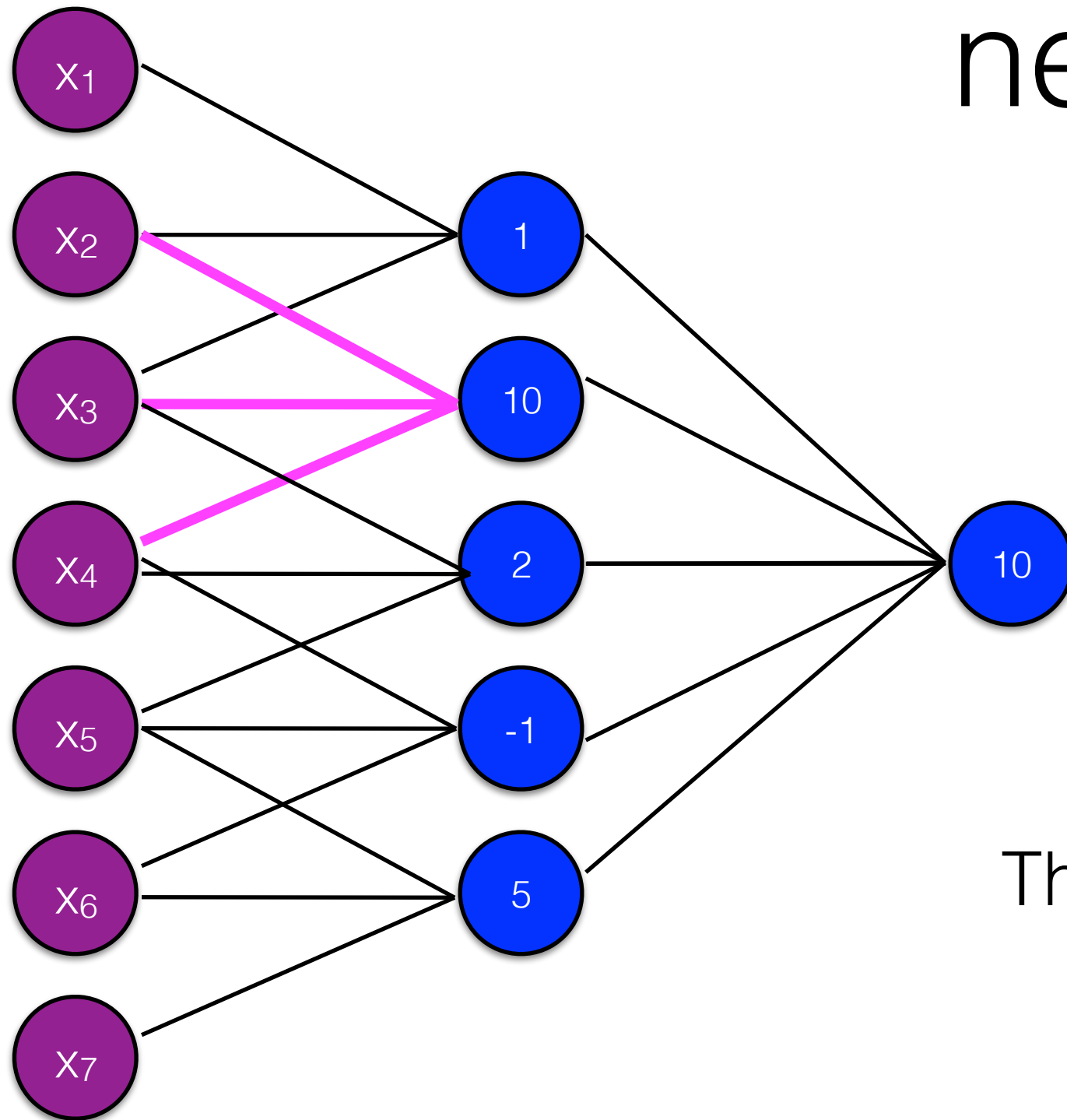
$$h_1 = \sigma(x_1 W_1 + x_2 W_2 + x_3 W_3)$$

# Pooling



- Down-samples a layer by selecting a single point from some set
- **Max-pooling** selects the largest value

# Convolutional networks

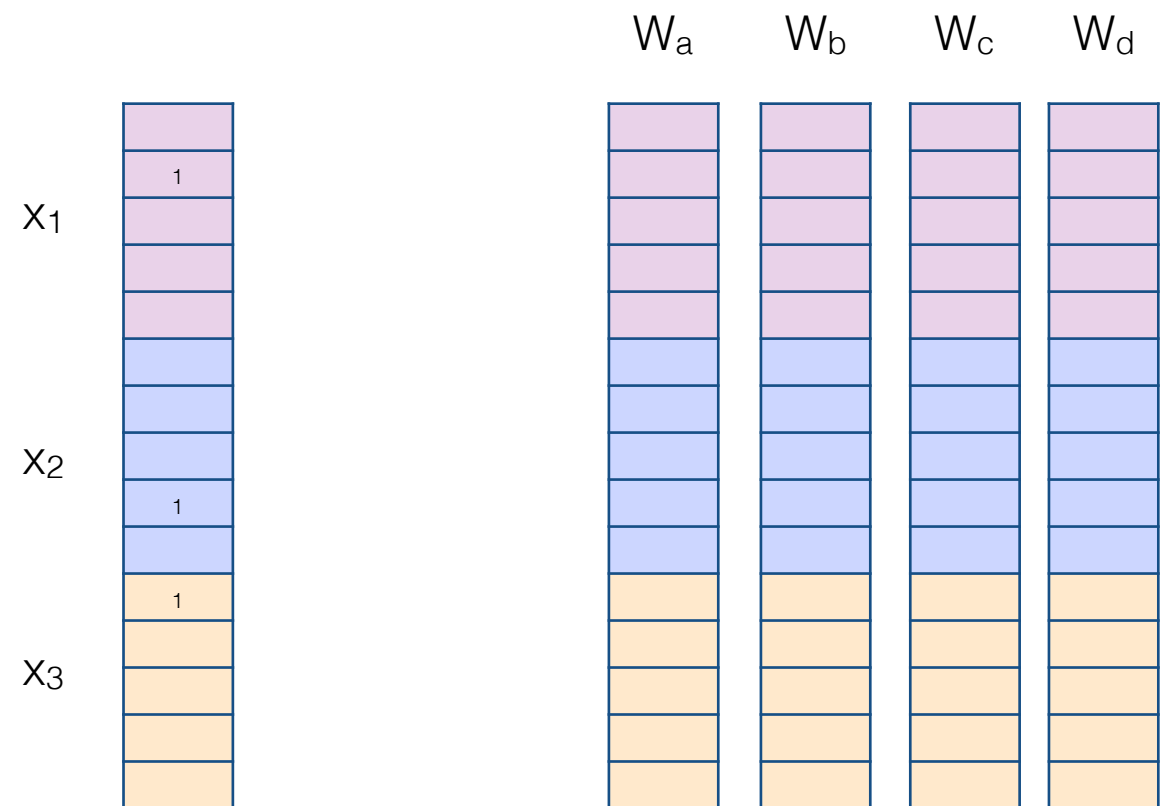
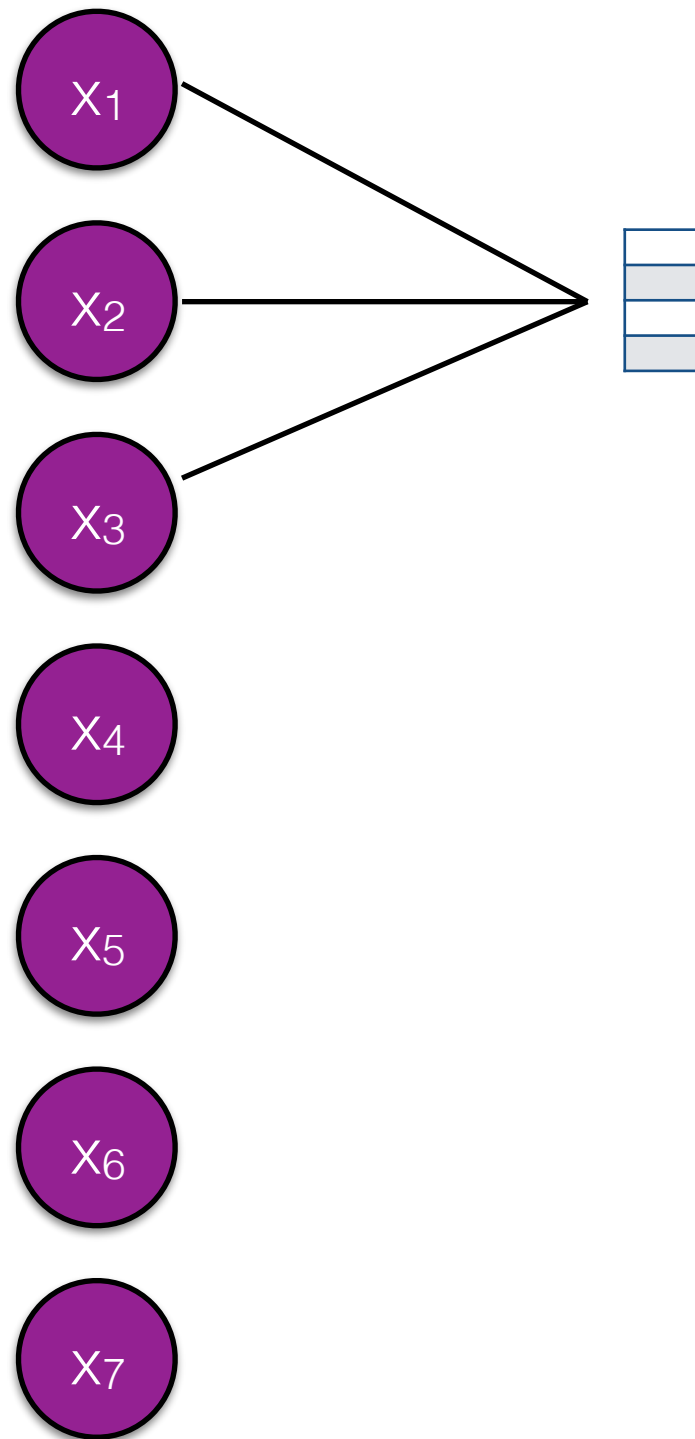


This defines one filter.

convolution

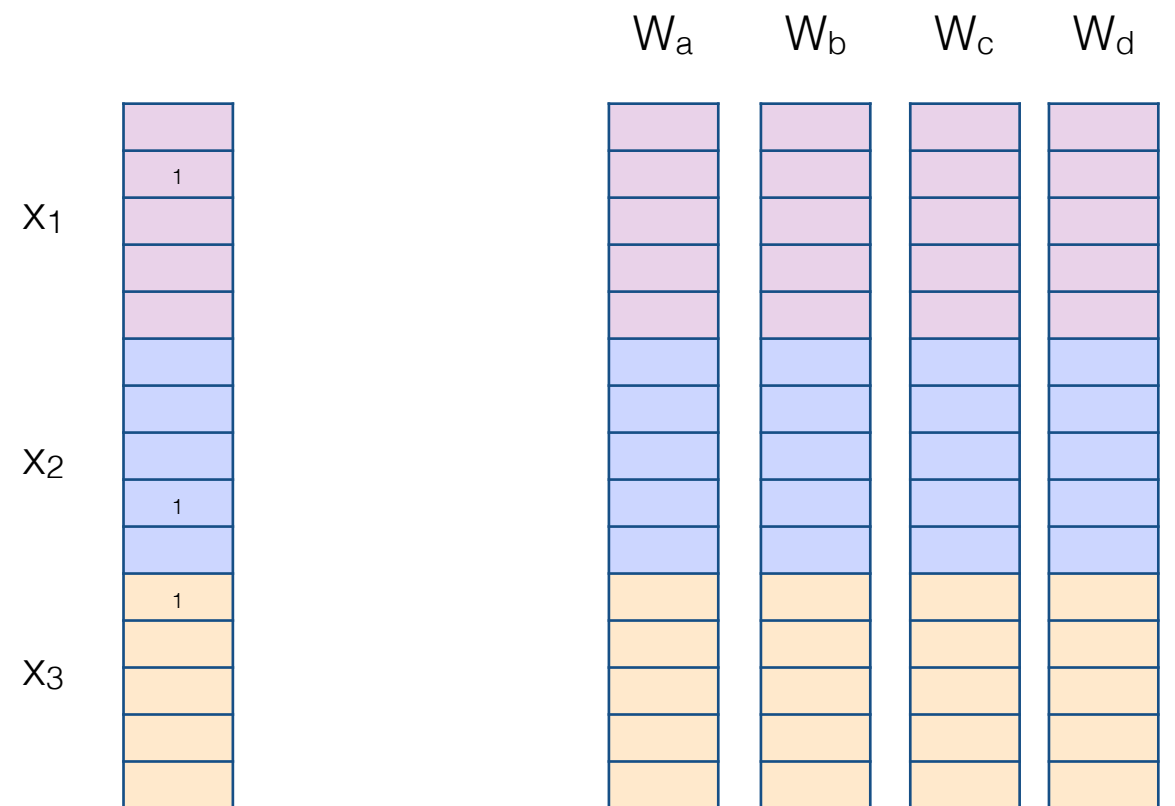
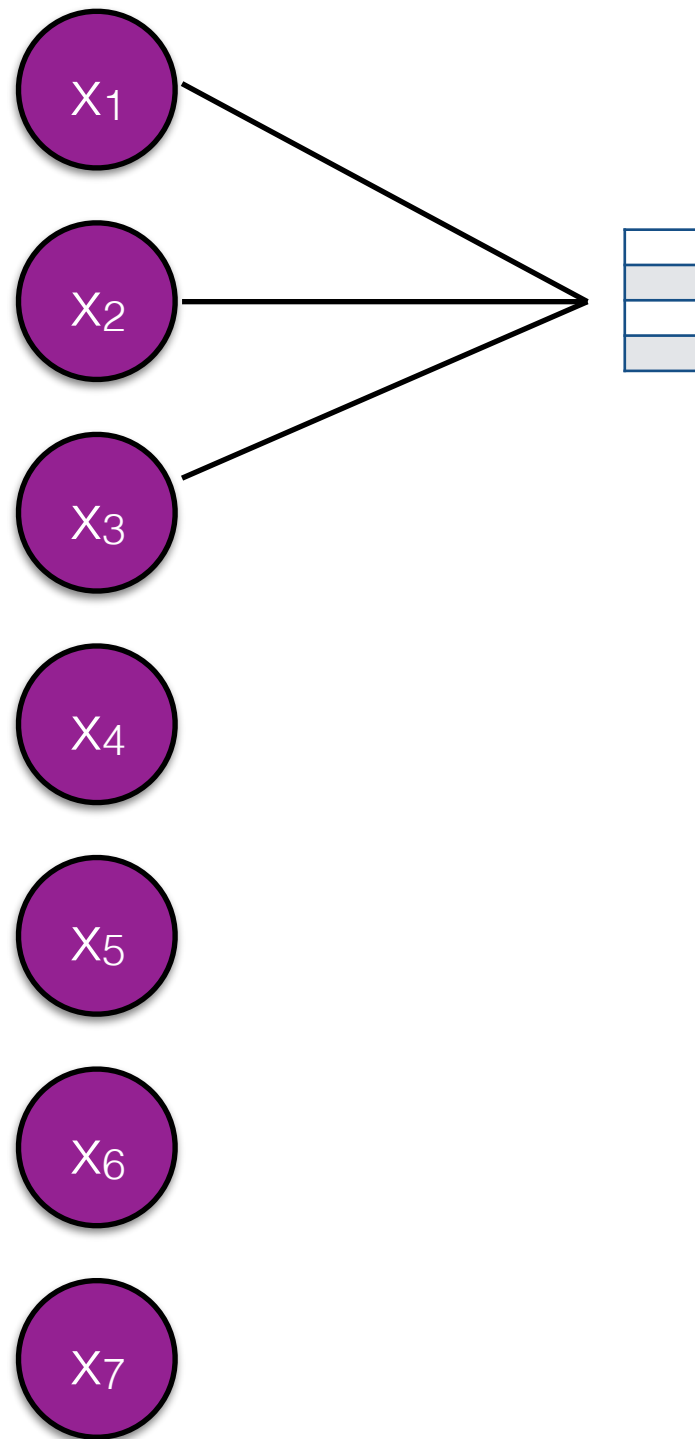
max pooling

We can specify multiple filters; each filter is a separate set of parameters to be learned



$$h_1 = \sigma(x^\top W) \in R^4$$

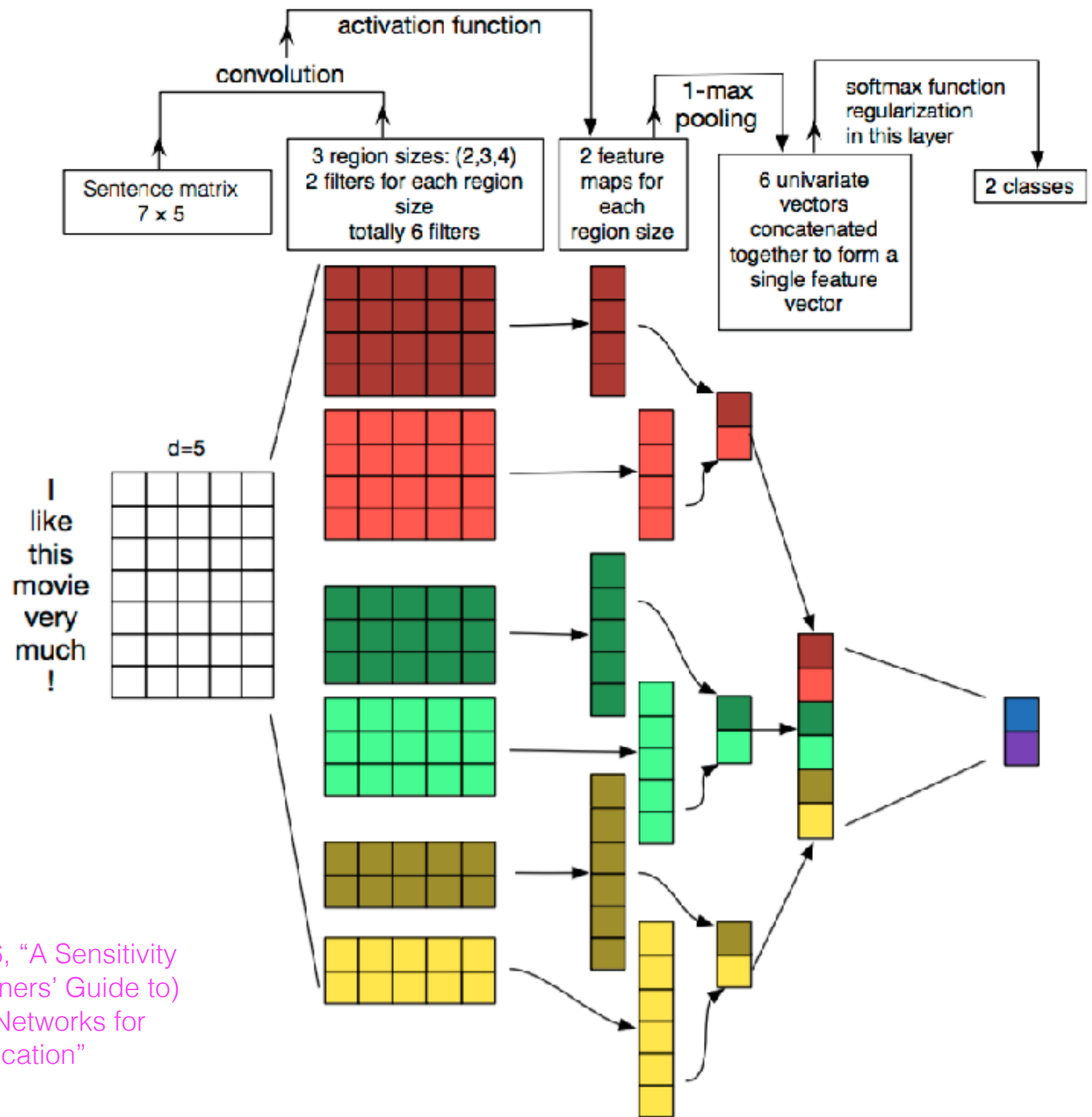
We can specify multiple filters; each filter is a separate set of parameters to be learned



$$h_1 = \sigma(x^\top W) \in R^4$$

# Convolutional networks

- With max pooling, we select a single number for each filter over all tokens
- (e.g., with 100 filters, the output of max pooling stage = 100-dimensional vector)
- If we specify multiple filters, we can also scope each filter over different window sizes



Zhang and Wallace 2016, "A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification"

# CNN as important ngram detector

Higher-order ngrams are much more informative than just unigrams (e.g., “i don’t like this movie” [“I”, “don’t”, “like”, “this”, “movie”])

We can think about a CNN as providing a mechanism for detecting important (sequential) ngrams without having the burden of creating them as unique features

|          | unique types |
|----------|--------------|
| unigrams | 50921        |
| bigrams  | 451,220      |
| trigrams | 910,694      |
| 4-grams  | 1,074,921    |

Uniquengrams (1-4) in Cornell movie review dataset

# 259 project proposal

## due 9/25

- Final project involving 1 to 3 students involving natural language processing -- either focusing on core NLP methods or using NLP in support of an empirical research question.
- Proposal (2 pages):
  - outline the work you're going to undertake
  - motivate its rationale as an interesting question worth asking
  - assess its potential to contribute new knowledge by situating it within related literature in the scientific community. (cite 5 relevant sources)
  - who is the team and what are each of your responsibilities (everyone gets the same grade)
- Feel free to come by my office hours and discuss your ideas!

# Thursday

- Read Hovy and Spruit (2016) and come prepared to discuss!