Natural Language Processing

Info 159/259

Lecture 4: Text classification 3 (Jan 30, 2020)

David Bamman, UC Berkeley

Generative vs. Discriminative models

- Generative models specify a joint distribution over the labels and the data. With this you could generate new data

$$P(X, Y) = P(Y)\, P(X \mid Y)$$

- Discriminative models specify the conditional distribution of the label y given the data x. These models focus on how to discriminate between the classes

$$P(Y \mid X)$$

Generative models

- With generative models (e.g., Naive Bayes), we ultimately also care about P(Y | X), but we get there by modeling more.

$$P(Y = y \mid X = x) = \frac{P(Y = y)\, P(X = x \mid Y = y)}{\sum_{y \in \mathcal{Y}} P(Y = y) P(X = x \mid Y = y)}$$

- Discriminative models focus on modeling P(Y | X) — *and only P(Y | X)* — directly.

Logistic regression

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

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

Features

- As a discriminative classifier, logistic regression doesn't assume features are independent like Naive Bayes does.

- Its power partly comes in the ability to create richly expressive features without the burden of independence.

- We can represent text through features that are not just the identities of individual words, but any feature that is scoped over the entirety of the input.

features
contains like
has word that shows up in positive sentiment dictionary
review begins with "I like"
at least 5 mentions of positive affectual verbs (like, love, etc.)

Logistic regression

- We want to find the value of β that leads to the highest value of the log likelihood:

$$\ell(\beta) = \sum_{i=1}^{N} \log P(y_i \mid x_i, \beta)$$

Algorithm 2 Logistic regression stochastic 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: **for** $i = 1$ to N **do**
5: $\beta_{t+1} = \beta_t + \alpha\left(y_i - \hat{p}(x_i)\right) x_i$
6: **end for**
7: **end while**

L2 regularization

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

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

L1 regularization

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

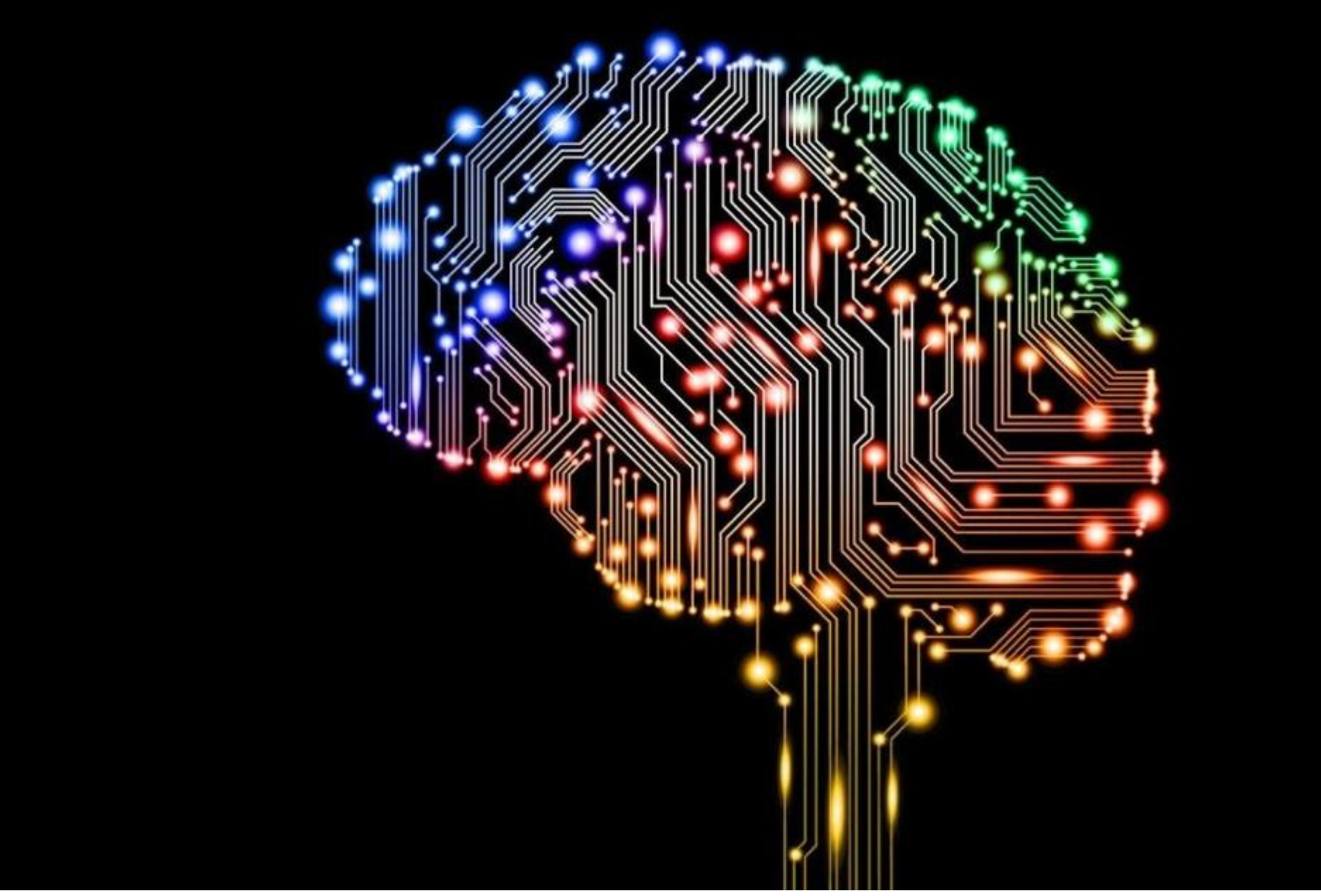
- L1 regularization encourages coefficients to be exactly 0.
- η again controls how much of a penalty to pay for coefficients that are far from 0 (optimize on development data)

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

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

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



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

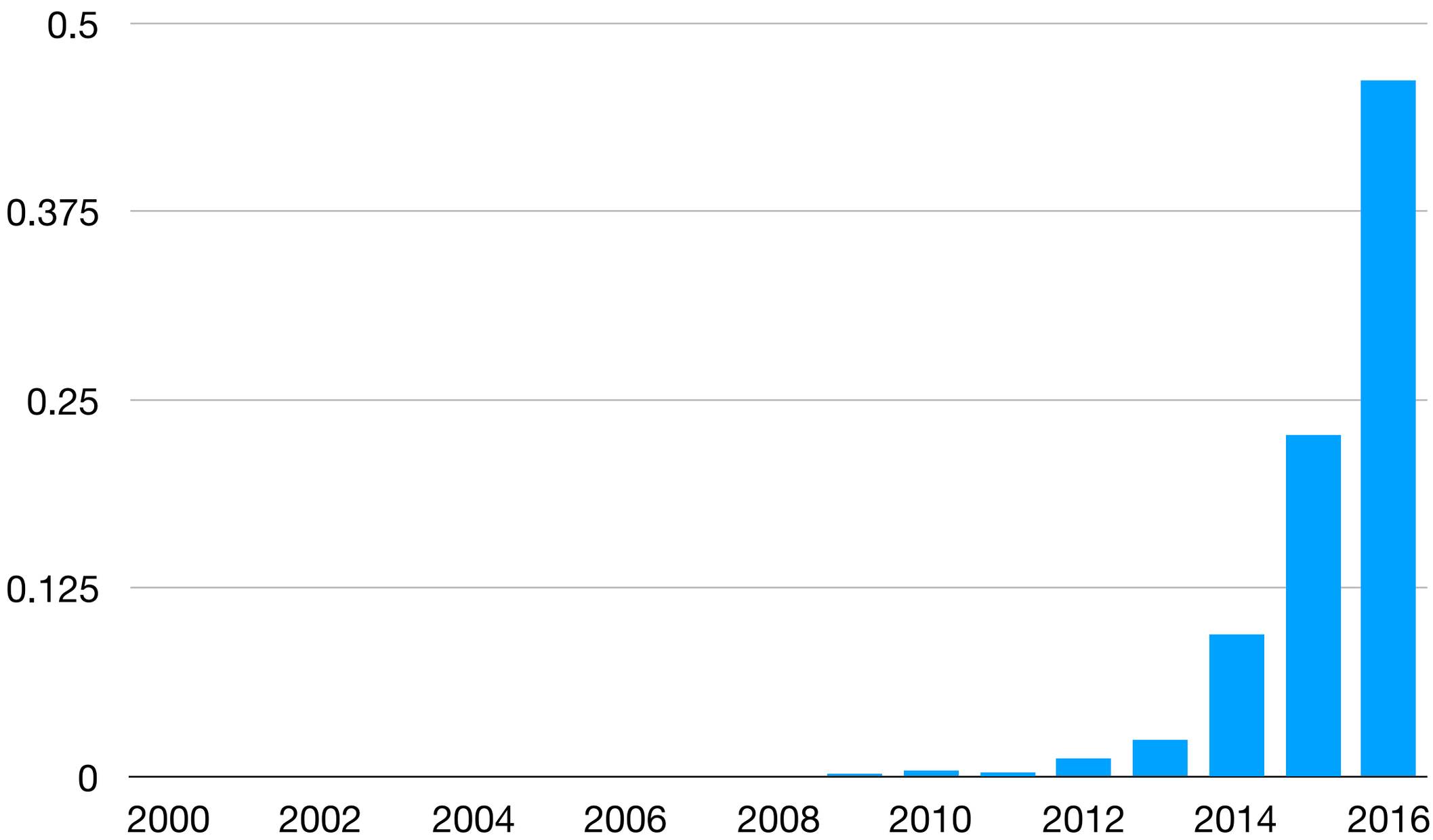
History of NLP

- 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)

“Word embedding” in NLP papers



ACL, EMNLP, NAACL, TACL, EACL, CONLL, CL (data from ACL Anthology Network)

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.

- Static representations -> contextual representations, where representations of words are sensitive to local context.

- Non-linear interactions of input features

- Multiple layers to capture hierarchical structure

Neural network libraries

Logistic regression

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

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

SGD

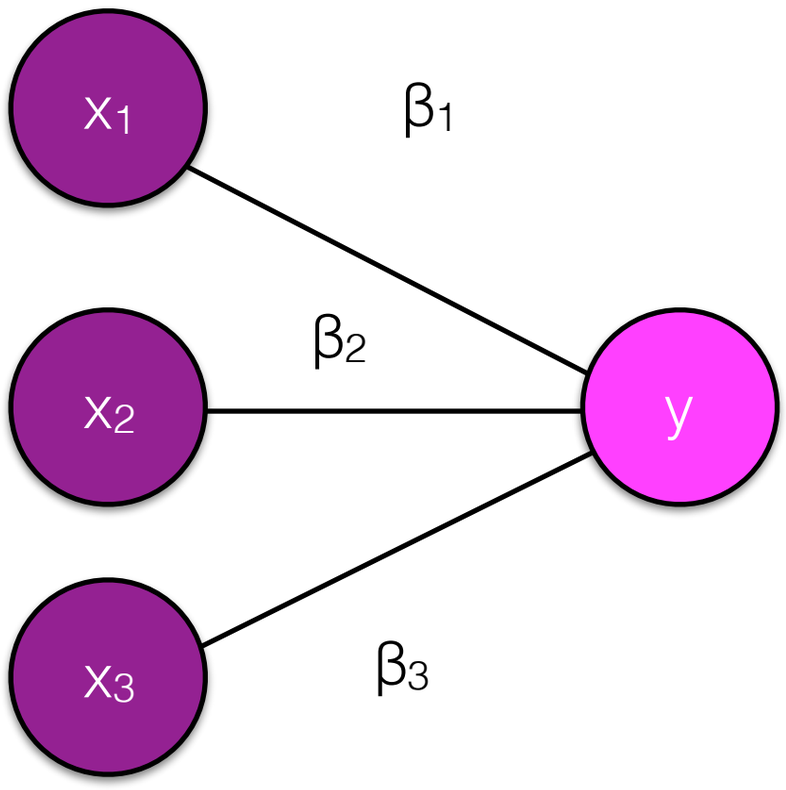
Algorithm 1 Logistic regression gradient descent

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

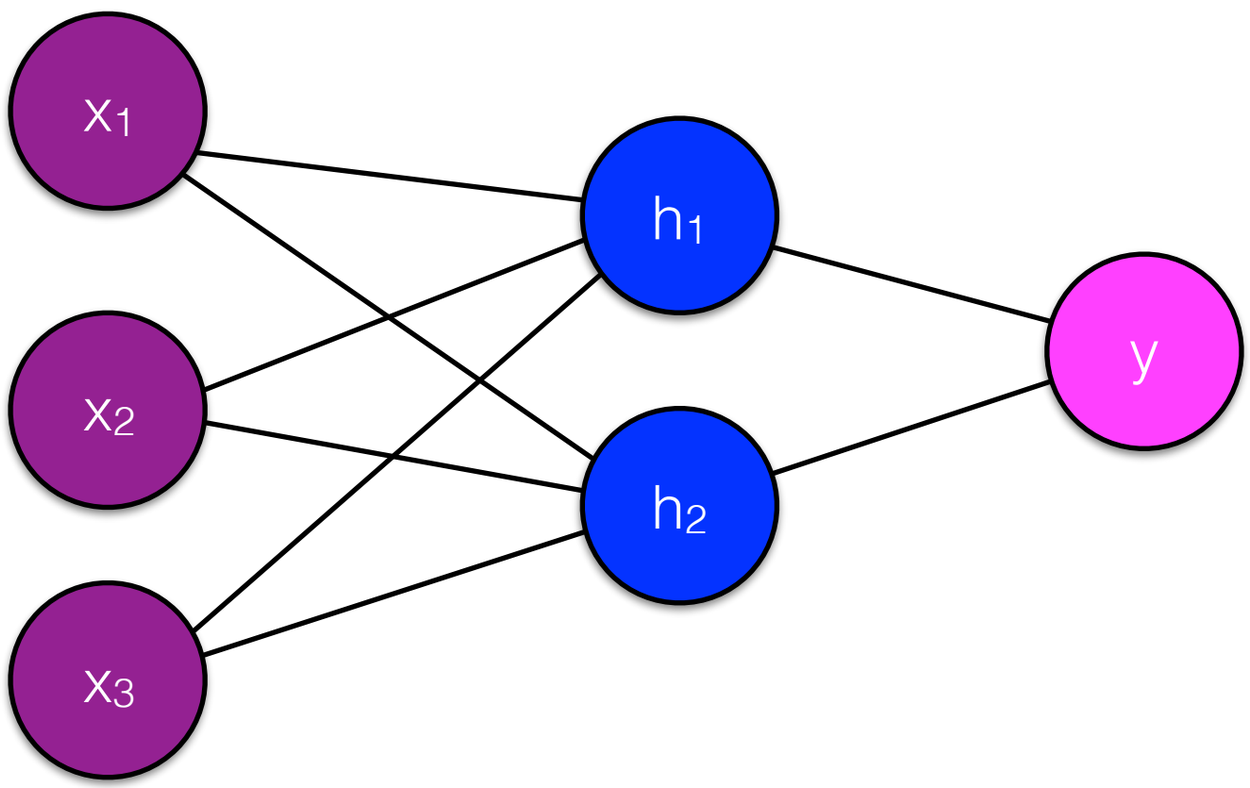
$$P(\hat{y} = 1) = \frac{1}{1 + \exp\left(-\sum_{i=1}^{F} x_i \beta_i\right)}$$



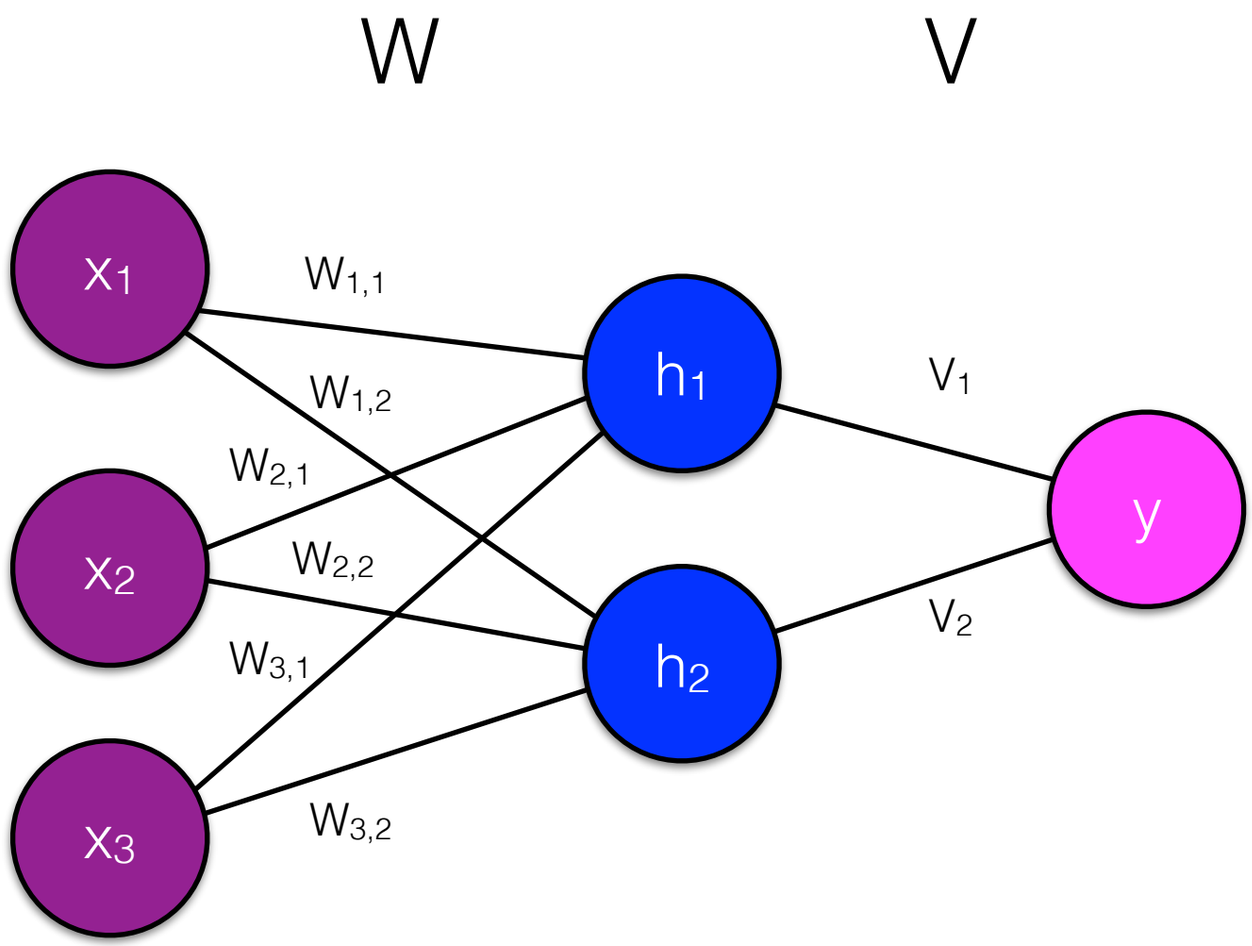
	x	β
not	1	-0.5
bad	1	-1.7
movie	0	0.3

Feedforward neural network

- Input and output are mediated by at least one hidden layer.



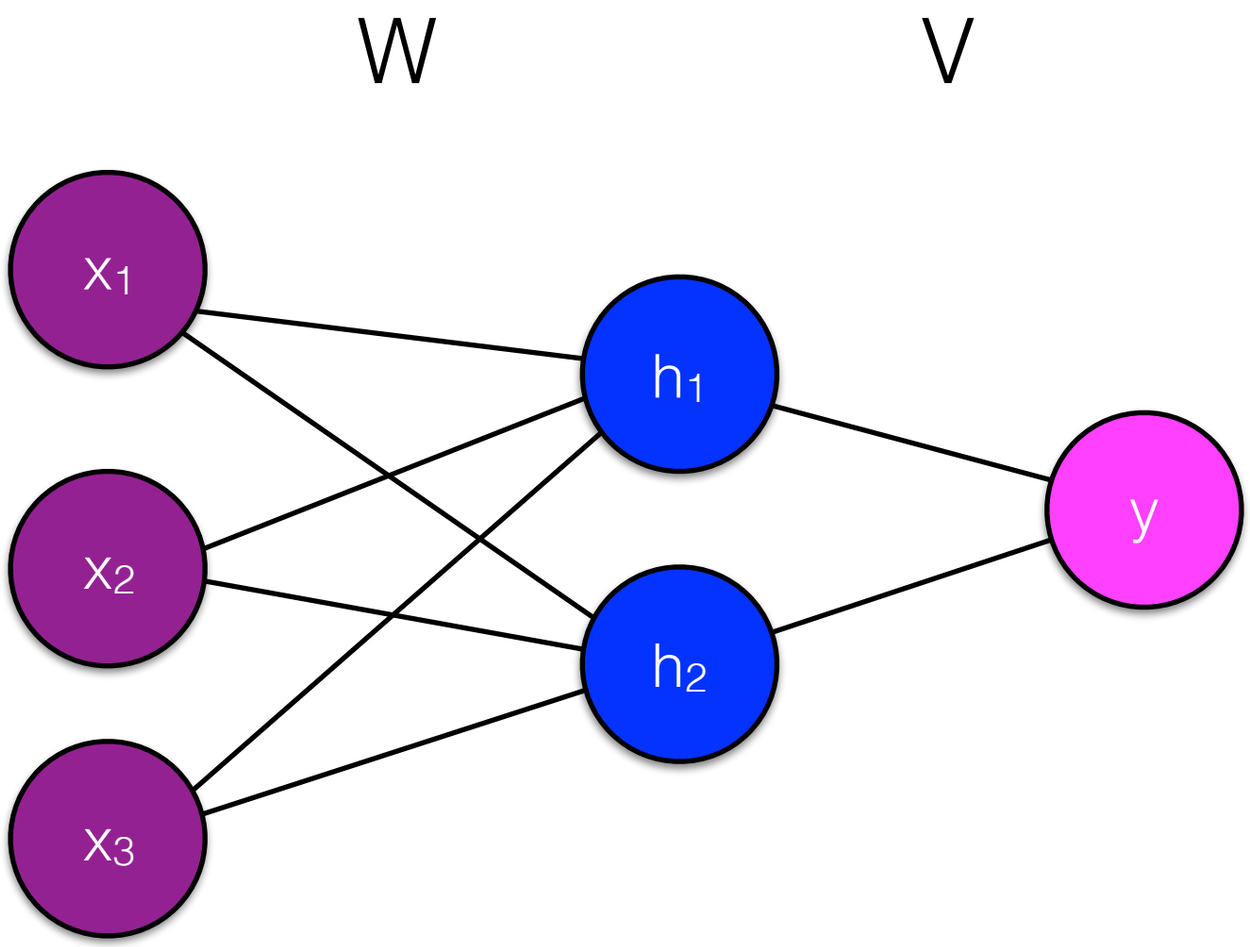
*For simplicity, we're leaving out the bias term, but assume most layers have them as well.



Input

"Hidden" Layer

Output

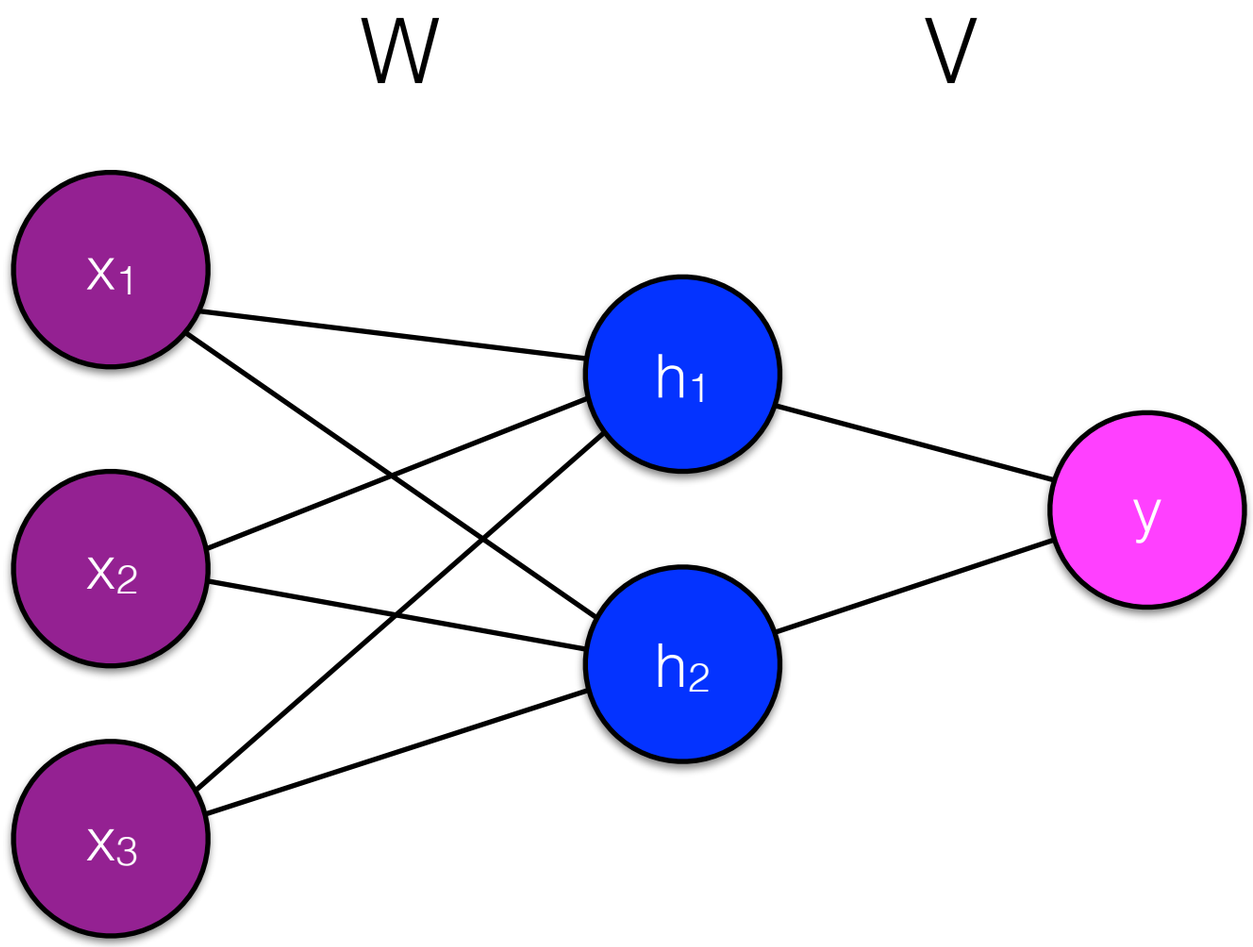


	x
not	1
bad	1
movie	0

W	
-0.5	1.3
0.4	0.08
1.7	3.1

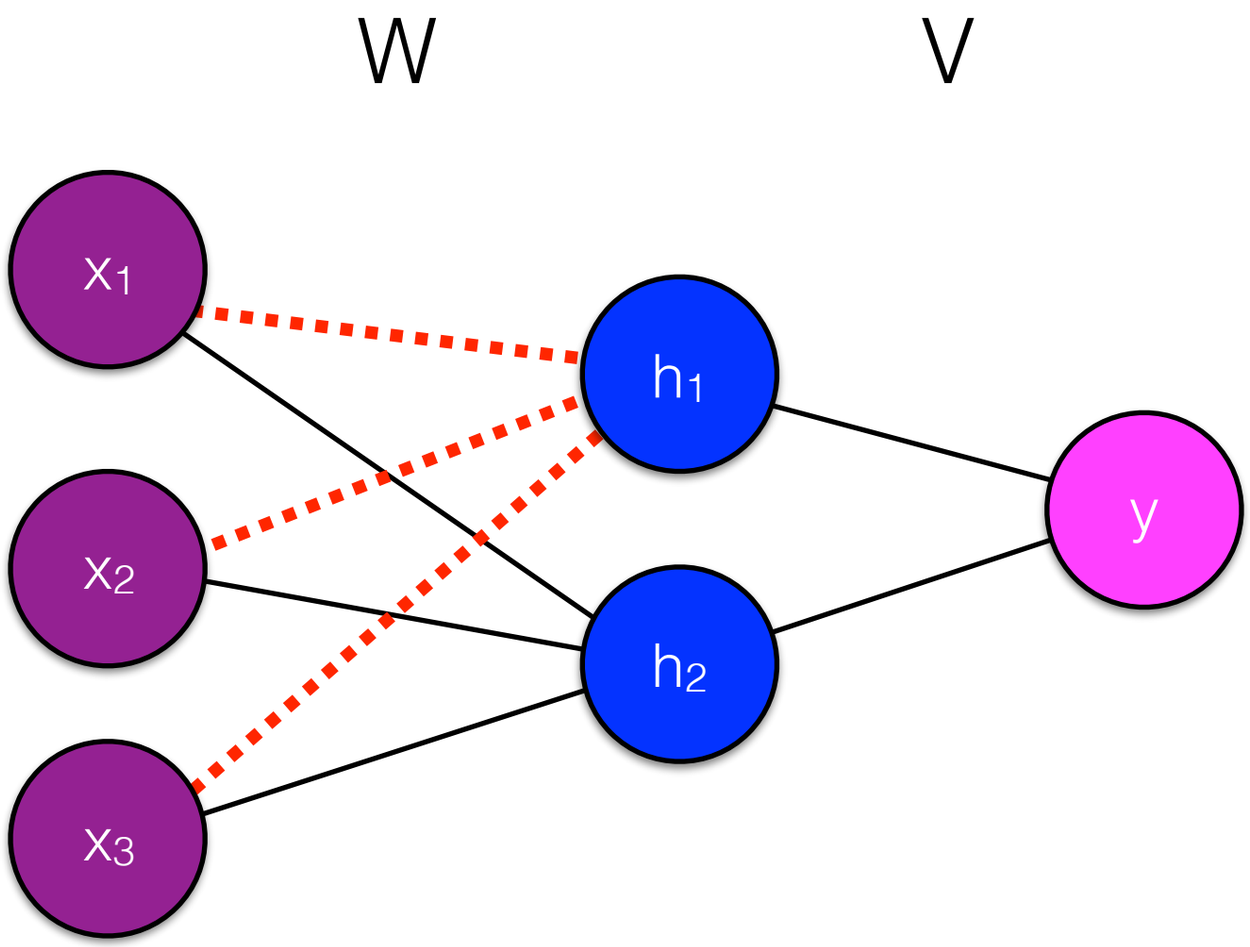
V
4.1
-0.9

y
1



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

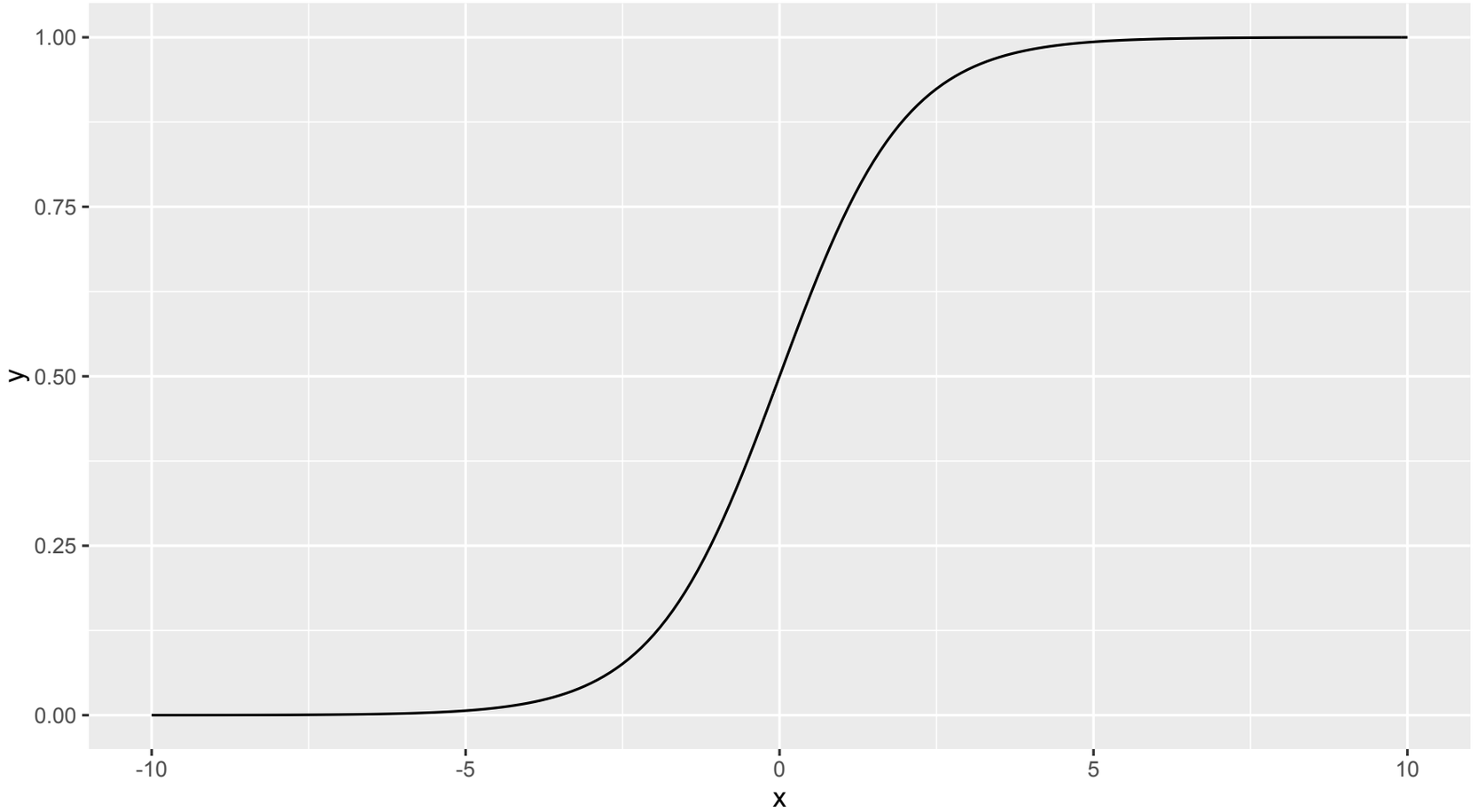
the hidden nodes are completely determined by the input and weights



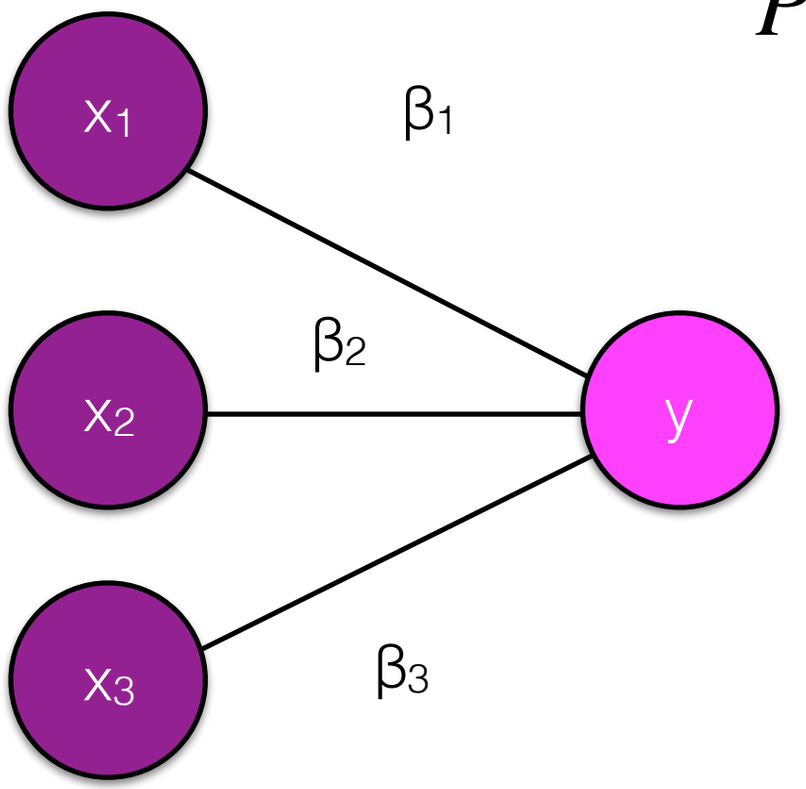
$$h_1 = f\left(\sum_{i=1}^{F} x_i W_{i,1}\right)$$

Activation functions

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



Logistic regression



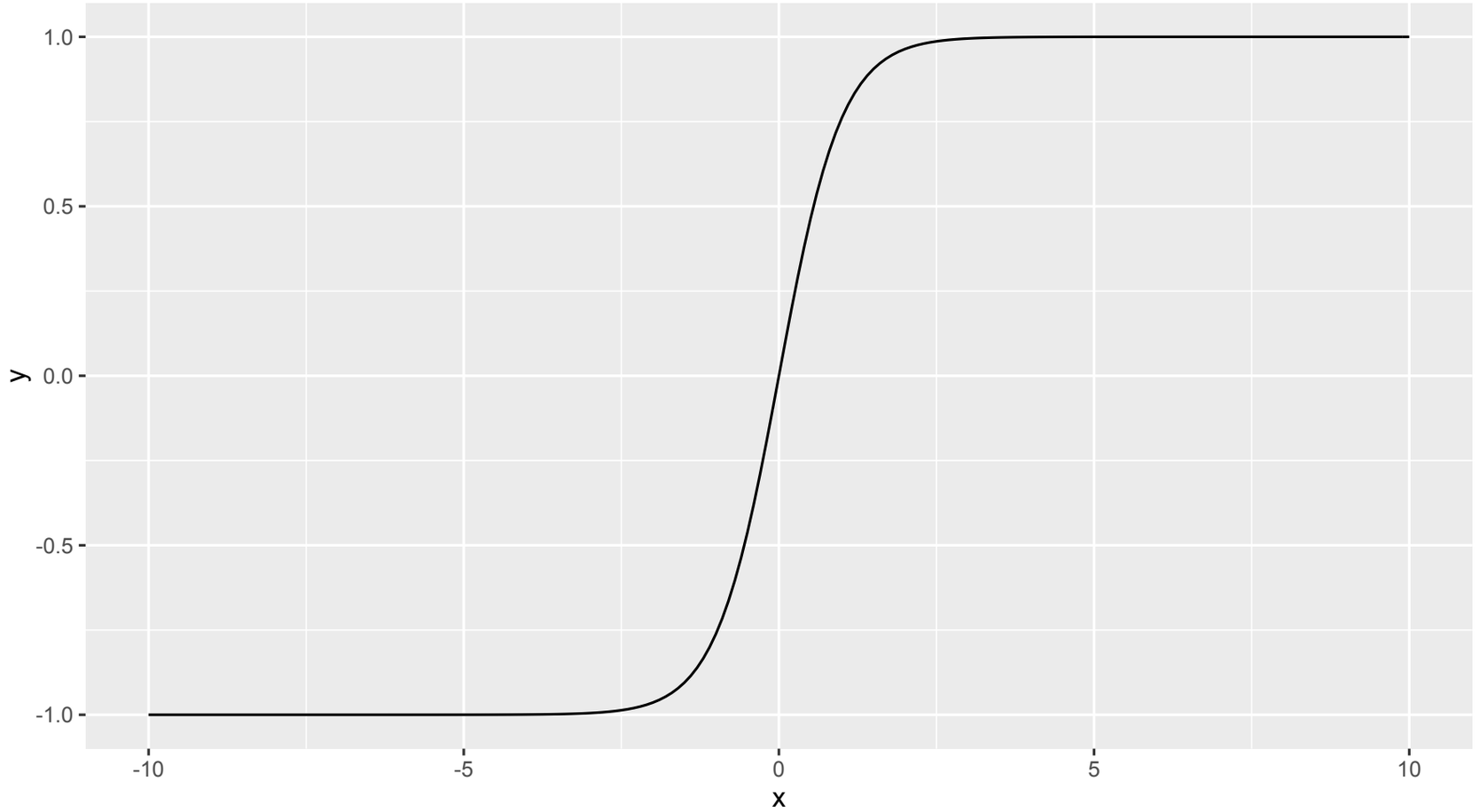
$$P(\hat{y} = 1) = \frac{1}{1 + \exp\left(-\sum_{i=1}^{F} x_i \beta_i\right)}$$

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

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

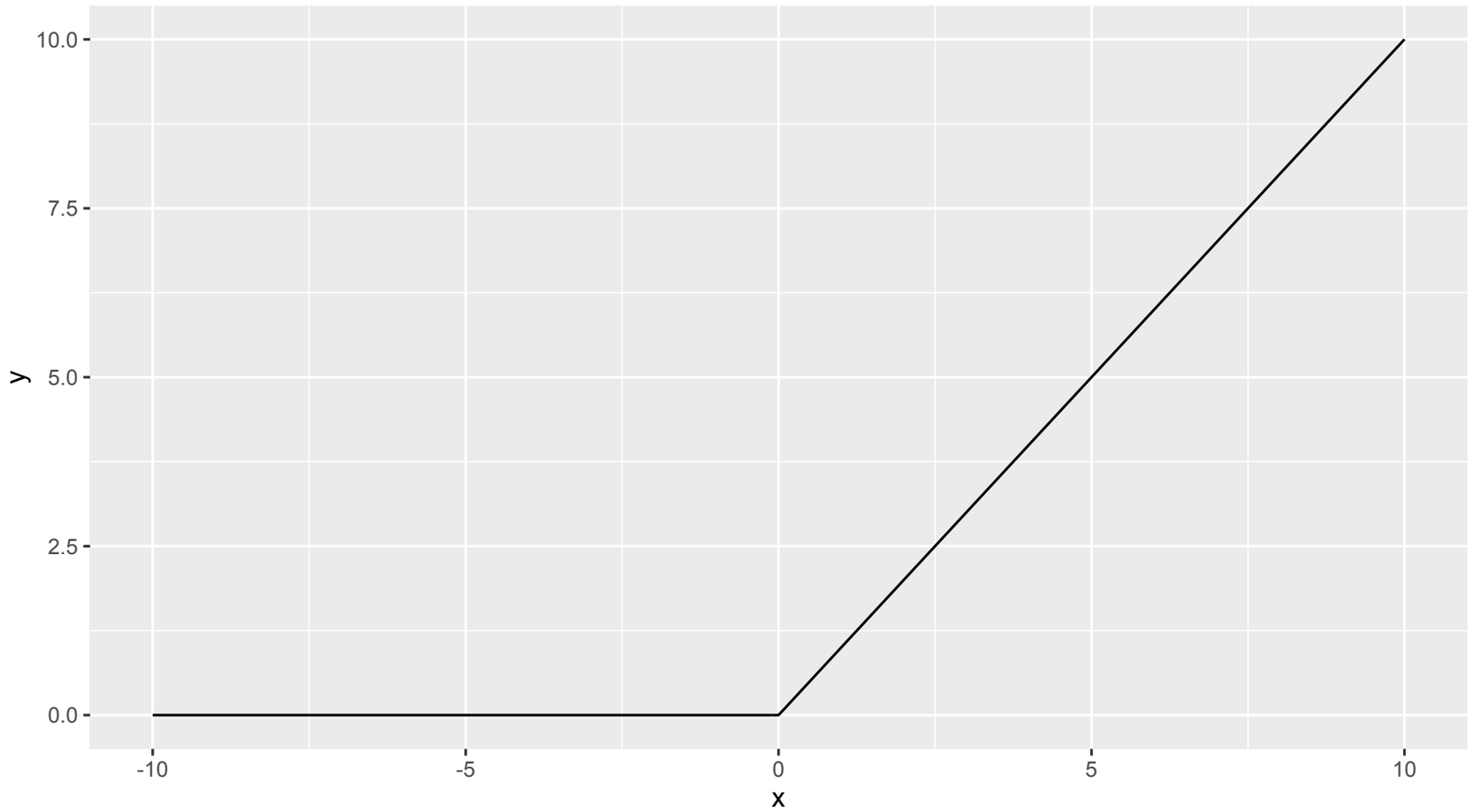
Activation functions

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



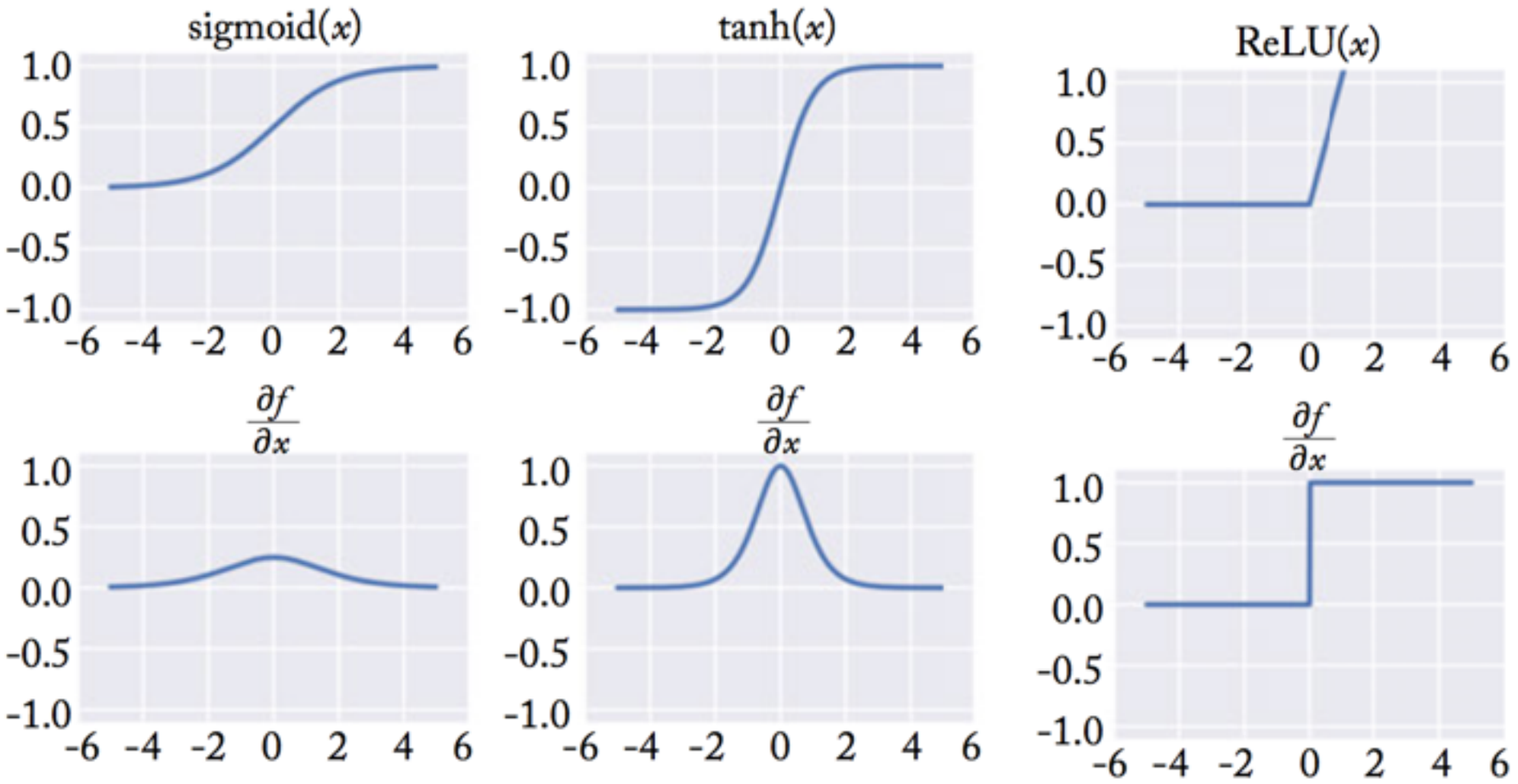
Activation functions

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



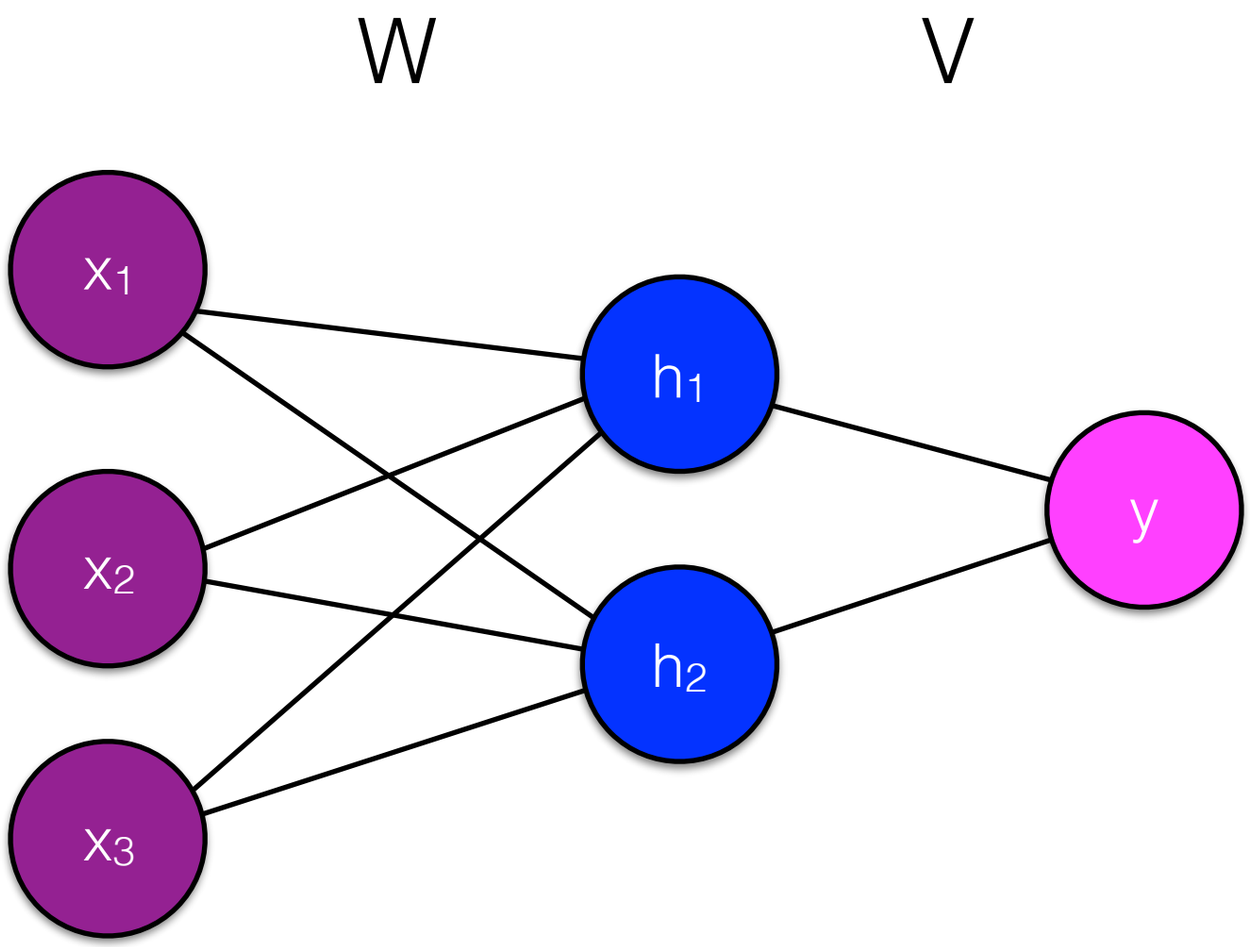
function

derivative



Goldberg 46

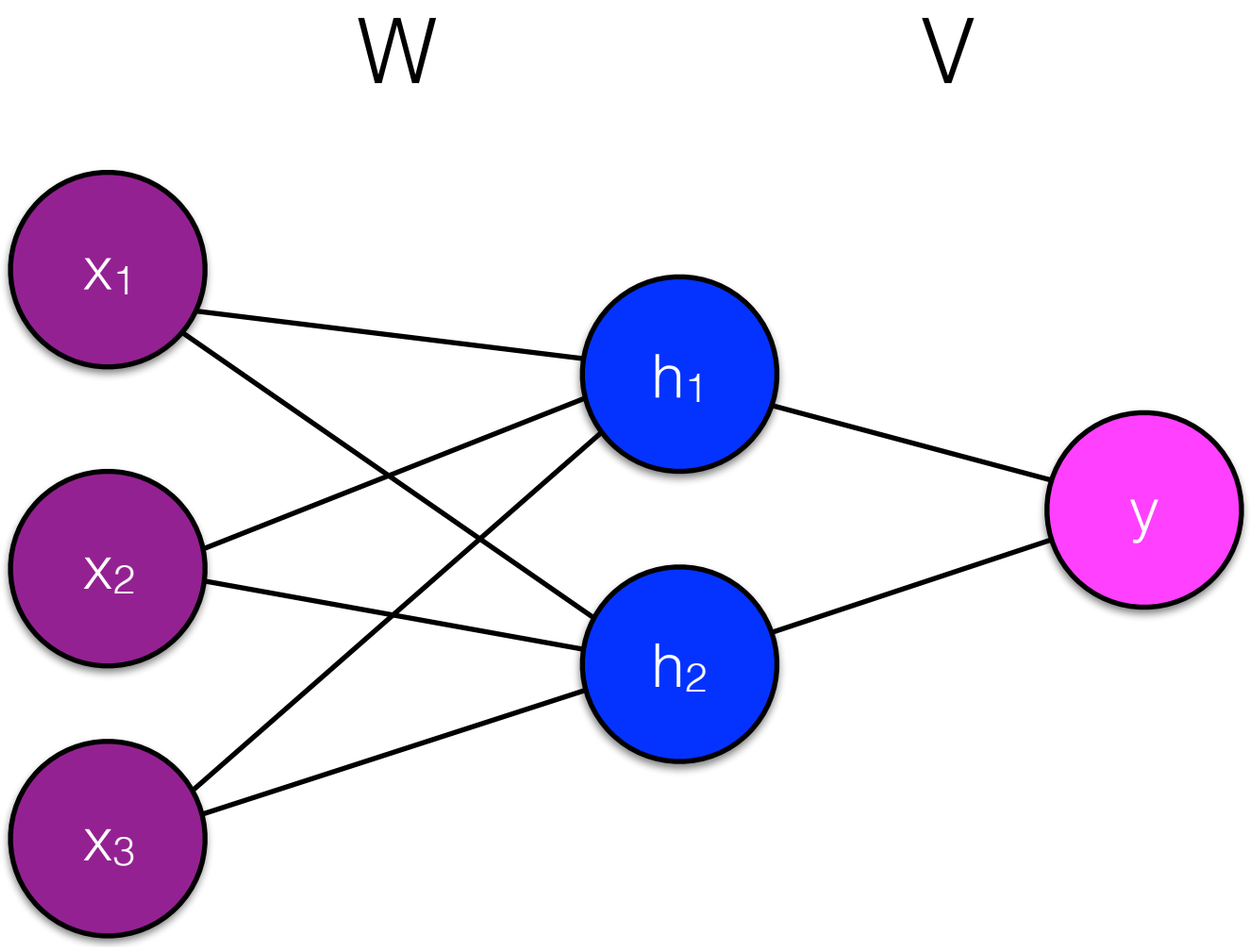
- ReLU and tanh are both used extensively in modern systems.
- Sigmoid is useful for final layer to scale output between 0 and 1, but is not often used in intermediate layers.



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

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

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



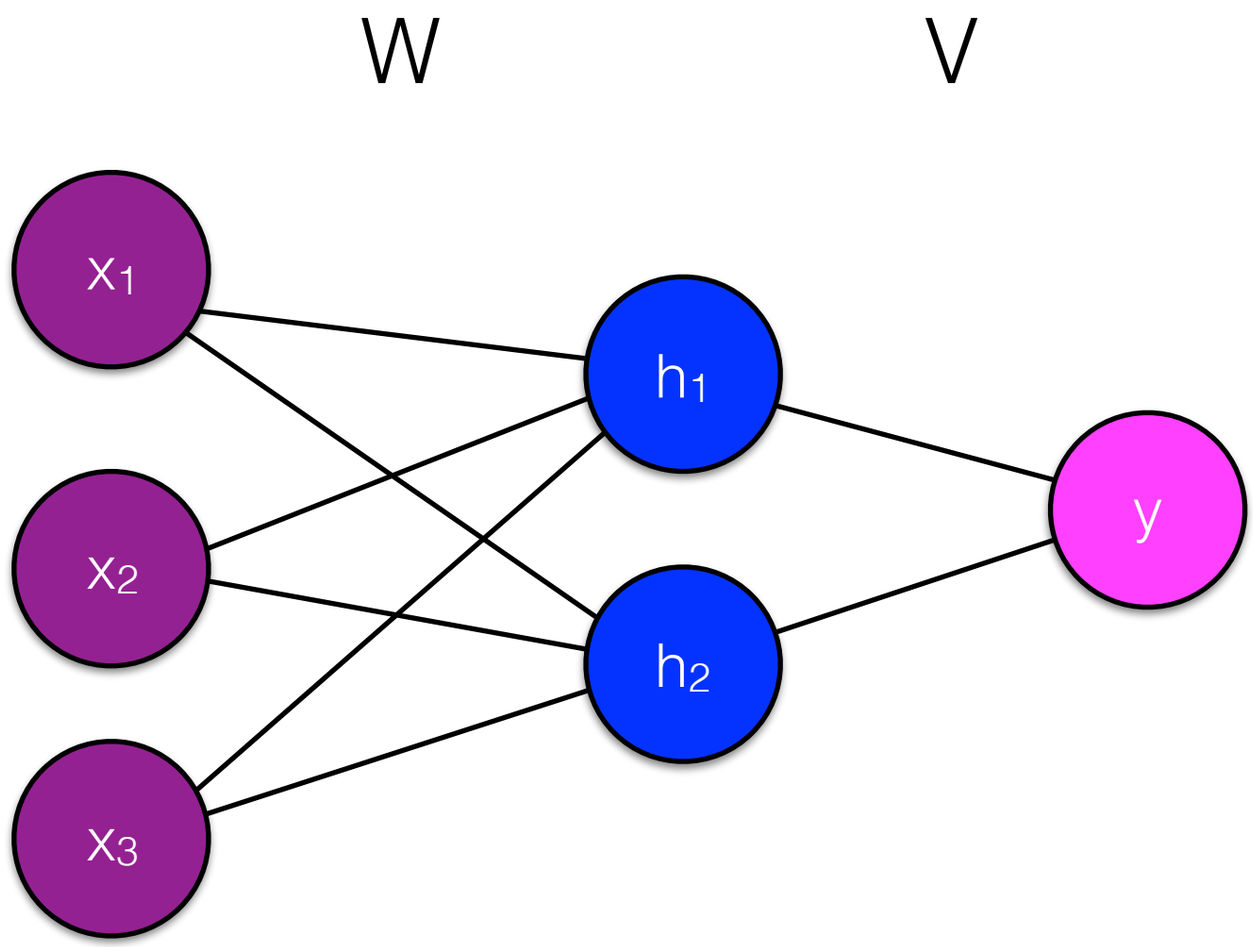
$$\hat{y} = \sigma\left[V_1\left(\sigma\left(\sum_i^F x_i W_{i,1}\right)\right) + V_2\left(\sigma\left(\sum_i^F x_i W_{i,2}\right)\right)\right]$$

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

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

The loss is another function chained on top

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

Chain rule

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

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

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

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

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

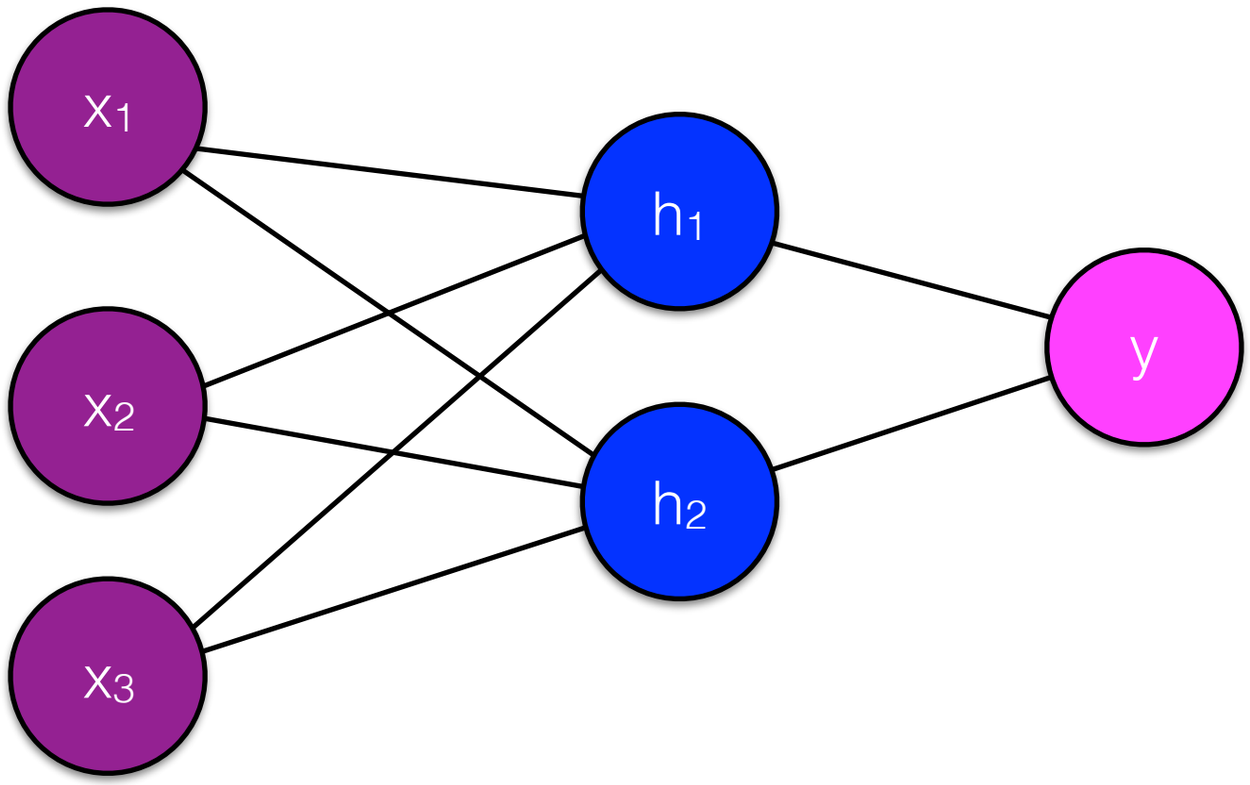
$$= \left(1 - \hat{y}\right)h$$

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.

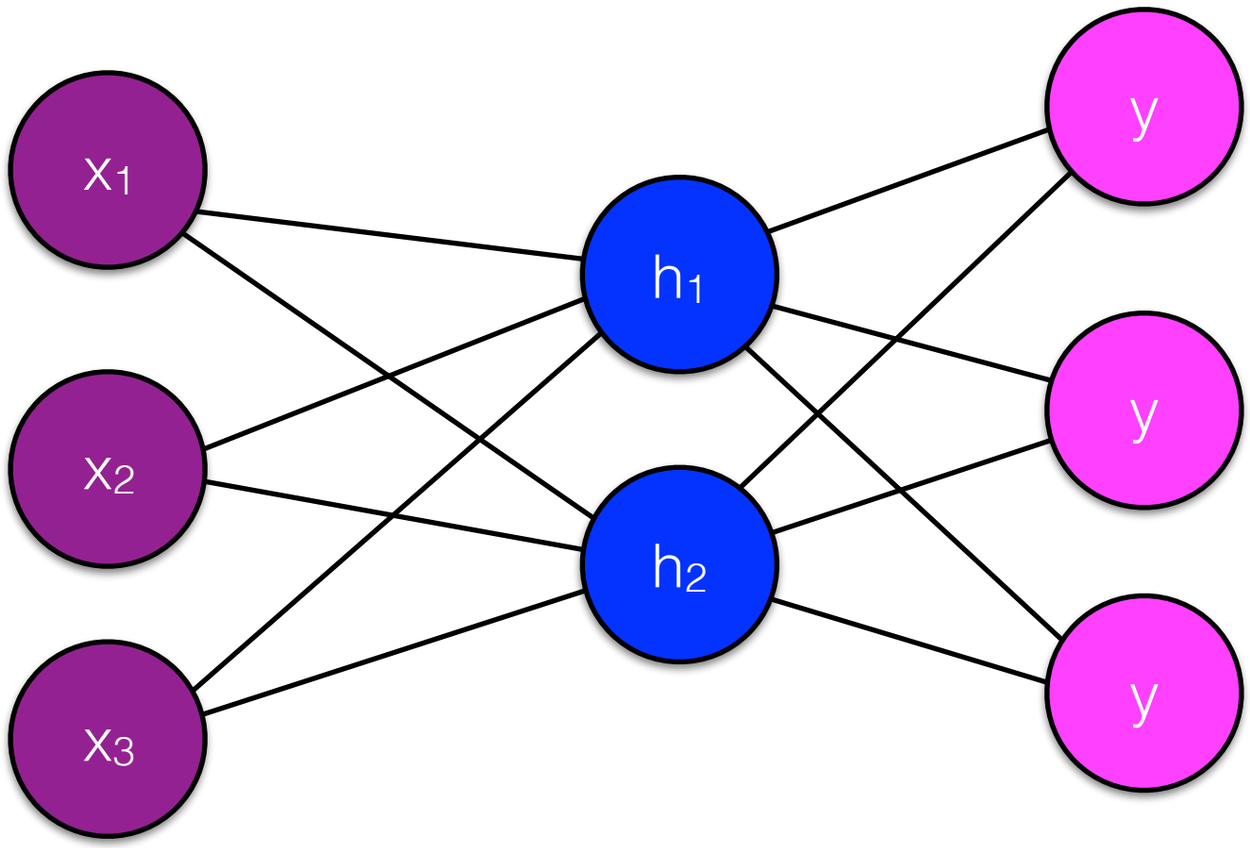
Neural network structures



1

Output one real value

Neural network structures



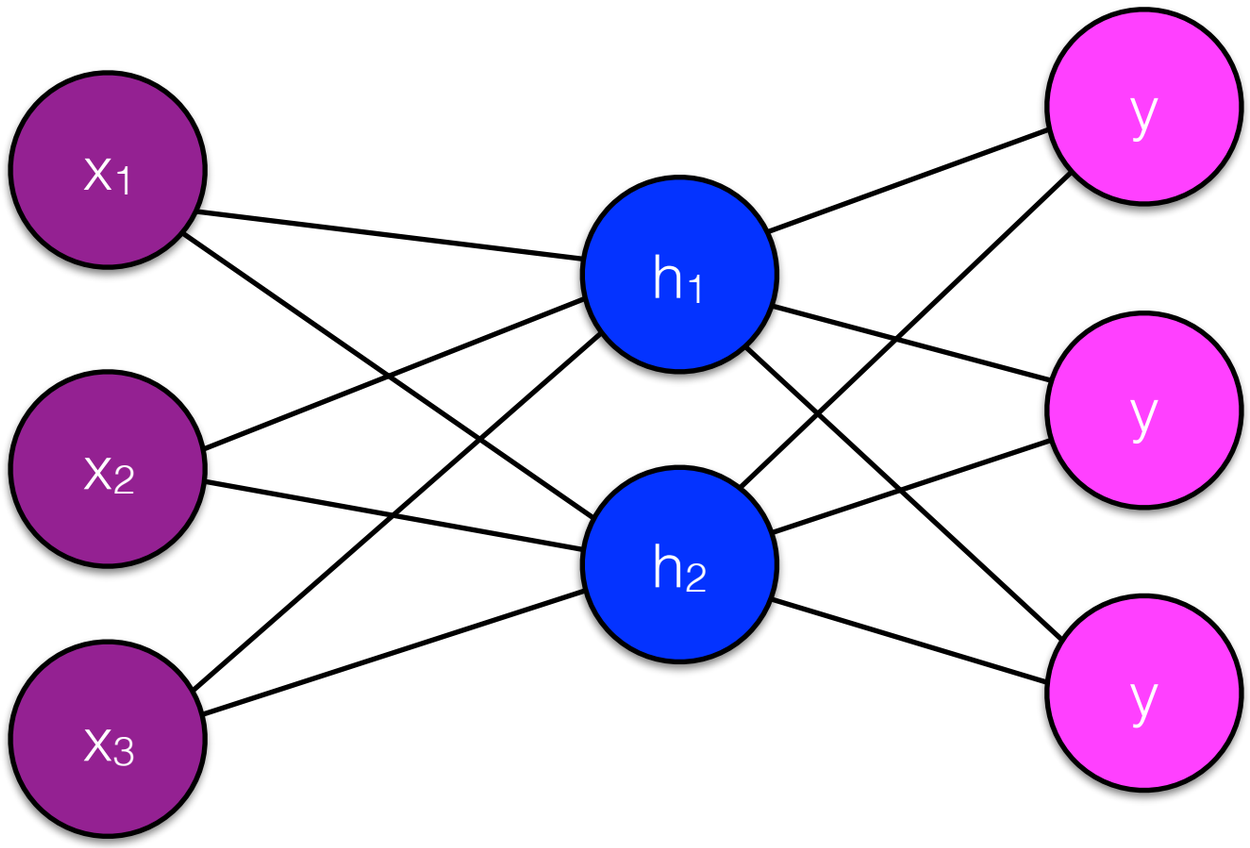
0

1

0

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

Neural network structures



1

1

0

output 3 values, several = 1 in training data

Regularization

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

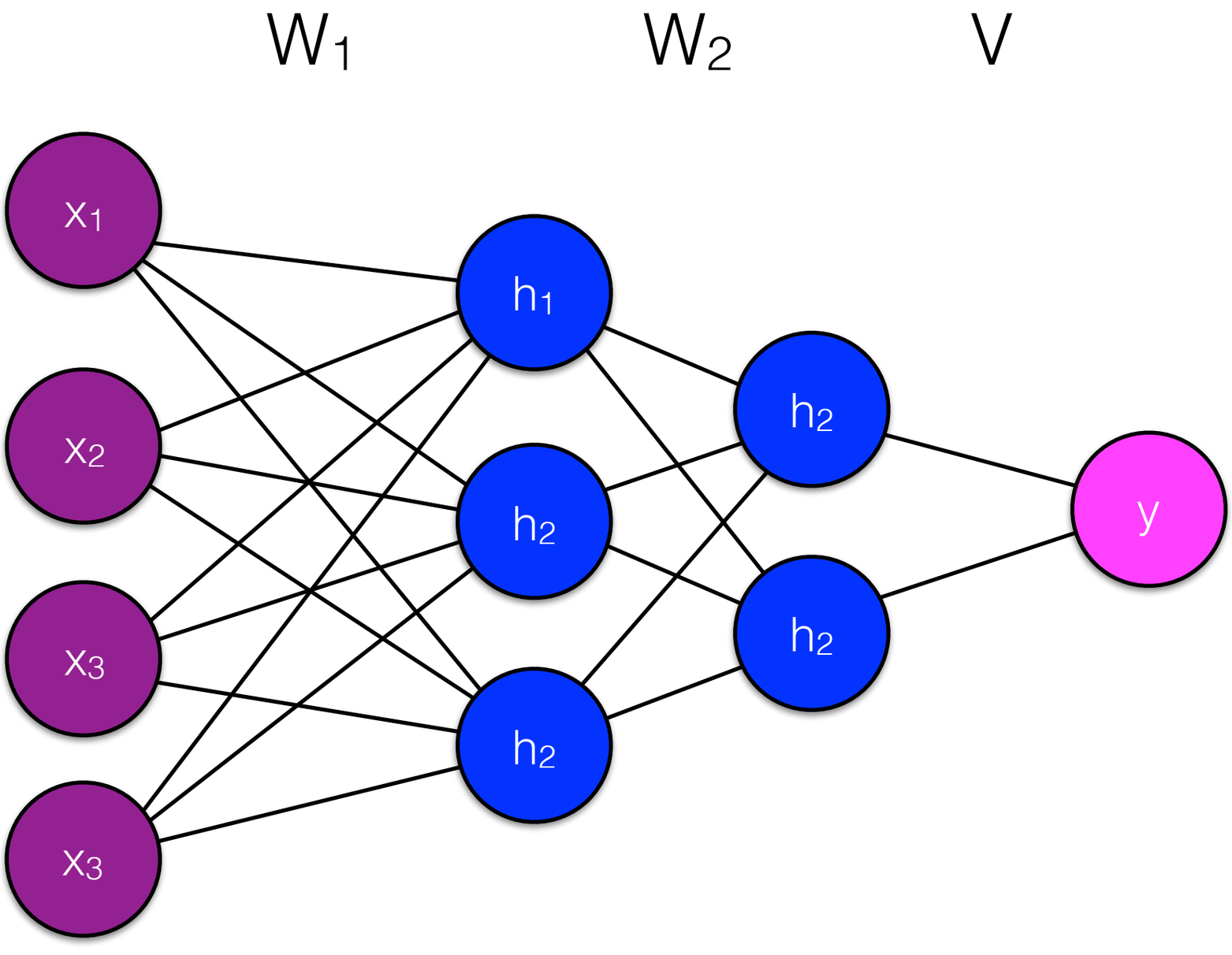
Regularization

- L2 regularization: penalize W and V for being too large

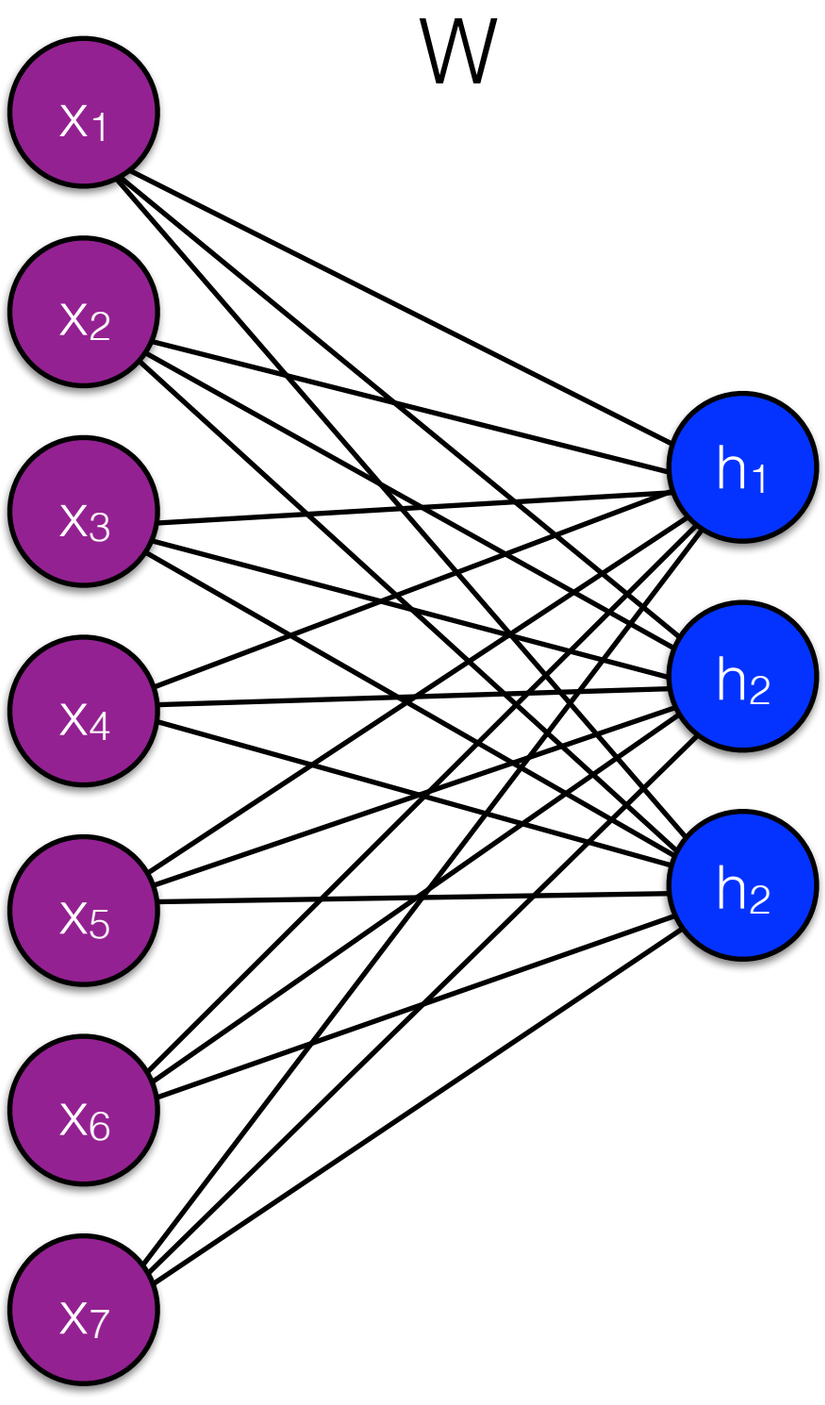
- Dropout: when training on a <x,y> pair, randomly remove some node and weights.

- Early stopping: Stop backpropagation before the training error is too small.

Deeper networks

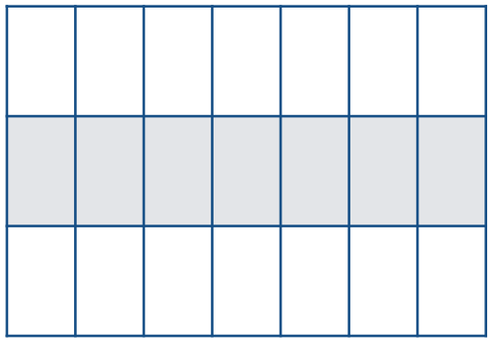


Densely connected layer

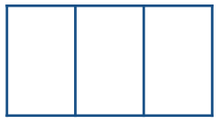


x

W



h



$$h = \sigma(xW)$$

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	0	0	0	0

blurring

http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/
https://docs.gimp.org/en/plug-in-convmatrix.html

1D Convolution

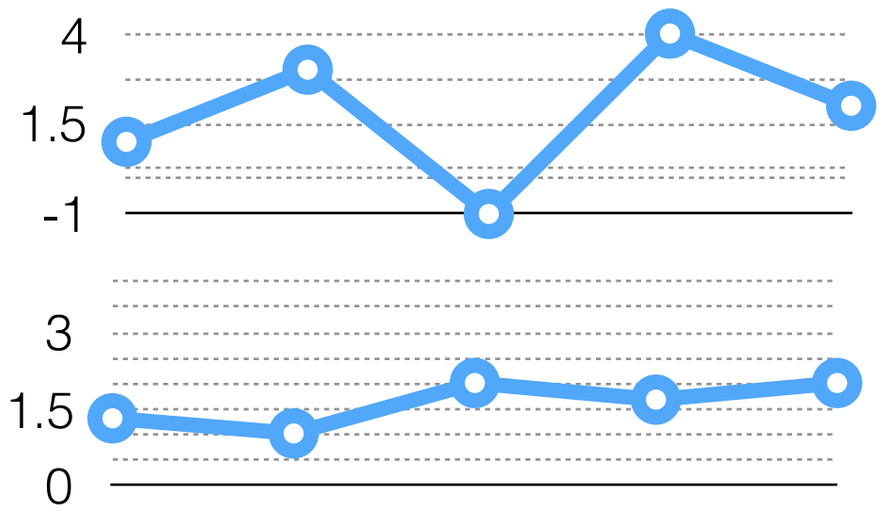
convolution K

1/3	1/3	1/3

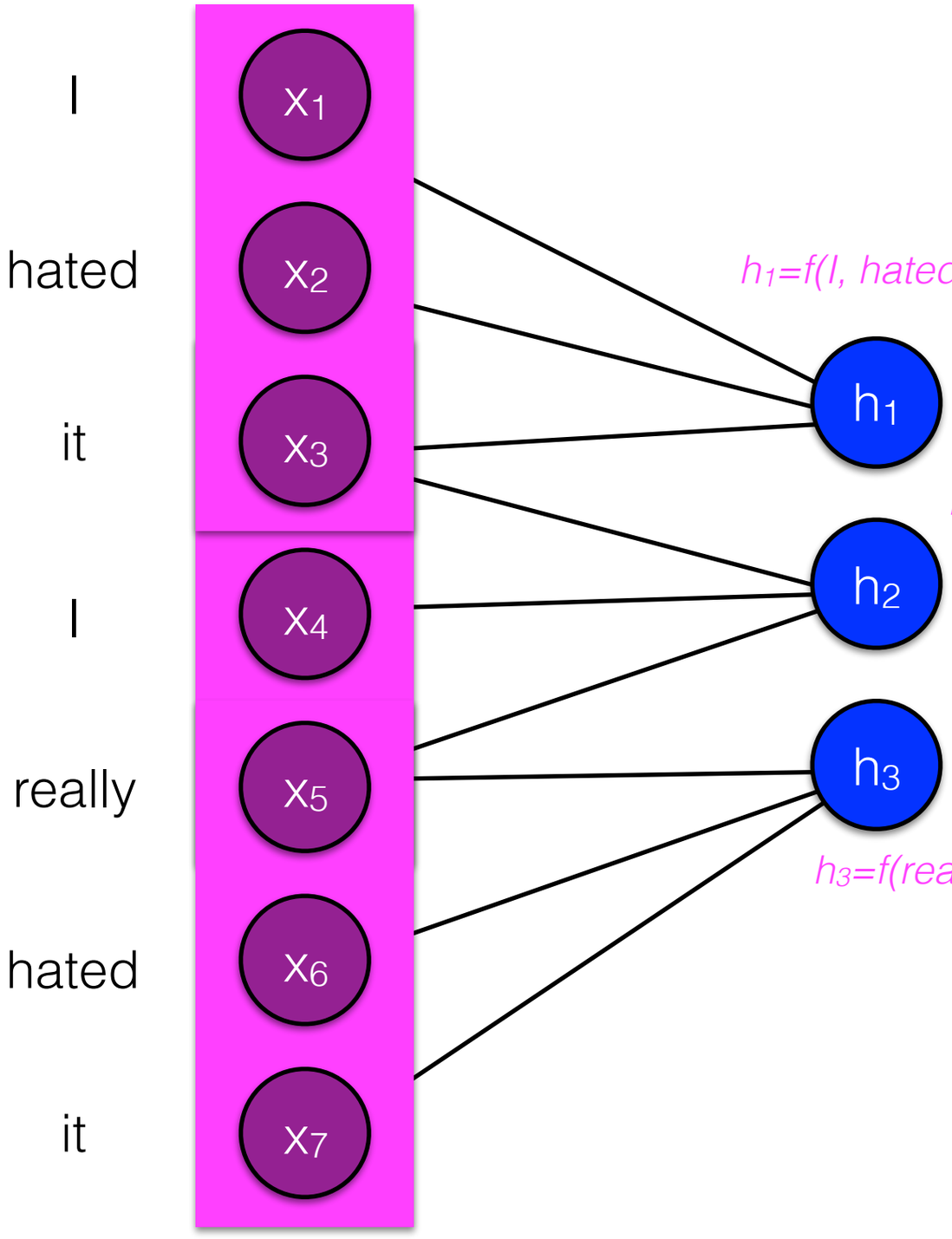
x

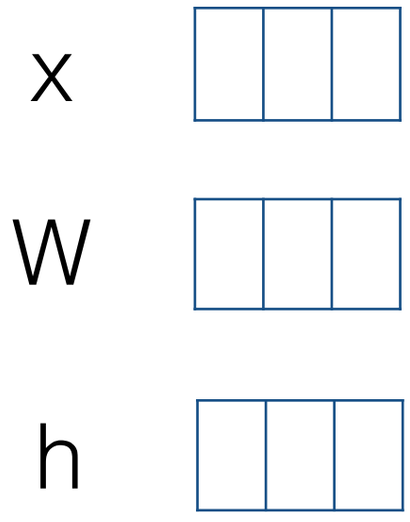
0	1	3	-1	4	2	0

1⅓	1	2	1⅔	2



Convolutional networks





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

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

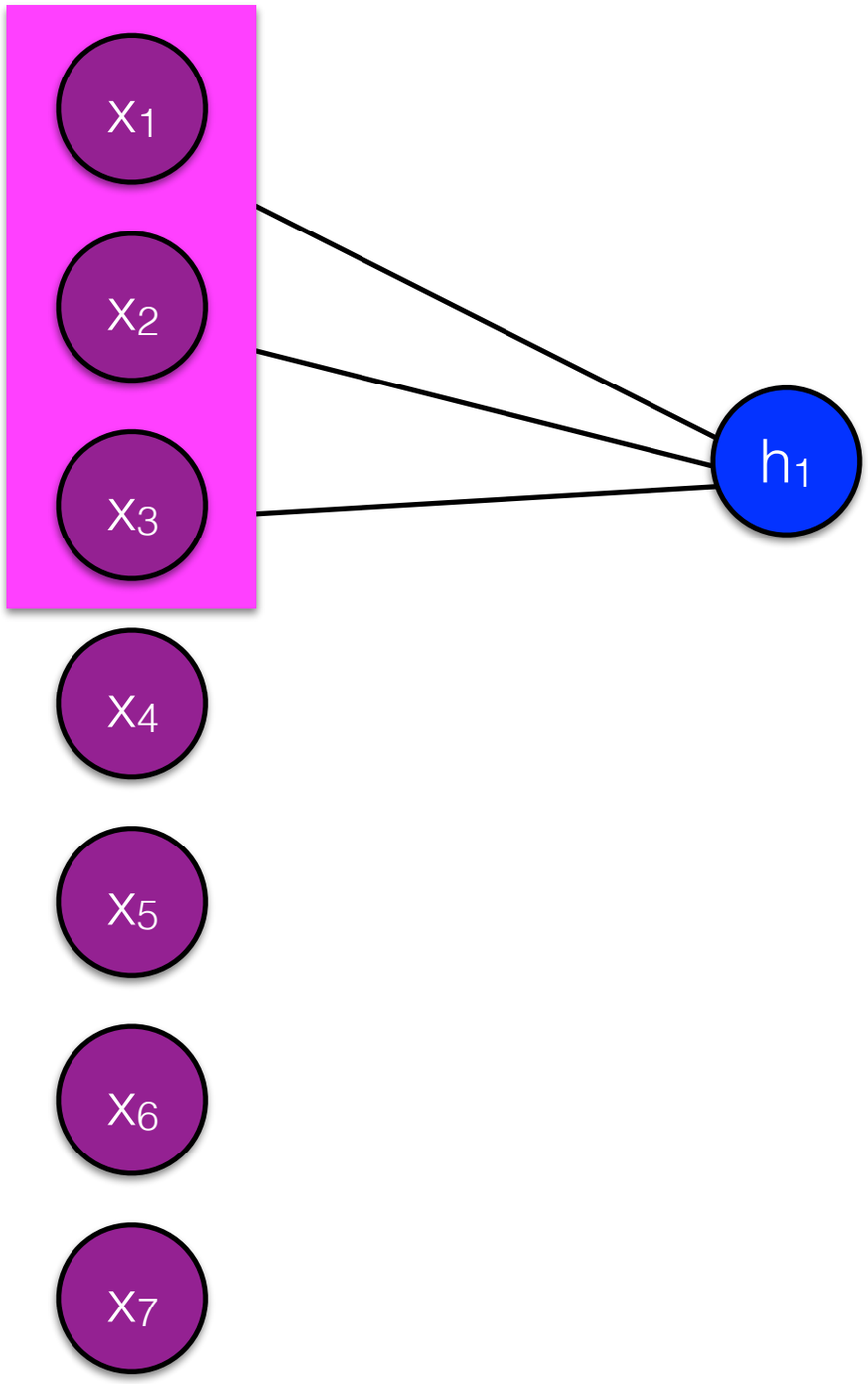
$$h_3 = \sigma(x_5 W_1 + x_6 W_2 + x_7 W_3)$$

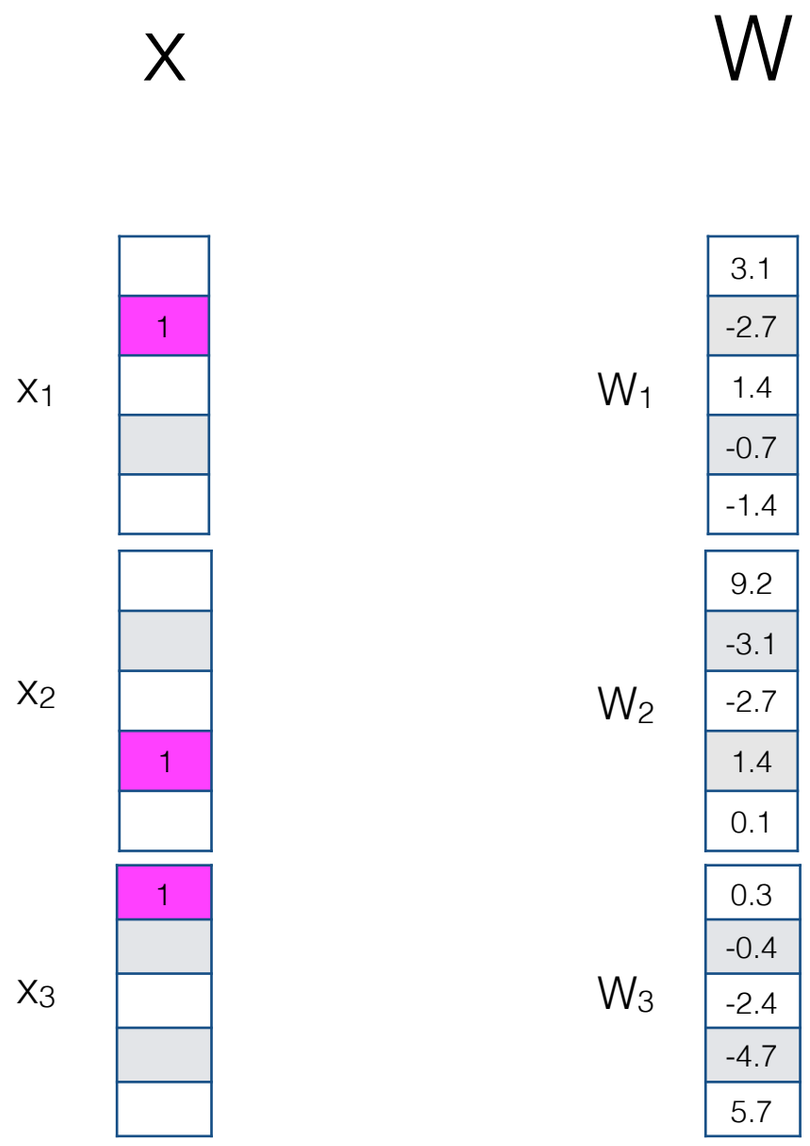
Indicator vector

- 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 on 2/13

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

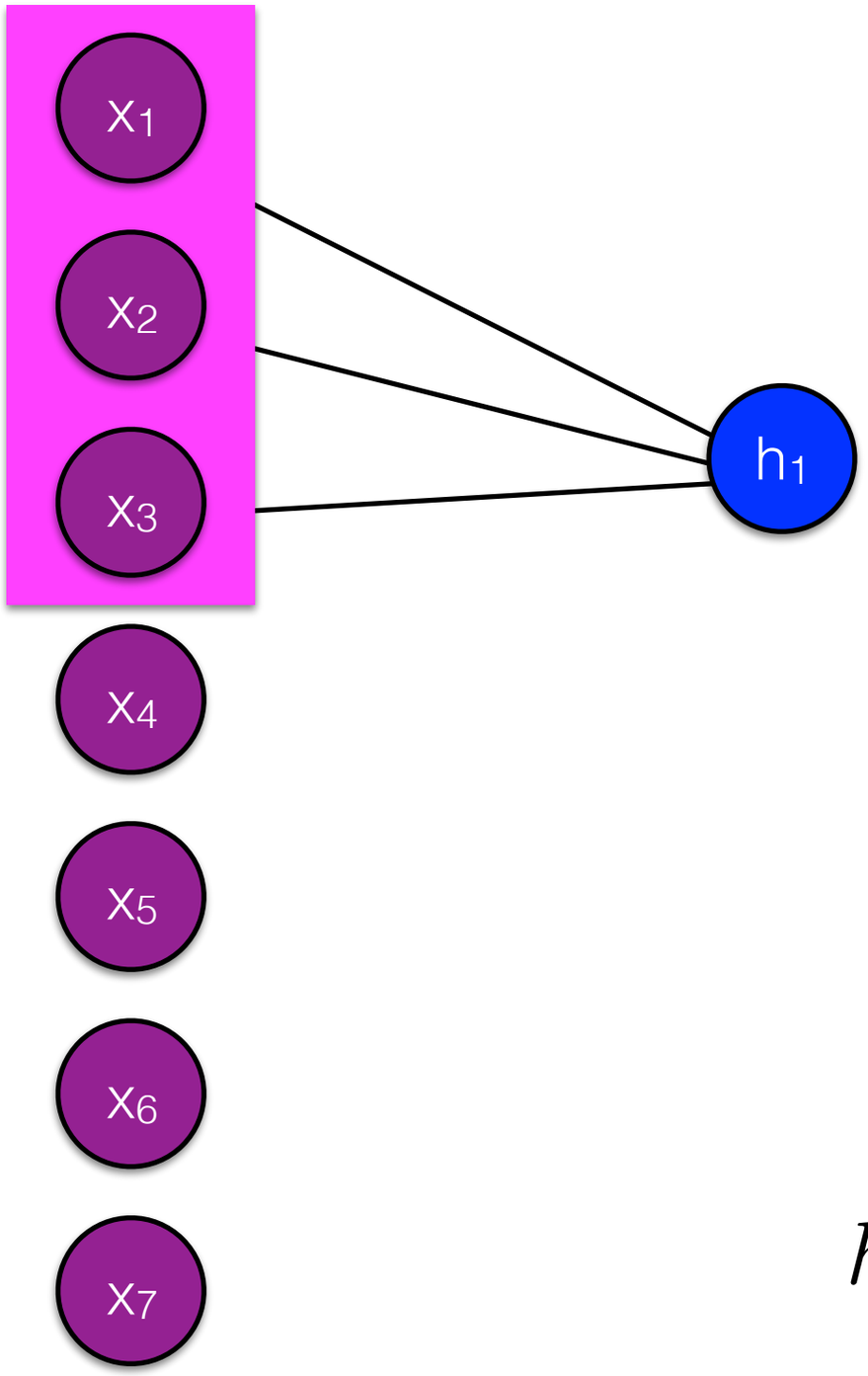


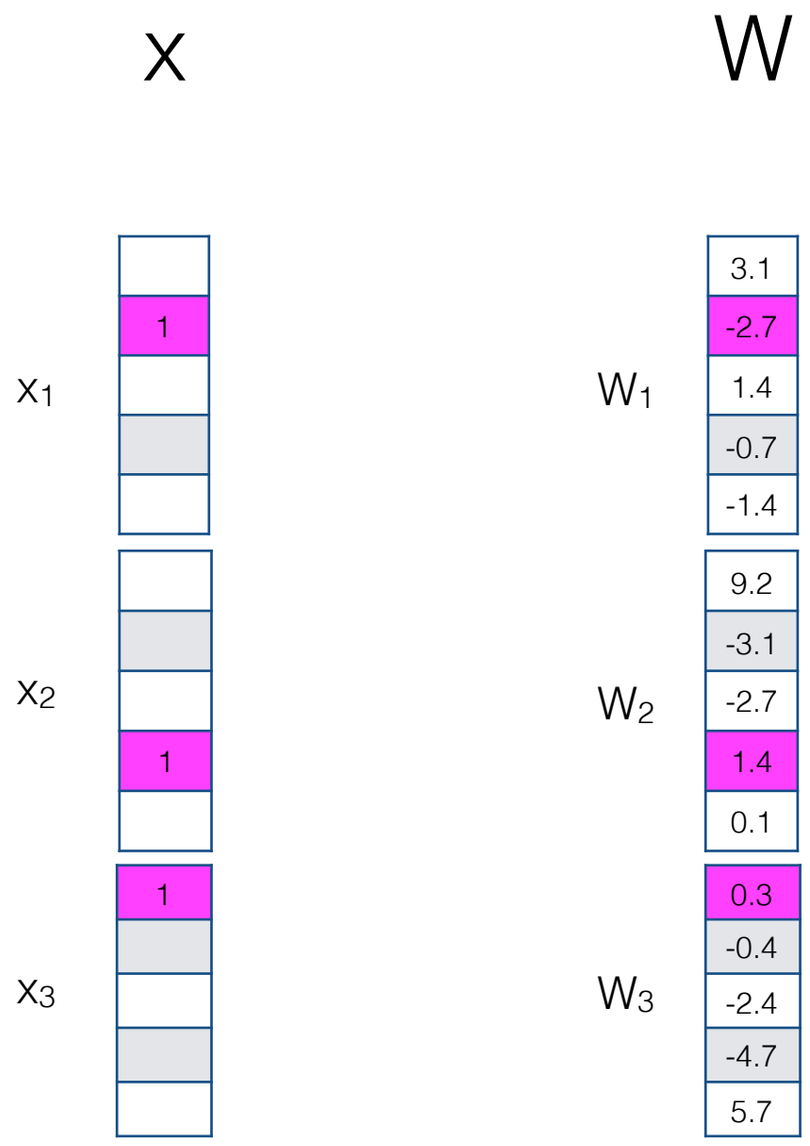


$$h_1 = \sigma(x_1 W_1 + x_2 W_2 + x_3 W_3)$$

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

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

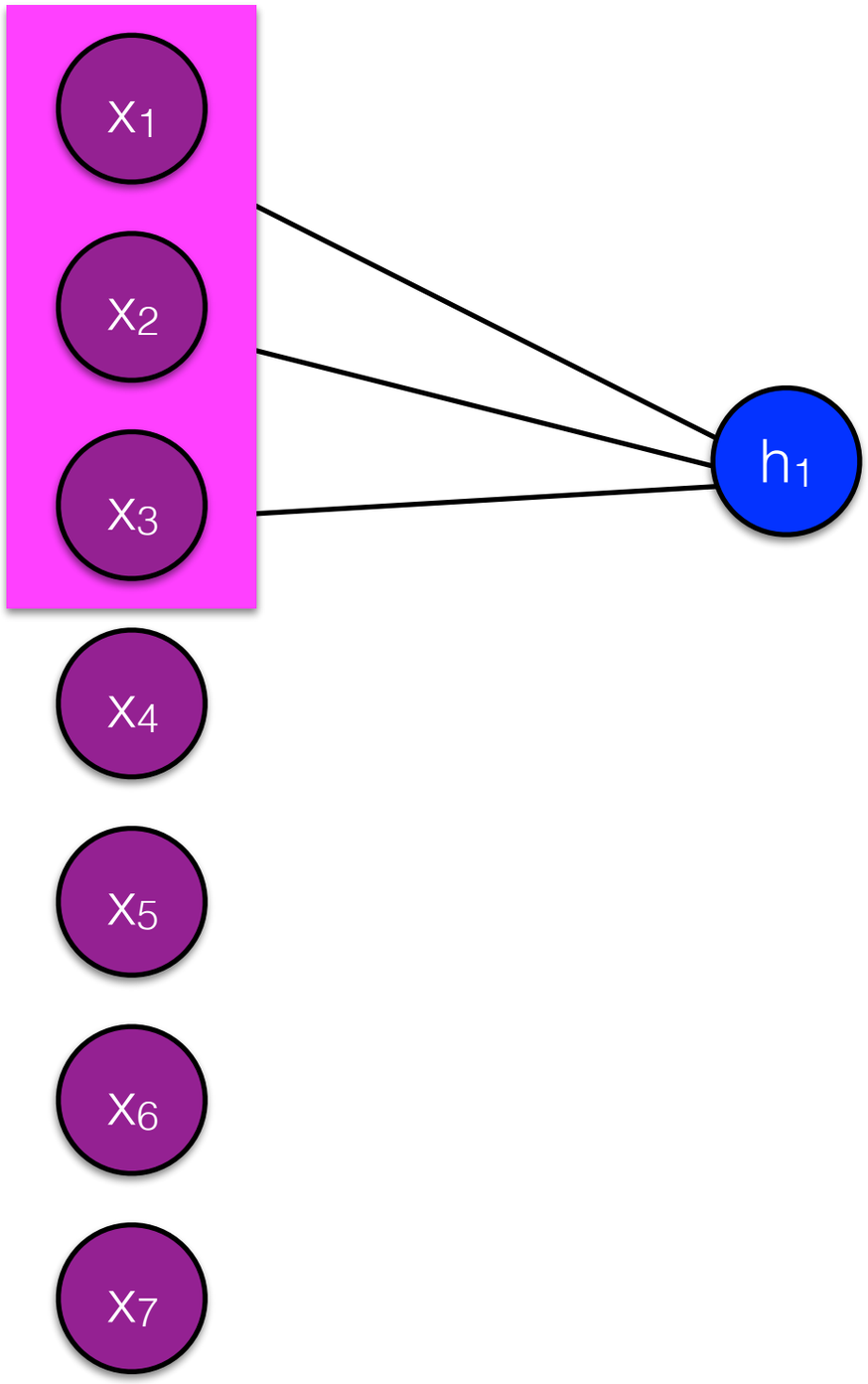




For indicator vectors, we're just adding these numbers together

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

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



x

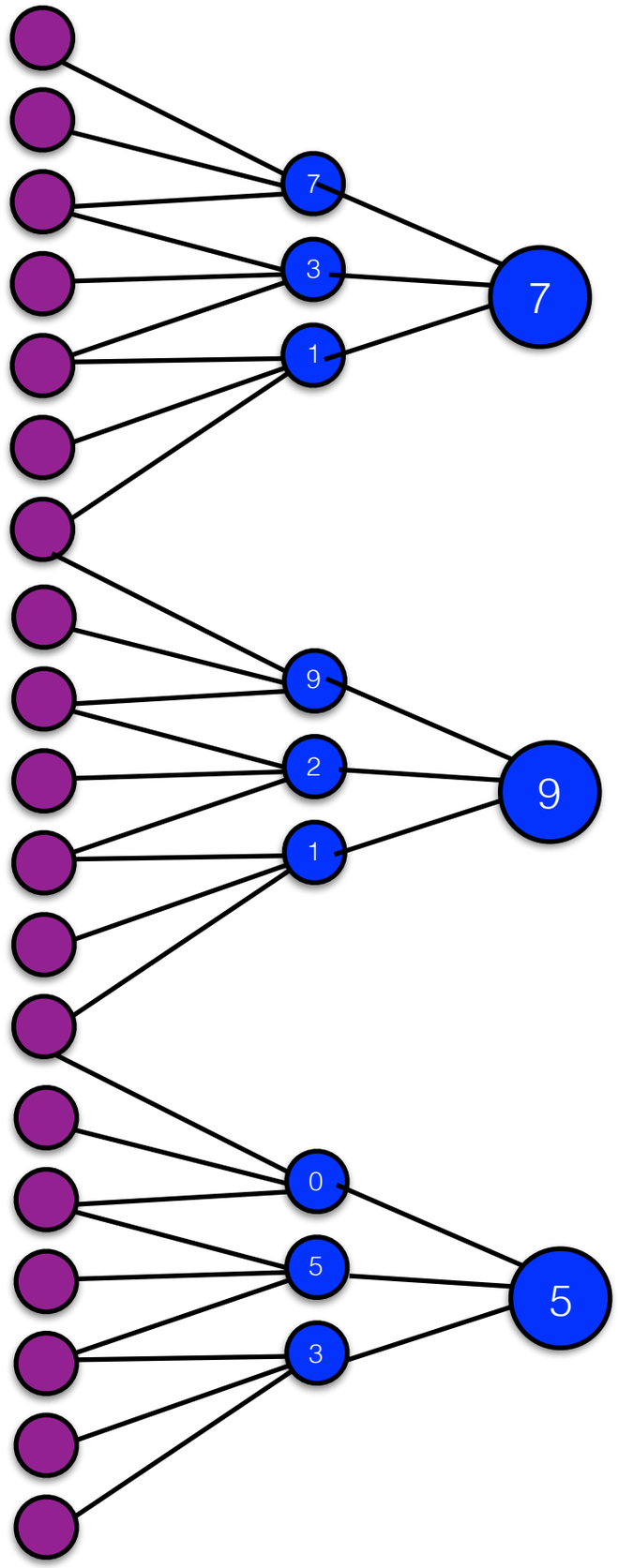
x_1	0.4
	0.8
	1.2
	-1.3
	0.4
x_2	0.2
	-5.3
	-1.2
	5.3
	0.4
x_3	2.6
	2.7
	-3.2
	6.2
	1.9

W

W_1	3.1
	-2.7
	1.4
	-0.7
	-1.4
W_2	9.2
	-3.1
	-2.7
	1.4
	0.1
W_3	0.3
	-0.4
	-2.4
	-4.7
	5.7

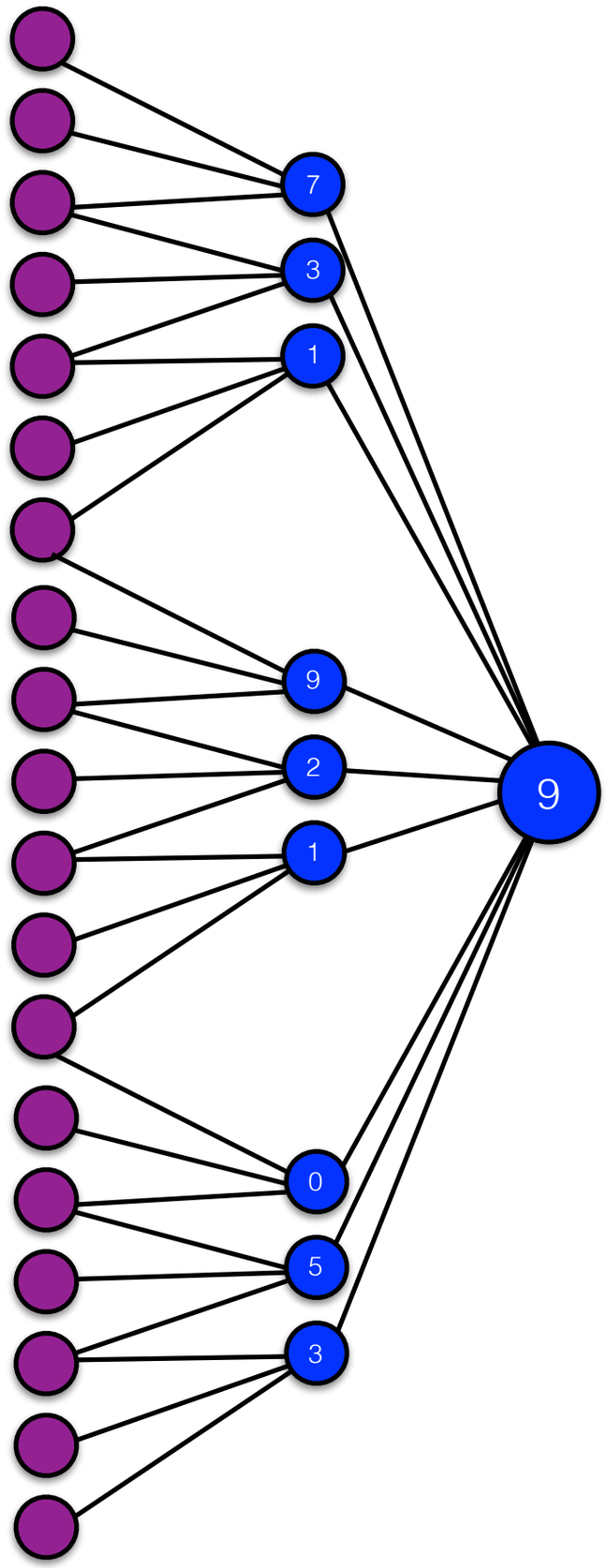
For dense input vectors (e.g., embeddings), full dot product

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



Pooling

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



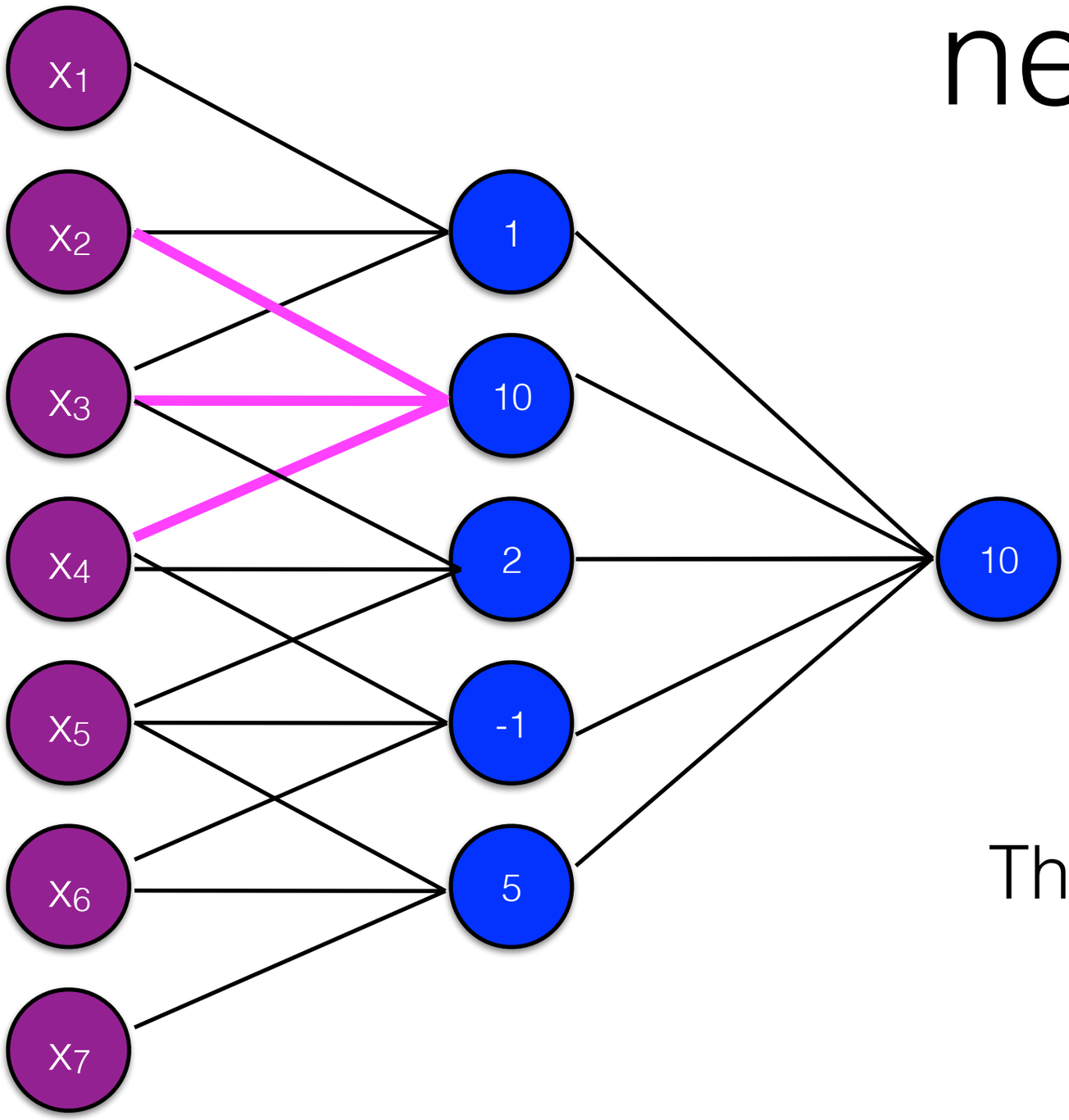
Global pooling

- Down-samples a layer by selecting a single point from some set

- Max-pooling over time (global max pooling) selects the largest value over an entire sequence

- Very common for NLP problems.

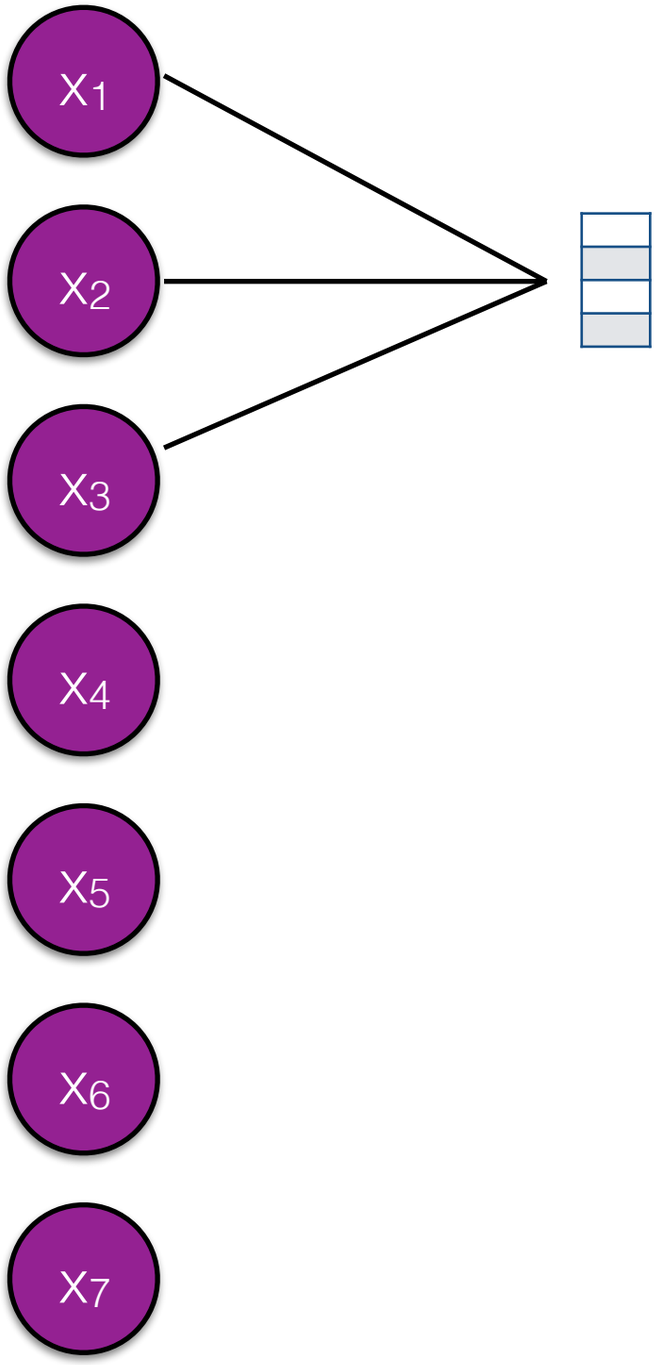
Convolutional networks

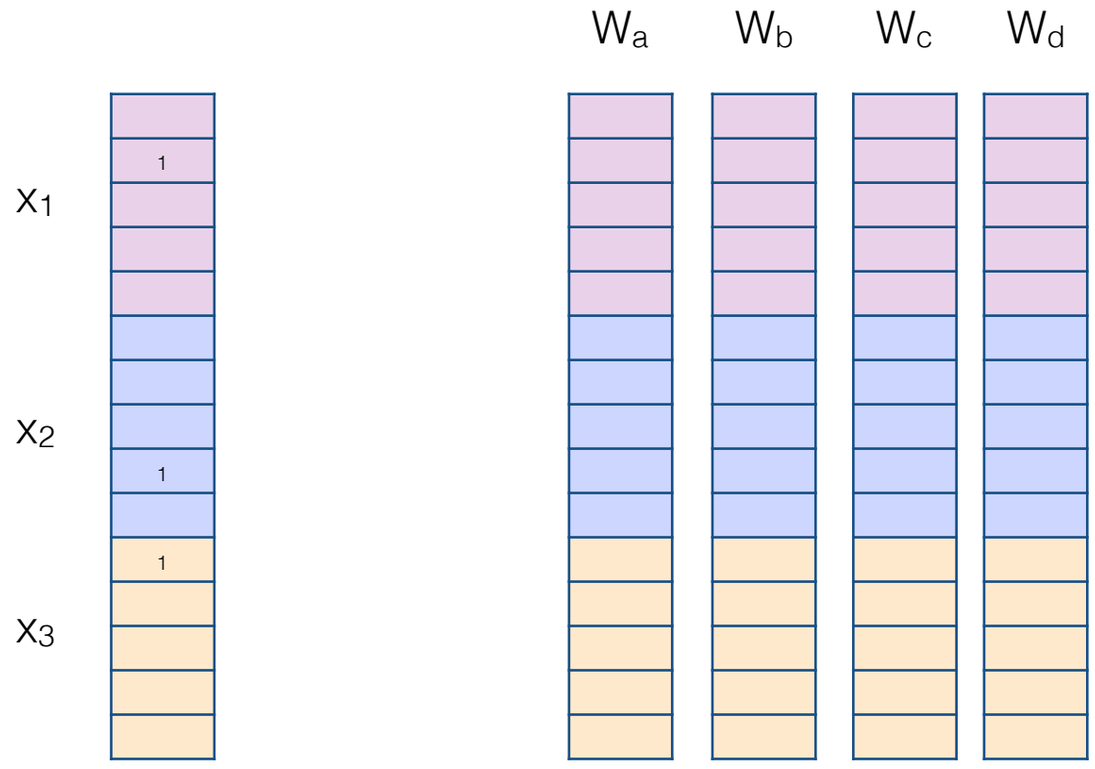


This defines one filter.

convolution max pooling

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

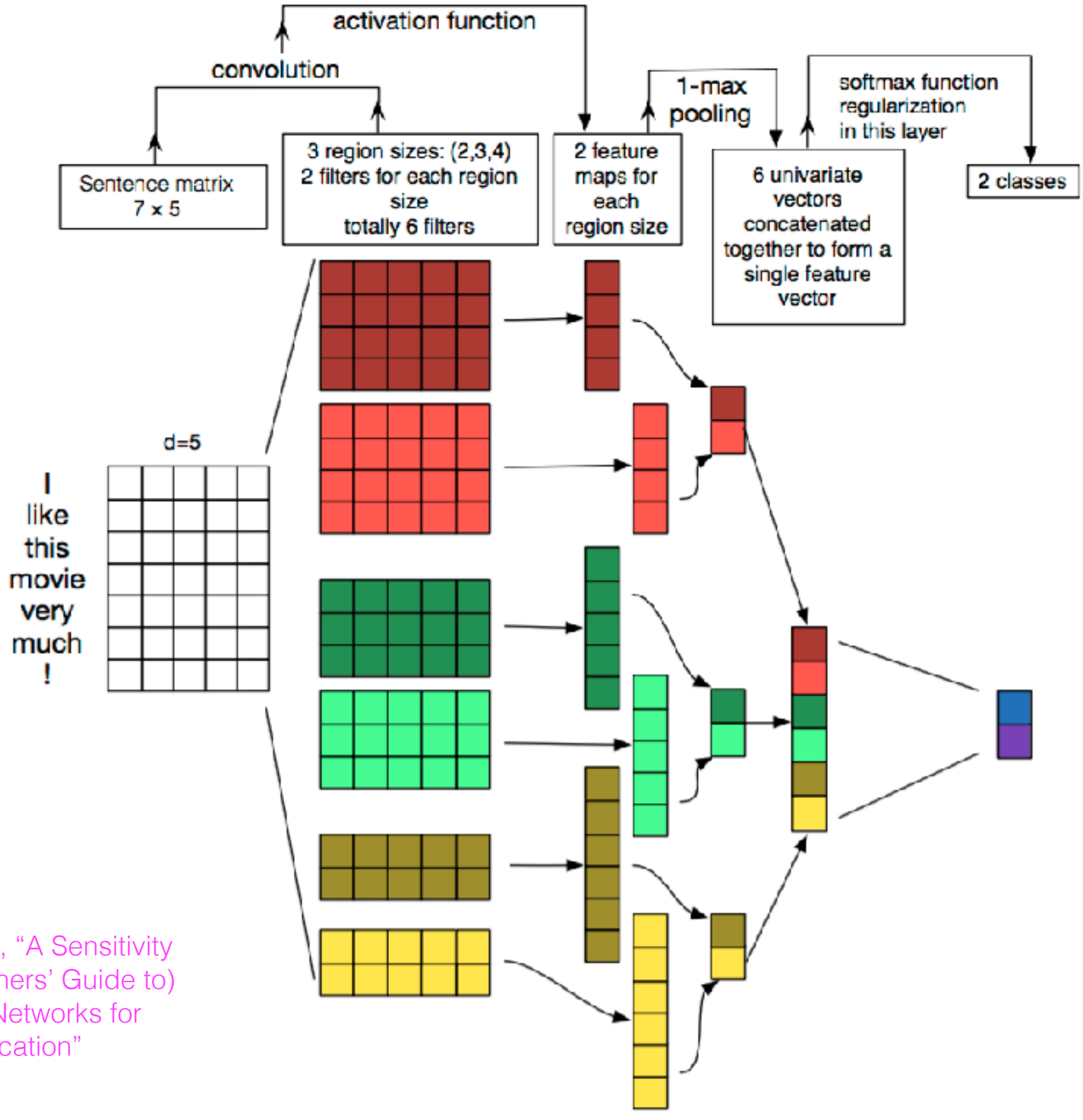




$$h_1 = \sigma(x^\top W) \in R^4$$

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

Unique ngrams (1-4) in Cornell movie review dataset

259 project proposal

due 2/18

- 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!

Tuesday

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