Natural Language Processing

Info 159/259

Lecture 4: Text classification with Neural Networks

Many slides & instruction ideas borrowed from:

David Bamman, Sofia Serrano, Dan Jurafsky & Mohit Iyyer

Logistics

- Quiz 1 is due tonight (11:59 pm)
    - Quiz 2 will be out this Friday (due next Monday Feb 5).
- Homework 1 is out & due next Tuesday, Feb 6 (11:59 pm)
    - Homework 2 will be out early next week.
- Info about the Annotation Project and final project/survey will be released gradually starting next week.
- Tonight Lecture: Text Classification with Neural Networks

Binary logistic regression

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

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

Multiclass logistic regression

$$P(Y = y \mid X = x; \beta) = \frac{\exp(x^\top \beta_y)}{\sum_{y' \in \mathcal{Y}} \exp(x^\top \beta_{y'})}$$

output space $\mathcal{Y} = \{1, \ldots, K\}$

Features

- As a discriminative classifier, logistic regression doesn't assume features are independent.
- 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 conditional log likelihood:

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

Algorithm 2 Logistic regression stochastic gradient descent

Data: training data $x \in \mathbb{R}^F, y \in \{0,1\}$
$\beta = 0^F$
while not converged **do**
  for $i = 1$ to N **do**
    $\beta_{t+1} = \beta_t + \alpha \left(y_i - \hat{p}(x_i)\right) x_i$
  end for
end while

L2 regularization

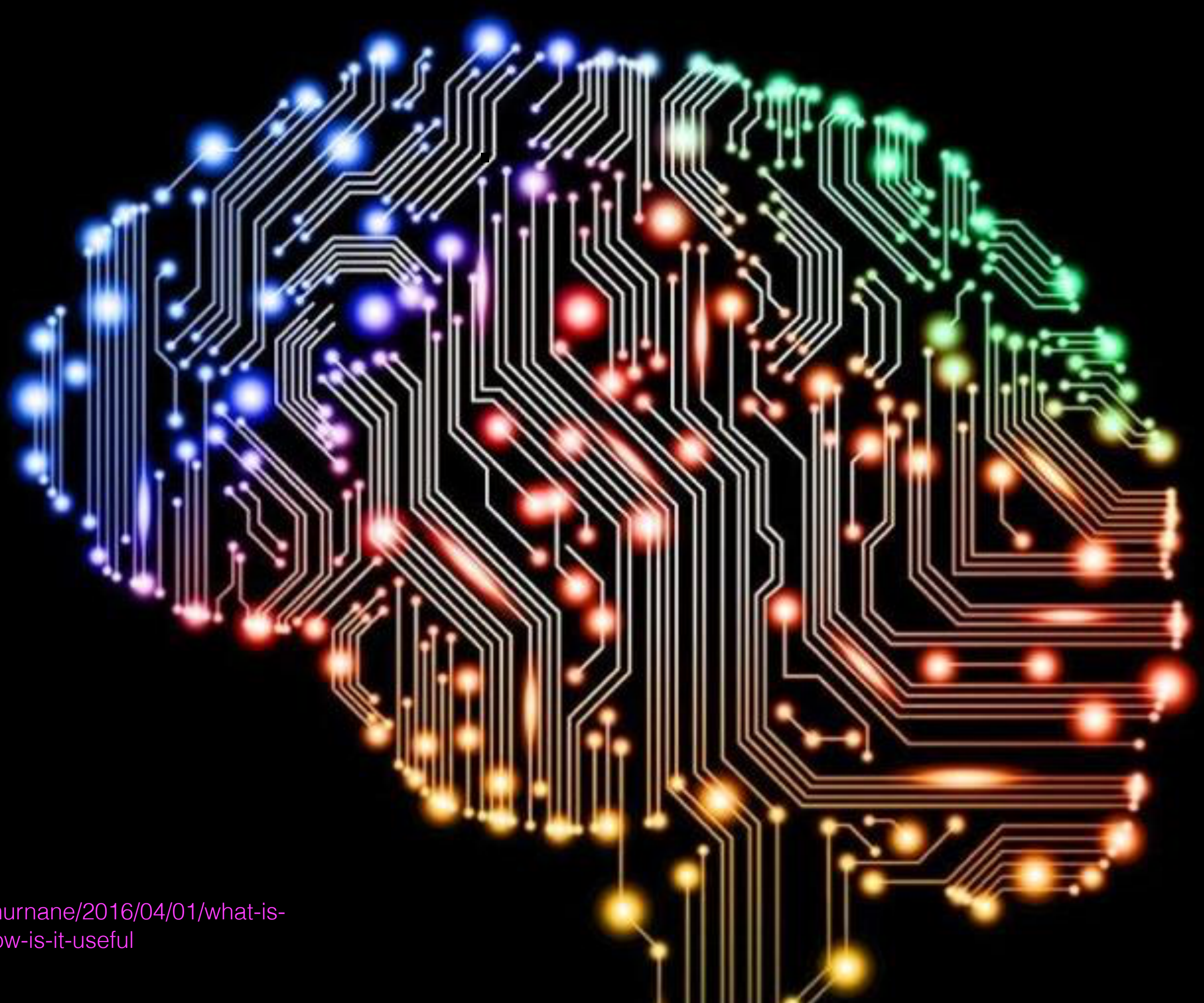
$$\ell(\beta) = \underbrace{\sum_{i=1}^{N} \log P(y_i \mid x_i, \beta)}_{\text{we want this to be high}} \quad - \quad \underbrace{\eta \sum_{j=1}^{F} \beta_j^2}_{\text{but we want this to be small}}$$

- 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

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

- 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)



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)
- Neural networks (~2014–today)

J&M 2008, ch 1

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]
- (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

TensorFlow

PYTORCH

K

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

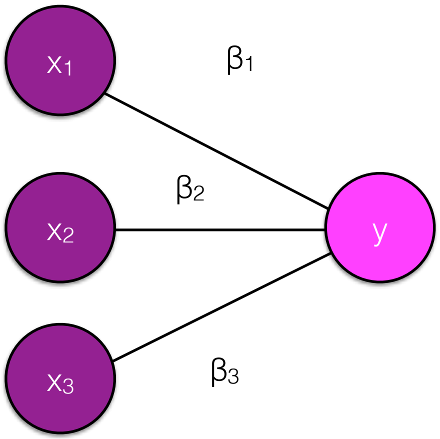
Algorithm 2 Logistic regression stochastic gradient descent

Data: training data $x \in \mathbb{R}^F, y \in \{0, 1\}$
$\beta = 0^F$
while not converged **do**
    for $i = 1$ to N **do**
        $\beta_{t+1} = \beta_t + \alpha \left(y_i - \hat{p}(x_i) \right) x_i$
    end for
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

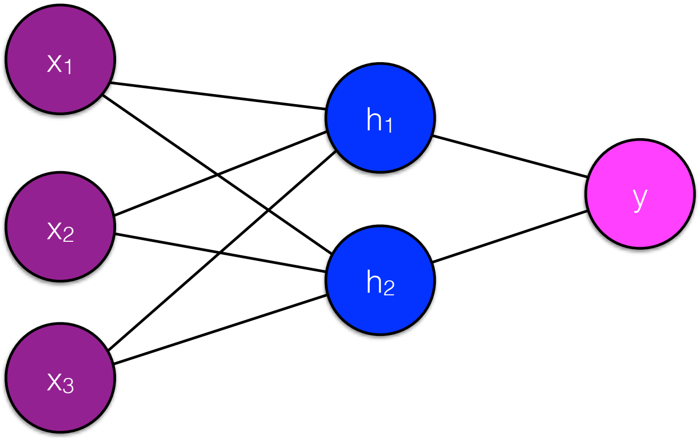
$$P(\hat{y} = 1) = \frac{1}{1 + \exp\left(-\sum_{i=1}^{F} x_i \beta_i\right)}$$



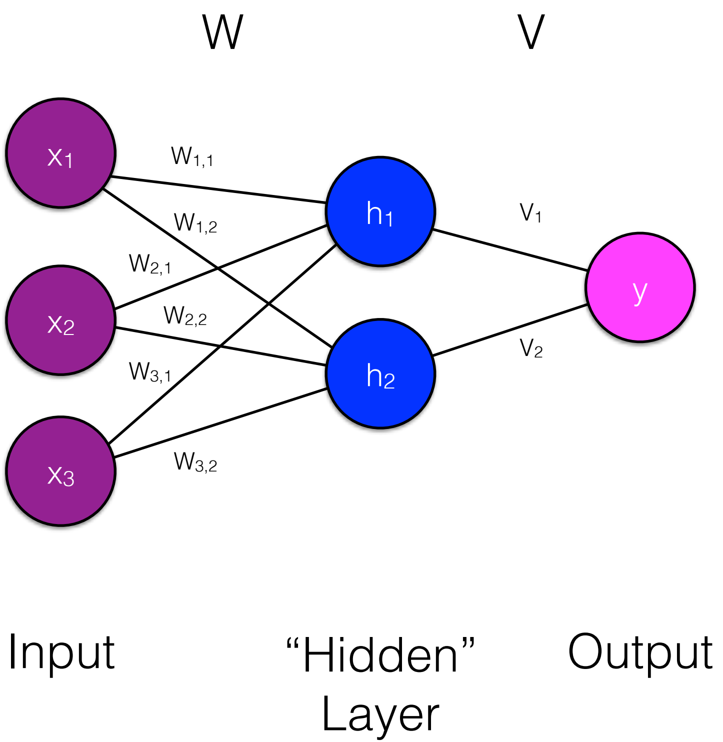
	x	β
not	1	-0.5
bad	1	-1.7
movie	0	0.3

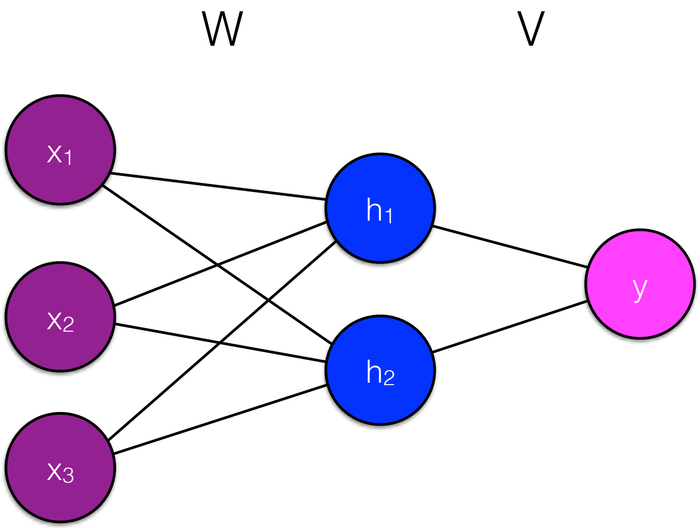
Feedforward neural network

- Input and output are mediated by at least one hidden layer.
- a.k.a. Multi Layer Perceptron (MLP)

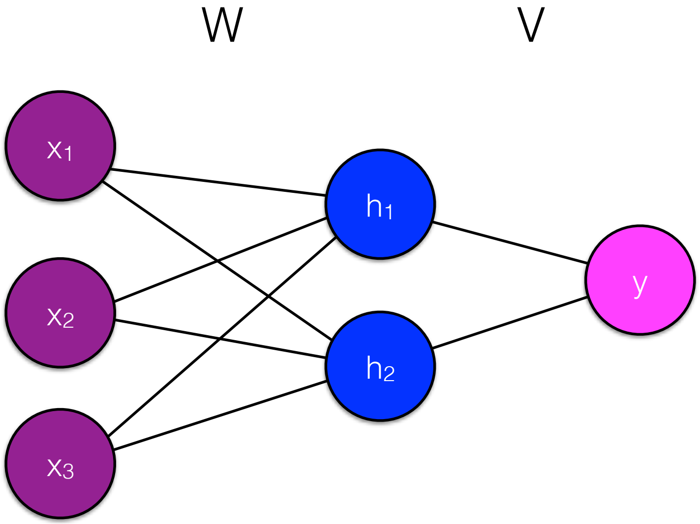


*For simplicity, we're leaving out the bias term, but assume most layers have them as well.



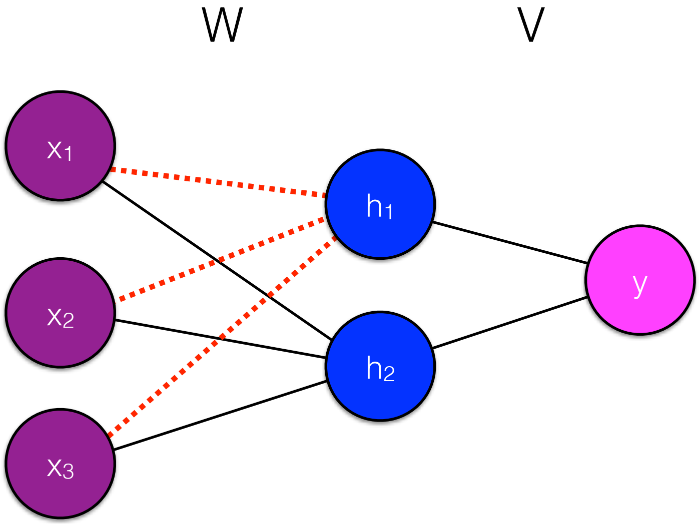


	x	W		V	y
not	1	-0.5	1.3	4.1	1
bad	1	0.4	0.08	-0.9	
movie	0	1.7	3.1		



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

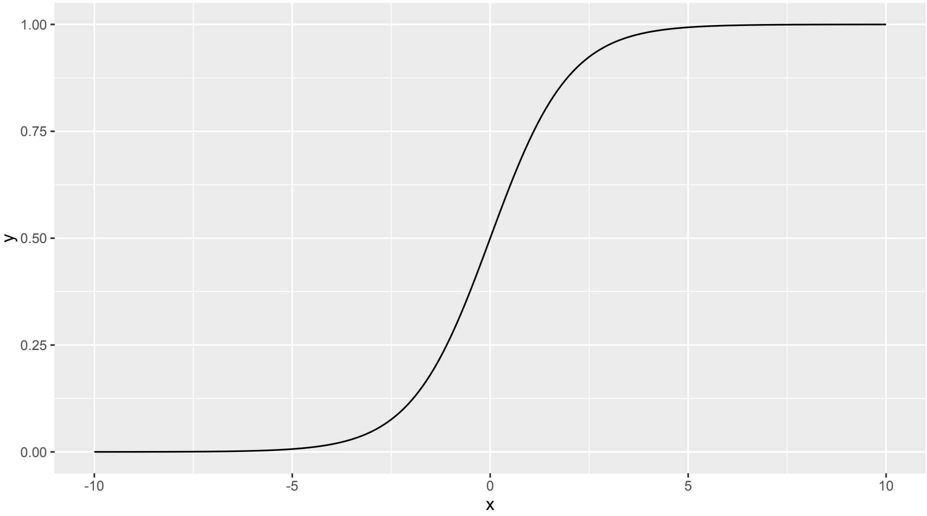
the hidden nodes are completely determined by the input and weights



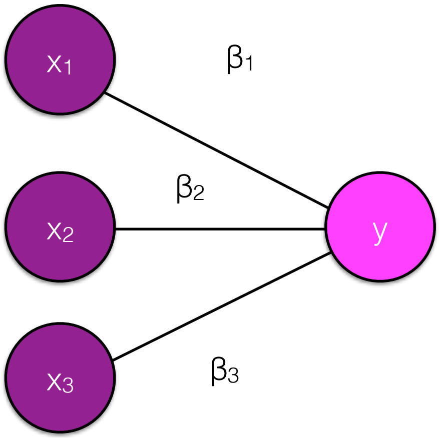
$$h_1 = f\left(\sum_{i=1}^{F} x_i W_{i,1}\right)$$

Activation functions

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



Logistic regression



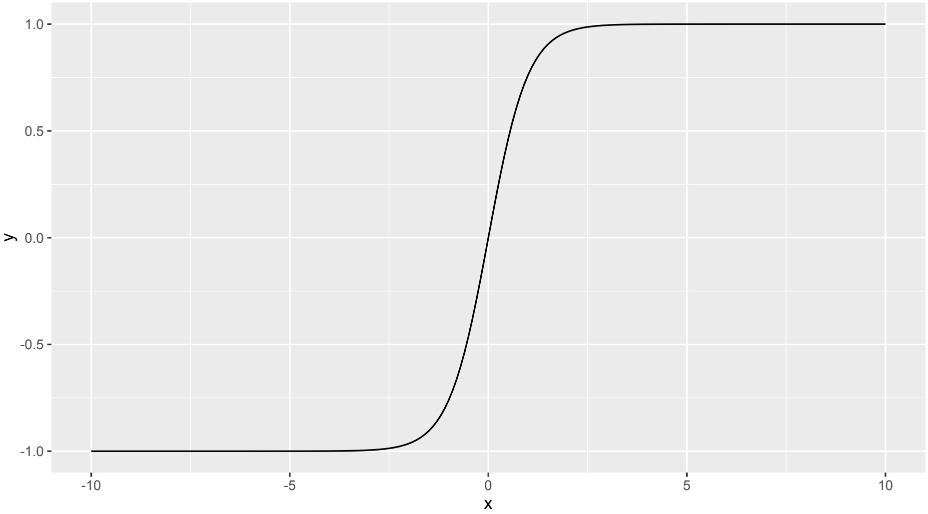
$$P(\hat{y} = 1) = \frac{1}{1 + \exp\left(-\sum_{i=1}^{F} x_i \beta_i\right)}$$

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

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

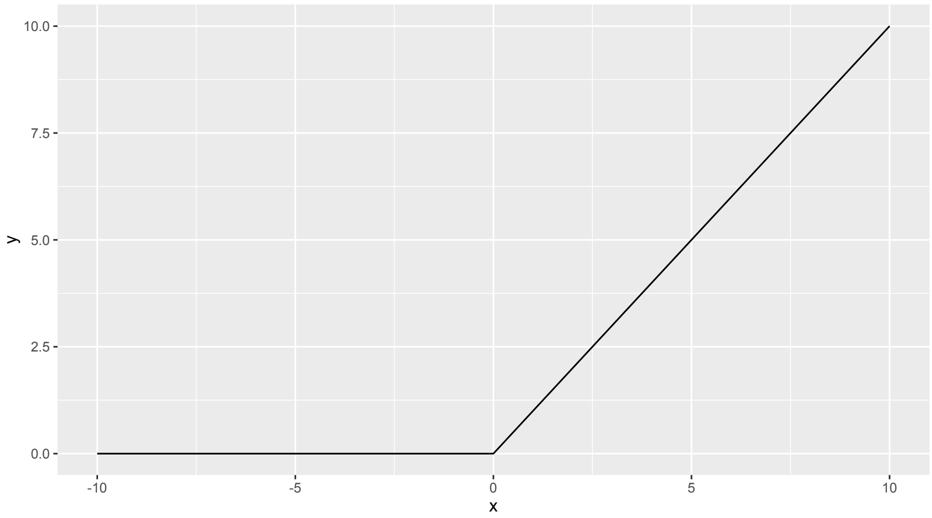
Activation functions

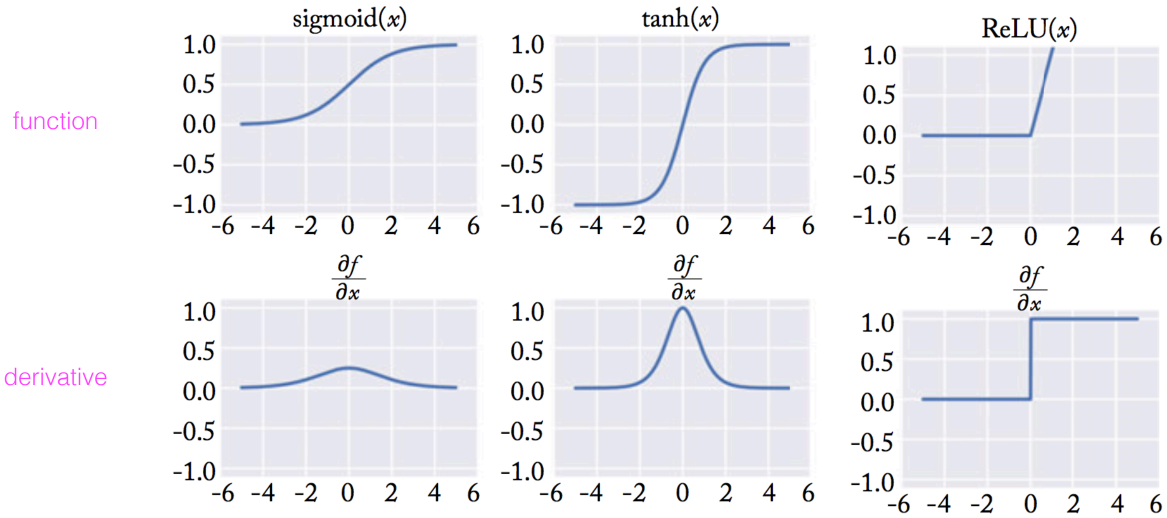
$$\tanh(z) = \frac{\exp(z) - \exp(-z)}{\exp(z) + \exp(-z)}$$



Activation functions

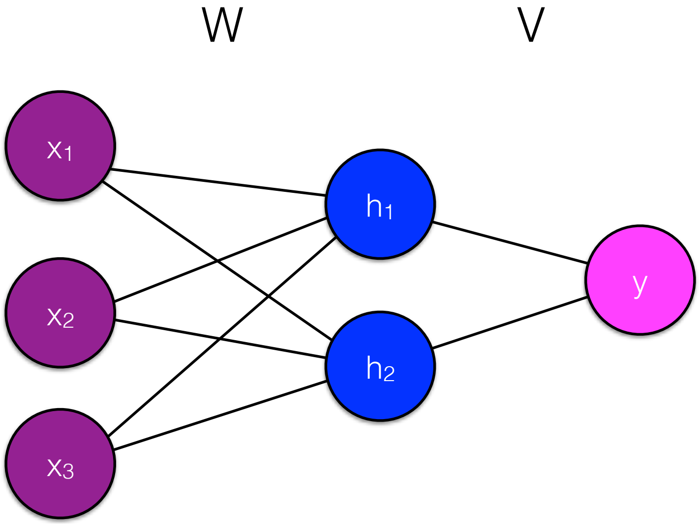
$$\text{ReLU}(z) = \max(0, z)$$





Goldberg 46

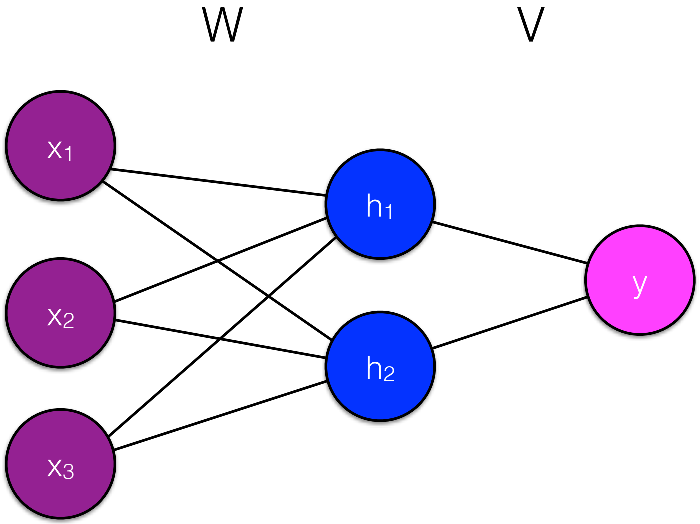
- ReLU and tanh are both used extensively in modern systems.
- Sigmoid is useful for final layer to scale output between 0 and 1, but is not often used in intermediate layers.



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

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

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



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

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.

Chain Rule (reminder)

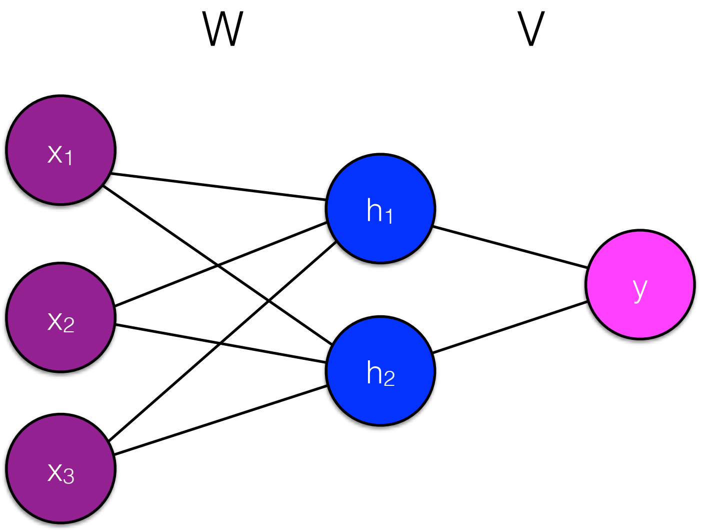
$$f(x) = u(v(x))$$

$$\frac{df}{dx} = \frac{du}{dv}\frac{dv}{dx}$$

$$f(x) = u(v(w(x)))$$

$$\frac{df}{dx} = \frac{du}{dv}\frac{dv}{dw}\frac{dw}{dx}$$

•



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

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

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

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

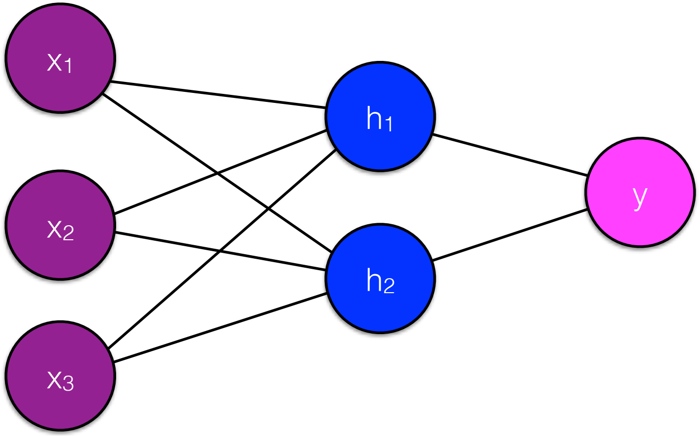
$$= (1 - \hat{y})h$$

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.

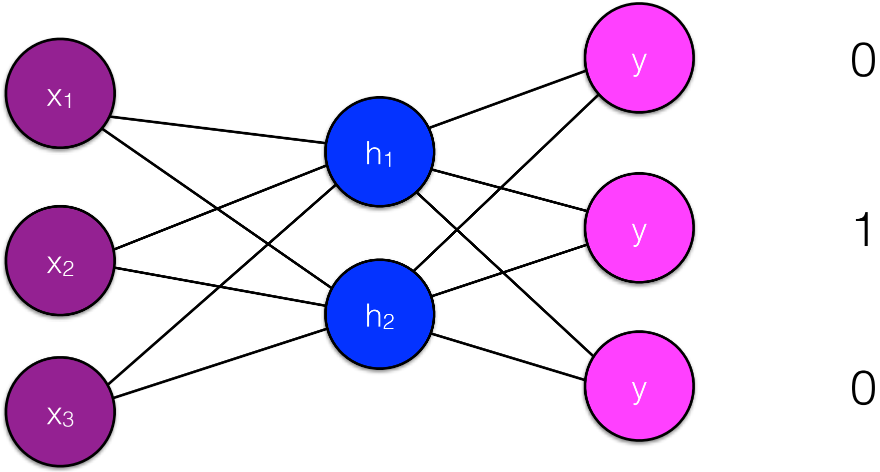
Neural network structures



1

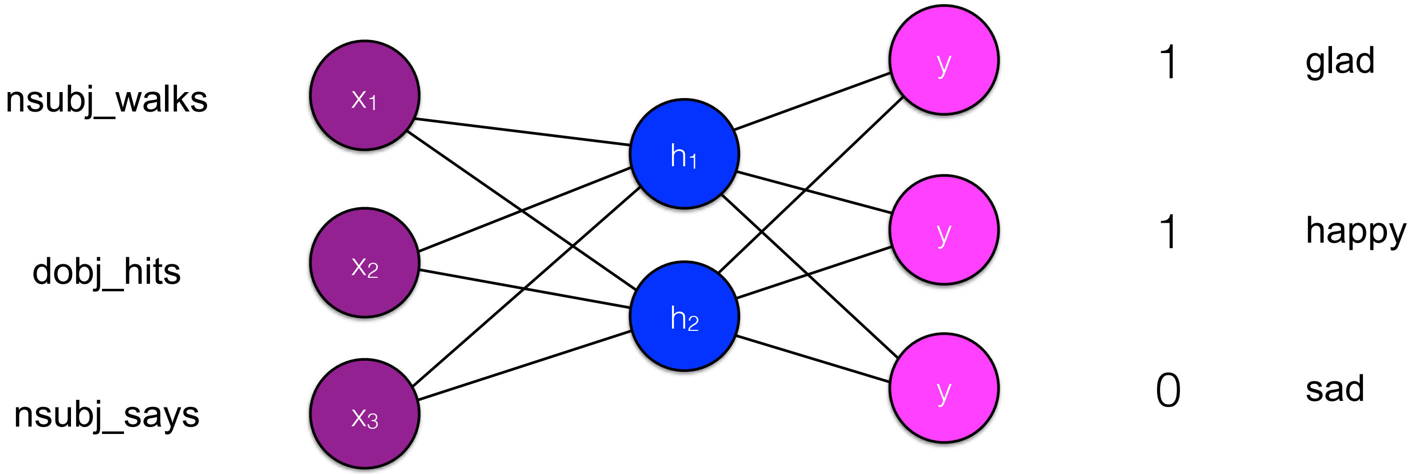
Output one real value; sigmoid function for output gives single probability between 0 and 1

Neural network structures



Multiclass: output 3 values, only one = 1 in training data; softmax function for output gives probability between 0 and 1 for each class (all class probabilities sum to 1); classes **compete** with each other.

Neural network structures



output 3 values, several = 1 in training data; sigmoid function for *each* output gives probability of presence of that label; classes do not compete with each other since multiple can be present together.

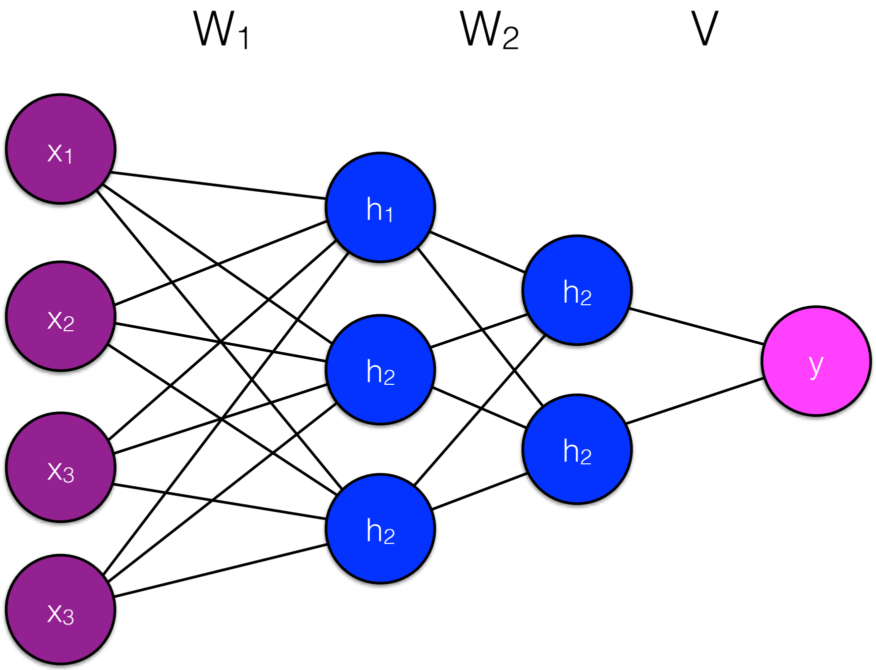
Regularization

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

Regularization

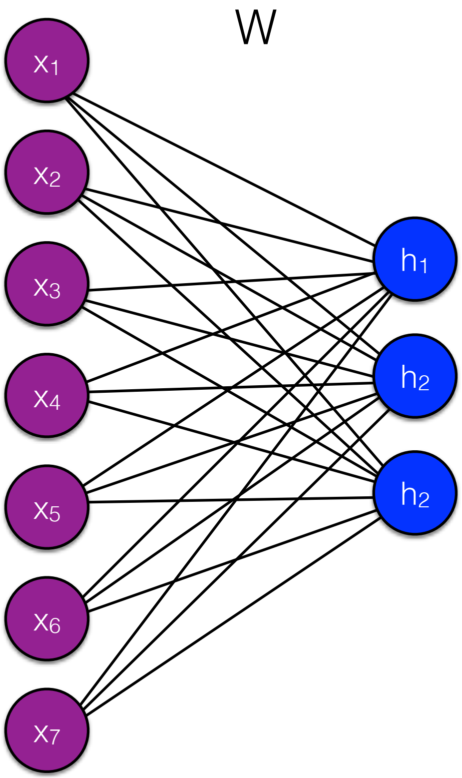
- L2 regularization: penalize W and V for being too large
- Dropout: when training on a <x,y> pair, randomly remove some node and weights.
- Early stopping: Stop backpropagation before the training error is too small.

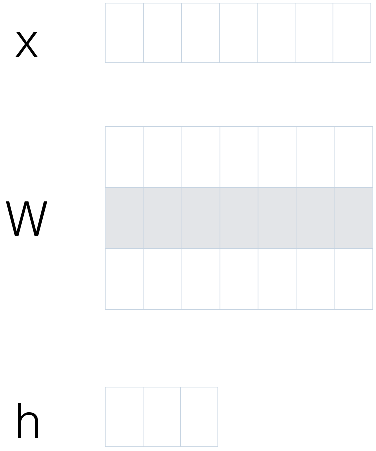
Deeper networks



Talk to your neighbor!

Densely connected layer





$$h = \sigma(xW)$$

Convolutional networks

- With convolution networks, the same operation (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*

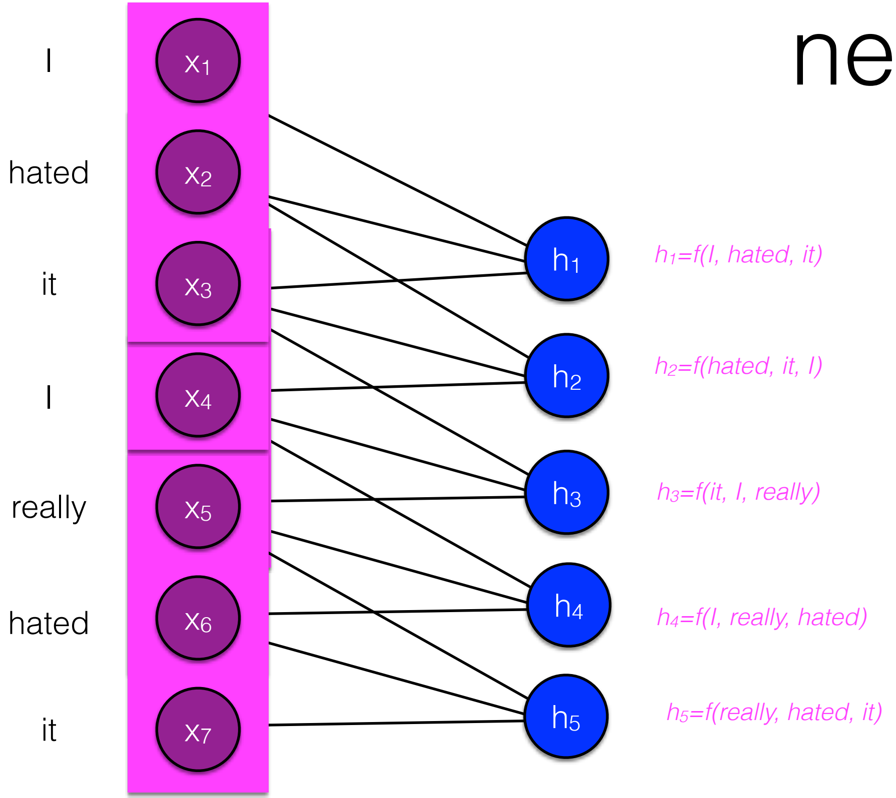
1/3	1/3	1/3

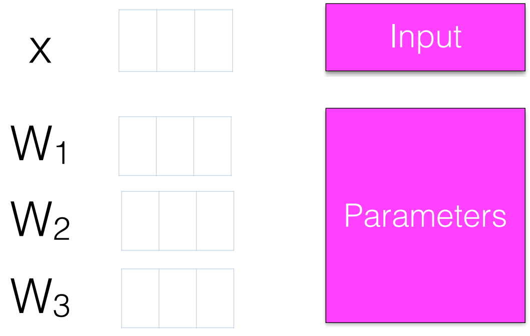
x

0	1	3	-1	4	2	0

1⅓	1	2	1⅔	2

Convolutional networks



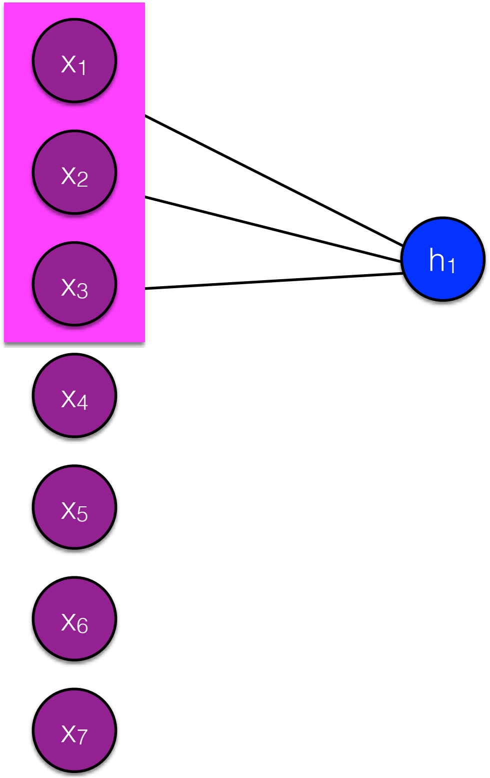


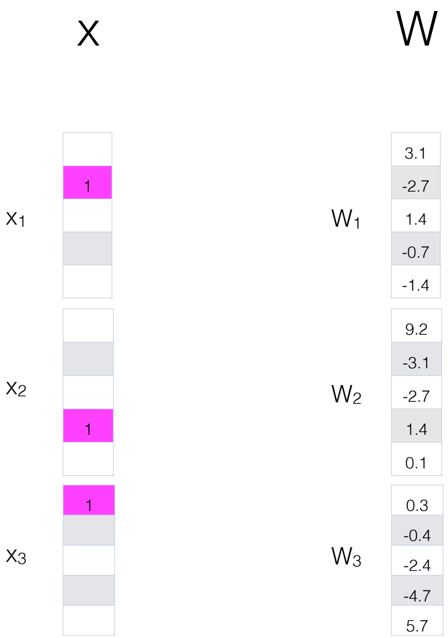
$$h_1 = \sigma(x_1 W_1 + x_2 W_2 + x_3 W_3)$$

Indicator vector

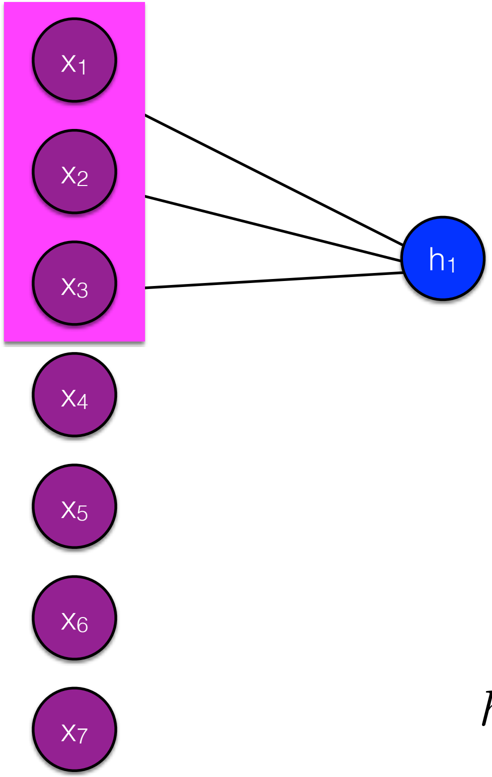
- Every token is a V-dimensional vector (size of the vocab) with a single 1 identifying the word

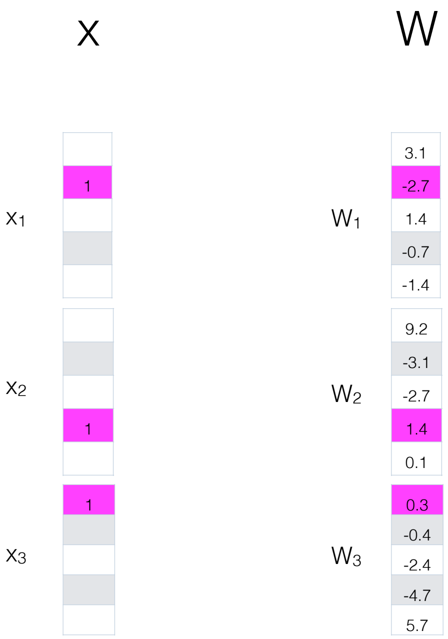
vocab item	indicator
a	0
aa	0
aal	0
aalii	0
aam	0
aardvark	1
aardwolf	0
aba	0





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

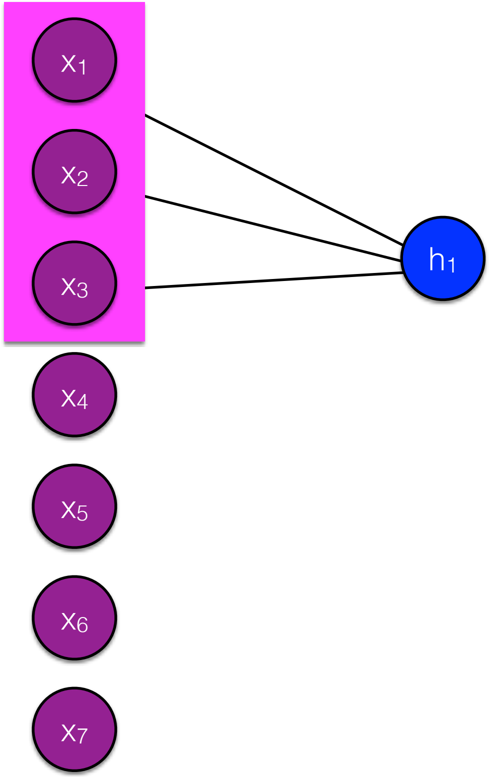




For indicator vectors, we're just adding these numbers together

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

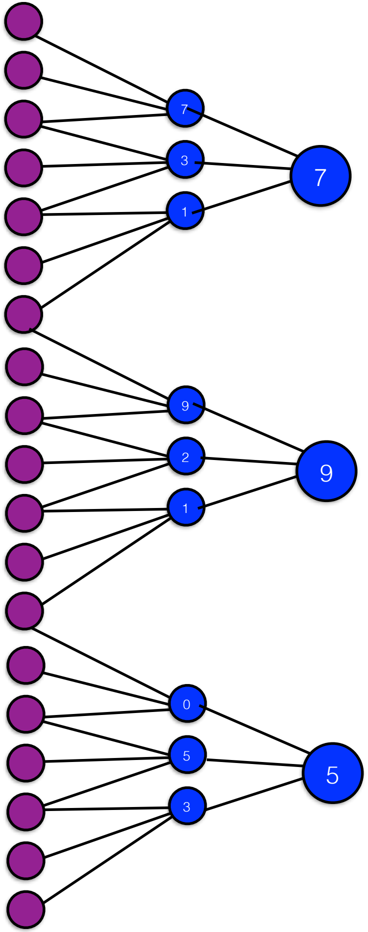
(Where x_n^{id} specifies the location of the 1 in the vector — i.e., the vocabulary id)



	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

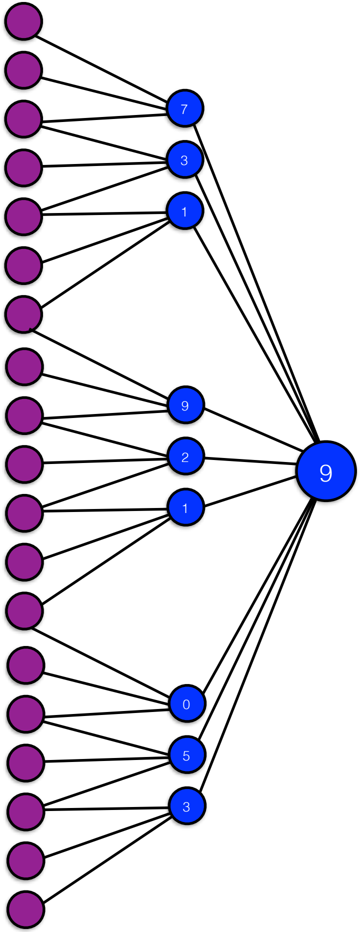
For dense input vectors (e.g., embeddings), full dot product

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



Pooling

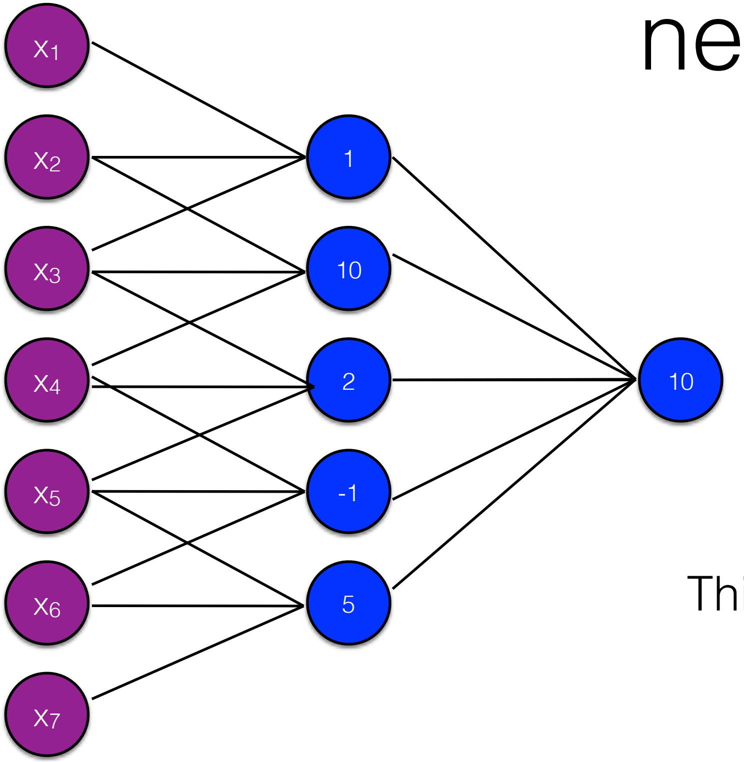
- Down-samples a layer by selecting a single point from some set
- Max-pooling selects the largest value



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.

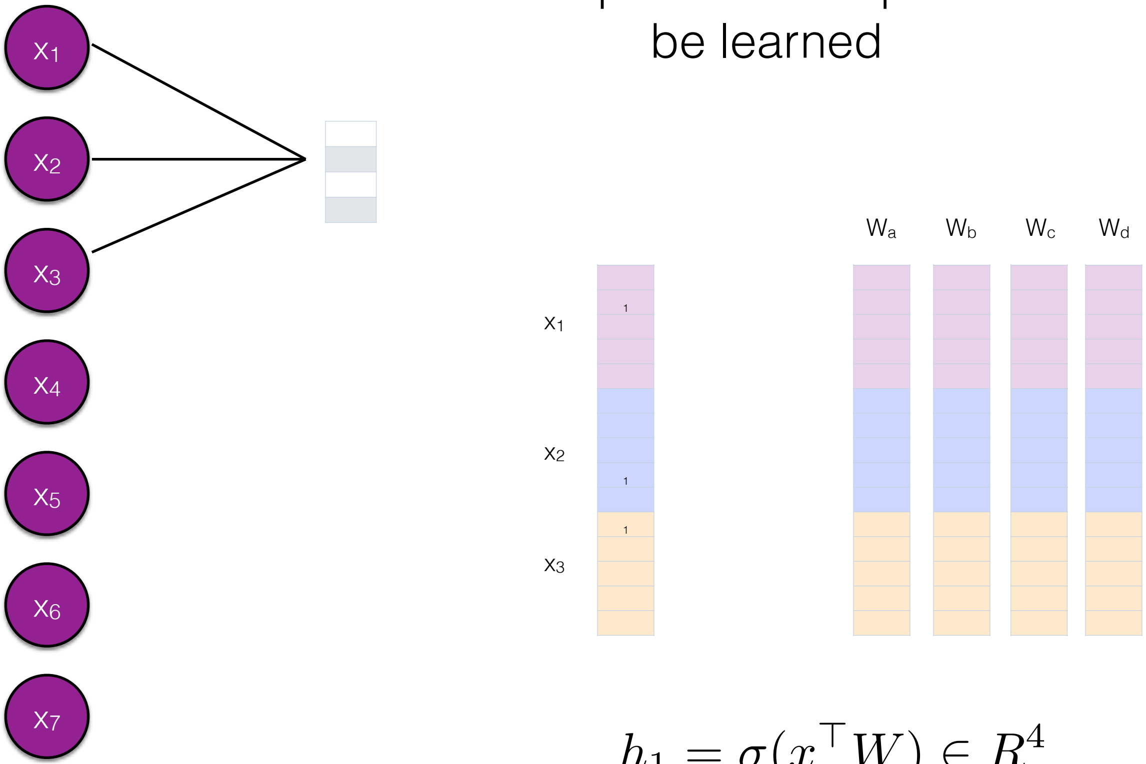
Convolutional networks



convolution max pooling

This defines one filter.

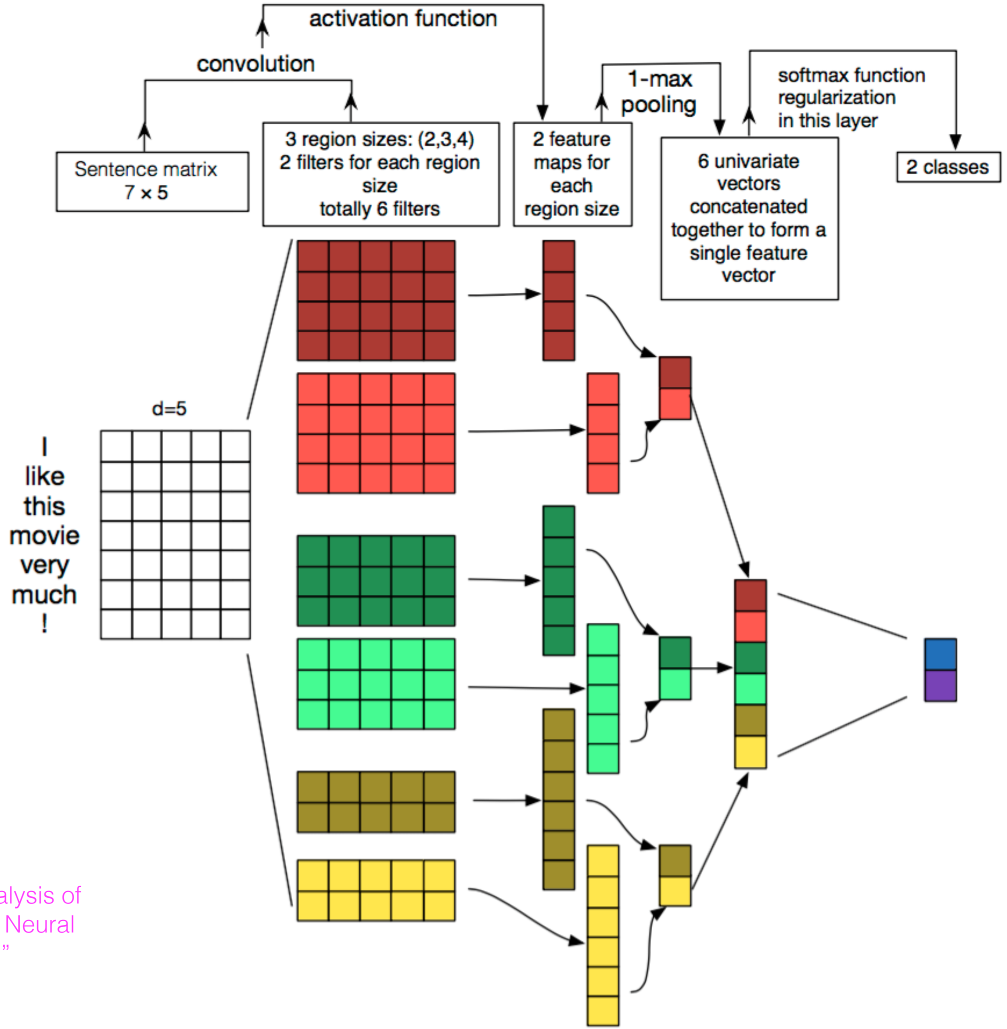
We can specify multiple filters; each filter is a separate set of parameters to be learned



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

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

Logistics

- Quiz 1 is due tonight (11:59 pm)
    - Quiz 2 will be out this Friday (due next Monday Feb 5).
- Homework 1 is out & due next Tuesday, Feb 6 (11:59 pm)
    - Homework 2 will be out early next week.
- Next Lecture: Text Classification with Contextual Embedding, BERT, etc.