SI 630
Natural Language Processing:
Algorithms and People

Lecture 2: Classification
Jan. 10, 2017

David Jurgens
jurgens@umich.edu

Some slides from David Bamman

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

$$P(y \mid x)$$

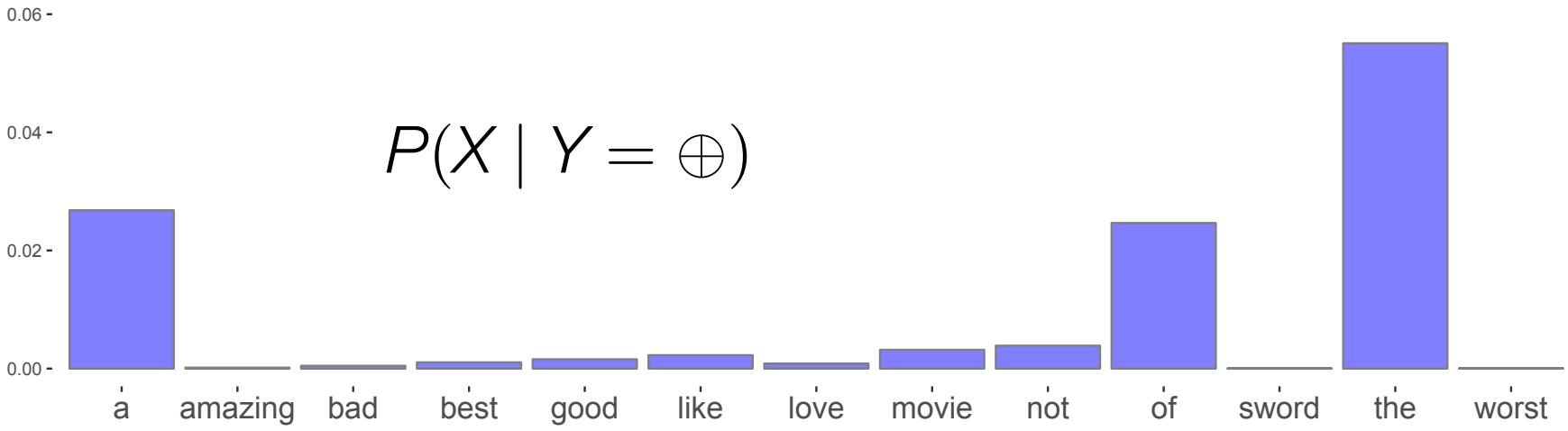
"… is a film which still causes real, not figurative, chills to run along my spine, and it is certainly the bravest and most ambitious fruit of Coppola's genius"

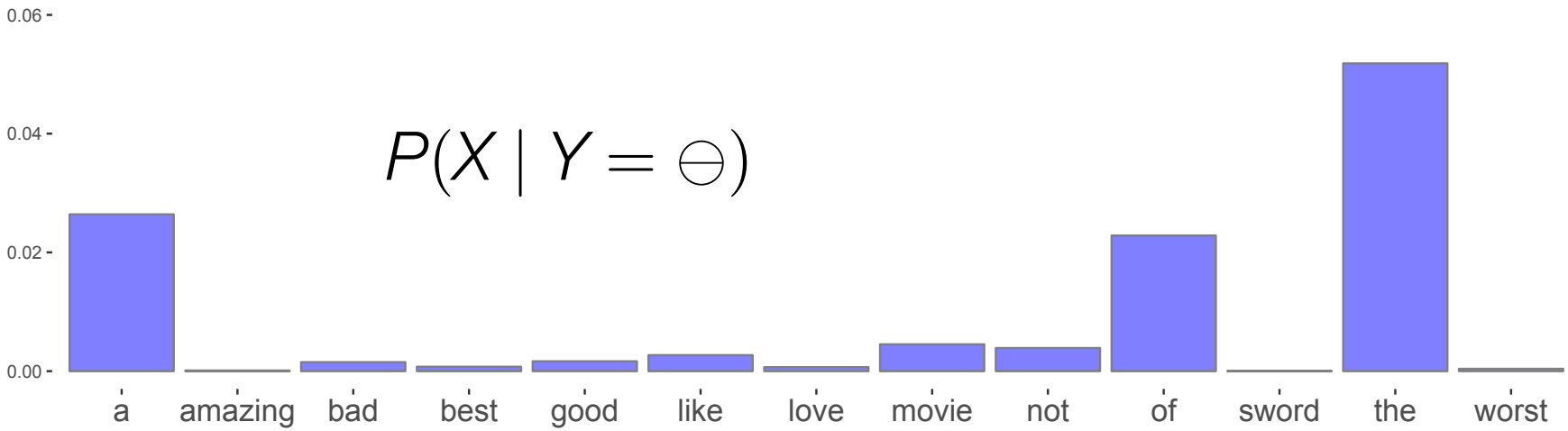
Roger Ebert, Apocalypse Now

"I hated this movie. Hated hated hated hated hated this movie. Hated it. Hated every simpering stupid vacant audience-insulting moment of it. Hated the sensibility that thought anyone would like it."

Roger Ebert, North

Generating


$P(X \mid Y = \oplus)$
0.06
0.04
0.02
0.00
a
amazing
bad
best
good
like
love
movie
not
of
sword
the
worst


$P(X \mid Y = \ominus)$
0.06
0.04
0.02
0.00
a
amazing
bad
best
good
like
love
movie
not
of
sword
the
worst

Generation

taking allen pete visual an lust be infinite corn physical here decidedly 1 for . never it against perfect the possible spanish of supporting this all this this pride turn that sure the a purpose in real . environment there's trek right . scattered wonder dvd three criticism his .

positive

us are i do tense kevin fall shoot to on want in (. minutes not problems unusually his seems enjoy that : vu scenes rest half in outside famous was with lines chance survivors good to . but of modern-day a changed rent that to in attack lot minutes

negative

Generative models

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

posterior

prior

likelihood

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

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

A few math things to remember

$$\sum_{i=1}^{F} x_i \beta_i = x_1\beta_1 + x_2\beta_2 + \ldots + x_F\beta_F$$

$$\prod_{i=1}^{F} x_i = x_i \times x_2 \times \ldots \times x_F$$

$$\exp(x) = e^x \approx 2.7^x \qquad \exp(x+y) = \exp(x)\exp(y)$$

$$\log(x) = y \rightarrow e^y = x \qquad \log(xy) = \log(x) + \log(y)$$

A few math things to remember

This equation $3x + 2y + 5z = 7$

can be written as a dot product of two vectors:

$$\begin{bmatrix} 3 & 2 & 5 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} = 7$$

A few math things to remember

Likewise, these equations

$$
\begin{aligned}
3x + 2y + 5z &= 7 \\
-1x + 3y + 9z &= 12 \\
4x + \tfrac{1}{2}y + 0z &= 8
\end{aligned}
$$

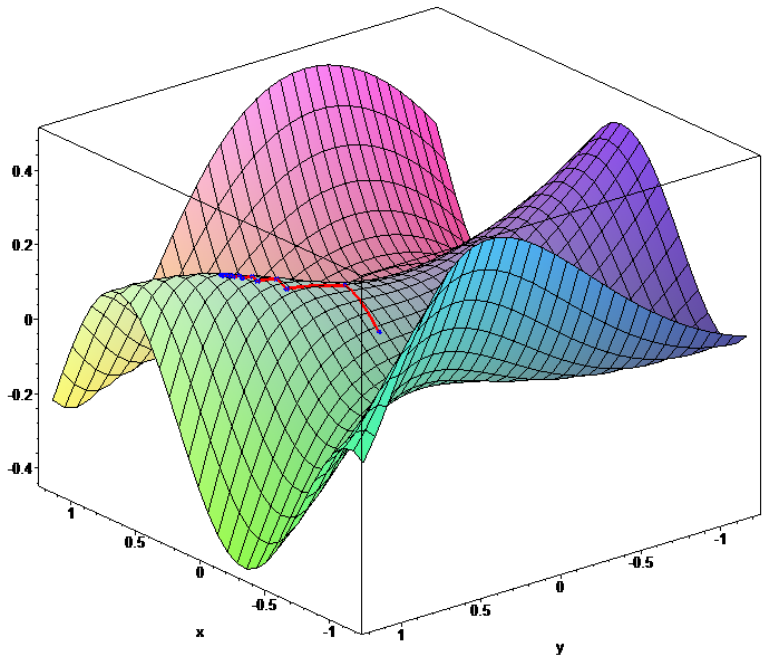
can be written as a dot product of a matrix and a vector:

$$
\begin{bmatrix} 3 & 2 & 5 \\ -1 & 3 & 9 \\ 4 & \frac{1}{2} & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 7 \\ 12 \\ 8 \end{bmatrix}
$$

Linear Algebra Takeaway:

- You can think of the dot product as applying a set of weights to your vector.
- The dot product of two vectors is a scalar (a normal number, like 7 or 0.34).
- The dot product of a matrix and a vector is a vector.

Quick Math Review: Derivatives



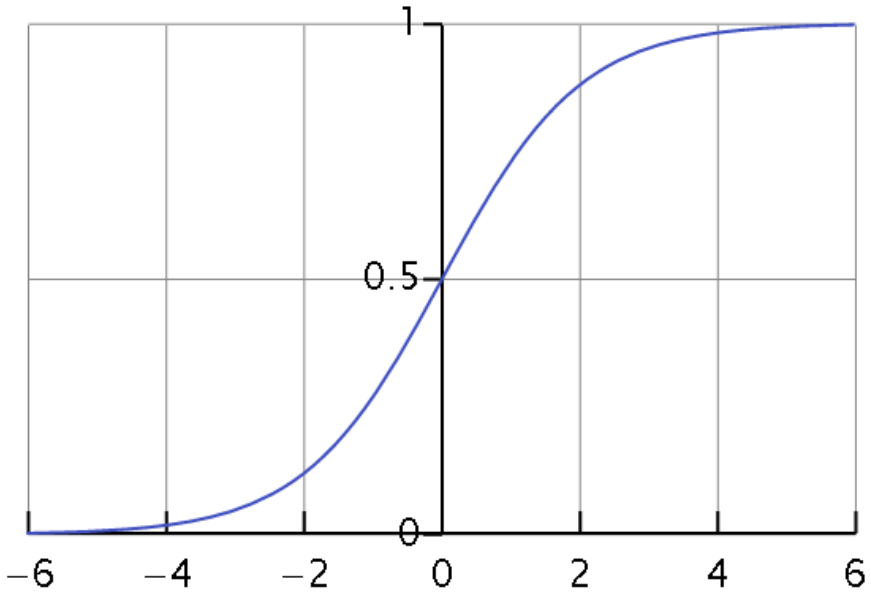
Derivative is a fancy word for slope.

If you are at the point where the green X is, and you want to find the point that minimizes the function, you can use the slope to figure out which way is downhill.

This works in spaces with more than two dimensions, also.

image credit: https://en.wikipedia.org/wiki/Gradient_descent

Quick Math Review: Sigmoid Function



The sigmoid function ranges from 0 to 1. Most of the time, it's much closer to one extreme than the other. This can be nice if you want to classify something as either 0 or 1.

Also called the logistic function, this is a key component of logistic regression.

Classification

Classification defines a mapping *h*

- from input data x (drawn from instance space $\mathcal{X}$)
- to a label (or labels) y from some enumerable output space $\mathcal{Y}$

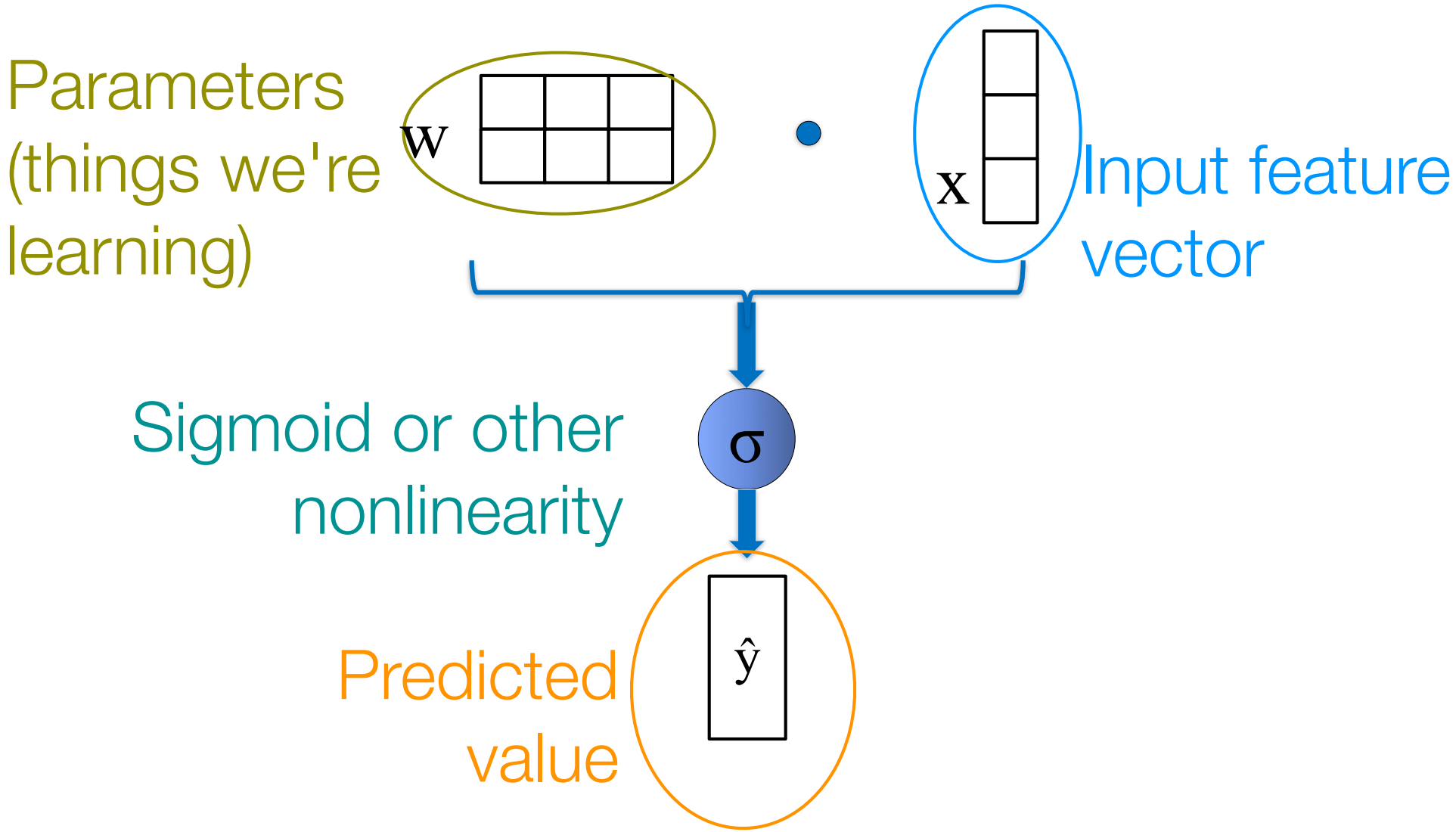
$\mathcal{X}$ = set of all movie reviews

$\mathcal{Y}$ = {good, bad}

x = a single document
y = bad

We will also call this mapping a function

Supervised Machine Learning



Training data

positive

"… is a film which still causes real, not figurative, chills to run along my spine, and it is certainly the bravest and most ambitious fruit of Coppola's genius"

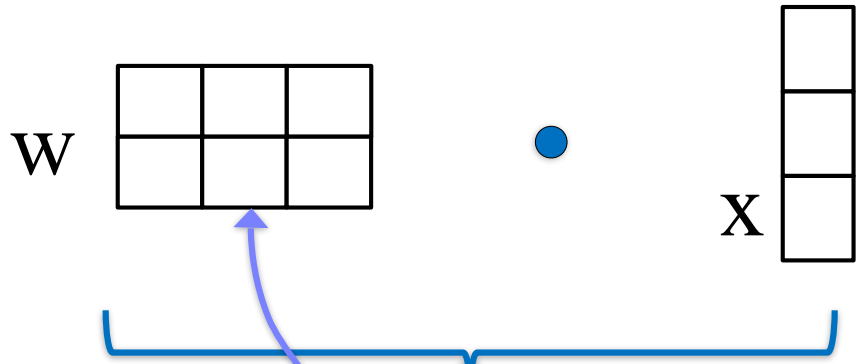
Roger Ebert, Apocalypse Now

- "I hated this movie. Hated hated hated hated hated this movie. Hated it. Hated every simpering stupid vacant audience-insulting moment of it. Hated the sensibility that thought anyone would like it."

Roger Ebert, North

negative

Supervised Machine Learning



Update parameters

How wrong were we?

cost(ŷ , y) Actual value

What Is the Feature Vector x?

- Typically a vector representation of a single character or word
- Often reflects the context in which that word is found
- Could just do counts, but that leads to sparse vectors
- Commonly used techniques: word2vec or GloVe word embeddings

https://wftda.com/wftda-leagues/tucson-roller-derby/

Logistic Regression

Logistic regression

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

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

x = feature vector

Feature	Value
the	0
and	0
bravest	0
love	0
loved	0
genius	0
not	0
fruit	1
BIAS	1

β = coefficients

Feature	β
the	0.01
and	0.03
bravest	1.4
love	3.1
loved	1.2
genius	0.5
not	-3.0
fruit	-0.8
BIAS	-0.1

	BIAS	love	loved
β	-0.1	3.1	1.2

	BIAS	love	loved	$a=\sum x_i\beta_i$	exp(-a)	1/(1+exp(-a))
x^1	1	1	0	3	0.05	0.952
x^2	1	1	1	4.2	0.015	0.985
x^3	1	0	0	-0.1	1.11	0.474

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 with out 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.)

Features

feature classes
unigrams (“like”)
bigrams (“not like”), higher order ngrams
prefixes (words that start with “un-”)
has word that shows up in positive sentiment dictionary

Features

Feature	Value
the	0
and	0
bravest	0
love	0
loved	0
genius	0
not	1
fruit	0
BIAS	1

Feature	Value
like	1
not like	1
did not like	1
in_pos_dict_MPQA	1
in_neg_dict_MPQA	0
in_pos_dict_LIWC	1
in_neg_dict_LIWC	0
author=ebert	1
author=siskel	0

How do we get good values for β?

β = coefficients

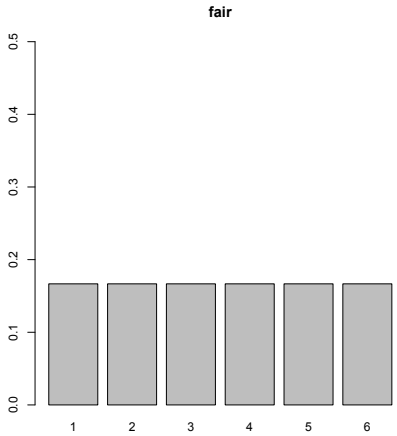
Feature	β
the	0.01
and	0.03
bravest	1.4
love	3.1
loved	1.2
genius	0.5
not	-3.0
fruit	-0.8
BIAS	-0.1

Likelihood

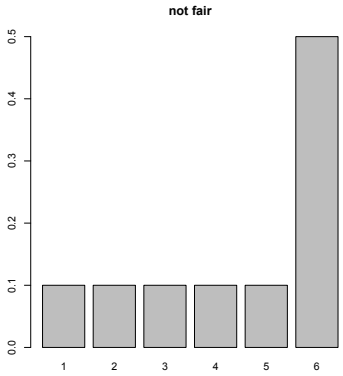
Remember the likelihood of data is its probability under some parameter values

In maximum likelihood estimation, we pick the values of the parameters under which the data is most likely.

Likelihood

P(2 6 6 | 

) =.17 x .17 x .17
= 0.004913

P(2 6 6 | 

) = .1 x .5 x .5
= 0.025

Conditional likelihood

$$\prod_{i}^{N} P(y_i \mid x_i, \beta)$$

For all training data, we want probability of the true label y for each data point x to high

	BIAS	love	loved	$a=\sum x_i\beta_i$	exp(-a)	1/(1+exp(-a))	true y
x^1	1	1	0	3	0.05	0.952	1
x^2	1	1	1	4.2	0.015	0.985	1
x^3	1	0	0	-0.1	1.11	0.475	0

Conditional likelihood

$$\prod_{i}^{N} P(y_i \mid x_i, \beta)$$

For all training data, we want probability of the true label y for each data point x to high

This principle gives us a way to pick the values of the parameters β that maximize the probability of the training data <x, y>

The value of β that maximizes likelihood also maximizes the log likelihood

$$\arg\max_{\beta} \prod_{i=1}^{N} P(y_i \mid x_i, \beta) = \arg\max_{\beta} \log \prod_{i=1}^{N} P(y_i \mid x_i, \beta)$$

The log likelihood is an easier form to work with:

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

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

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

We want to find the values of β that make the value of this function the greatest

$$\sum_{<x,y=+1>} \log P(1 \mid x, \beta) + \sum_{<x,y=0>} \log P(0 \mid x, \beta)$$

$$\frac{\partial}{\partial \beta_i} \ell(\beta) = \sum_{<x,y>} (y - \hat{p}(x))\, x_i$$

Gradient descent

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**

If y is 1 and p(x) = 0, then this still pushes the weights a lot

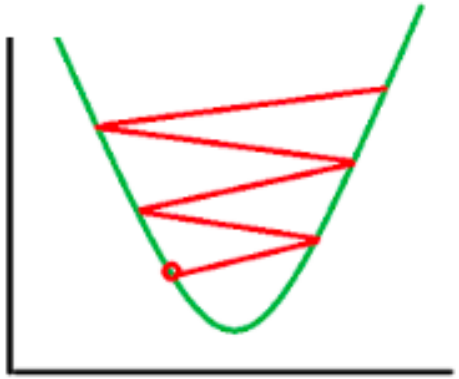
If y is 1 and p(x) = 0.99, then this still pushes the weights just a little bit

Learning rates

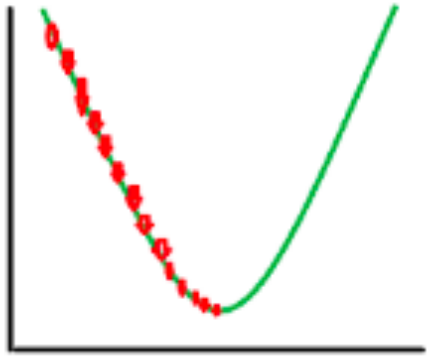
Algorithm 1 Logistic regression gradient descent

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**

The learning rate decides how much we update each iteration.
Usually named α or η



High α



Low α

Stochastic Gradient Descent (SGD)

- Batch gradient descent reasons over every training data point for each update of β. This can be slow to converge.
- Stochastic gradient descent updates β after each data point.

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**

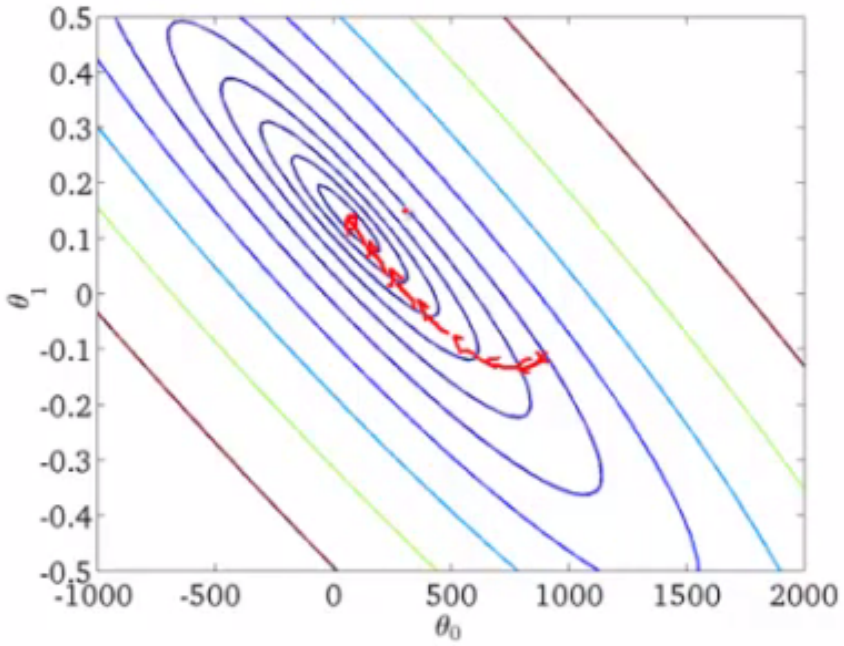
Gradient Descent vs. Stochastic Gradient Descent

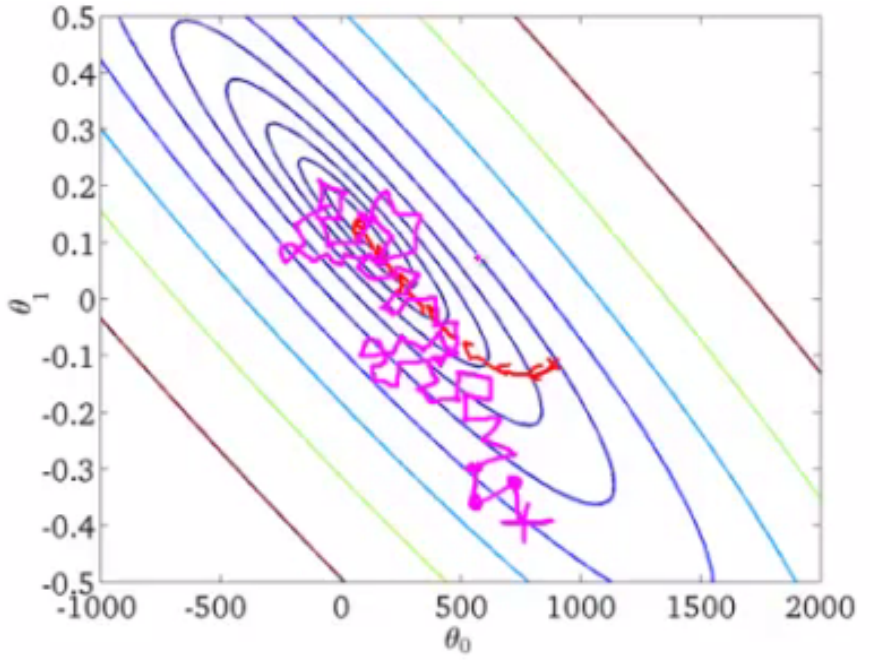
Algorithm 1 Logistic regression gradient descent

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**

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: $\quad$ **for** $i = 1$ to N **do**
5: $\quad\quad \beta_{t+1} = \beta_t + \alpha\, (y_i - \hat{p}(x_i))\, x_i$
6: $\quad$ **end for**
7: **end while**





Practicalities

- When calculating the P(y | x) or in calculating the gradient, you don't need to loop through all features — only those with nonzero values

- (Which makes sparse, binary values useful)

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

$$\frac{\partial}{\partial \beta_i} \ell(\beta) = \sum_{<x,y>} (y - \hat{p}(x))\, x_i$$

Quiz time: Which of the following is true?

- In text classification most words are

  A.rare

  B.not correlated with any class

  C.given low weights in the LR classifier

  D.unlikely to affect classification

  E.not very interesting

$$\frac{\partial}{\partial \beta_i} \ell(\beta) = \sum_{<x,y>} (y - \hat{p}(x))\, x_i$$

If a feature x_i only shows up with the positive class (e.g., positive sentiment), what are the possible values of its corresponding β_i?

$$\frac{\partial}{\partial \beta_i} \ell(\beta) = \sum_{<x,y>} (1 - 0)1$$

$$\frac{\partial}{\partial \beta_i} \ell(\beta) = \sum_{<x,y>} (1 - 0.9999999)1$$

always positive

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

β = coefficients

Feature	β
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 ∧ dog ∧ starts with “in”	30.1

Feature selection

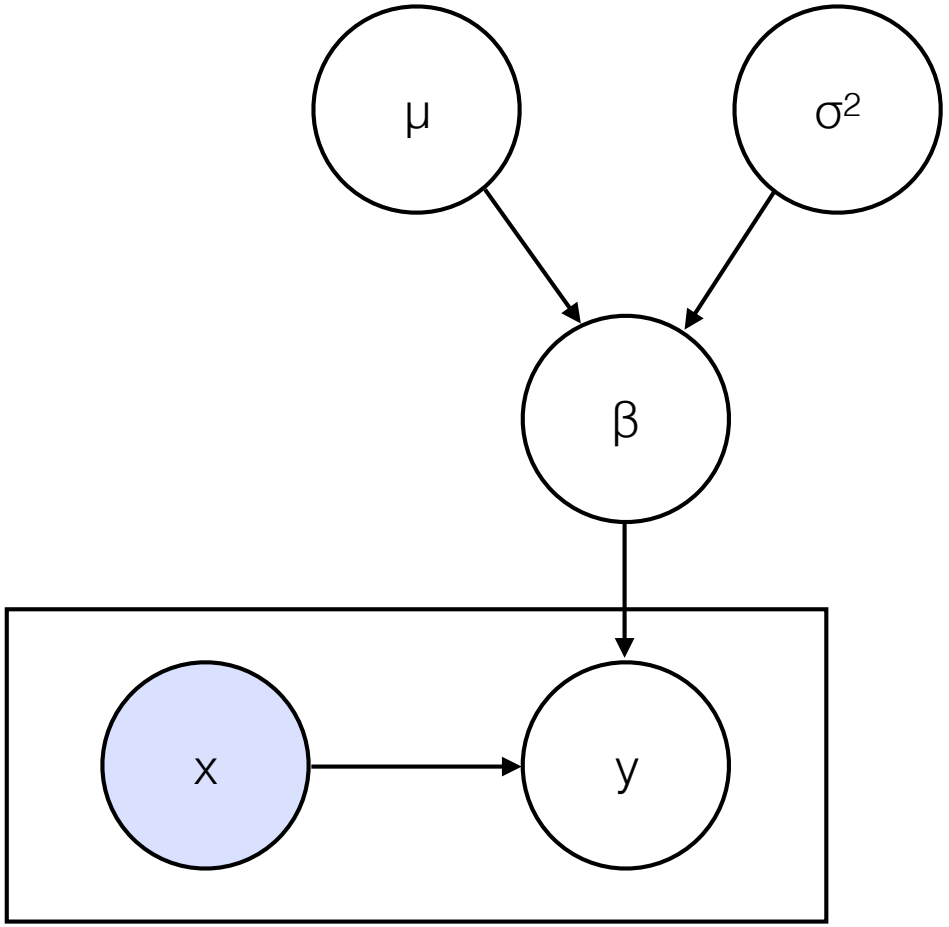
- We could threshold features by minimum count but that also throws away information
- We can take a probabilistic approach and encode a prior belief that all β should be 0 unless we have strong evidence otherwise

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)

no L2 regularization		some L2 regularization		high L2 regularization	
33.83	Won Bin	2.17	Eddie Murphy	0.41	Family Film
29.91	Alexander Beyer	1.98	Tom Cruise	0.41	Thriller
24.78	Bloopers	1.70	Tyler Perry	0.36	Fantasy
23.01	Daniel Brühl	1.70	Michael Douglas	0.32	Action
22.11	Ha Jeong-woo	1.66	Robert Redford	0.25	Buddy film
20.49	Supernatural	1.66	Julia Roberts	0.24	Adventure
18.91	Kristine DeBell	1.64	Dance	0.20	Comp Animation
18.61	Eddie Murphy	1.63	Schwarzenegger	0.19	Animation
18.33	Cher	1.63	Lee Tergesen	0.18	Science Fiction
18.18	Michael Douglas	1.62	Cher	0.18	Bruce Willis



$$\beta \sim \text{Norm}(\mu, \sigma^2)$$

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

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)

What do the coefficients mean?

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

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

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

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

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

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

This is the odds of y occurring

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

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

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

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

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

This is the odds of y occurring

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

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

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

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

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


https://www.guitarworld.com/artists/keith-richards-isolated-guitar-gimme-shelter-rolling-stones

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

theano

(Keras)

(Dynet)

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

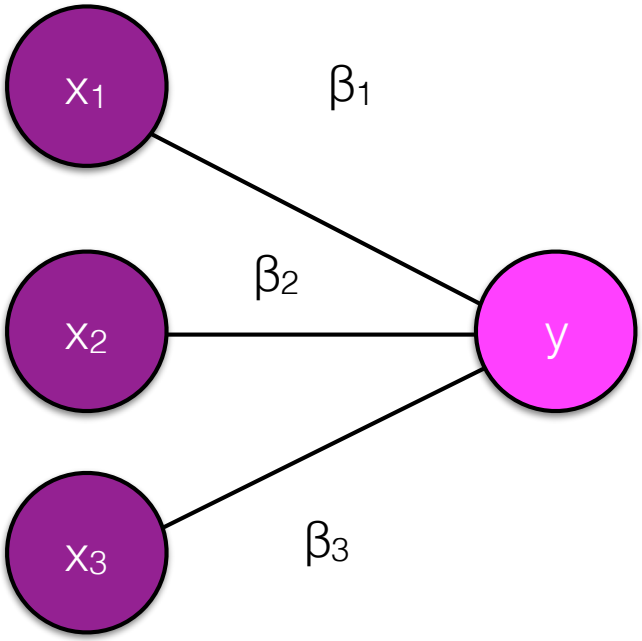
Algorithm 1 Logistic regression gradient descent

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

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

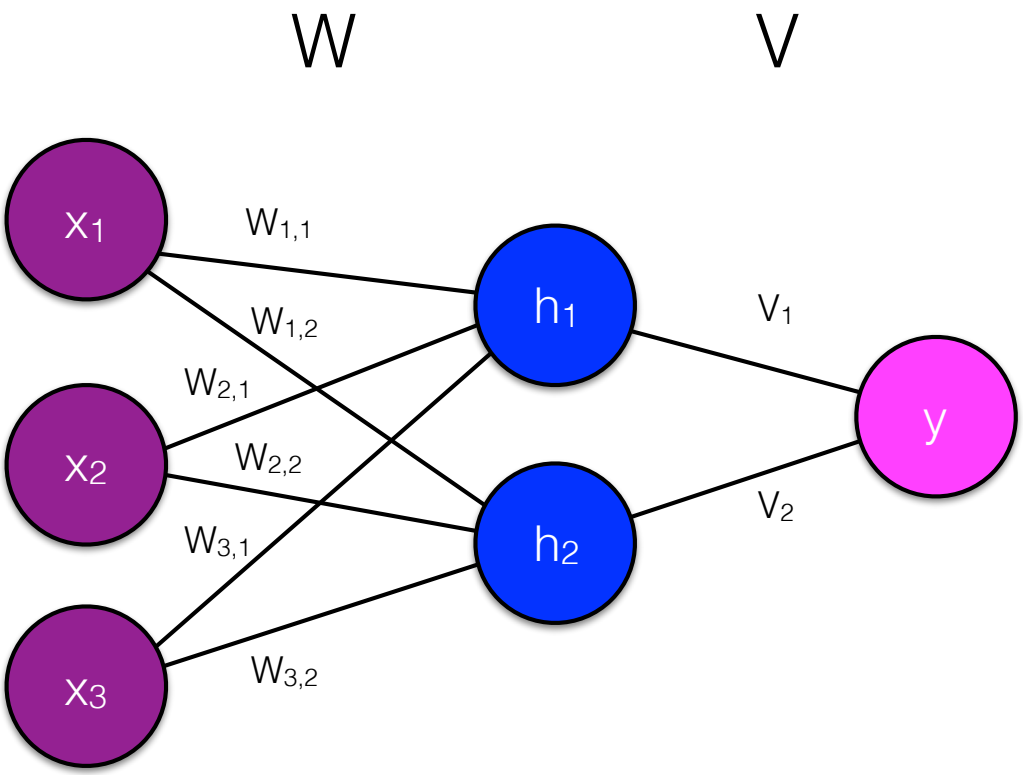


	x	β
not	1	-0.5
bad	1	-1.7
movie	0	0.3

Neural networks

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

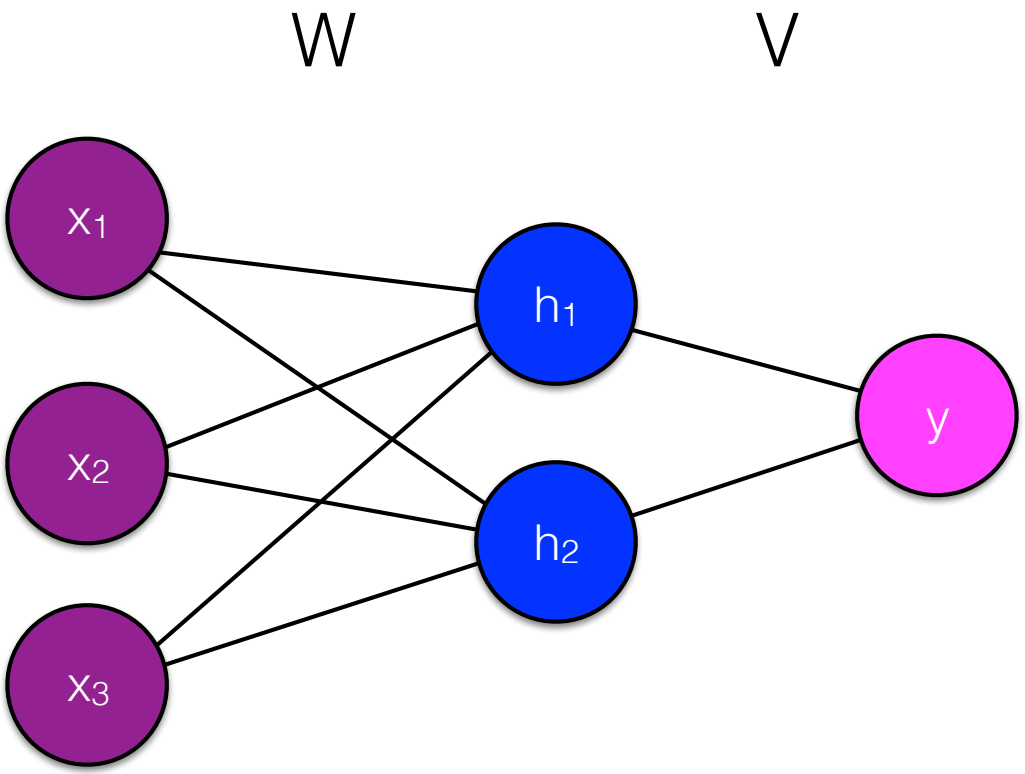
*For simplicity, we're leaving out the bias term, but assume most layers have them as well.



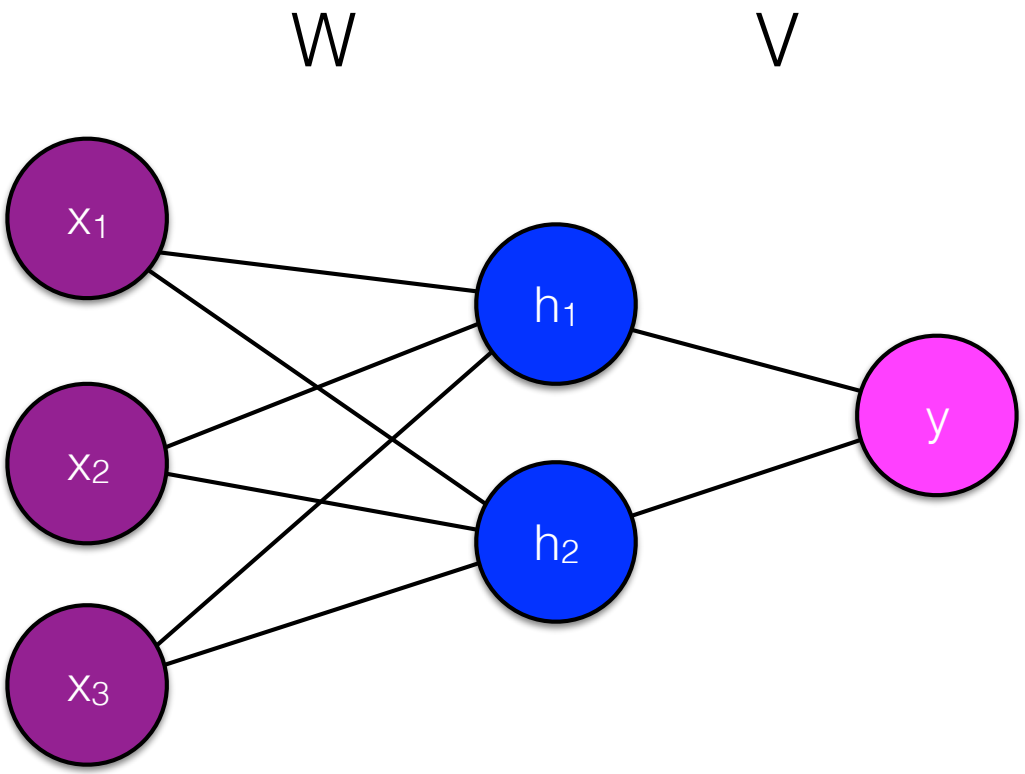
Input

"Hidden" Layer

Output

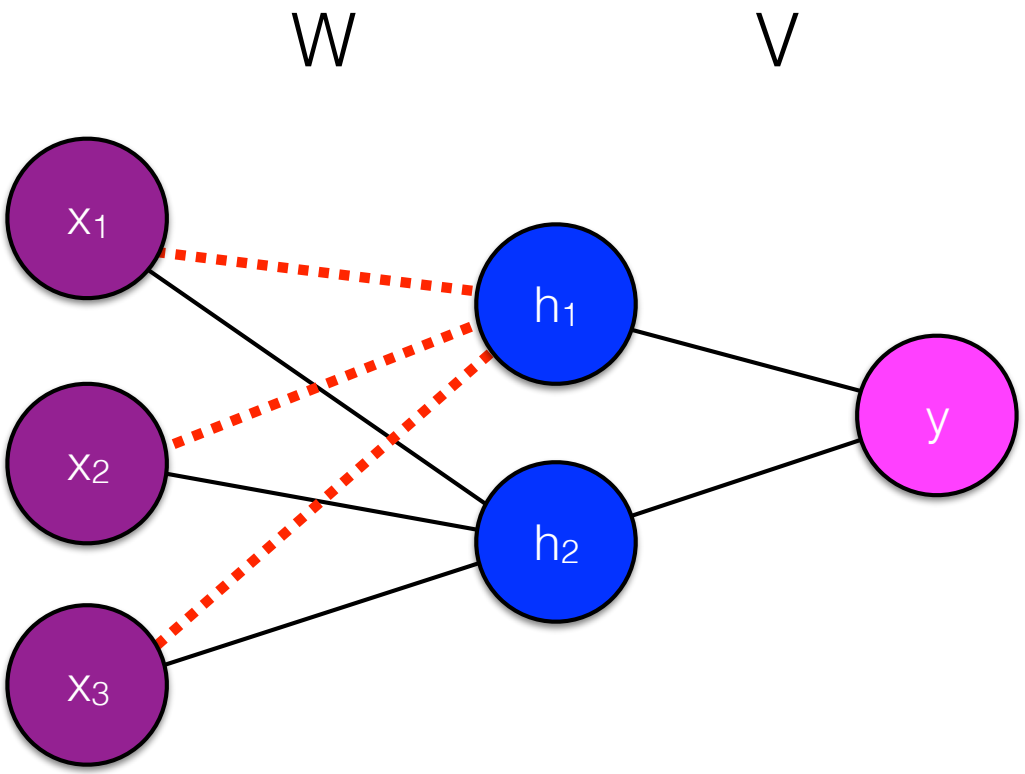


	x	W		V	y
not	1	-0.5	1.3	4.1	1
bad	1	0.4	0.08	-0.9	
movie	0	1.7	3.1		



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

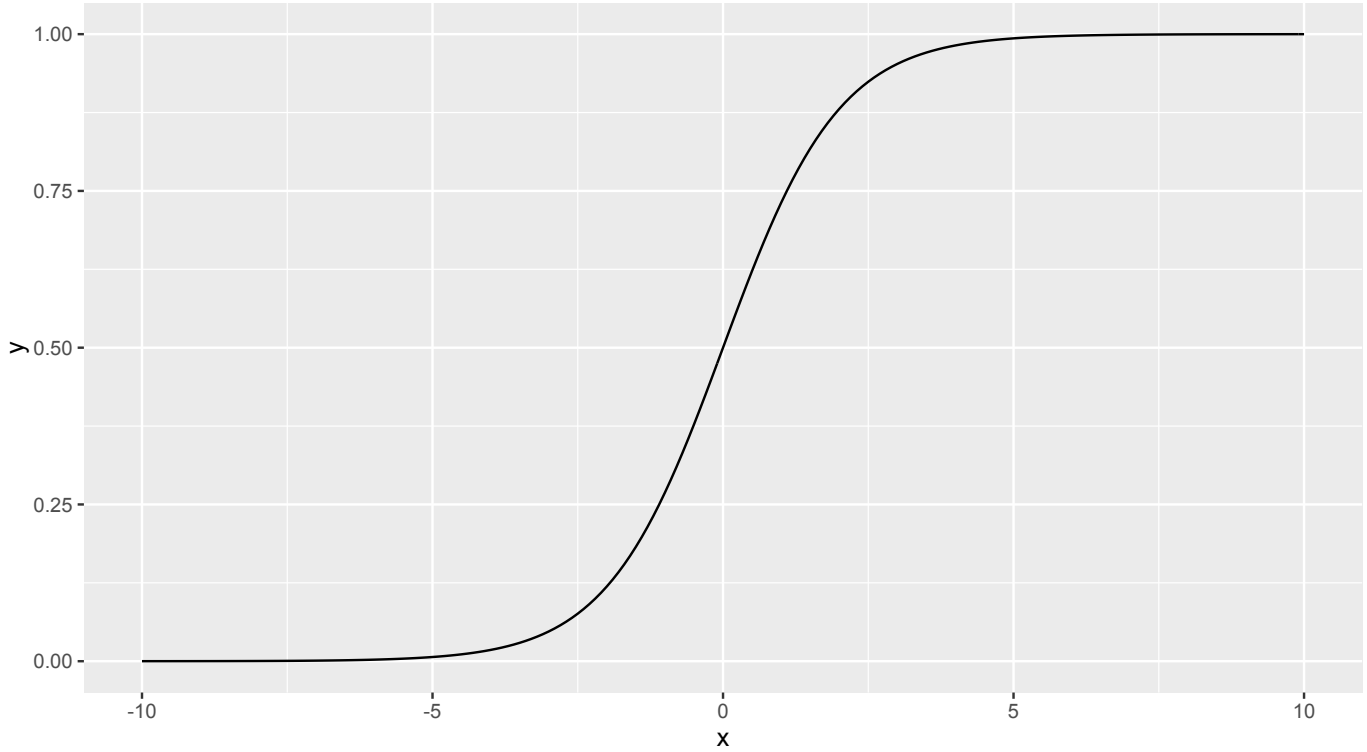
the hidden nodes are completely determined by the input and weights



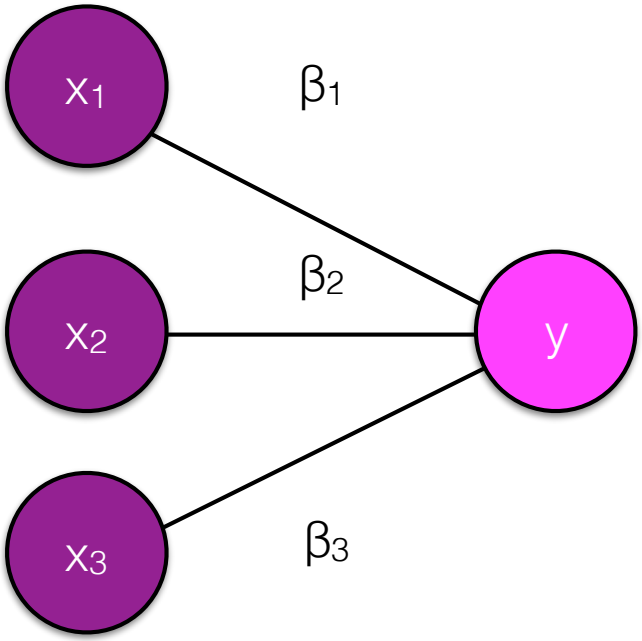
$$h_1 = f\left(\sum_{i=1}^{F} x_i W_{i,1}\right)$$

Activation functions

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



Logistic regression



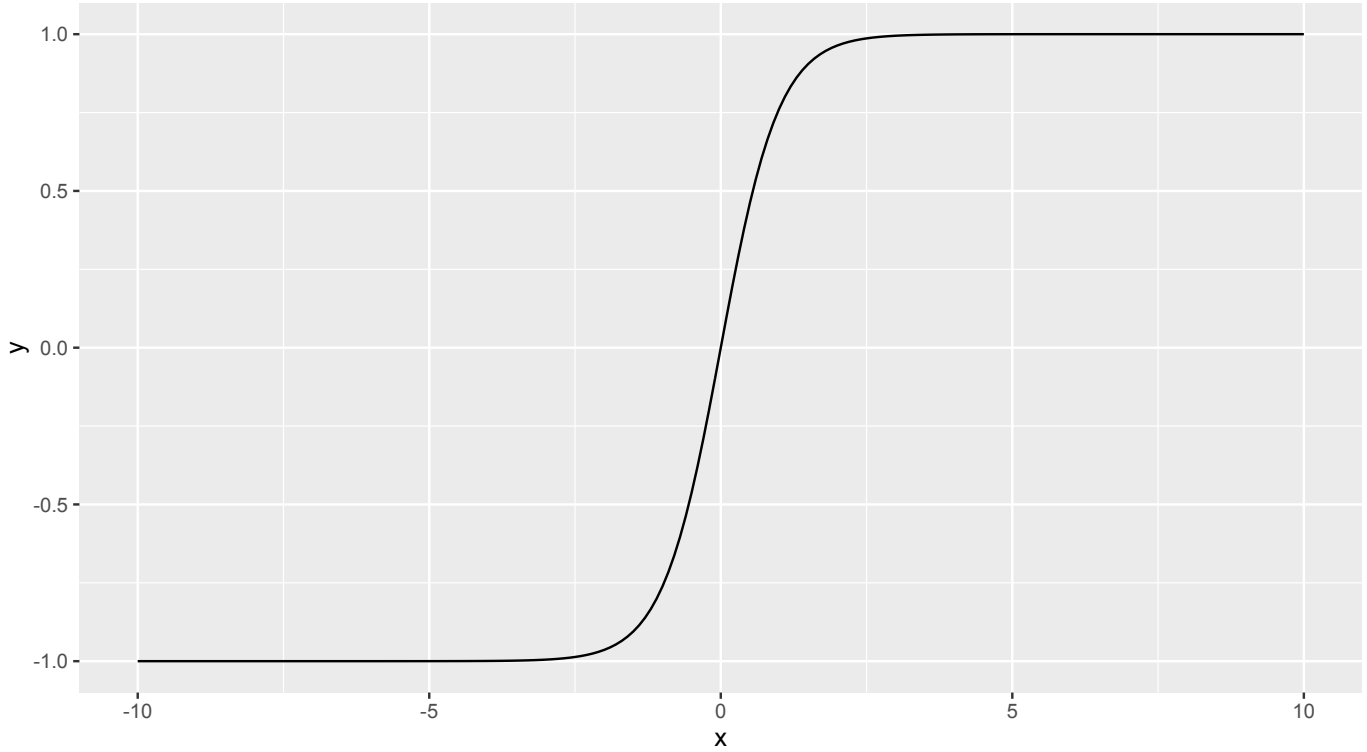
$$\hat{y} = \frac{1}{1 + \exp\left(-\sum_{i=1}^{F} x_i \beta_i\right)}$$

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

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

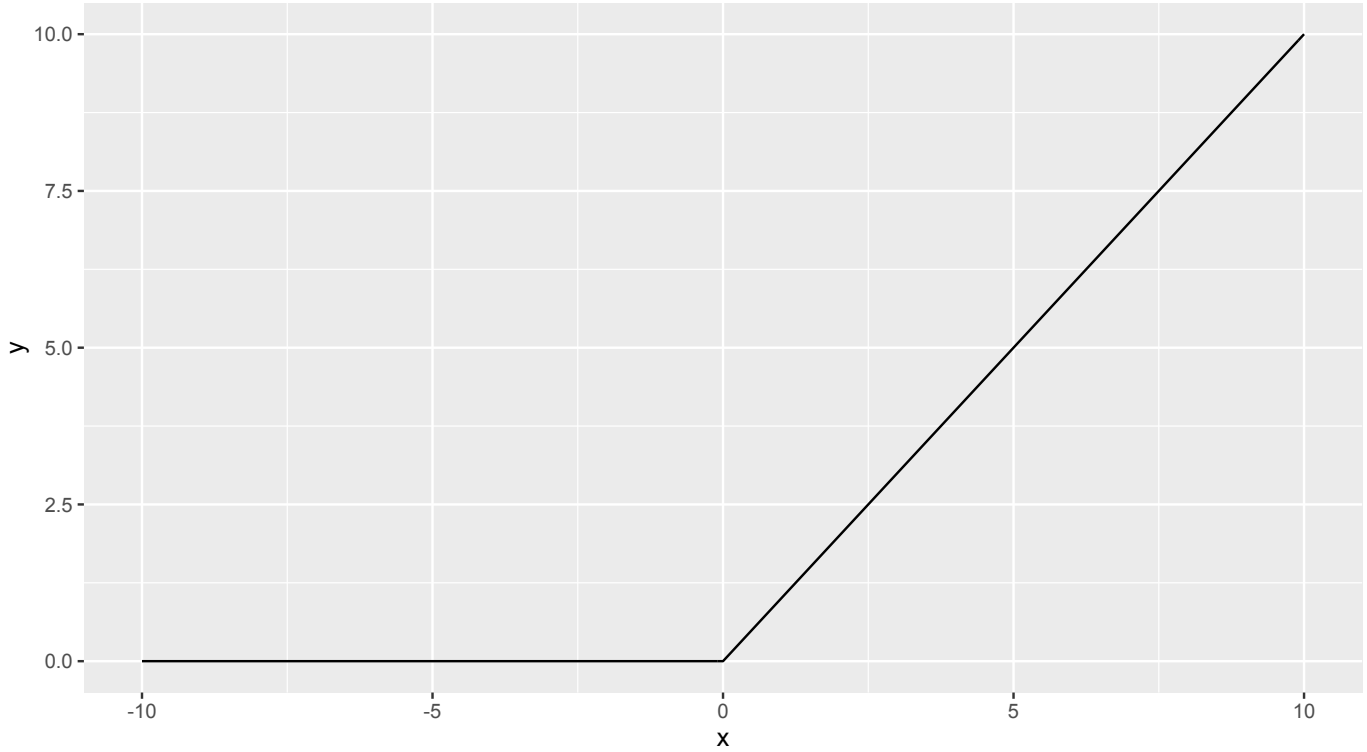
Activation functions

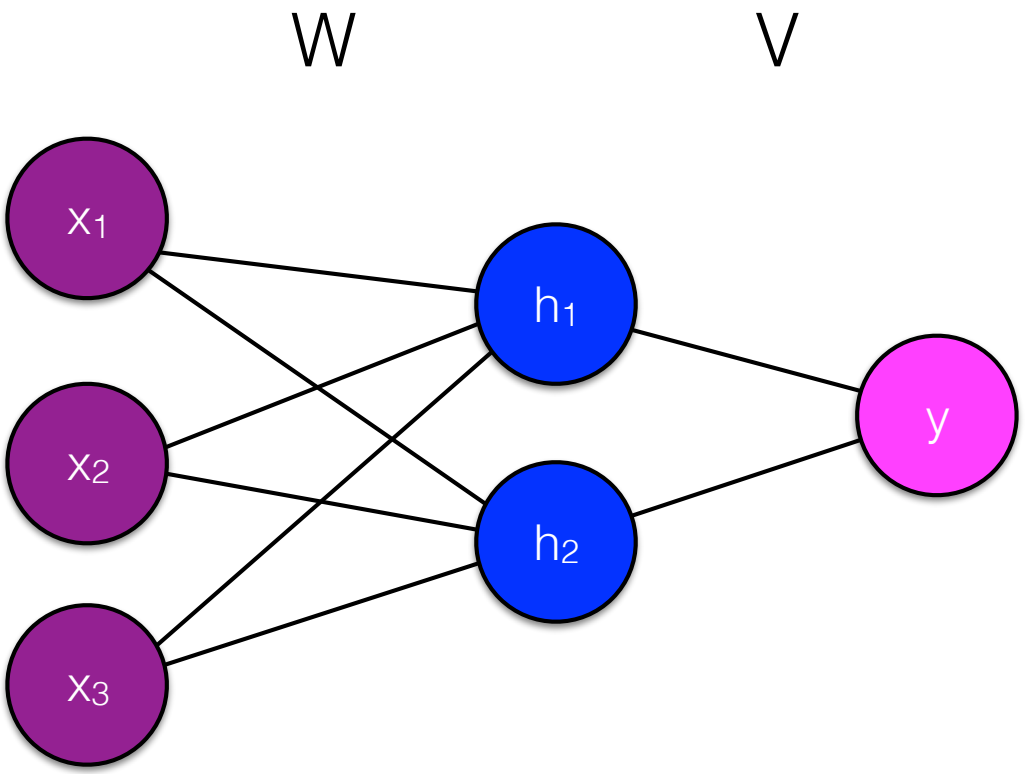
$$\tanh(z) = \frac{\exp(z) - \exp(-z)}{\exp(z) + \exp(-z)}$$



Activation functions

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

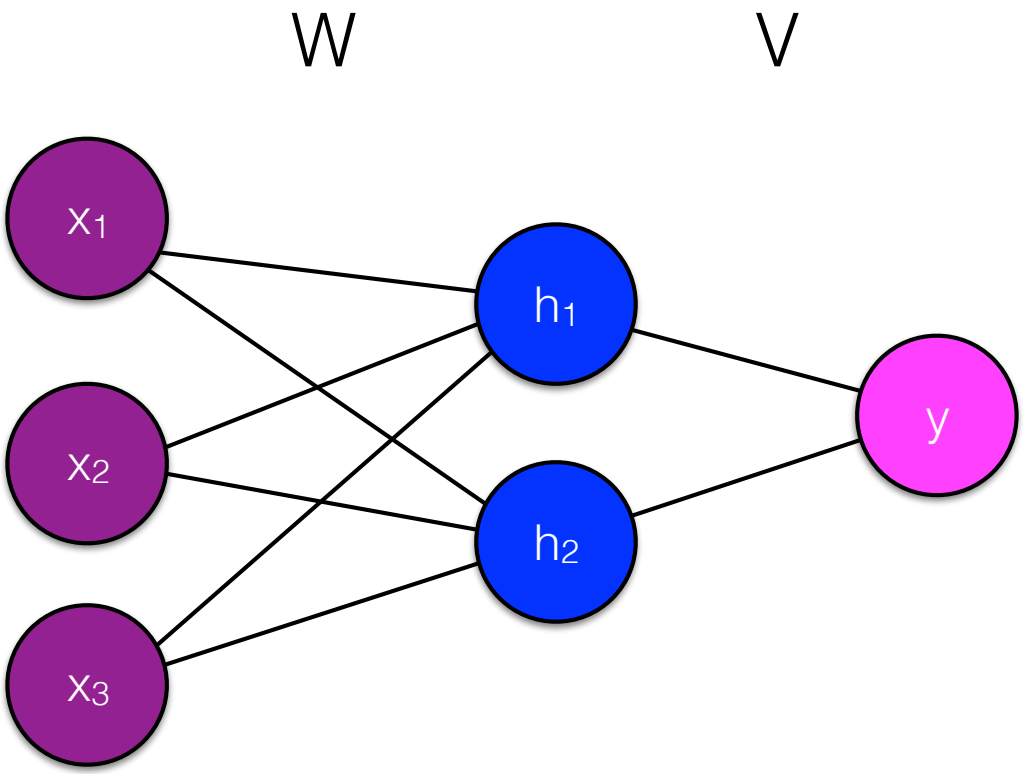




$$h_1 = \sigma\left(\sum_{i=1}^{F} x_i W_{i,1}\right)$$

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

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



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

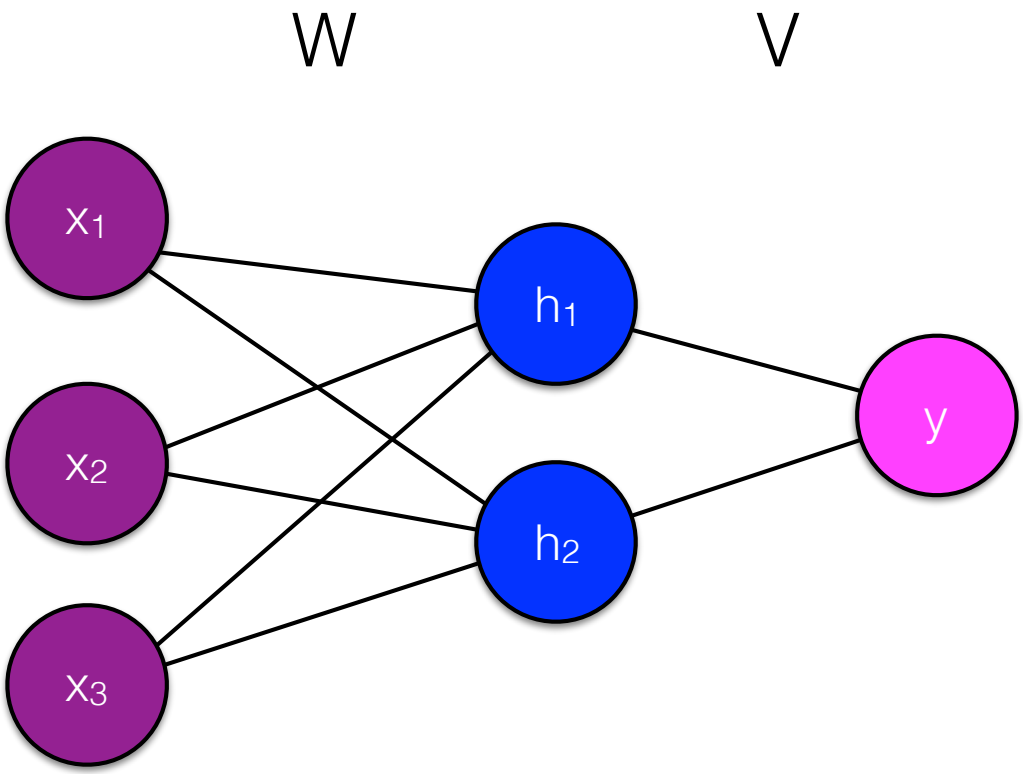
we can express y as a function only of the input x and the weights W and V


What a great sounding neural network!
But how to I get it to sound like?

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

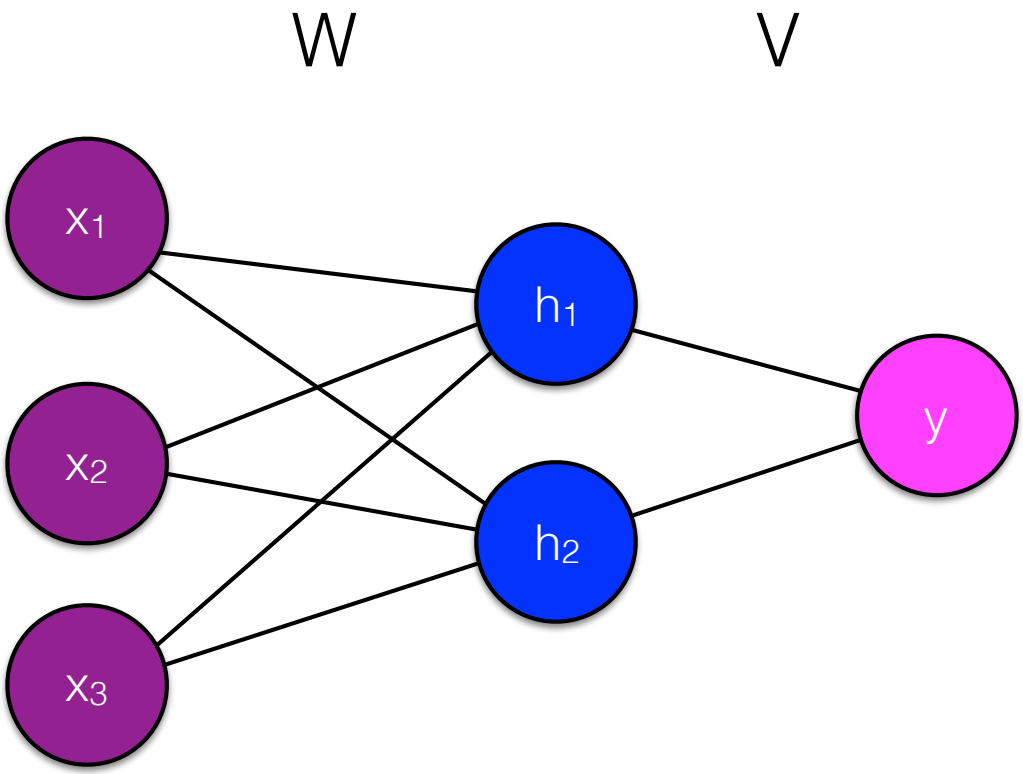
This is hairy, but differentiable

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

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



Neural networks are a series of functions chained together

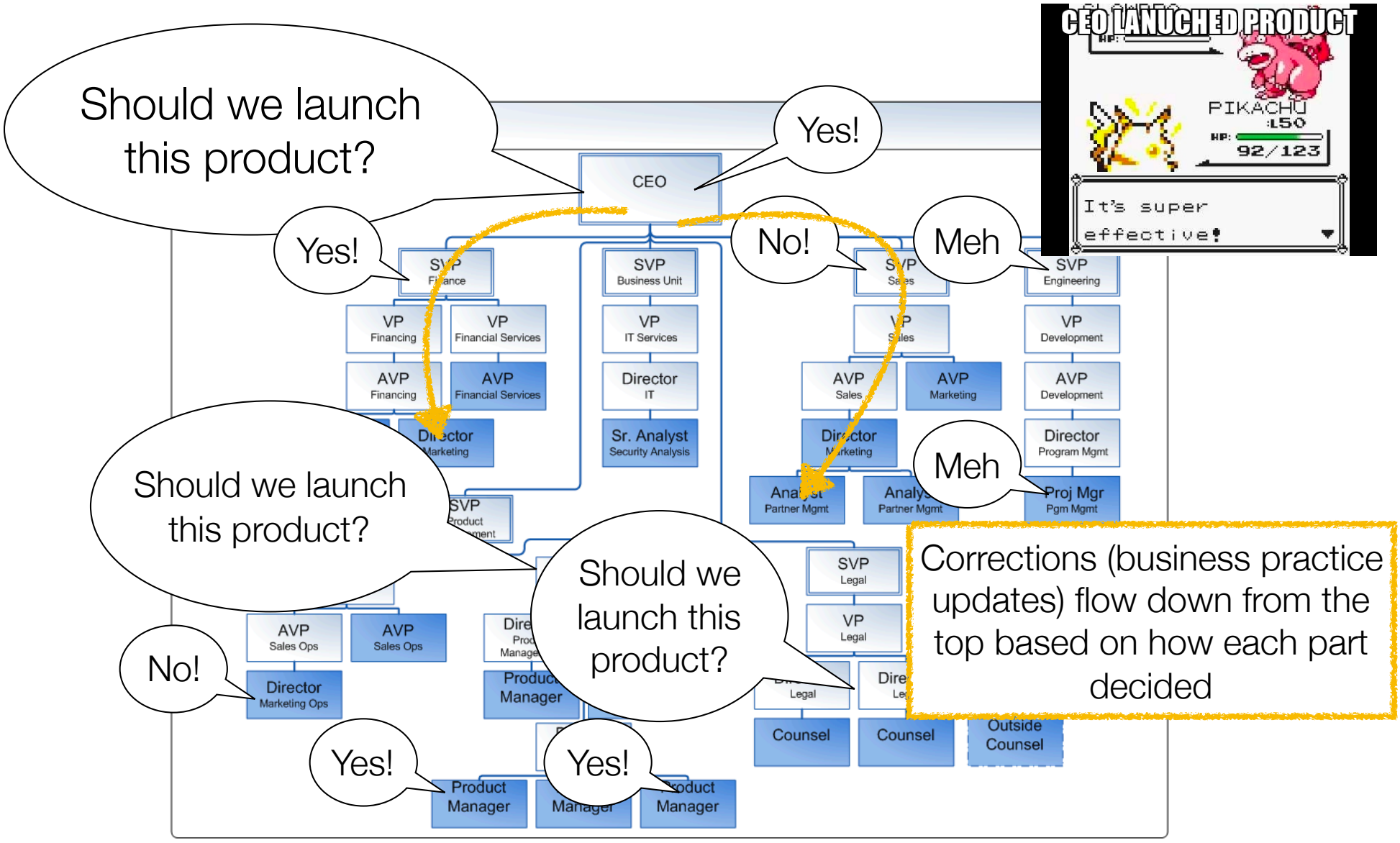
$$xW \rightarrow \sigma(xW) \rightarrow \sigma(xW)V \rightarrow \sigma(\sigma(xW)V)$$

The loss is another function chained on top

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

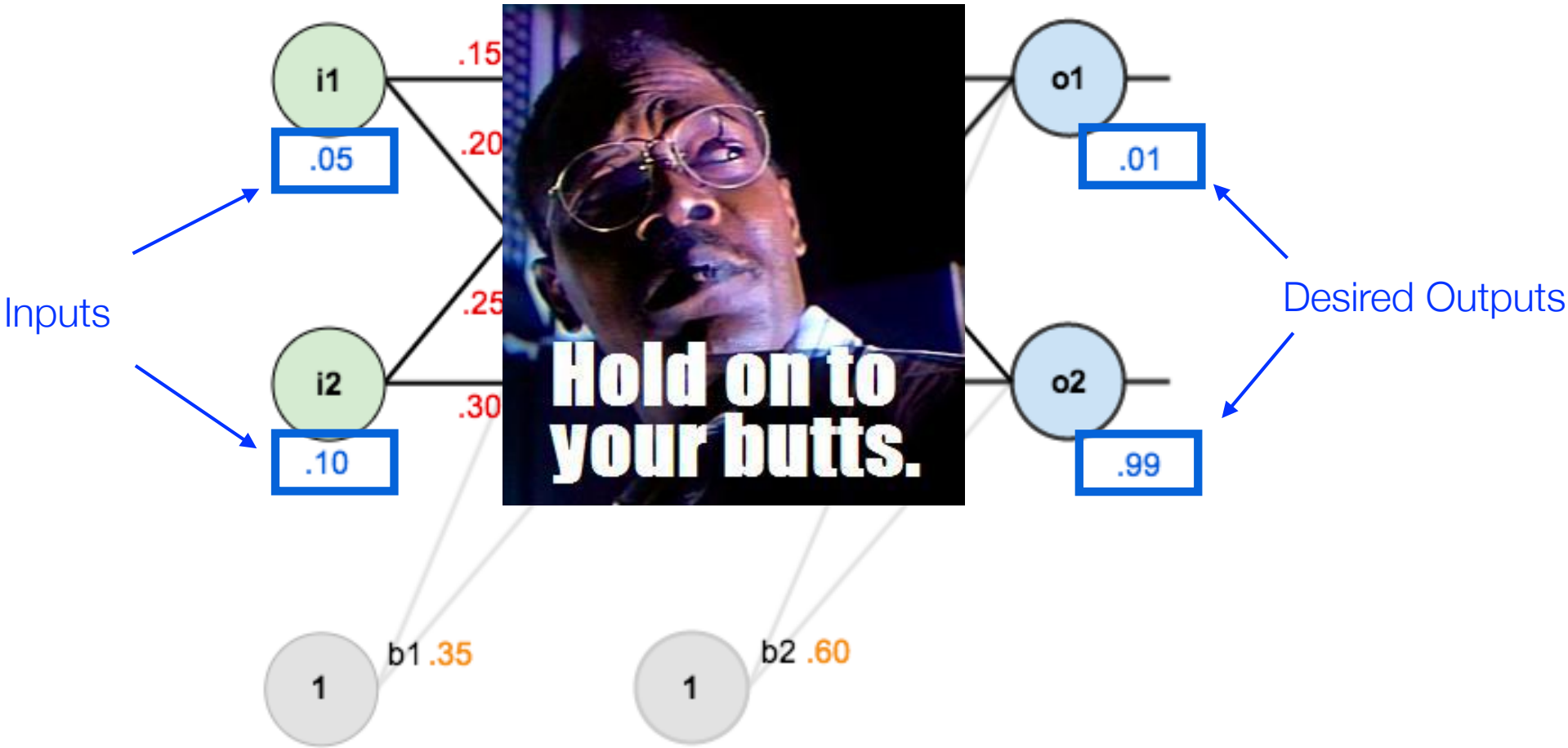
The loss tells us how much the prediction was wrong

An analog on how to update a neural network



Seilevel

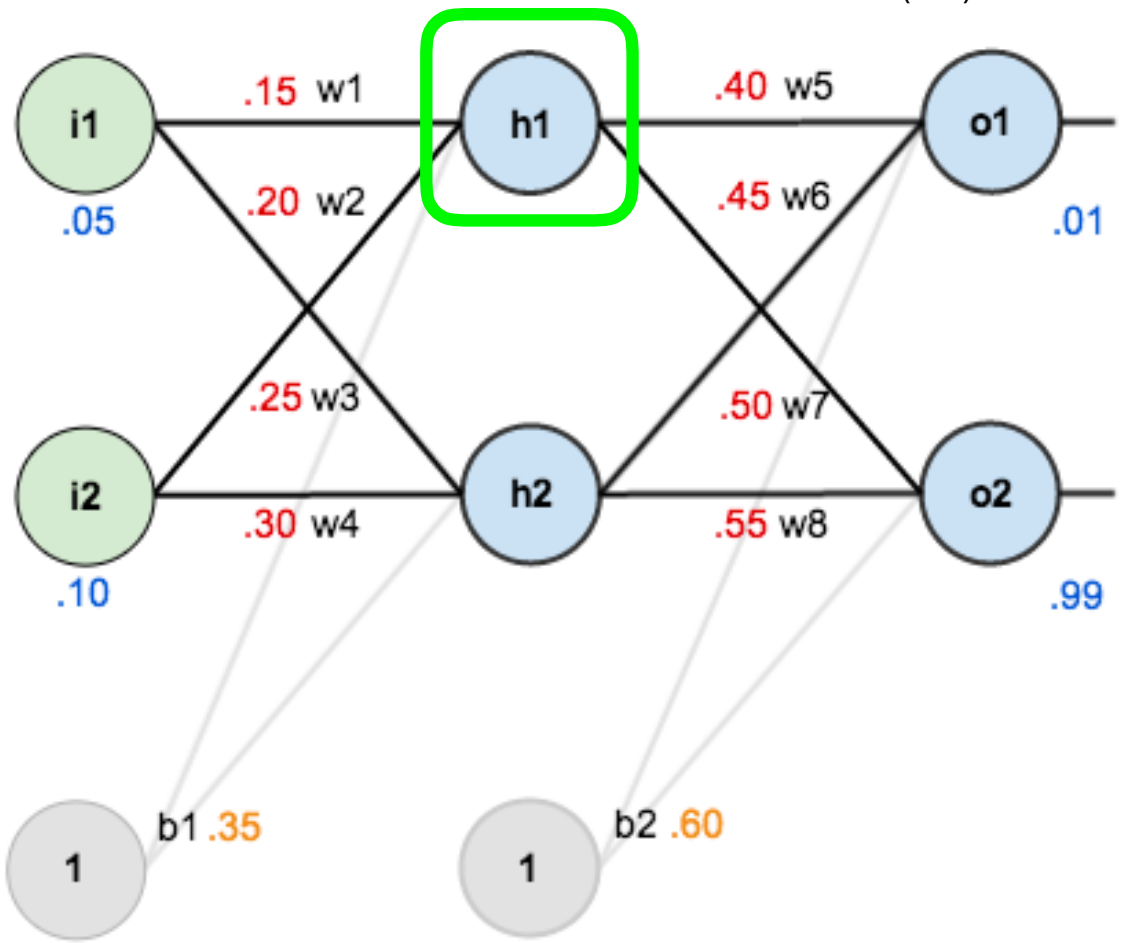
Let's try updating a real network!



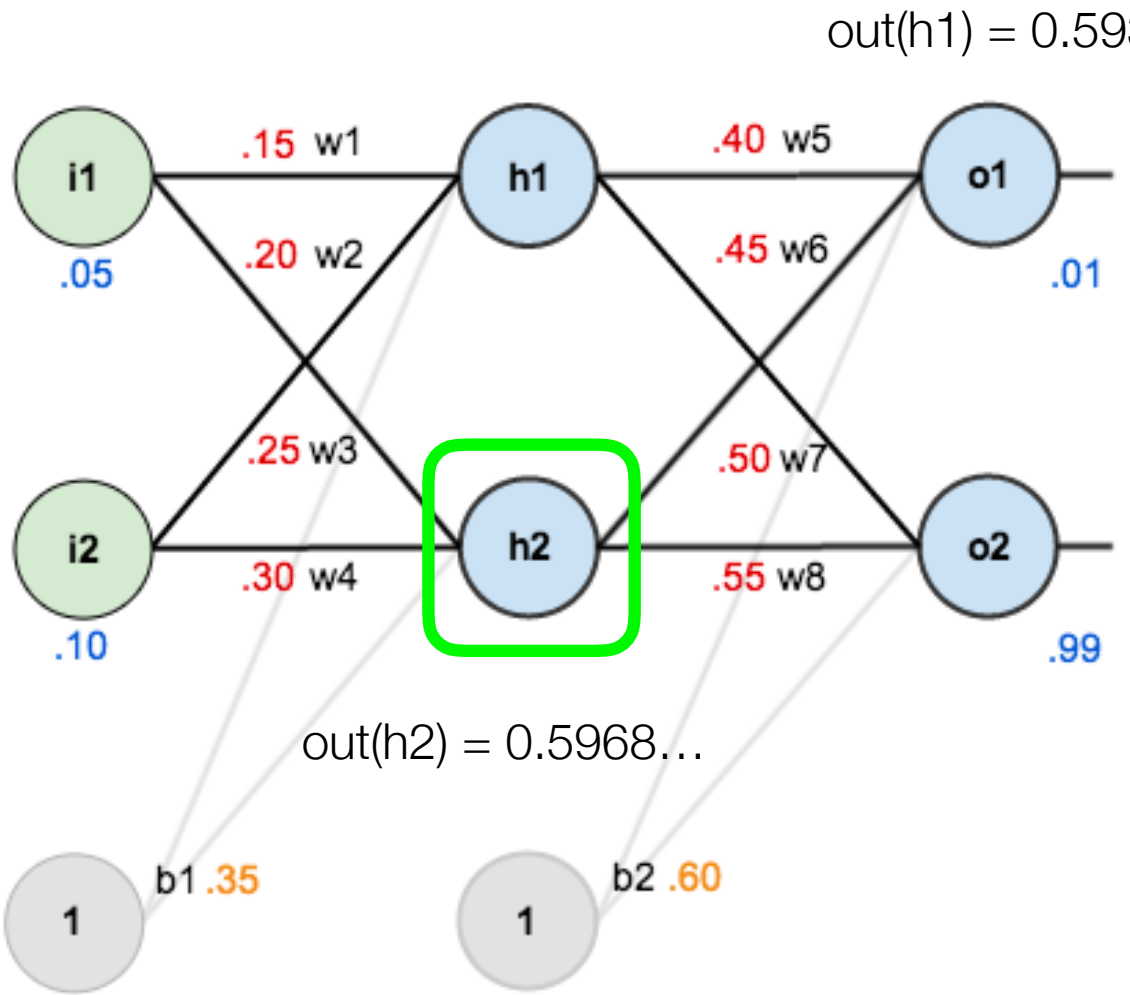
Let's try updating a real network!

net(h1) = w1*i1 + w2*i2 + b1
= 0.15*0.05 + 0.20*0.10 + 0.35
= 0.3775

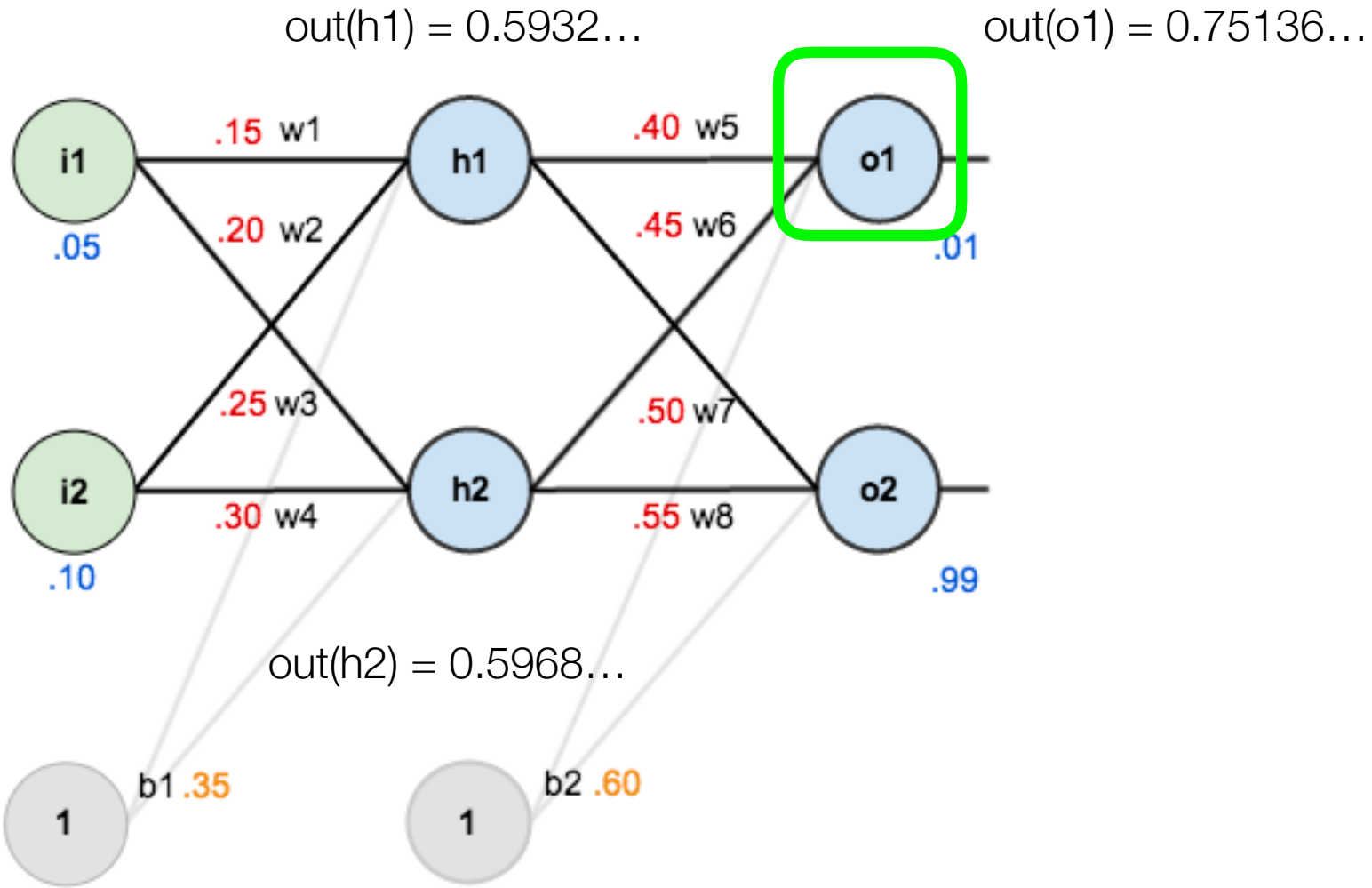
out(h1) = sigmoid(net(h1))
out(h1) = 1/ (1 + e^-0.3775)
out(h1) = 0.5932…



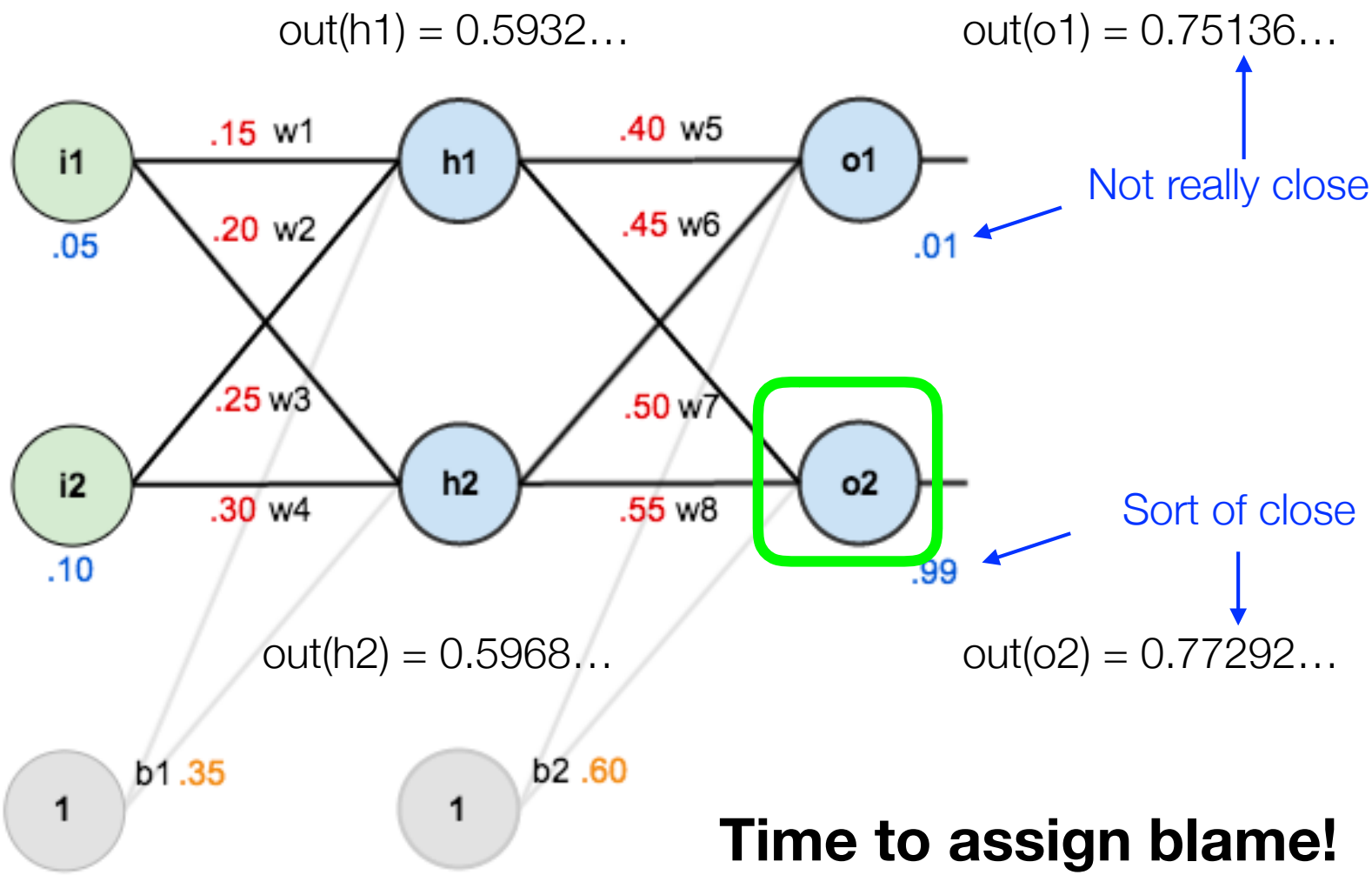
Let's try updating a real network!



Let's try updating a real network!



Let's try updating a real network!



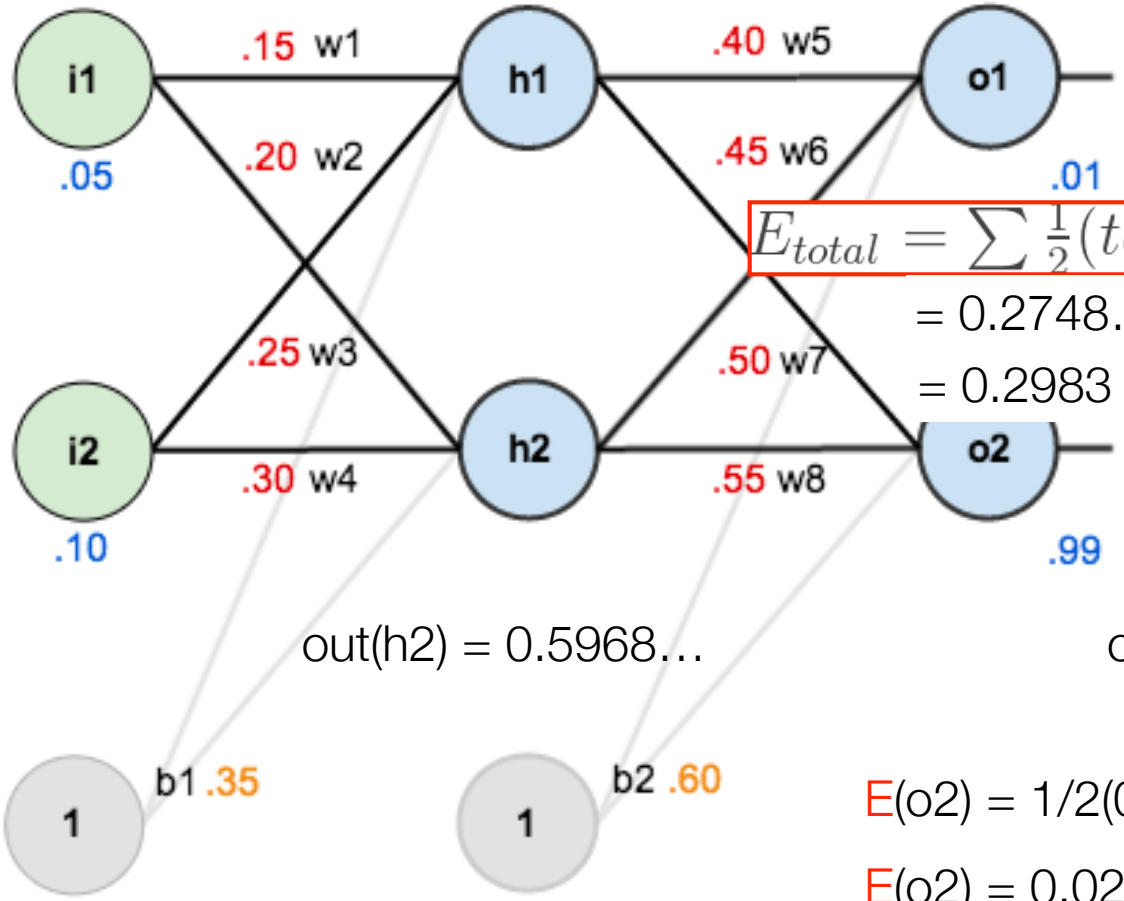
Let's try updating a real network!

E(o1) = 1/2(0.01 - 0.75136…)^2

E(o1) = 0.27481…

out(h1) = 0.5932…

out(o1) = 0.75136…



$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

= 0.2748… + 0.02456…

= 0.2983

out(h2) = 0.5968…

out(o2) = 0.77292…

E(o2) = 1/2(0.99 - 0.77292…)^2

E(o2) = 0.02356…

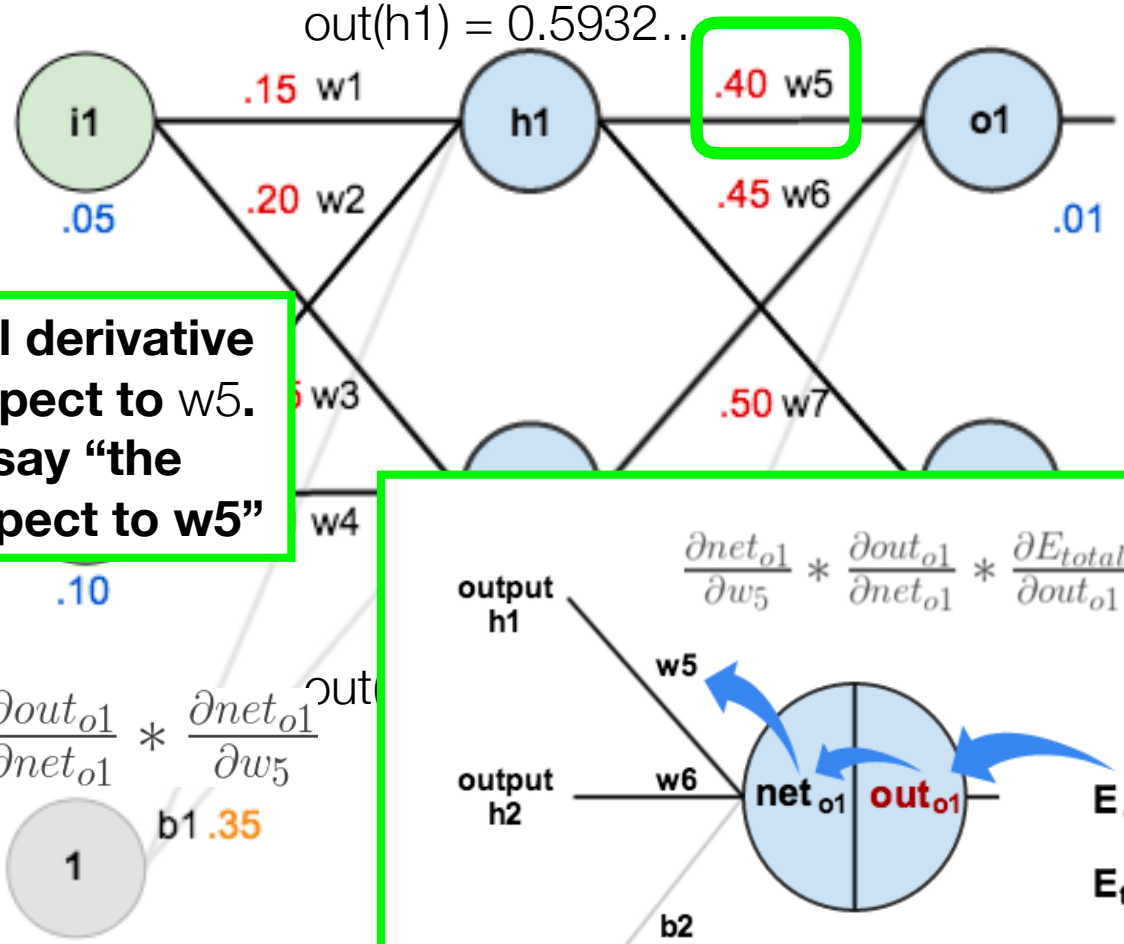
Let's try updating a real network!

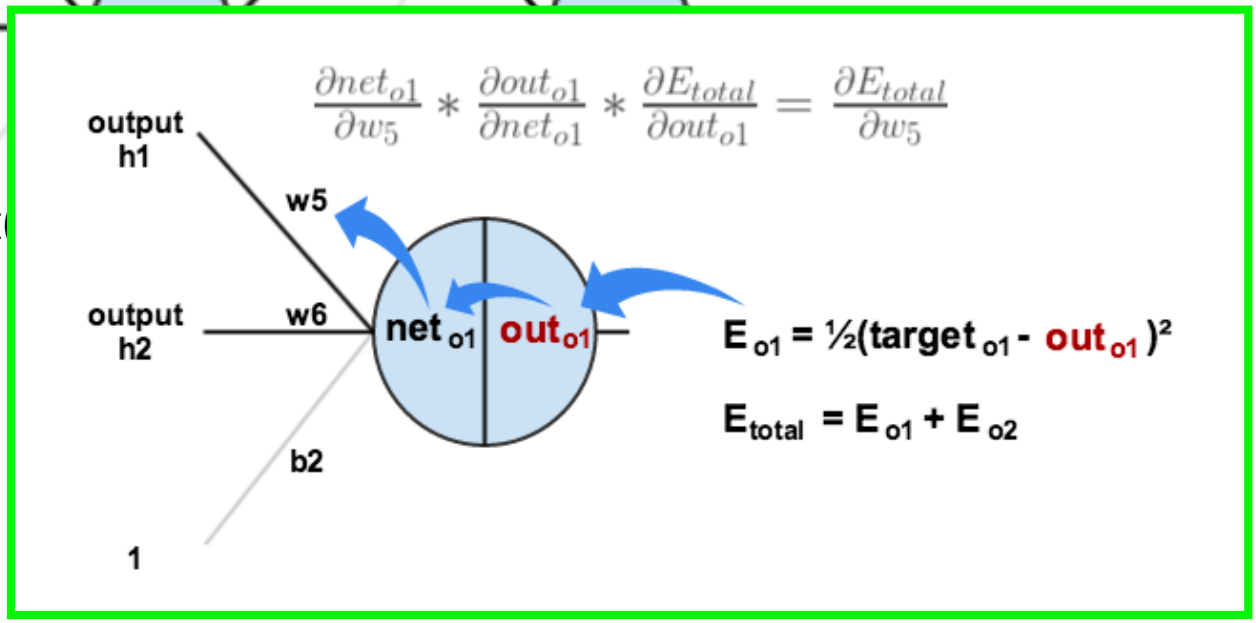
How much should we change w5 to reduce the total error?

$$\frac{\partial E_{total}}{\partial w_5}$$

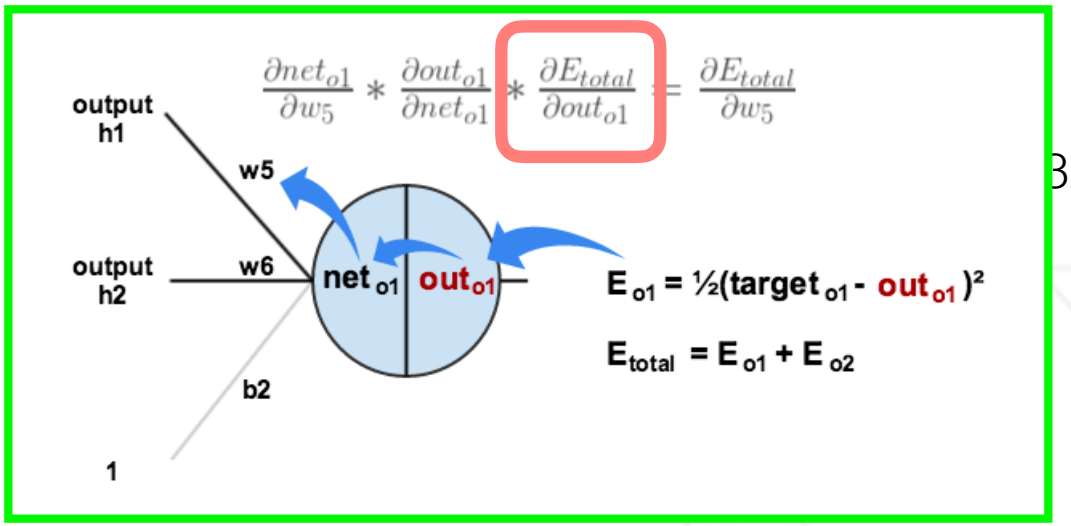
This is the partial derivative of E *total* **with respect to** w5**. You can also say "the gradient with respect to w5"**

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$





Let's try updating a real network!



We're asking how much does the total error change with respect to the output

$$E_{total} = \frac{1}{2}(target_{o1} - out_{o1})^2 + \frac{1}{2}(target_{o2} - out_{o2})^2$$

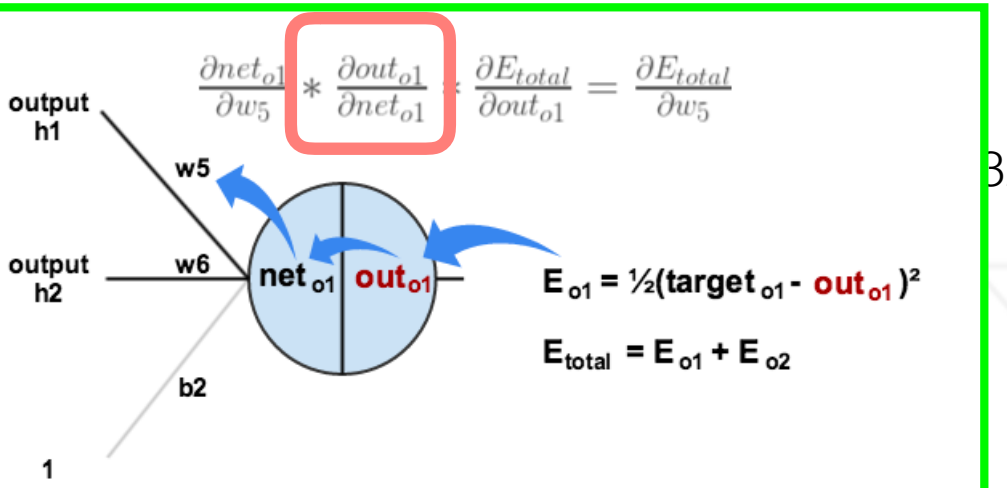
$$\frac{\partial E_{total}}{\partial out_{o1}} = 2 * \frac{1}{2}(target_{o1} - out_{o1})^{2-1} * -1 + 0$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = -(target_{o1} - out_{o1})$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = -(0.01 - 0.75136507) = 0.74136507$$

$\frac{\partial E_{total}}{\partial w_5}$ **is the partial derivative of** E *total* **with respect to** w5**. You can also say "the gradient with respect to w5"**

Let's try updating a real network!



32.

We're asking how much does the output of o1 change with respect to the total net input (i.e., the sum before the sigmoid)

$$out_{o1} = \frac{1}{1+e^{-net_{o1}}}$$

$$\frac{\partial out_{o1}}{\partial net_{o1}} = out_{o1}(1 - out_{o1})$$

$$= 0.75136507(1 - 0.75136507)$$

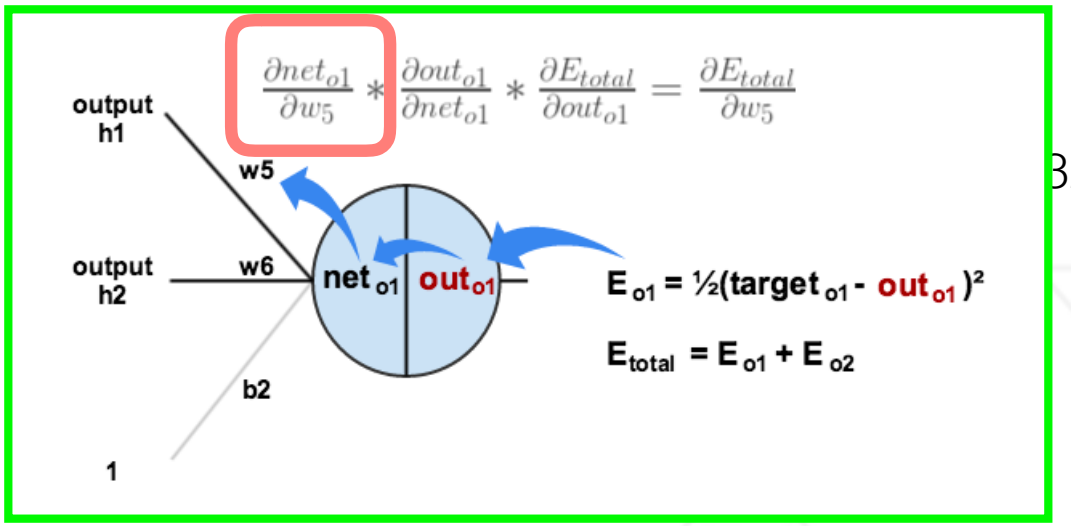
$$= 0.186815602$$

$\frac{\partial E_{total}}{\partial w_5}$ **is the partial derivative of** E *total* **with respect to** w5**. You can also say "the gradient with respect to w5"**

Let's try updating a

We're asking how much does the total net output of o1 change with respect to w5

32…



$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

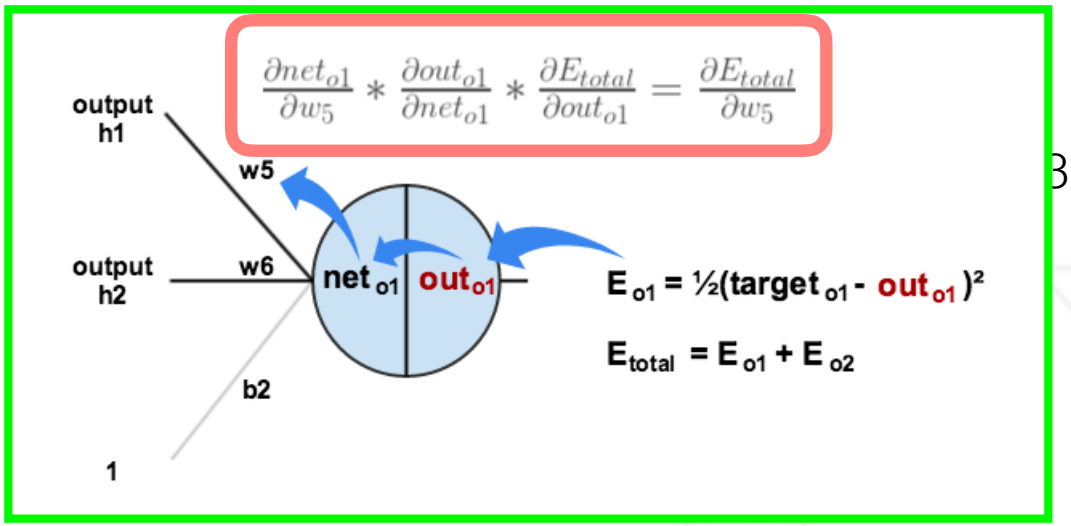
$$\frac{\partial net_{o1}}{\partial w_5} = 1 * out_{h1} * w_5^{(1-1)} + 0 + 0$$

$$\frac{\partial net_{o1}}{\partial w_5} = 1 * out_{h1} * w_5^{(1-1)} + 0 + 0 = out_{h1} = 0.593269992$$

$$= out_{h1} = 0.593269992$$

$\frac{\partial E_{total}}{\partial w_5}$ **is the partial derivative of** E *total* **with respect to** w5**. You can also say "the gradient with respect to w5"**

Let's try updating a real network!



32…

E(o1) = 0.27481…

out(o1) = 0.75136…

E *total*= 0.2983

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

$$\frac{\partial E_{total}}{\partial w_5} = 0.74136507 * 0.186815602 * 0.593269992$$

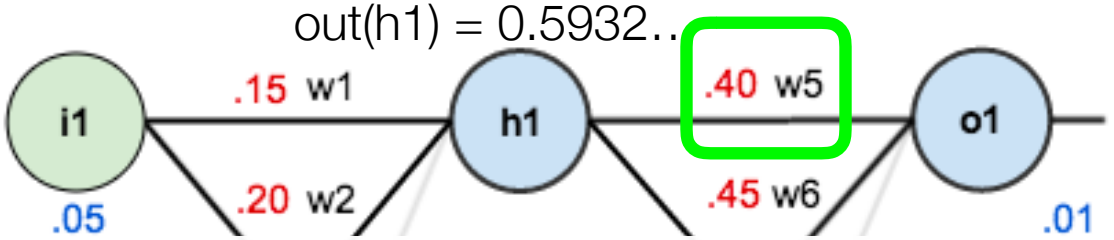
$$\frac{\partial E_{total}}{\partial w_5} = 0.082167041$$

$\frac{\partial E_{total}}{\partial w_5}$ **is the partial derivative of** E *total* **with respect to** w5**. You can also say "the gradient with respect to w5"**

Let's try updating a real network!

How much should we change w5 to reduce the total error?

$\frac{\partial E_{total}}{\partial w_5}$



The new value of w5

$$w_5^+ = w_5 - \eta * \frac{\partial E_{total}}{\partial w_5}$$

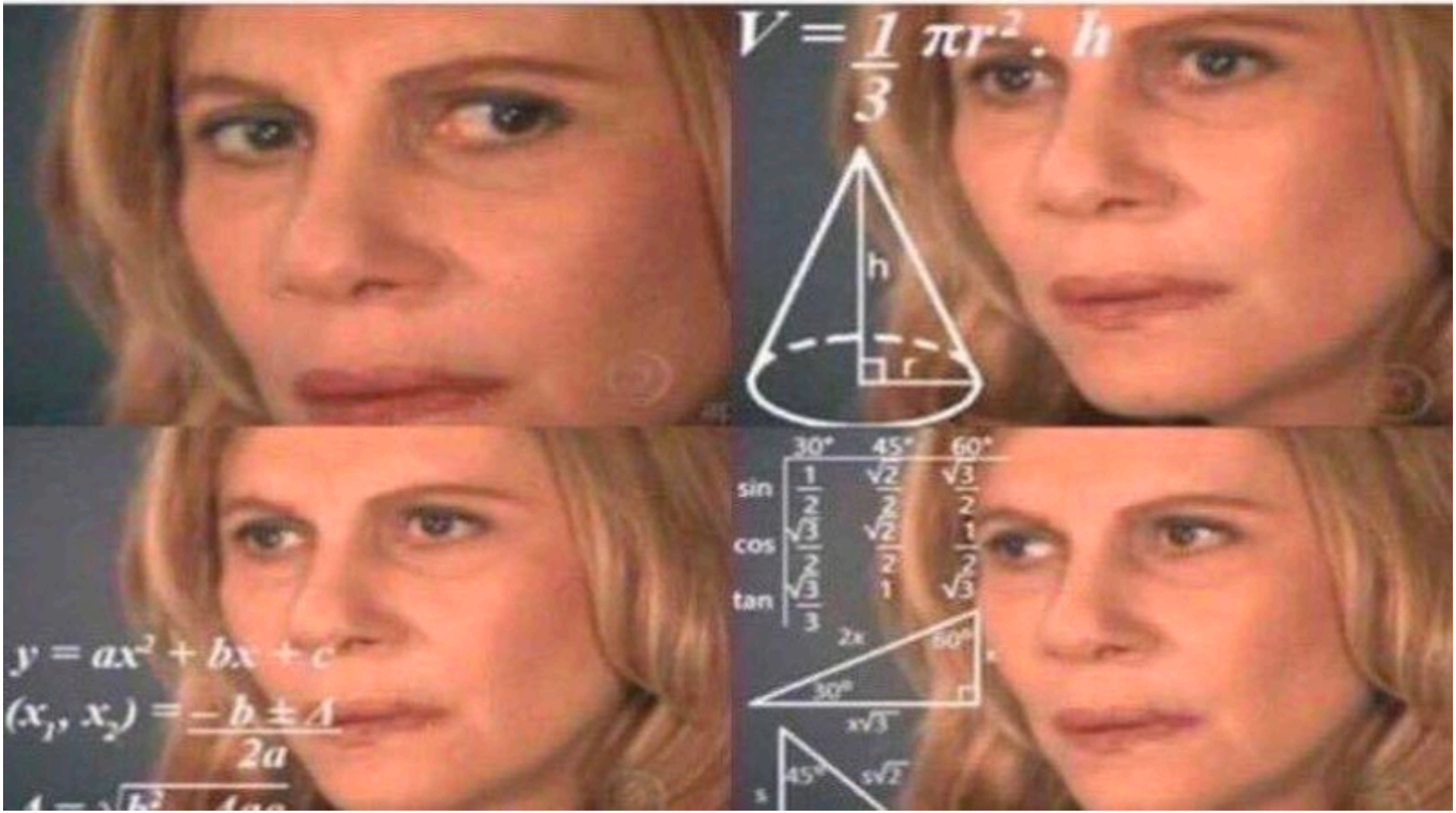
This is the *learning rate* for how much we update the network at each time step

$$w_5^+ = 0.4 - 0.5 * 0.082167041$$

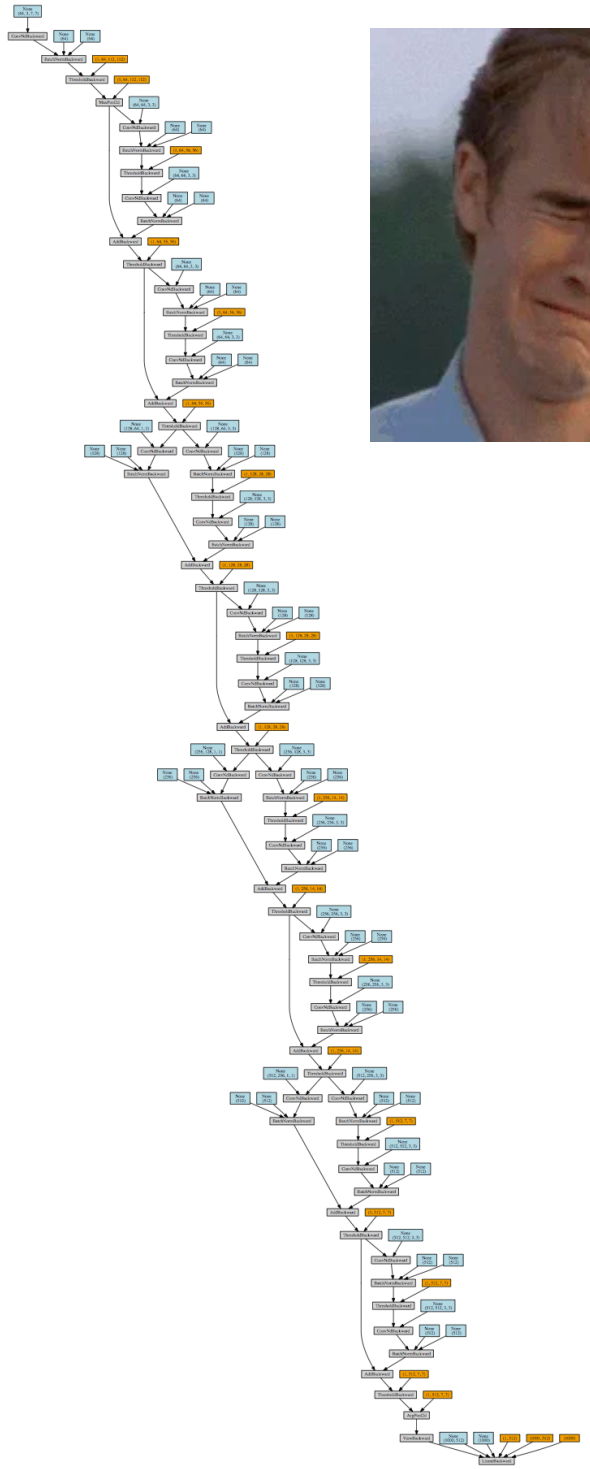
$$= 0.35891648$$

$\frac{\partial E_{total}}{\partial w_5}$ **is the partial derivative of** E *total* **with respect to** w5**. You can also say "the gradient with respect to w5"**

Your turn: solve for all the nodes!



You can walk through the whole example with more detail here:
https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/



Deep learning sounds impossible

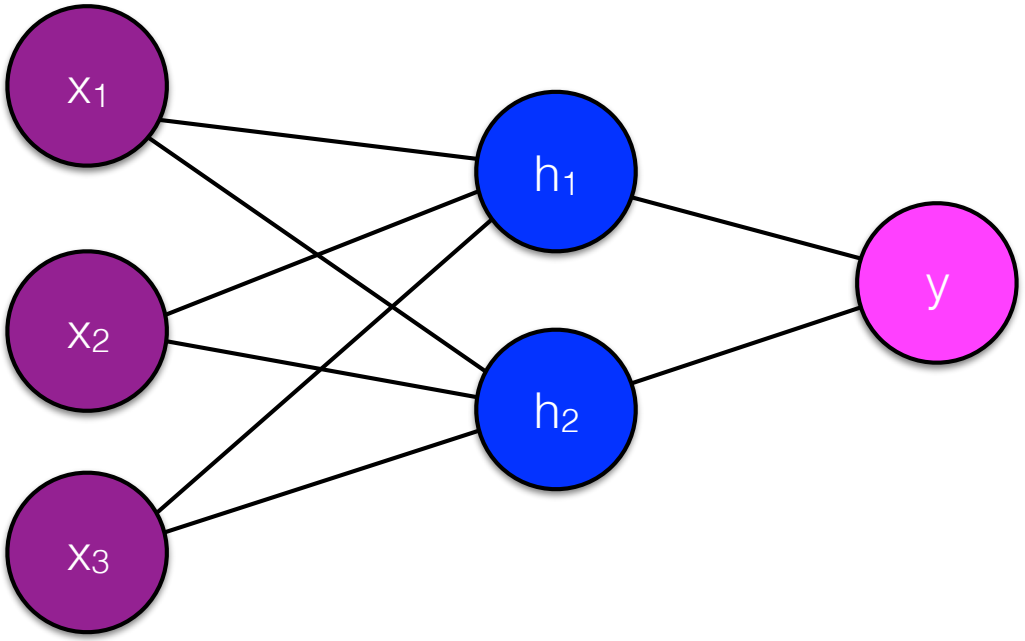
Recent advancements in neural networks are partly due to better hardware (GPUs), libraries, and algorithms like autograd that compute the gradients (differentials) for you

https://www.smithsonianmag.com/science-nature/what-can-rodents-tell-us-about-why-humans-love-180949441/

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.

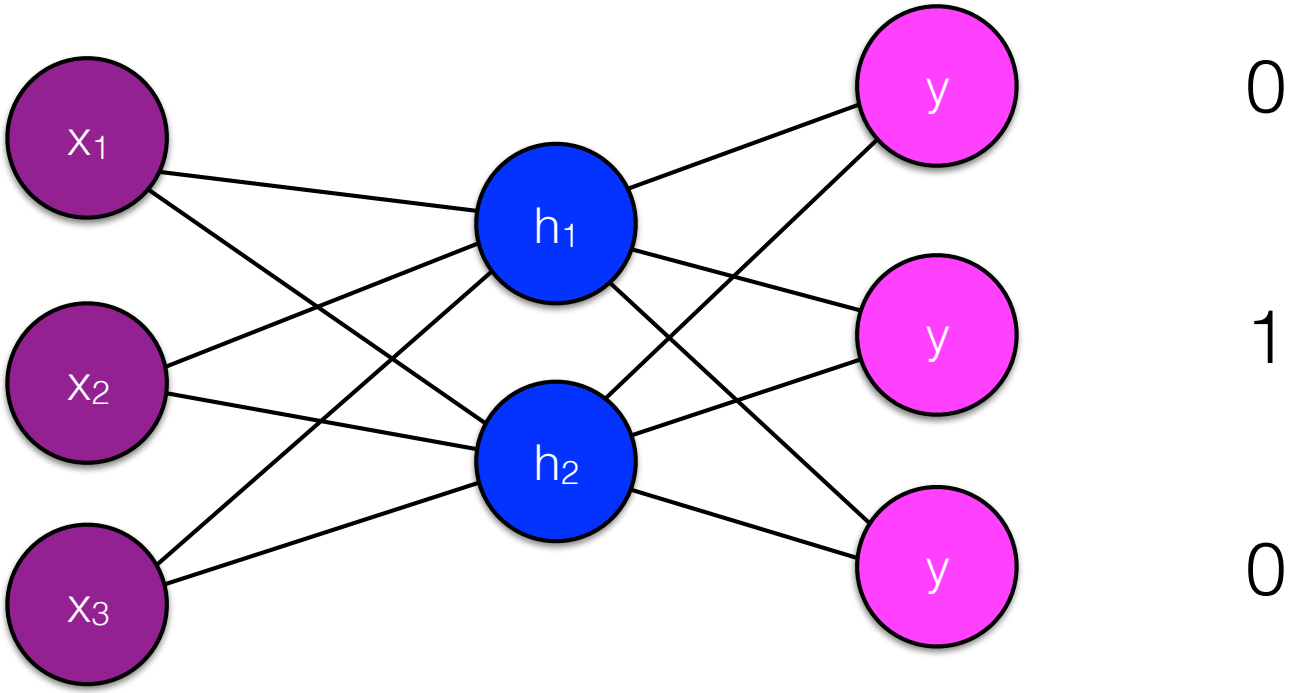
Neural network structures



1

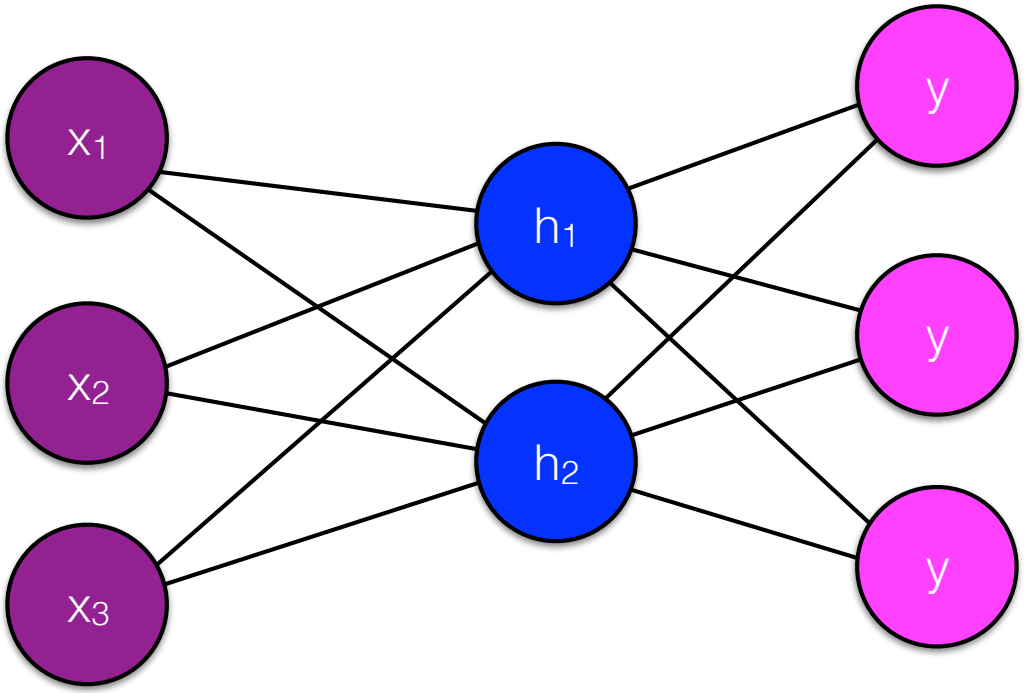
Output one real value

Neural network structures



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

Neural network structures



output 3 values, several = 1 in training data

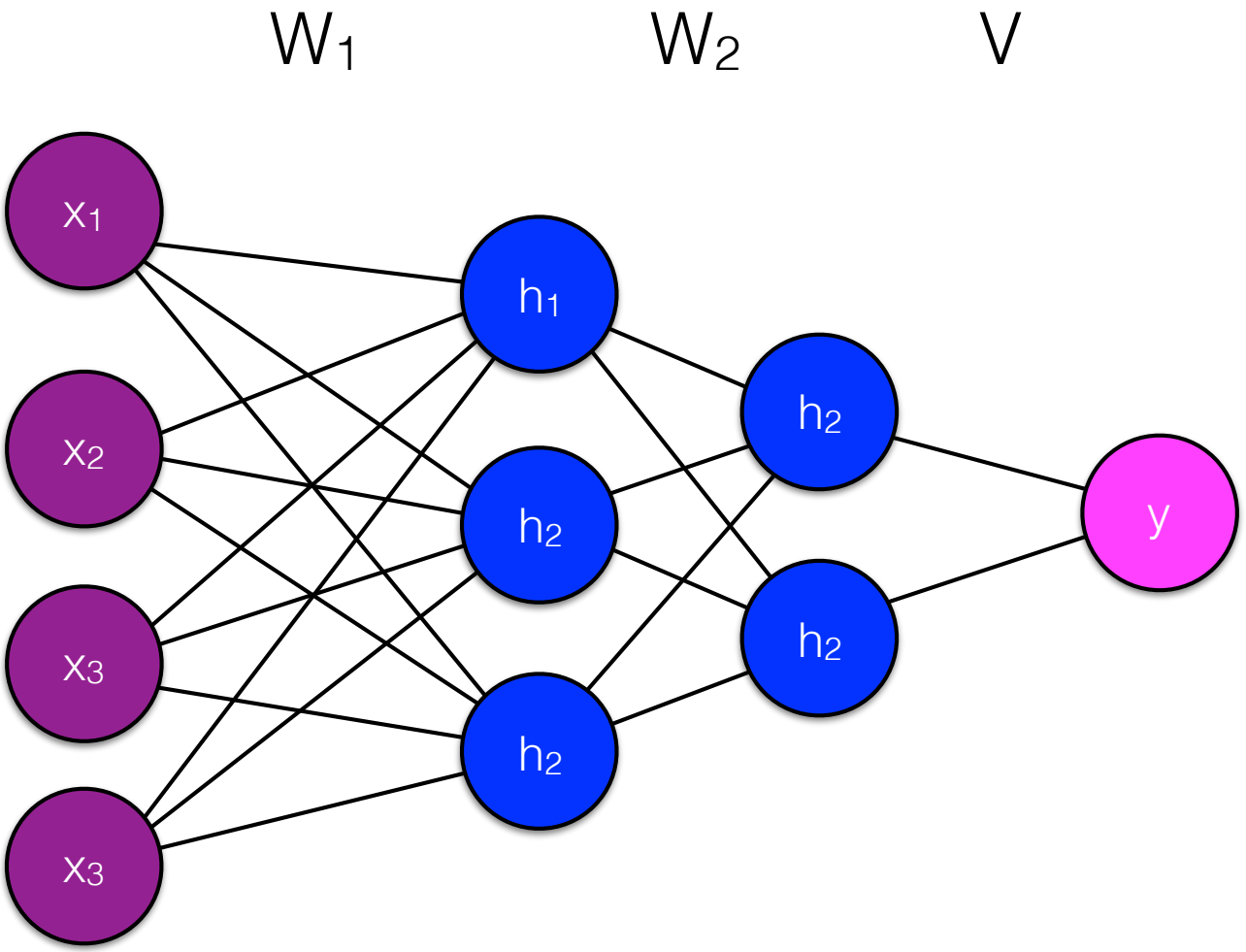
Regularization

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

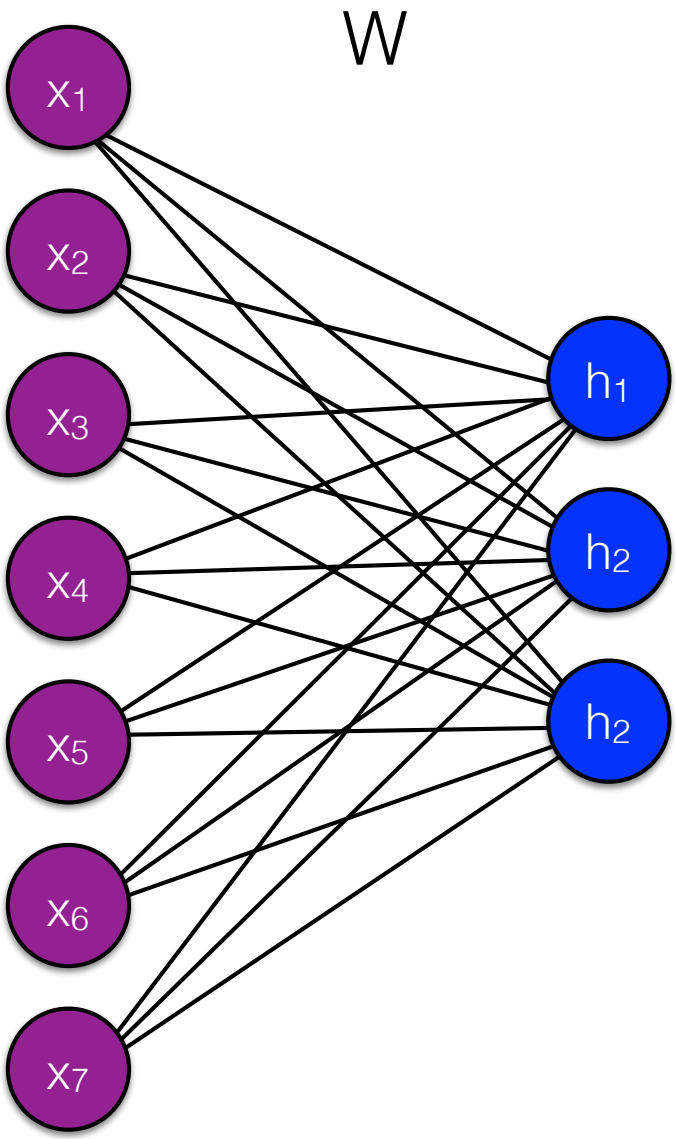
Regularization

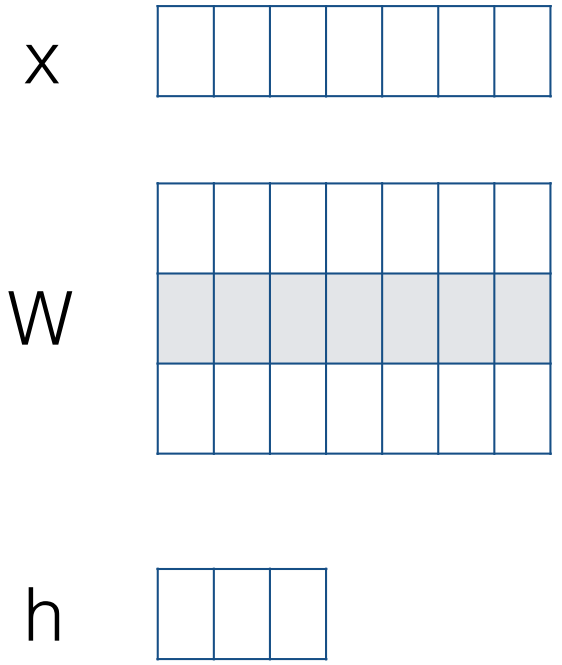
- L2 regularization: penalize W and V for being too large
- Dropout: when training on a <x,y> pair, randomly remove some node and weights.
- Early stopping: Stop backpropagation before the training error is too small.

Deeper networks



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

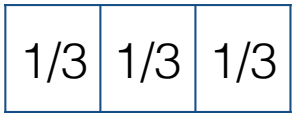
0	0	0	0	0
0	1	1	1	0
0	1	1	1	0
0	1	1	1	0
0	0	0	0	0

blurring

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

1D Convolution

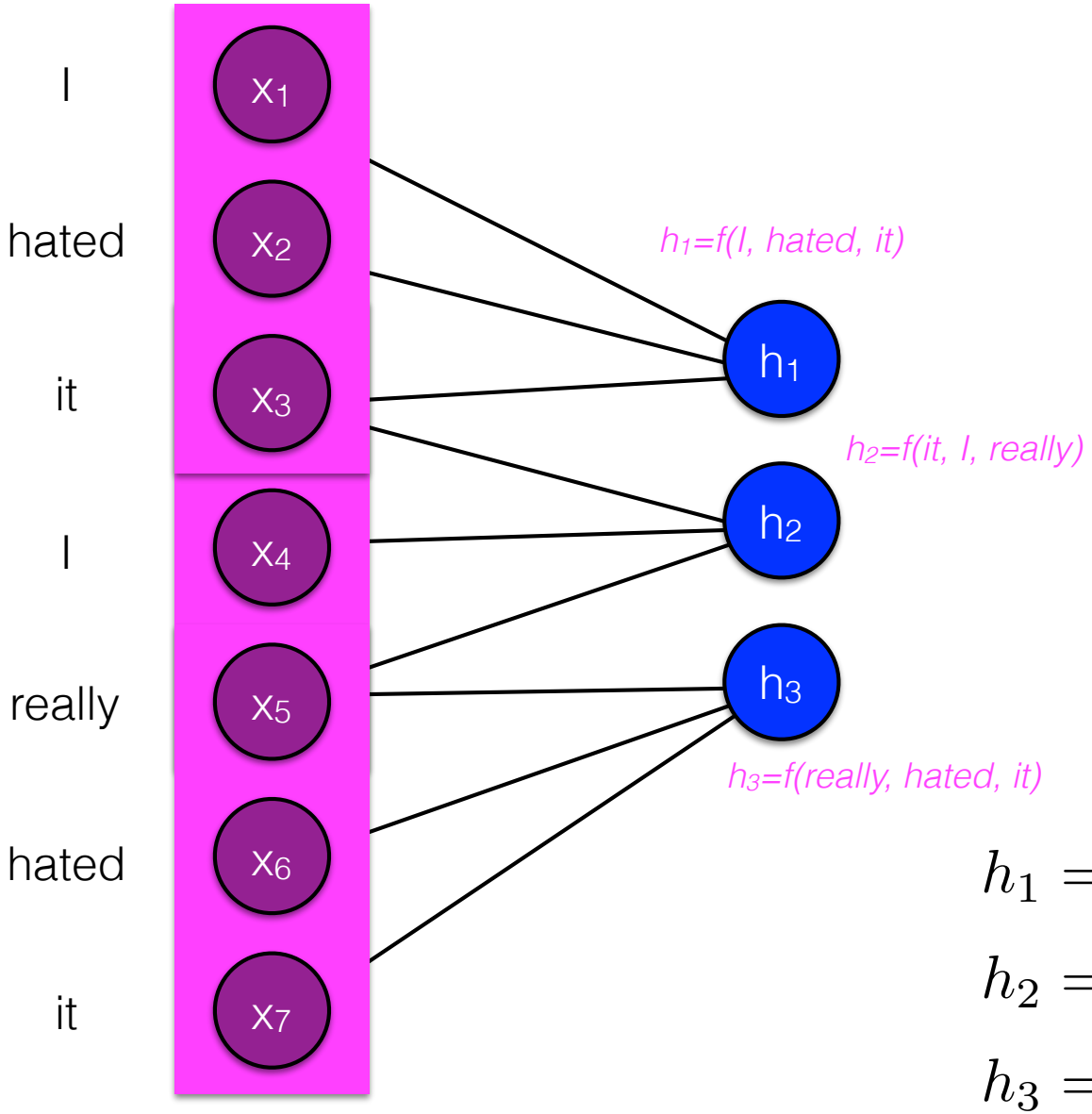
convolution *K*

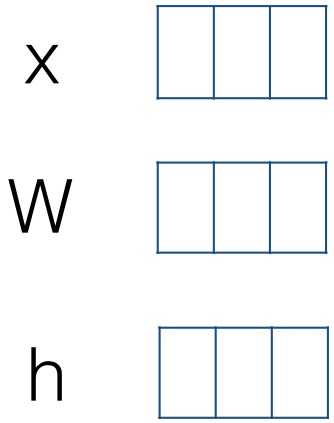


x

$x_{1:4}^\top K$ = moving average

Convolutional networks



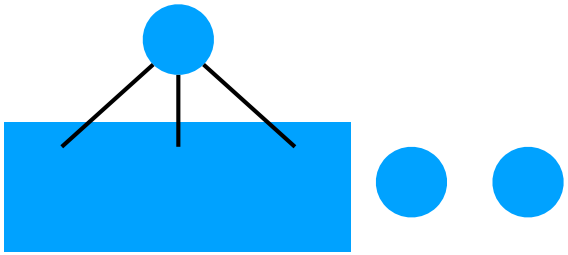


$$h_1 = \sigma(x_1 W_1 + x_2 W_2 + x_3 W_3)$$

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

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

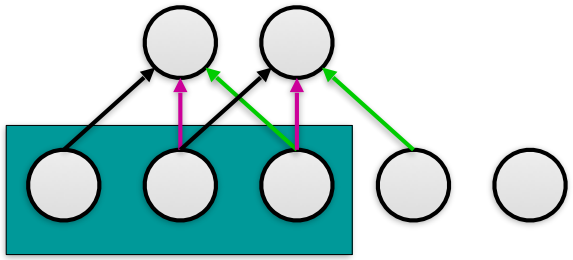
Basic Idea of CNNs



Hidden layer

Input

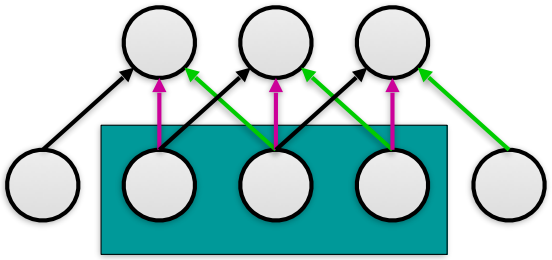
Basic Idea of CNNs



Hidden layer

Input

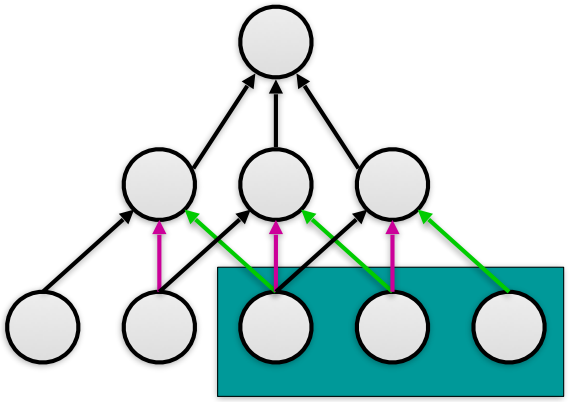
Basic Idea of CNNs



Hidden layer

Input

Basic Idea of CNNs



Hidden layer

Input

Cats' Brains and CNNs

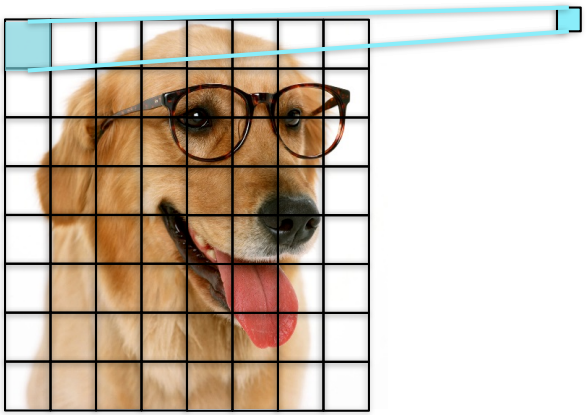
(Be nice to your cats)

With the exception of distinguishing cucumbers from deadly snakes, cats' brains are really good at image classification.

Yann LeCun et al. (1998) found that a neural network that works like a cat's visual cortex was good at digit classification.

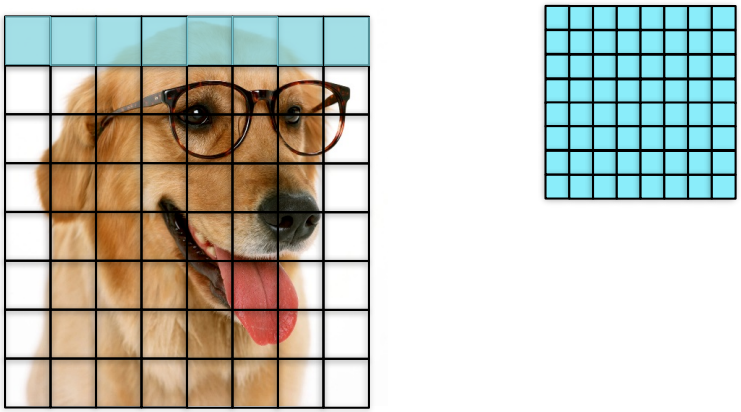
When machines got powerful enough, it also turned out to be great at image classification. (Krizhevsky et al. 2012)

CNN for Image Classification



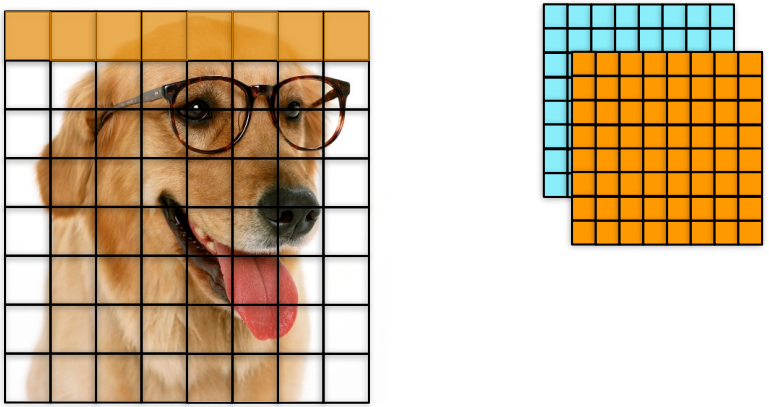
Convolutional Layer

CNN for Image Classification



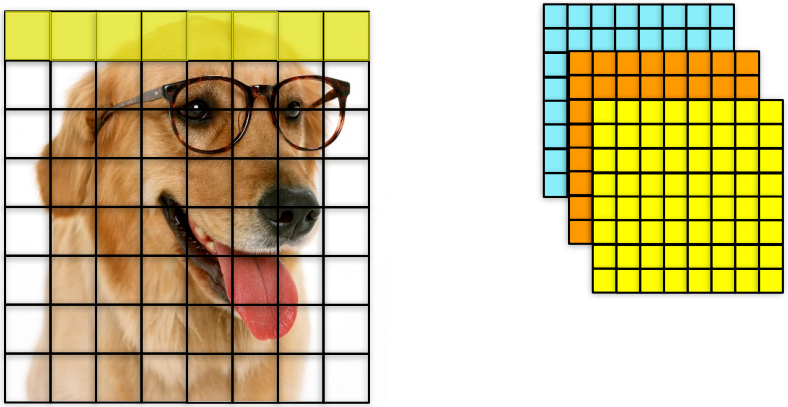
Convolutional Layer

CNN for Image Classification



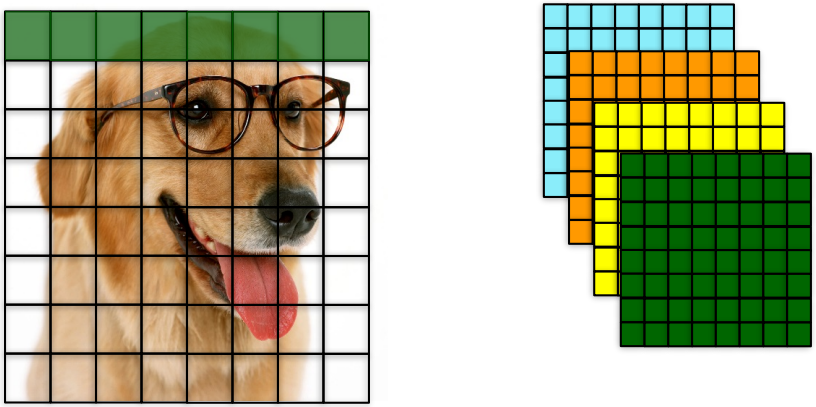
Convolutional Layer

CNN for Image Classification

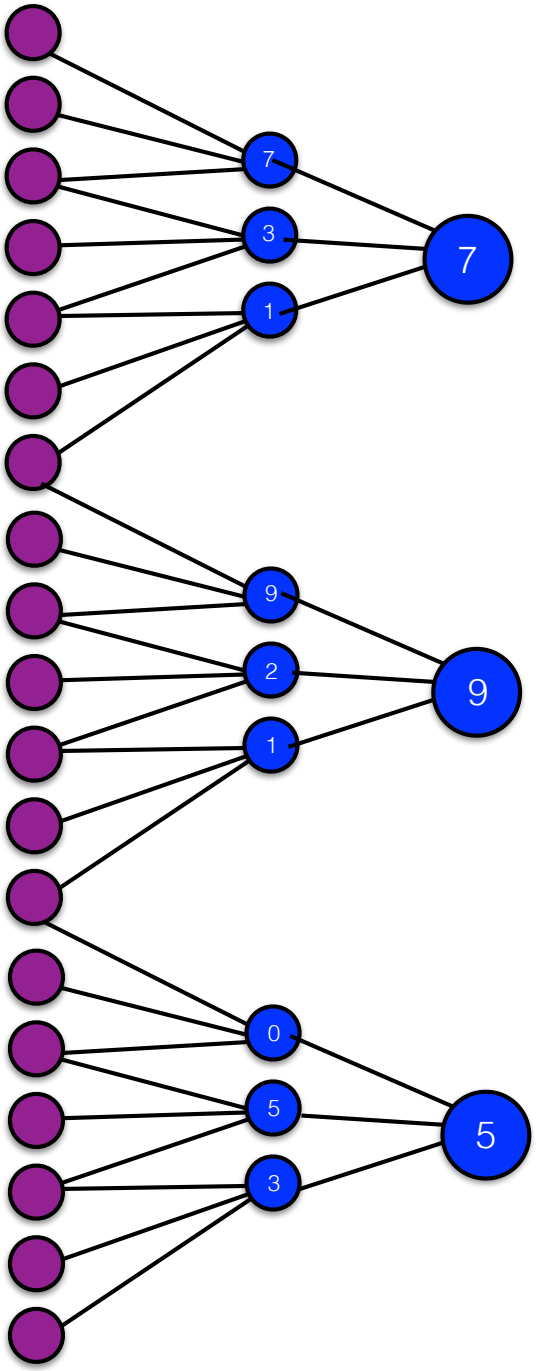


Convolutional Layer

CNN for Image Classification



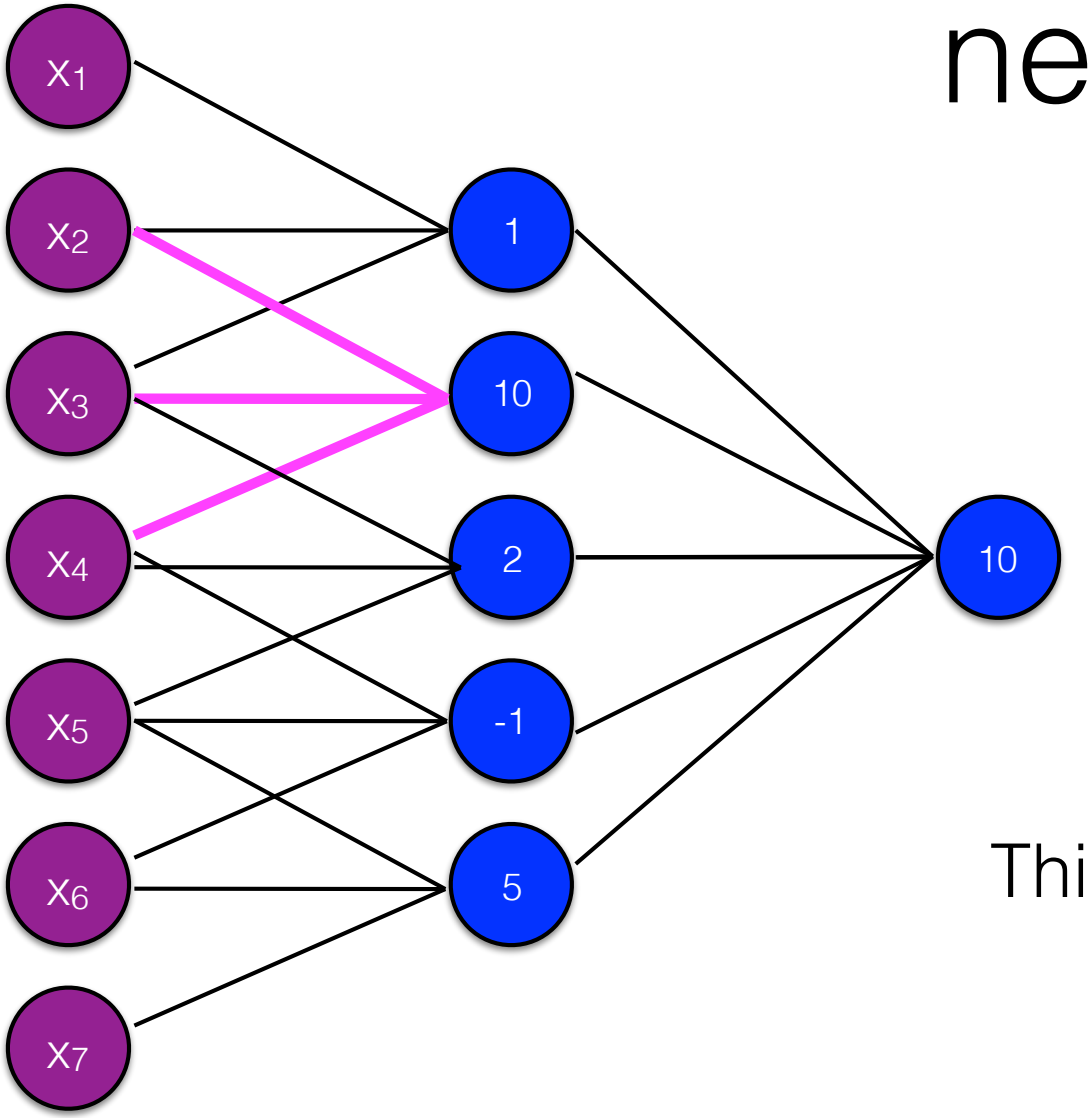
Convolutional Layer



Pooling

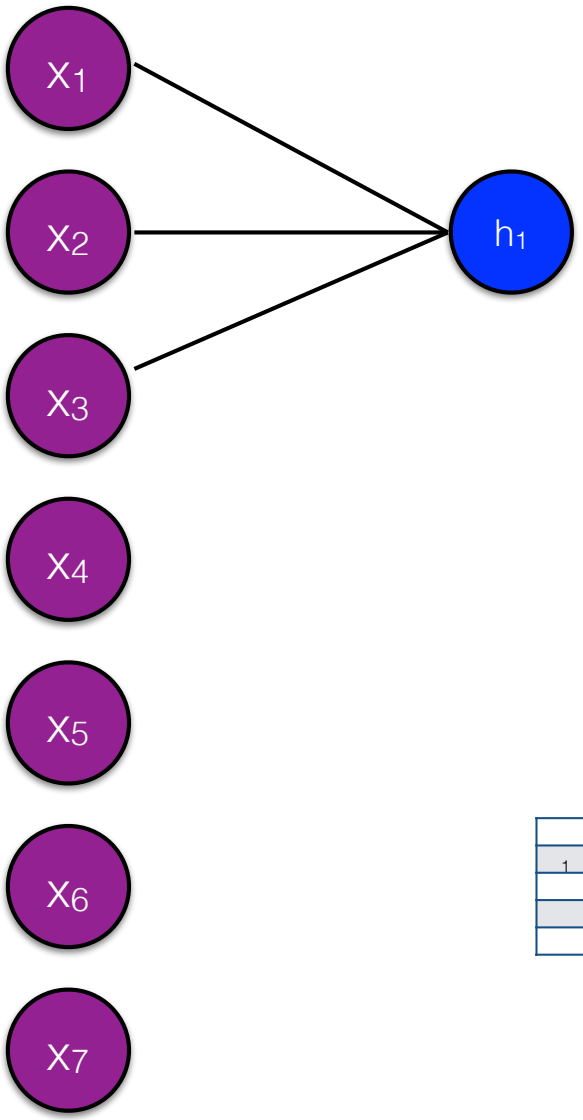
- Down-samples a layer by selecting a single point from some set
- Max-pooling selects the largest value

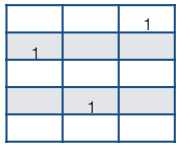
Convolutional networks



This defines one filter.

convolution max pooling

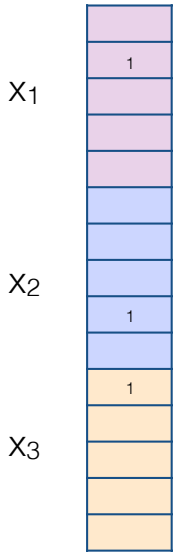




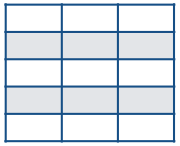
x

x_1 x_2 x_3

vectorize

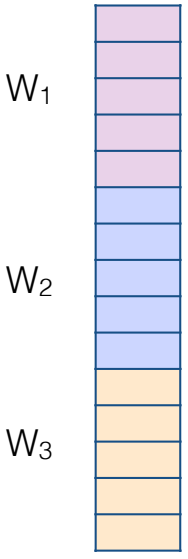


W



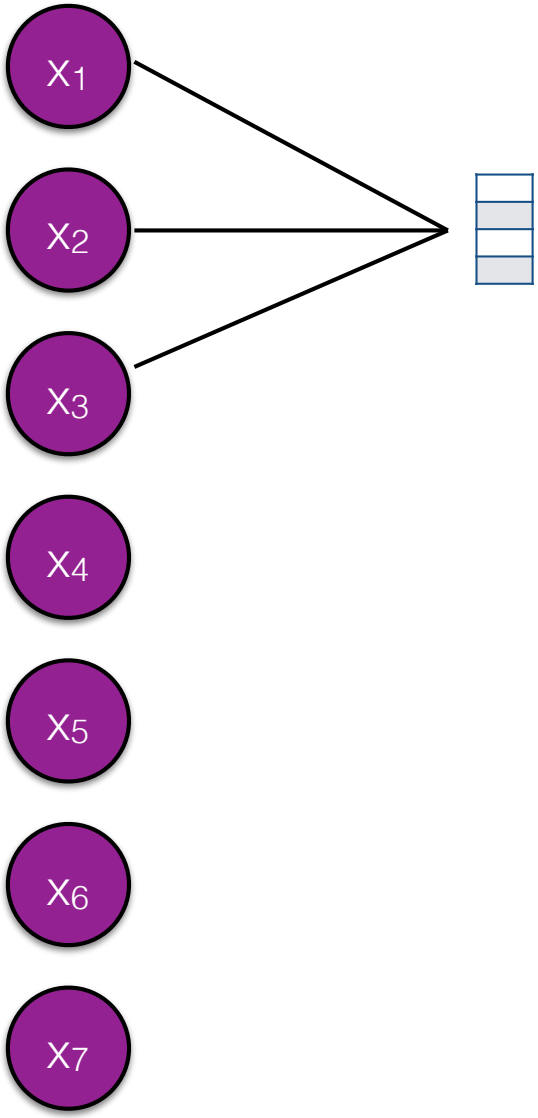
W_1 W_2 W_3

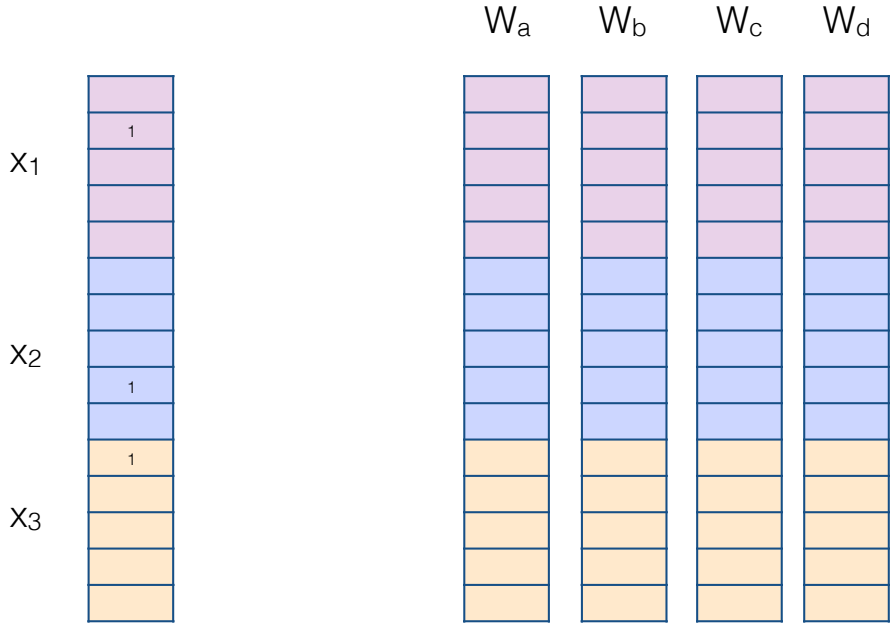
vectorize



$$h_1 = \sigma(x^\top W)$$

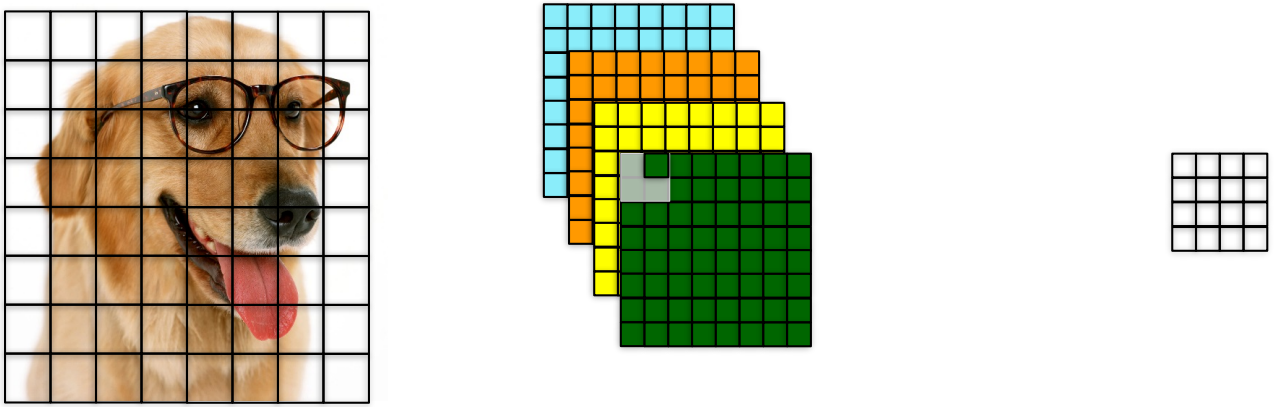
We can specify multiple filters; each filter is a separate set of parameters to be learned





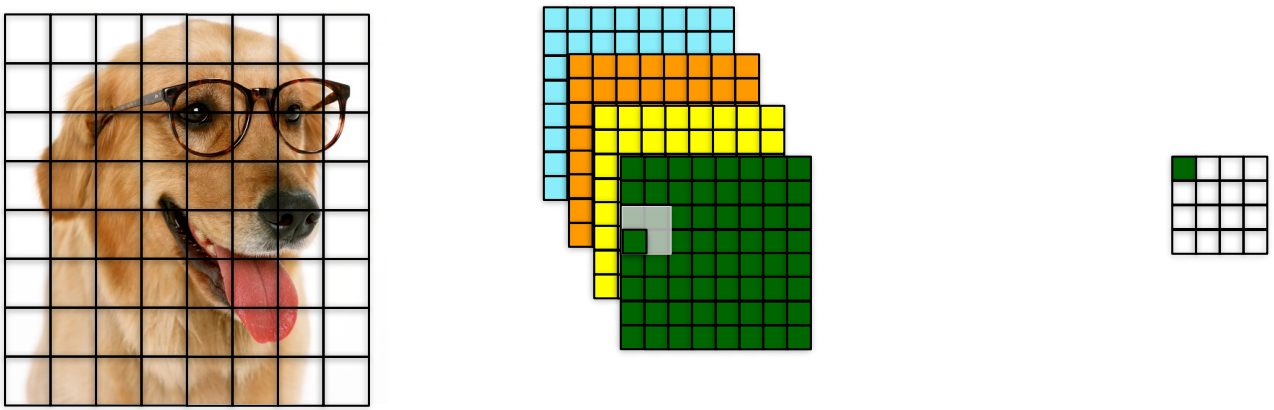
$$h_1 = \sigma(x^\top W) \in R^4$$

CNN for Image Classification



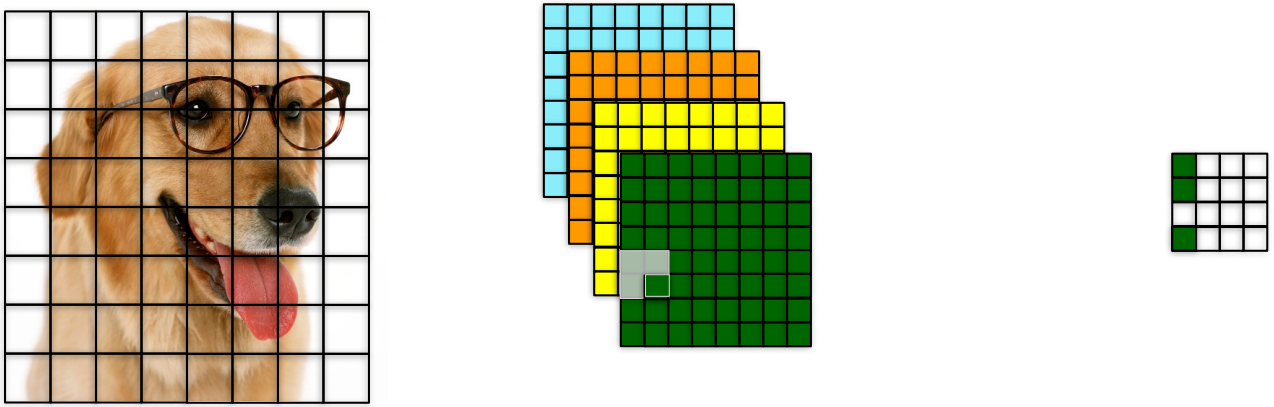
Max Pooling

CNN for Image Classification



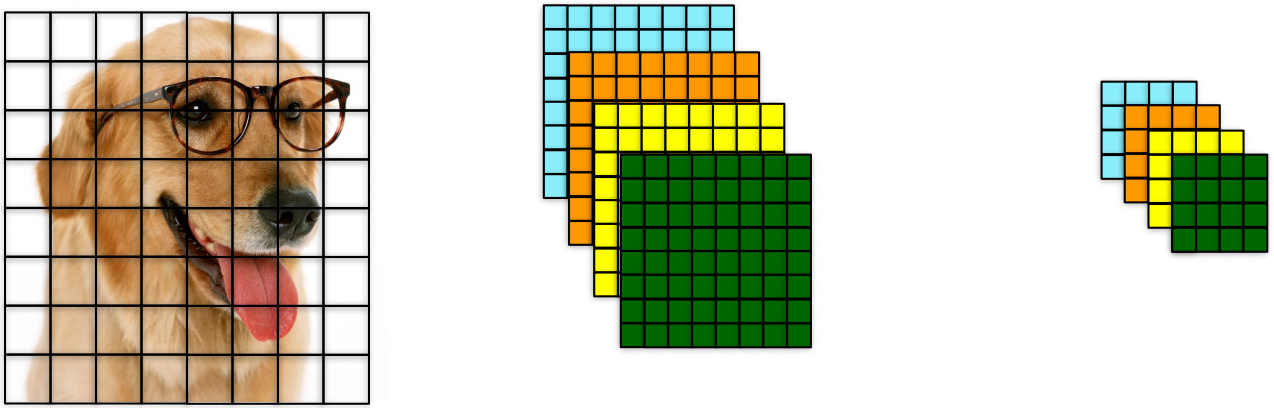
Max Pooling

CNN for Image Classification



Max Pooling

CNN for Image Classification

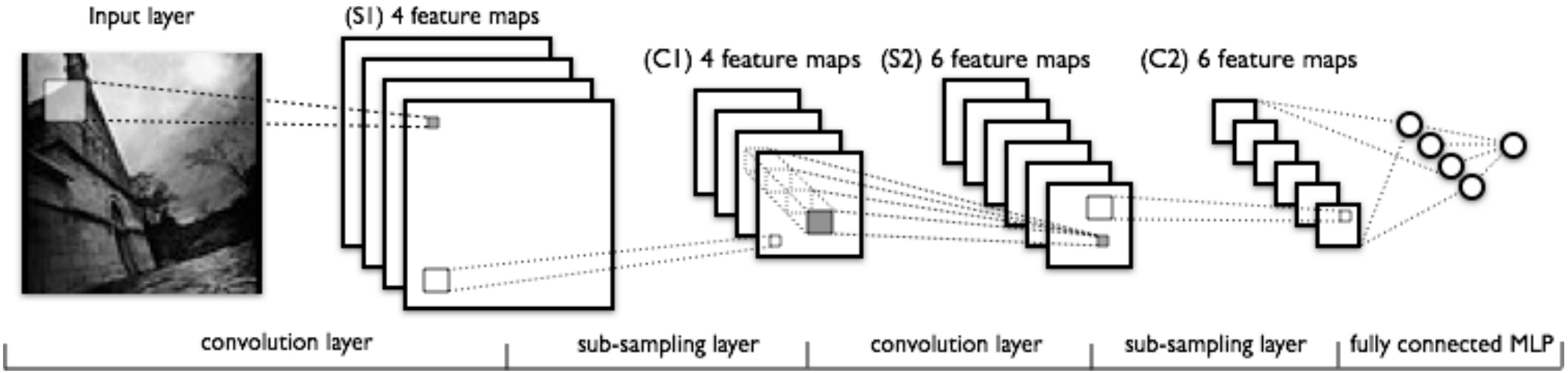


Max Pooling

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

CNN for Image Classification

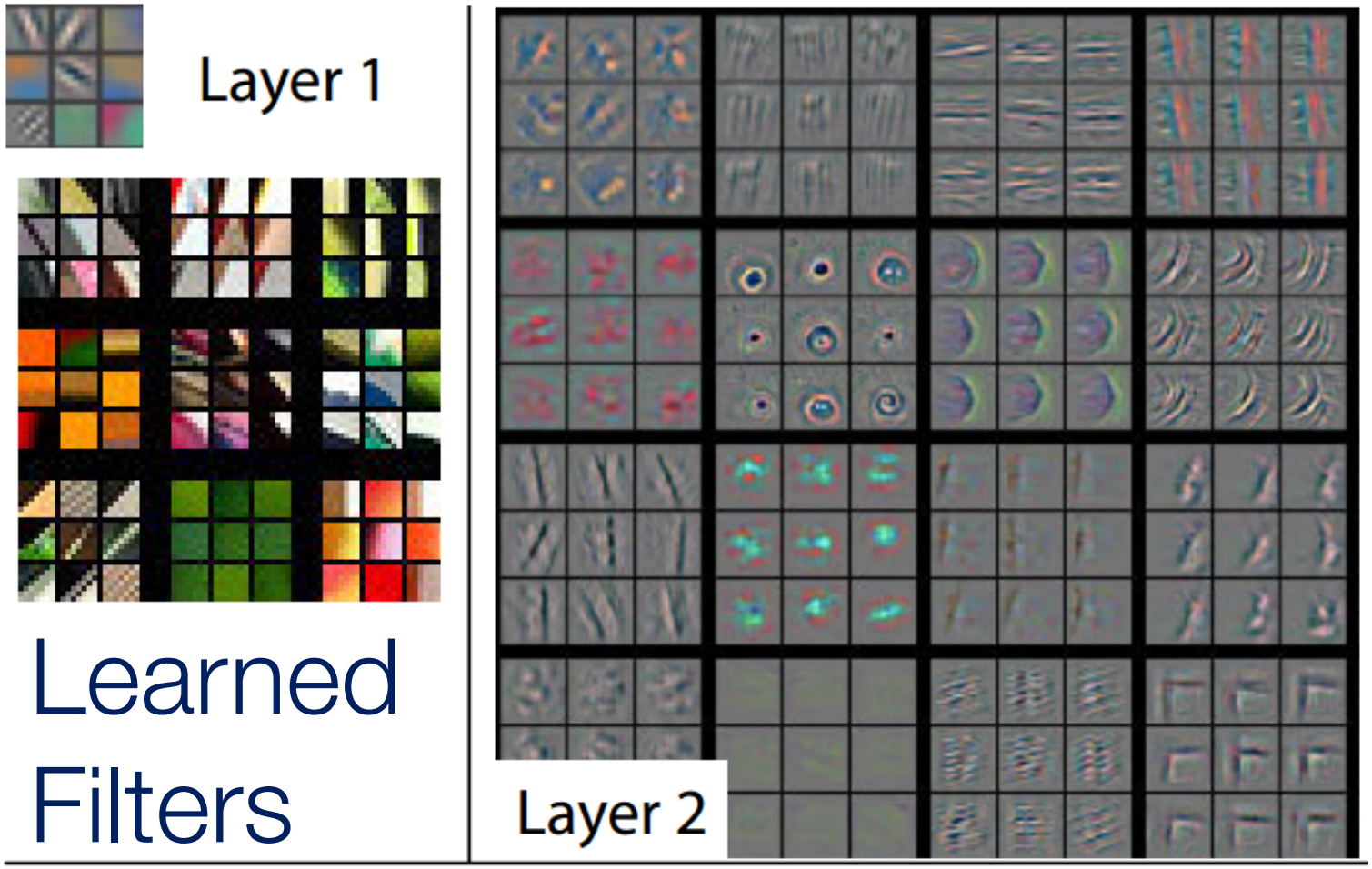


http://deeplearning.net/tutorial/lenet.html

CNN for Image Classification

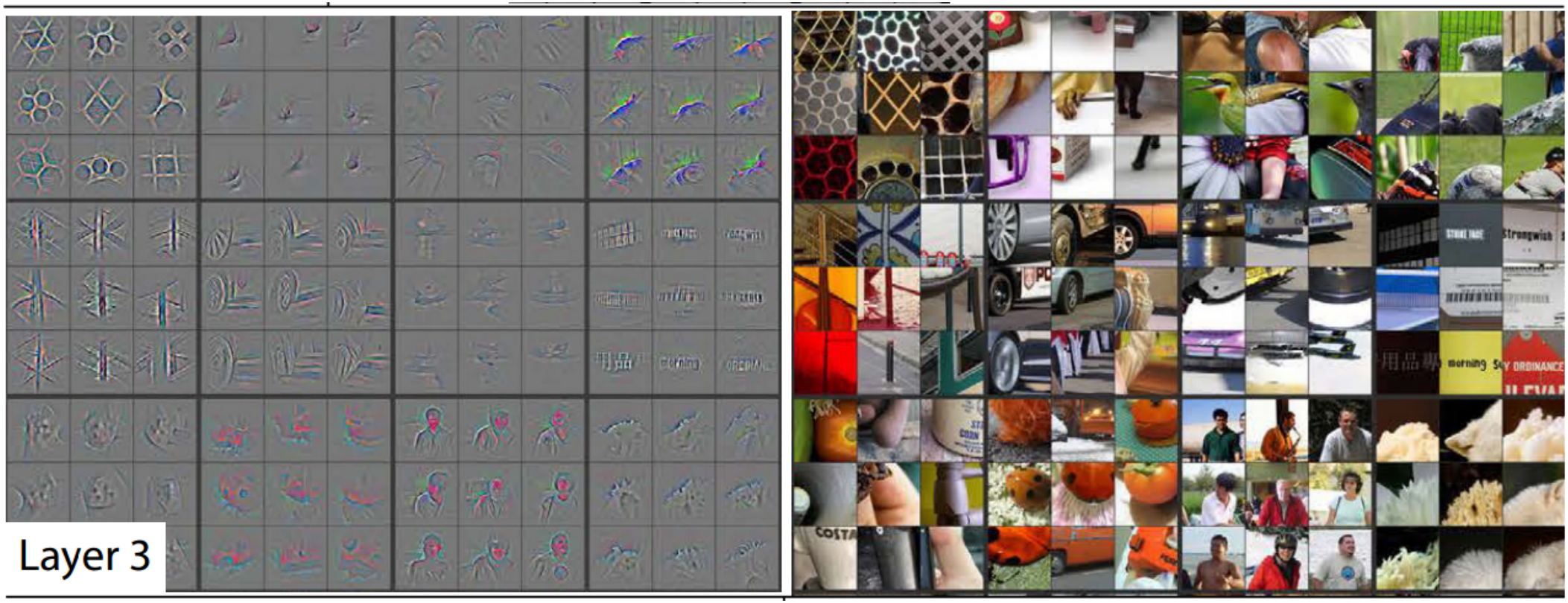
- How good are CNNs?
- “We also entered a variant of this model in the ILSVRC-2012 competition and achieved a winning top-5 test error rate of **15.3%**, compared to **26.2%** achieved by the second-best entry.”
  - Krizhevsky et al., 2012: ImageNet Classification with Deep Convolutional Neural Networks
  - Competition to classify photos from ImageNet, http://www.image-net.org/

CNN for Image Classification



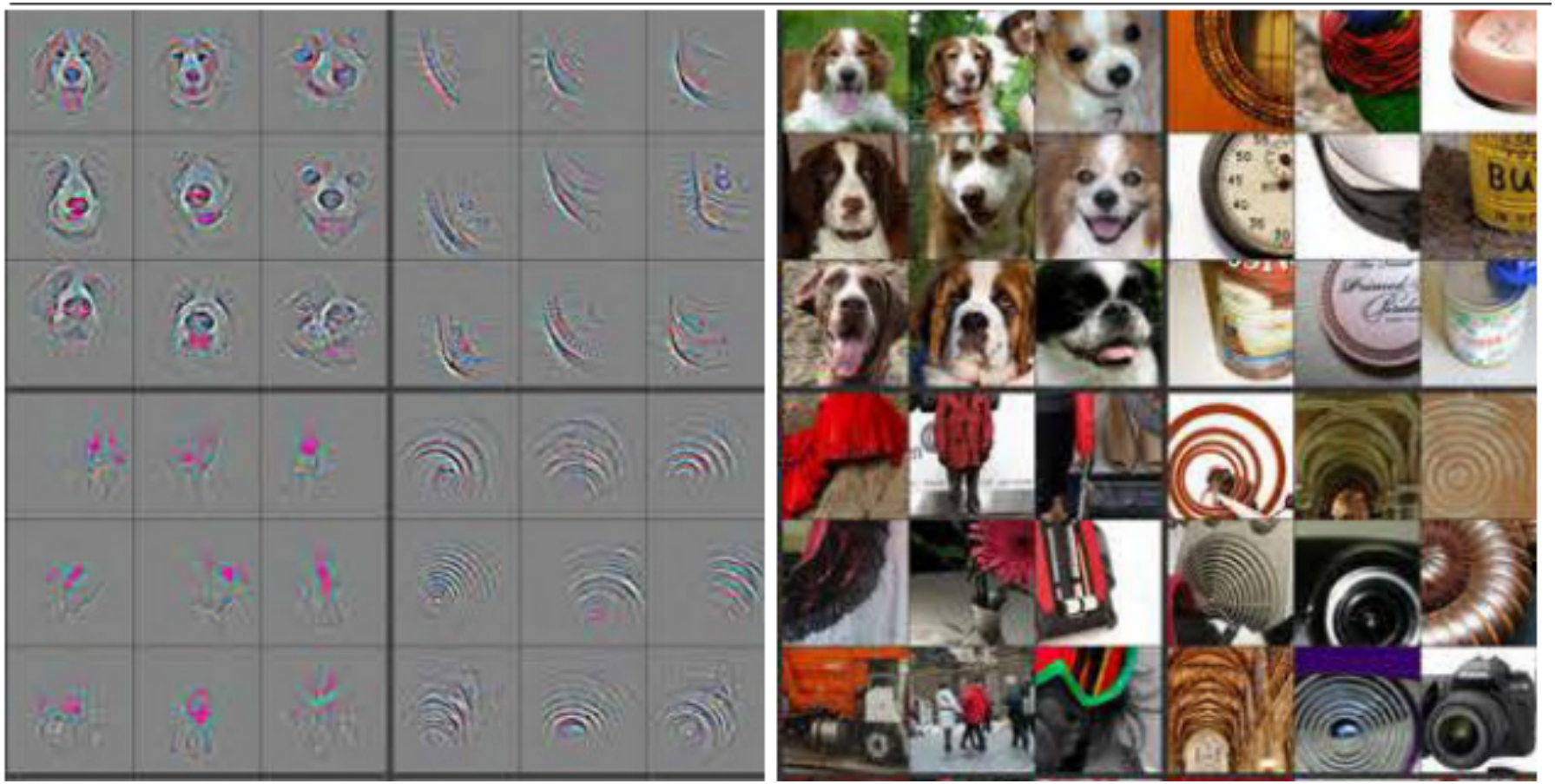
Zeiler and Fergus 2014

CNN for Image Classification



Zeiler and Fergus 2014

CNN for Image Classification



Layer 4

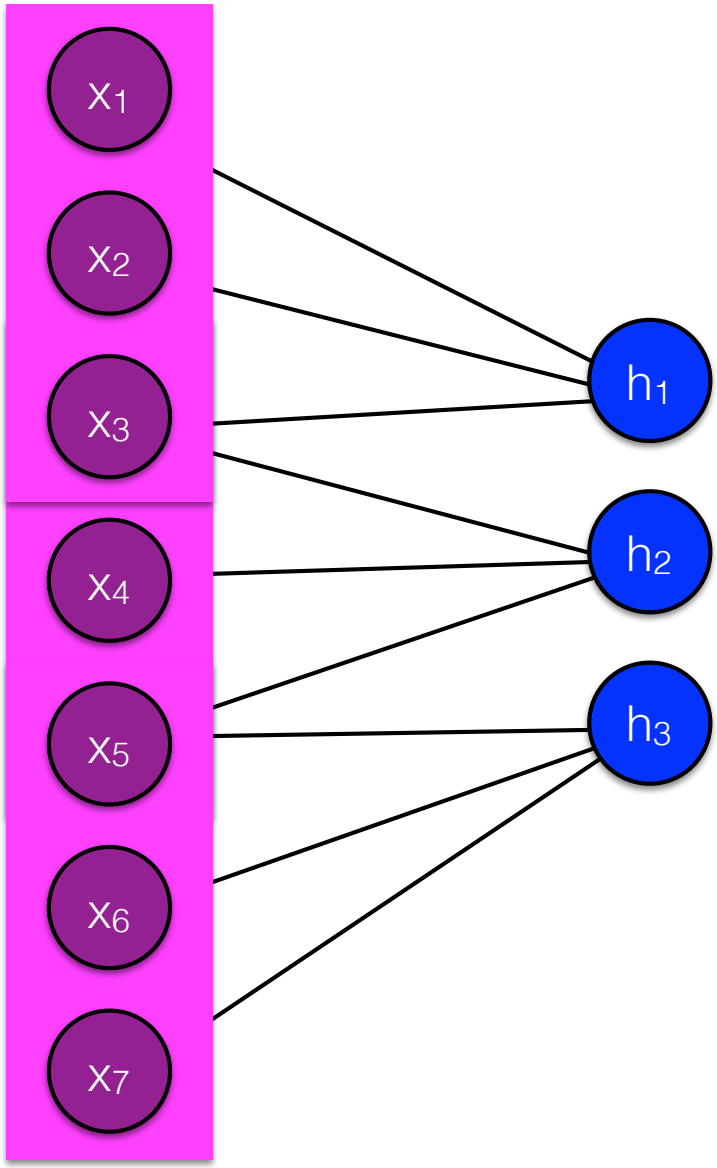
Zeiler and Fergus 2014

CNNs for NLP

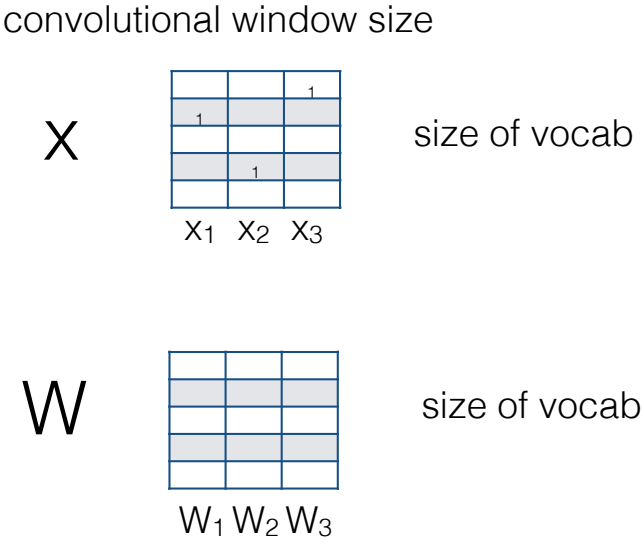
- Represent words with indicator vectors
- Every token is a V-dimensional vector (size of the vocab) with a single 1 identifying the word
- We'll get to distributed representations of words in on 9/19

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

Convolutional networks



convolutional window size

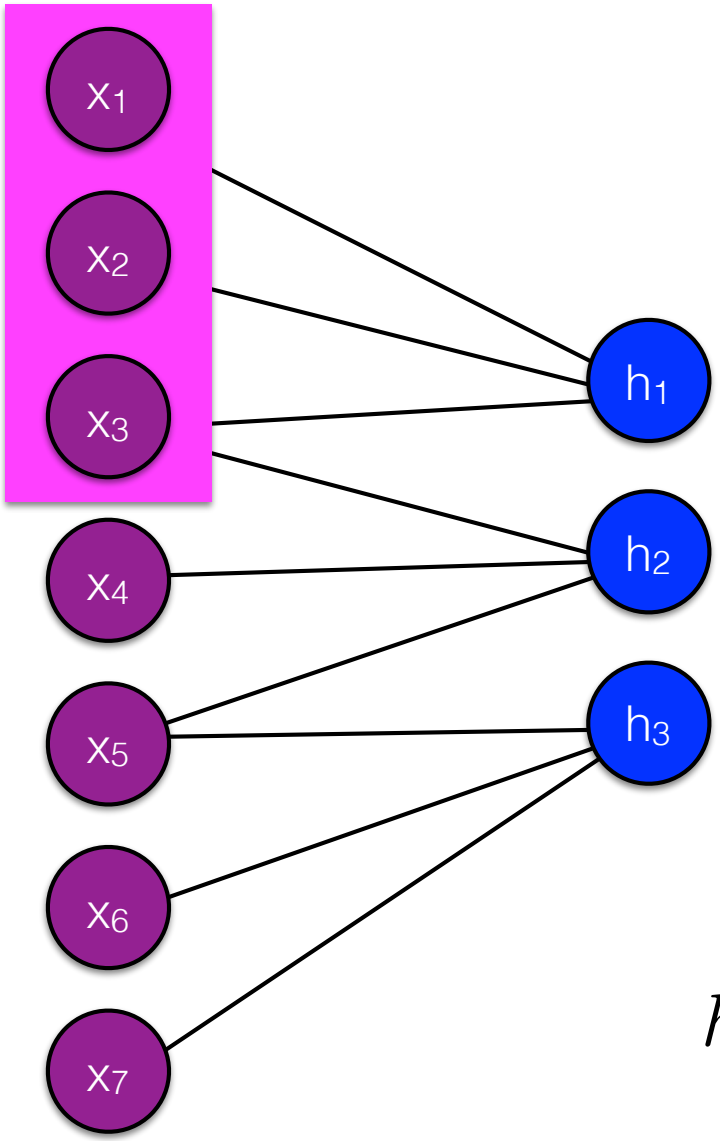


$$h_1 = \sigma(x_1 W_1 + x_2 W_2 + x_3 W_3)$$

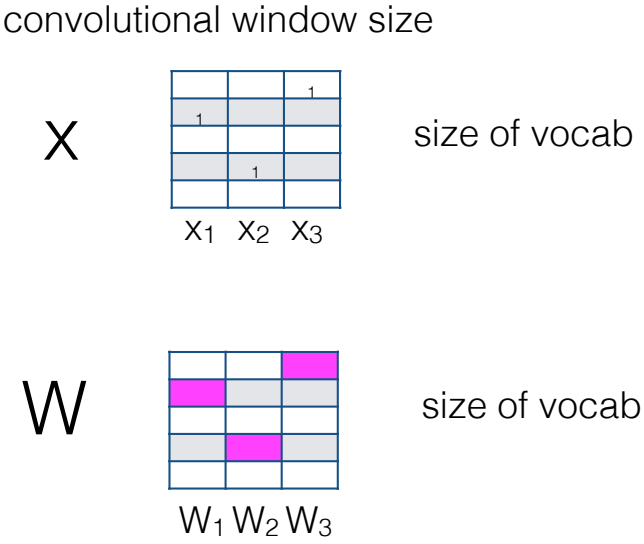
$$h_2 = \sigma(x_3 W_1 + x_4 W_2 + x_5 W_3)$$

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

Convolutional networks



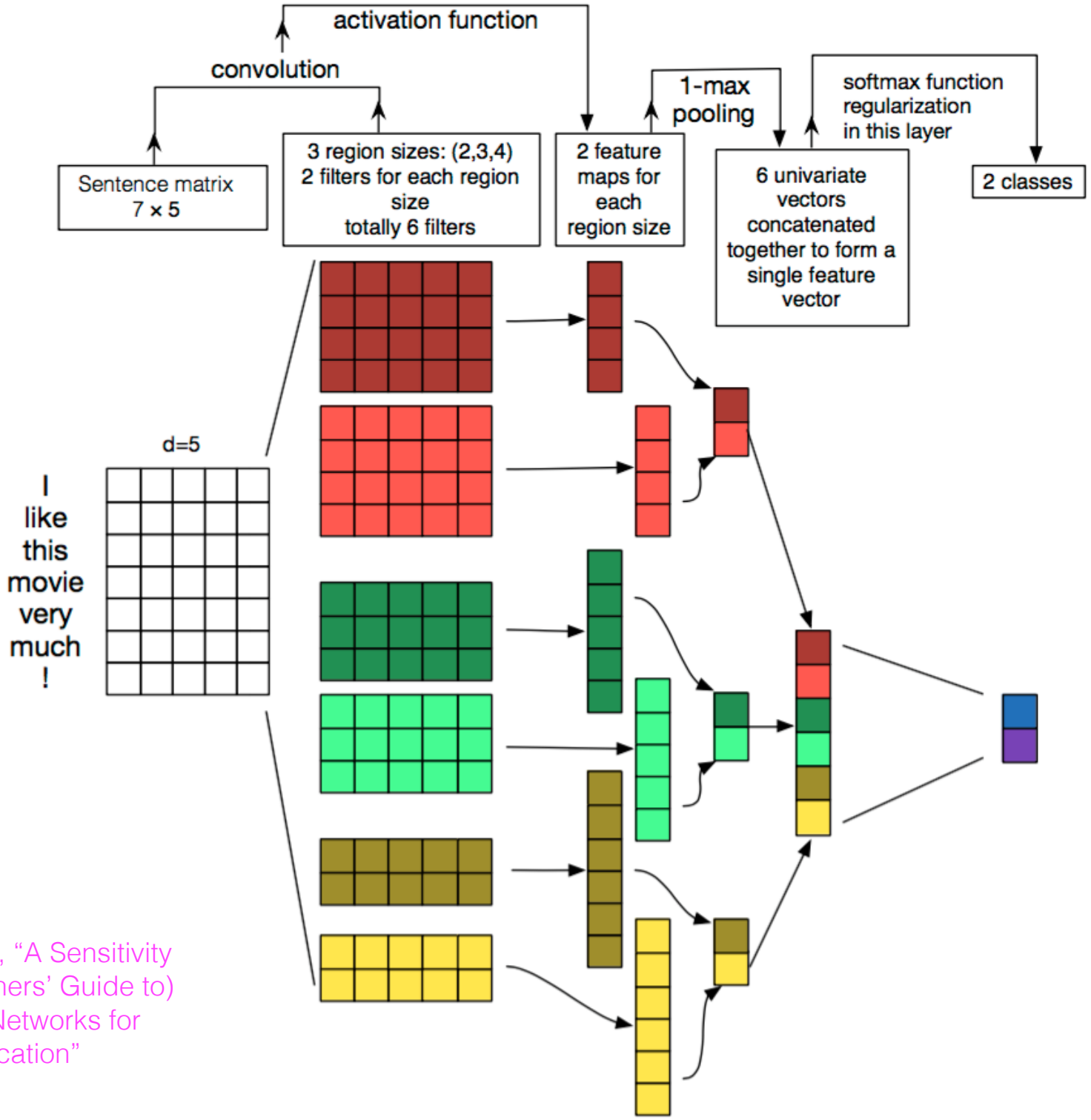
convolutional window size



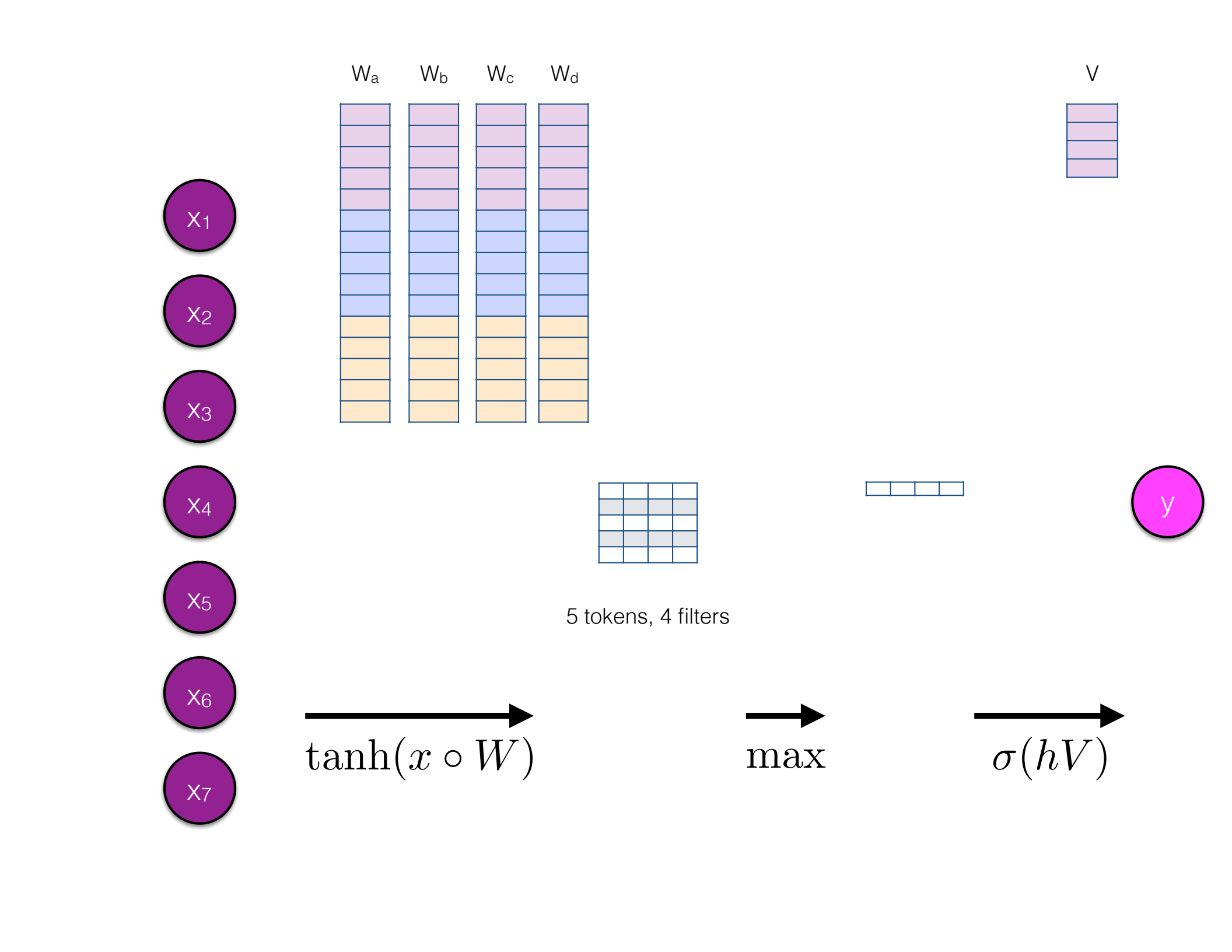
For indicator vectors, we're just adding these numbers together

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

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



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


W_a
W_b
W_c
W_d
V
x_1
x_2
x_3
x_4
x_5
x_6
x_7
y
5 tokens, 4 filters
$\tanh(x \circ W)$
max
$\sigma(hV)$

CNN as important ngram detector

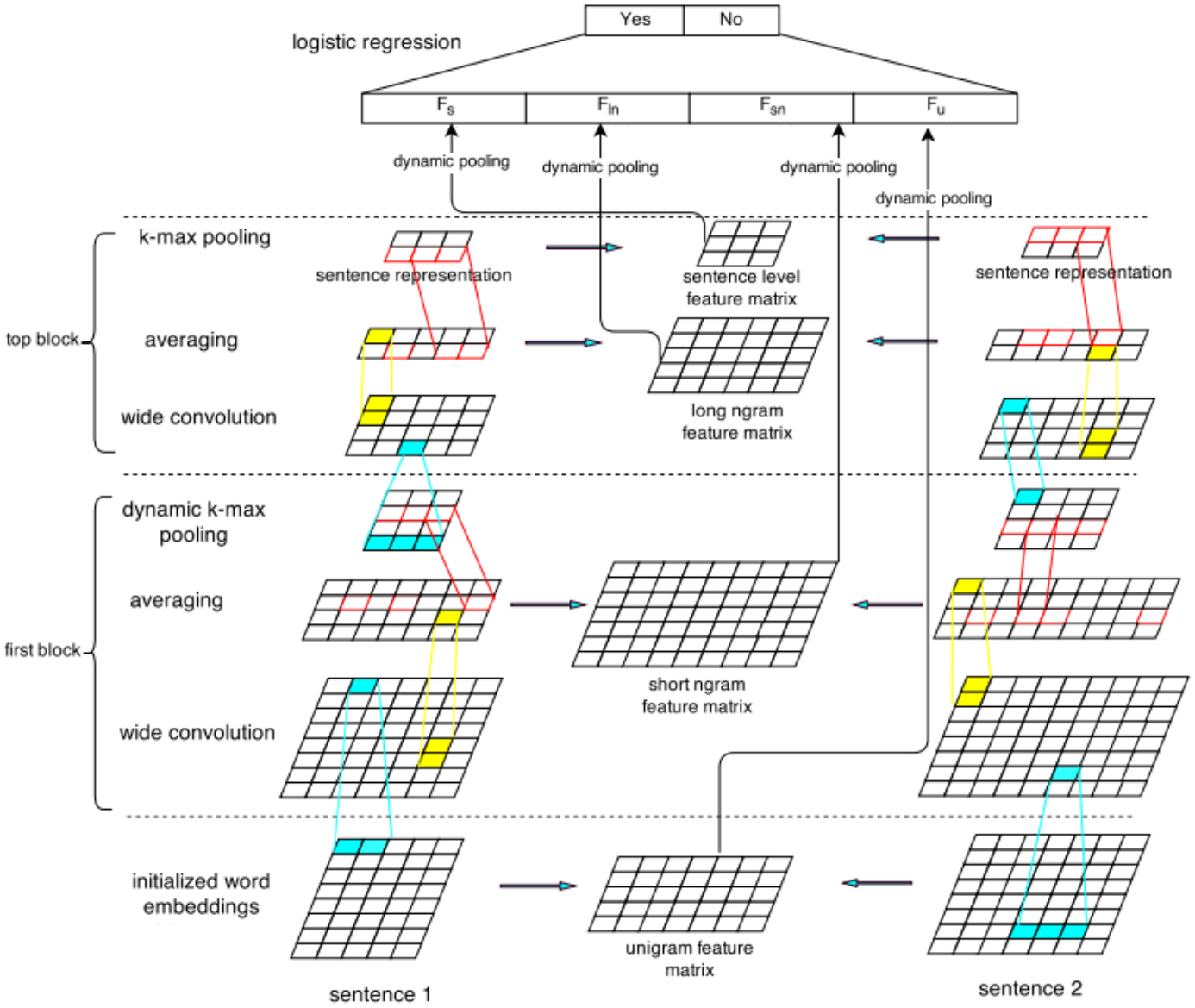
Higher-order ngrams are much more informative than just unigrams (e.g., “i don’t like this movie” [“I”, “don’t”, “like”, “this”, “movie”])

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

Example Application: Paraphrase Detection



(Yin & Schütze 2015)

So, What Can't They Do for NLP?

- Convolutional Neural Network for Paraphrase Identification (Yin & Schütze 2015)
- Summarization-based Video Caption via Deep Neural Networks (Li et al. 2015)
- Question Answering over Freebase with Multi-Column Convolutional Neural Networks (Dong et al. 2015)
- Convolutional Neural Network Architectures for Matching Natural Language Sentences (Hu et al. 2015)
- Learning Semantic Representations Using Convolutional Neural Networks for Web Search (Sheng et al. 2015)
- Deep Convolutional Neural Networks for Sentiment Analysis of Short Texts (dos Santos & Gatti 2014)
- Relation Extraction: Perspective from Convolutional Neural Networks (Nguyen & Grishman 2015)
- Modeling Mention, Context and Entity with Neural Networks for Entity Disambiguation (Sun et al. 2015)
- Modeling Interestingness with Deep Neural Networks (Gao 2015)

… and so on

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

$$P(y \mid x)$$

Summary

- Discriminative classifiers directly model the conditional, p(y|x)
- Logistic regression is a simple linear classifier, that retains a probabilistic semantics
- Parameters in LR are learned by iterative optimization (e.g. SGD)
- Regularization helps to avoid overfitting

Administrative Matters

- Paper Readings will go out tonight. Reading responses will start this week
- Model Exam Questions will also go out this week
- Reminder: Regex Homework is due next Weds!

Homework 1 is out!

- You'll implement Naive Bayes!
- You'll implement Logistic Regression!
- You'll finally put an end to bad behavior online!