ANTONIS ANASTASOPOULOS
CS499 INTRODUCTION TO NLP

CLASSIFICATION

https://cs.gmu.edu/~antonis/course/cs499-spring21/

With adapted slides by David Mortensen and Alan Black

STRUCTURE OF THIS LECTURE

1. Classification
2. Naive Bayes Classifier
3. Features and Embeddings
4. Discussion

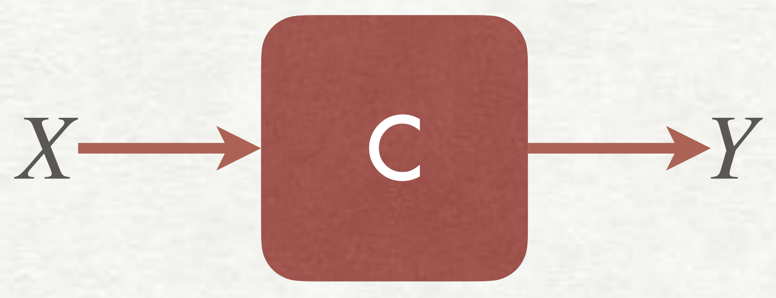
CLASSIFICATION

NOTATION

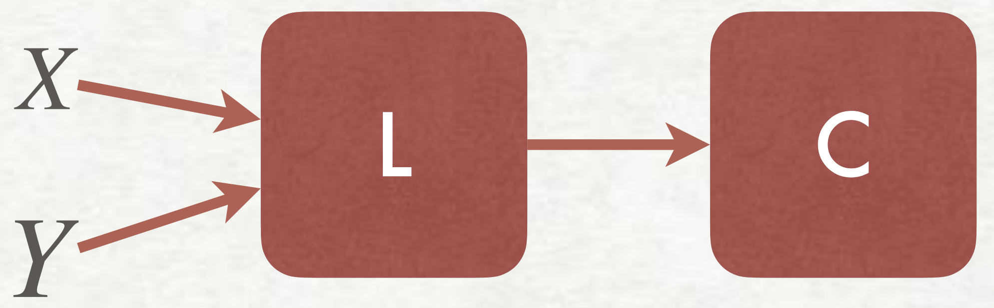
Training examples $\mathbf{x} = (x_1, x_2, \ldots, x_N)$

Their categories (labels) $\mathbf{y} = (y_1, y_2, \ldots, y_N)$

A classifier C seeks to map x_i to y_i



A learner L tries to infer C from $(\mathbf{x}, \mathbf{y})$



PROBABILISTIC CLASSIFIERS

X

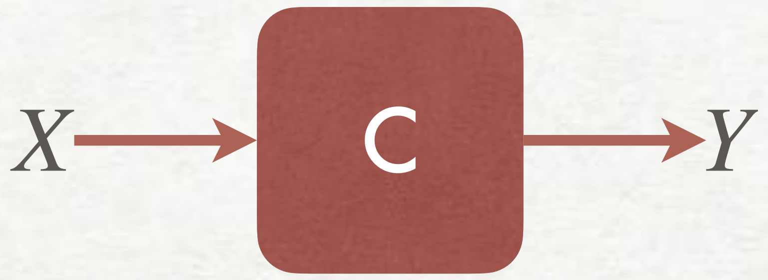
return

$$\arg\max_{y'} p(y' \mid x)$$

C

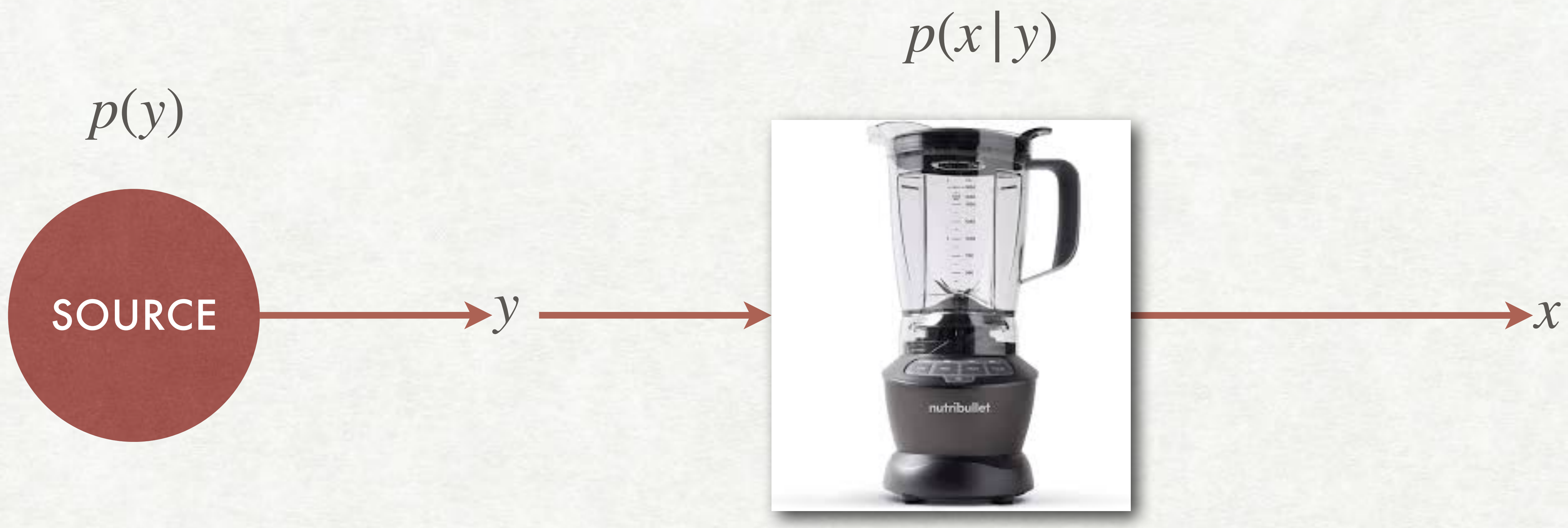
Y

GENERAL NOISY CHANNEL MODEL



GENERAL NOISY CHANNEL MODEL

A "story" of how the observed data came to be.



What portion of emails are expected to be spam vs. not spam?

What proportion of product reviews are expected to get 1,2,3,4,5 stars?

NOISY CHANNEL CLASSIFIERS

return

$$\arg\max_{y'} p(y') \times p(x \mid y')$$

$X \rightarrow$ C $\rightarrow Y$

REPRESENTATION

Representing labels is easy (e.g. can be easily mapped to integers)

Representing input text: features

Any object $x \in \mathcal{X}$ you might be given can be represented as a vector in a **vector space**
(as we already saw vectors for text are often sparse and high-dimensional)

Designing $\mathbf{\Phi}$ ("feature engineering")

What information do you need to solve the problem?
What information do you need to avoid mistakes?

NAÏVE BAYES CLASSIFIER

X

$\phi_j \leftarrow [\mathbf{\Phi}(x)]_j$

return $\arg\max_{y'} p(y') \times \Pi_j(\phi_j \mid y')$

Y

C

NAÏVE BAYES LEARNER

X

Y

$\phi_j \leftarrow [\boldsymbol{\Phi}(x)]_j$

return

$\arg\max_{y'} p(y') \times \Pi_j(\phi_j \mid y')$

C

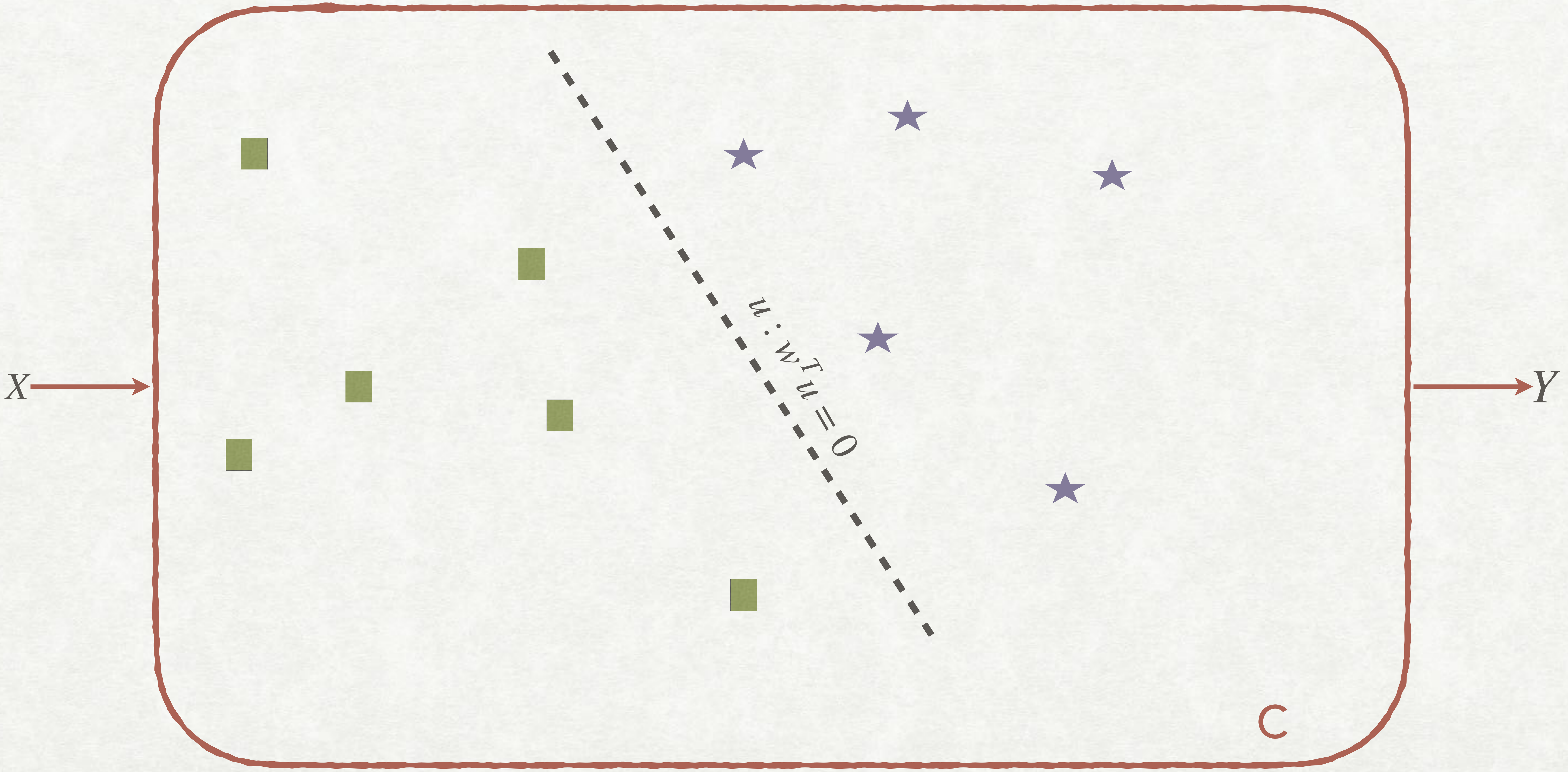
$$\forall y, p(y) \leftarrow \frac{\text{count}(y)}{N}$$

$$\forall y, \forall j, \forall f, \quad p(\phi_j(x) = f \mid y) \leftarrow \frac{\text{count}(f, y)}{\text{count}(y)}$$

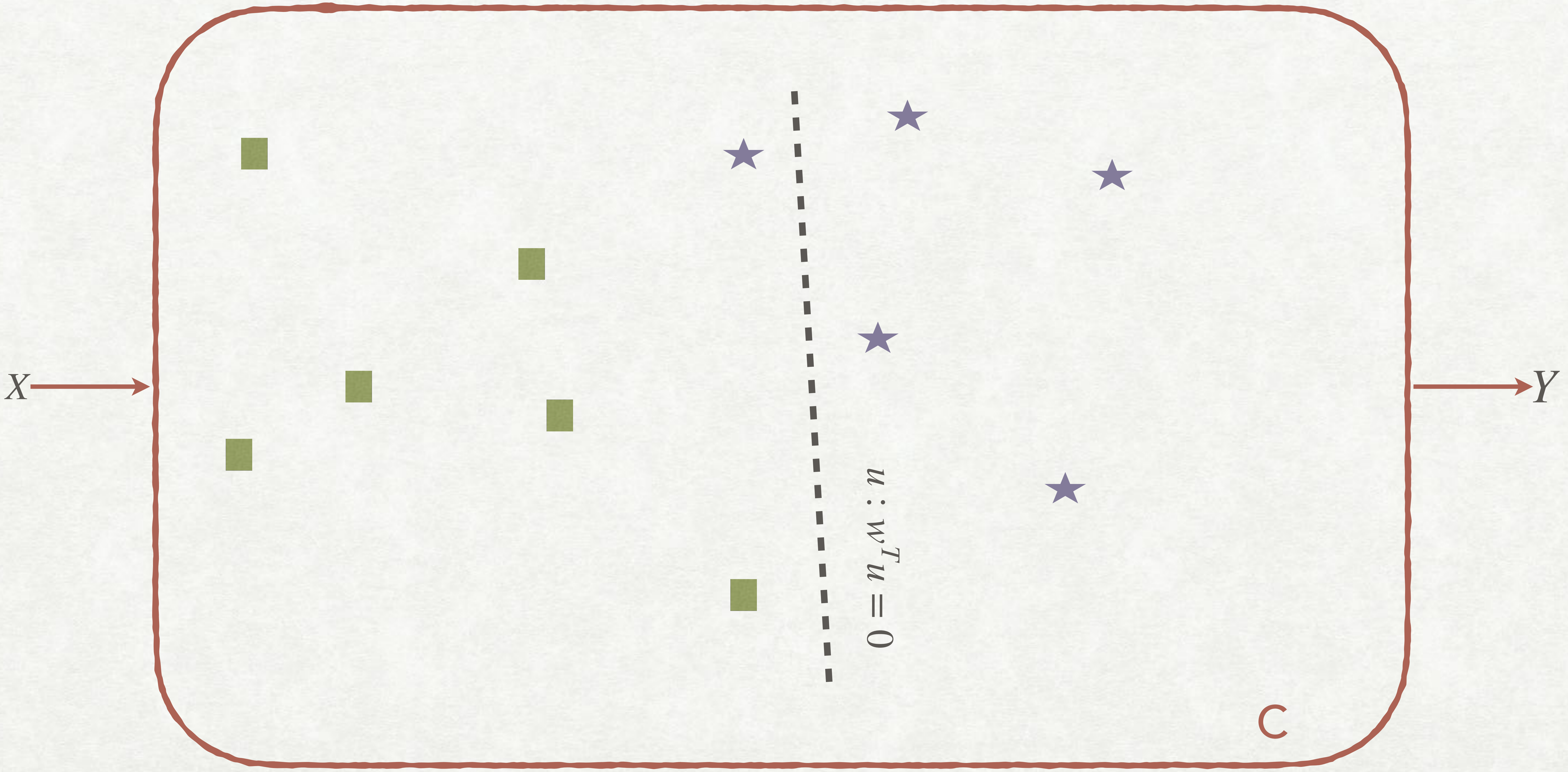
L

$p(\cdot)$

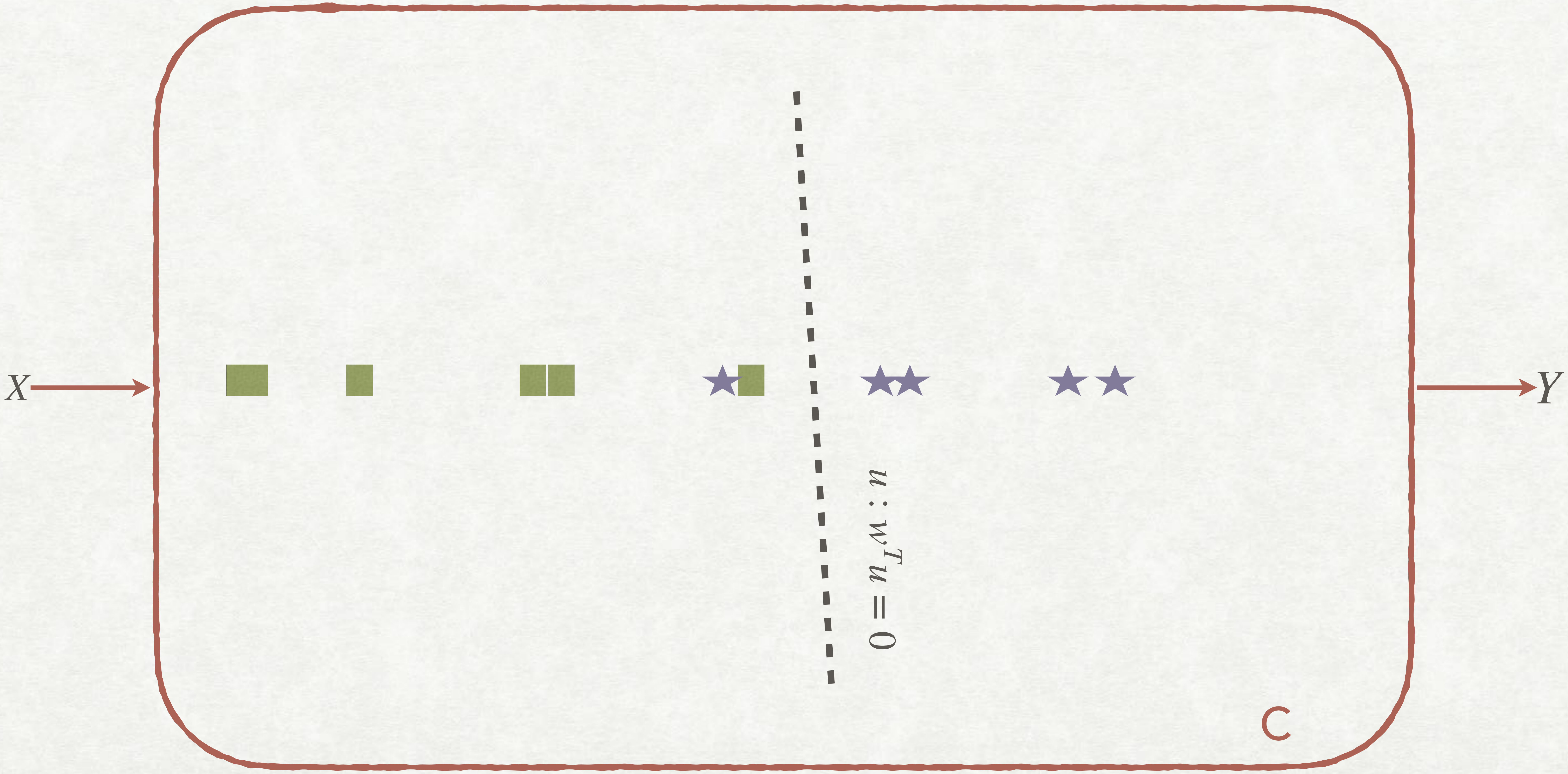
LINEAR CLASSIFIER



LINEAR CLASSIFIER



LINEAR CLASSIFIER



LINEAR CLASSIFIER (>2 CLASSES)

return

$$\arg\max_{y'} \mathbf{w}^T \mathbf{\Phi}(x, y)$$

X

Y

C

PERCEPTRON LEARNER

$\mathbf{w} \leftarrow 0$

for $t = 1 \ldots T$:

  select (x_t, y_t)

  # run current classifier

  $y \leftarrow \arg\max_{y'} \mathbf{w}^T \mathbf{\Phi}(x, y')$

  if y!=y_t then *#mistake*

    $\mathbf{w} \leftarrow \mathbf{w} + \alpha[\mathbf{\Phi}(x_t, y_t) - \mathbf{\Phi}(x_y, y)]$

return $\mathbf{w}$

return

$\arg\max_{y'} \mathbf{w}^T \mathbf{\Phi}(x, y)$

C

L

X

Y

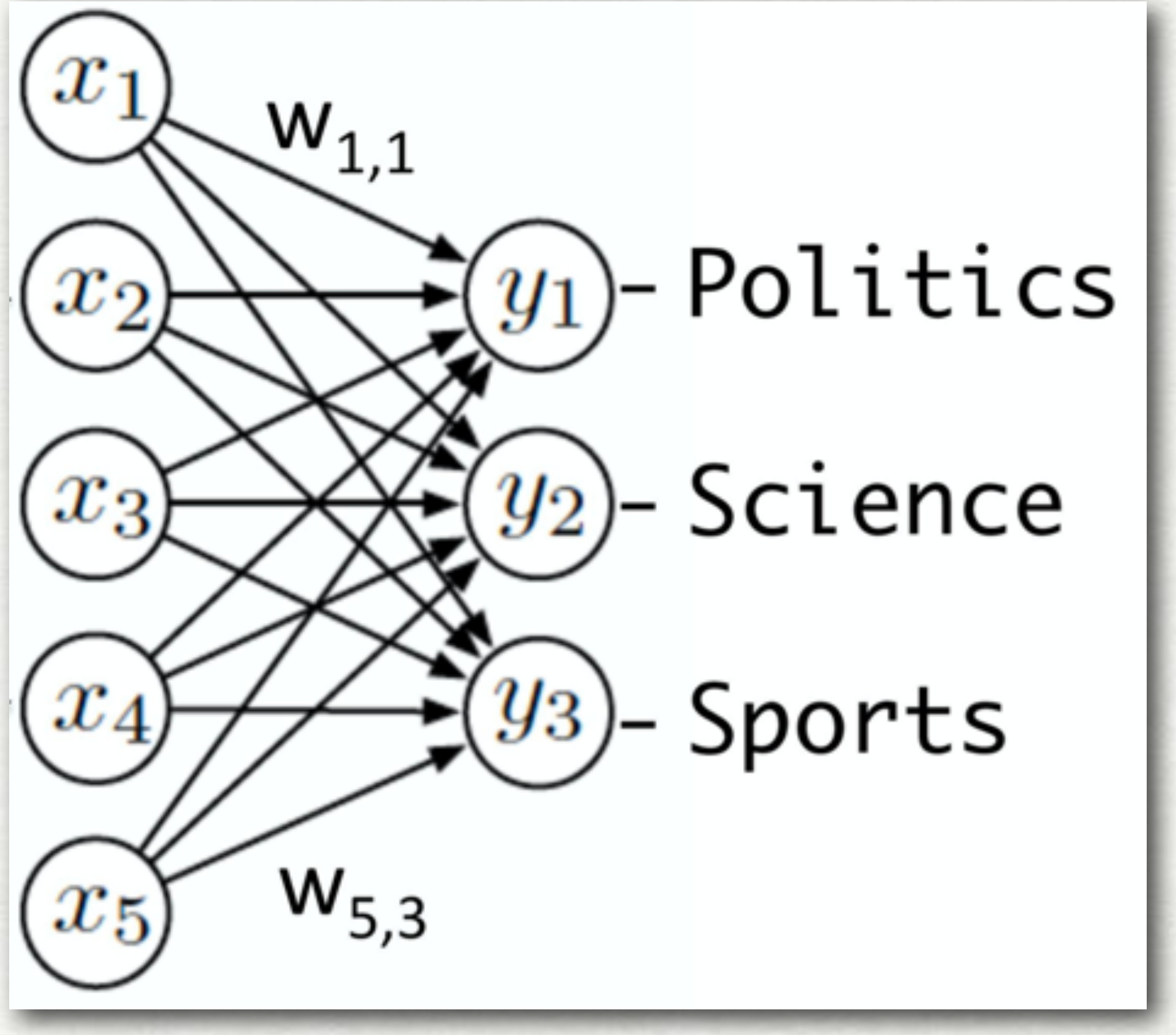
$\mathbf{w}$

NEURAL NETWORK V1.0: LINEAR MODEL

Linear models: $f(l, d) = w \cdot g(l, d) = w(l) \cdot x(d)$

e.g. $y_1 = x_1 w_{1,1} + x_2 w_{2,1} + x_3 w_{3,1} + x_4 w_{4,1} + x_5 w_{5,1} = w(1) \cdot x(d)$

Number of times *Lost* appears in a document

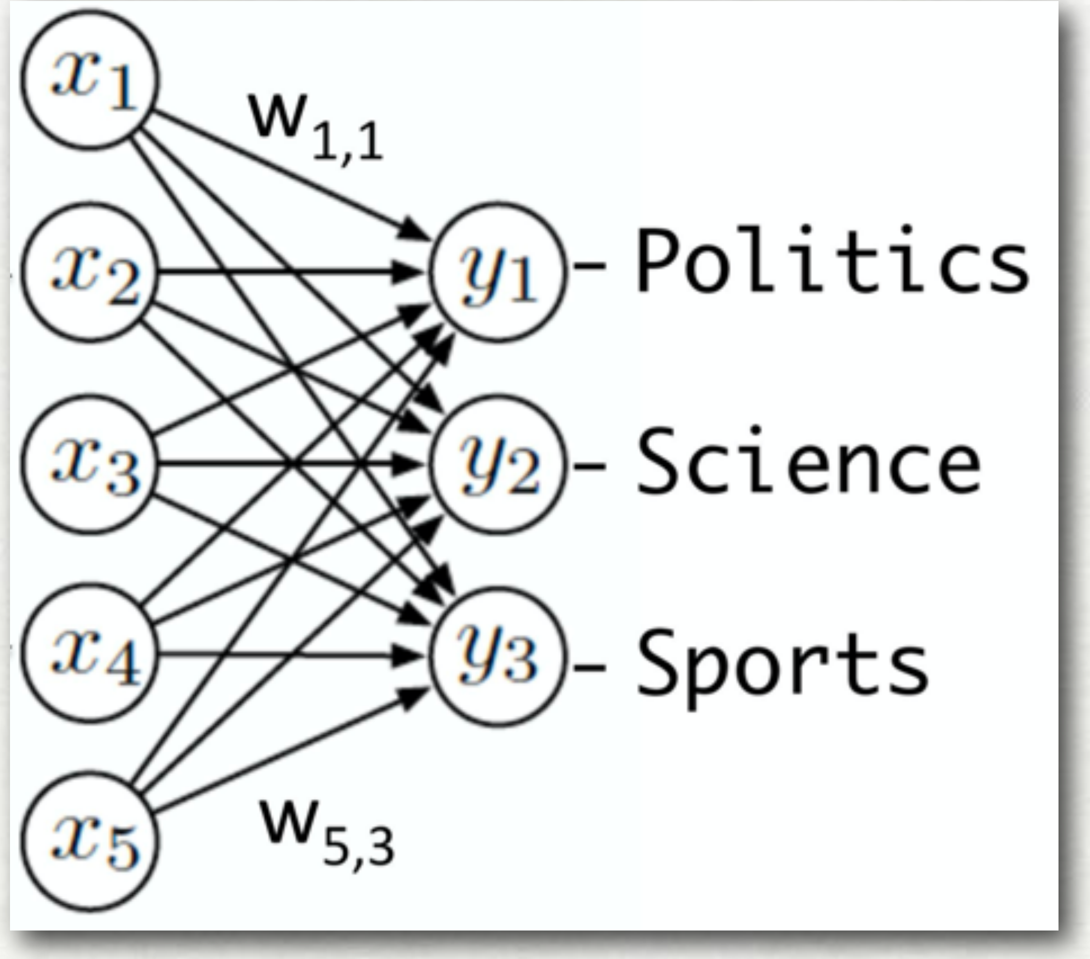


Number of times *Barcelona* appears in a document

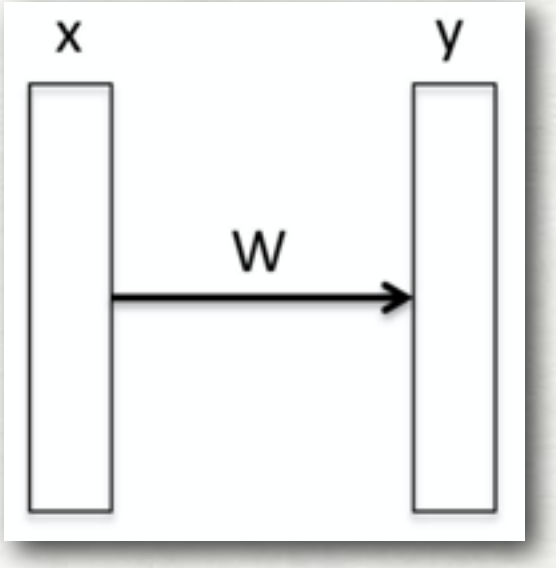
NEURAL NETWORK V1.0: LINEAR MODEL

Linear models: $f(l, d) = w \cdot g(l, d) = w(l) \cdot x(d)$

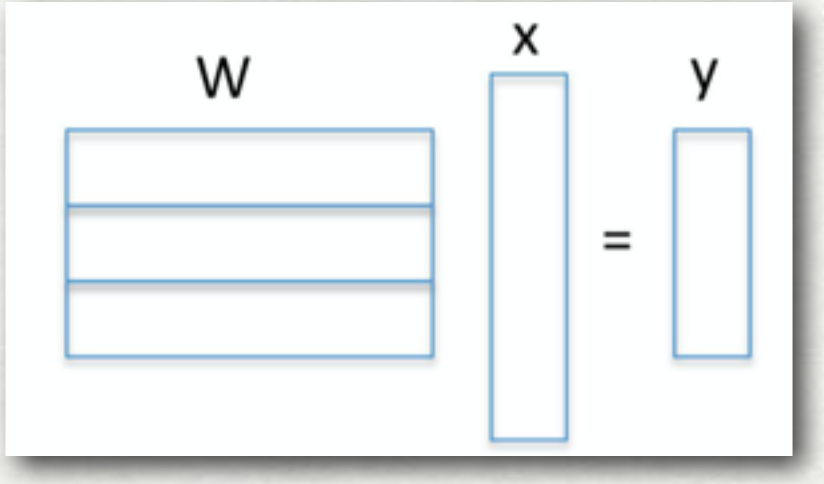
e.g. $y_1 = x_1 w_{1,1} + x_2 w_{2,1} + x_3 w_{3,1} + x_4 w_{4,1} + x_5 w_{5,1} = w(1) \cdot x(d)$



same as:

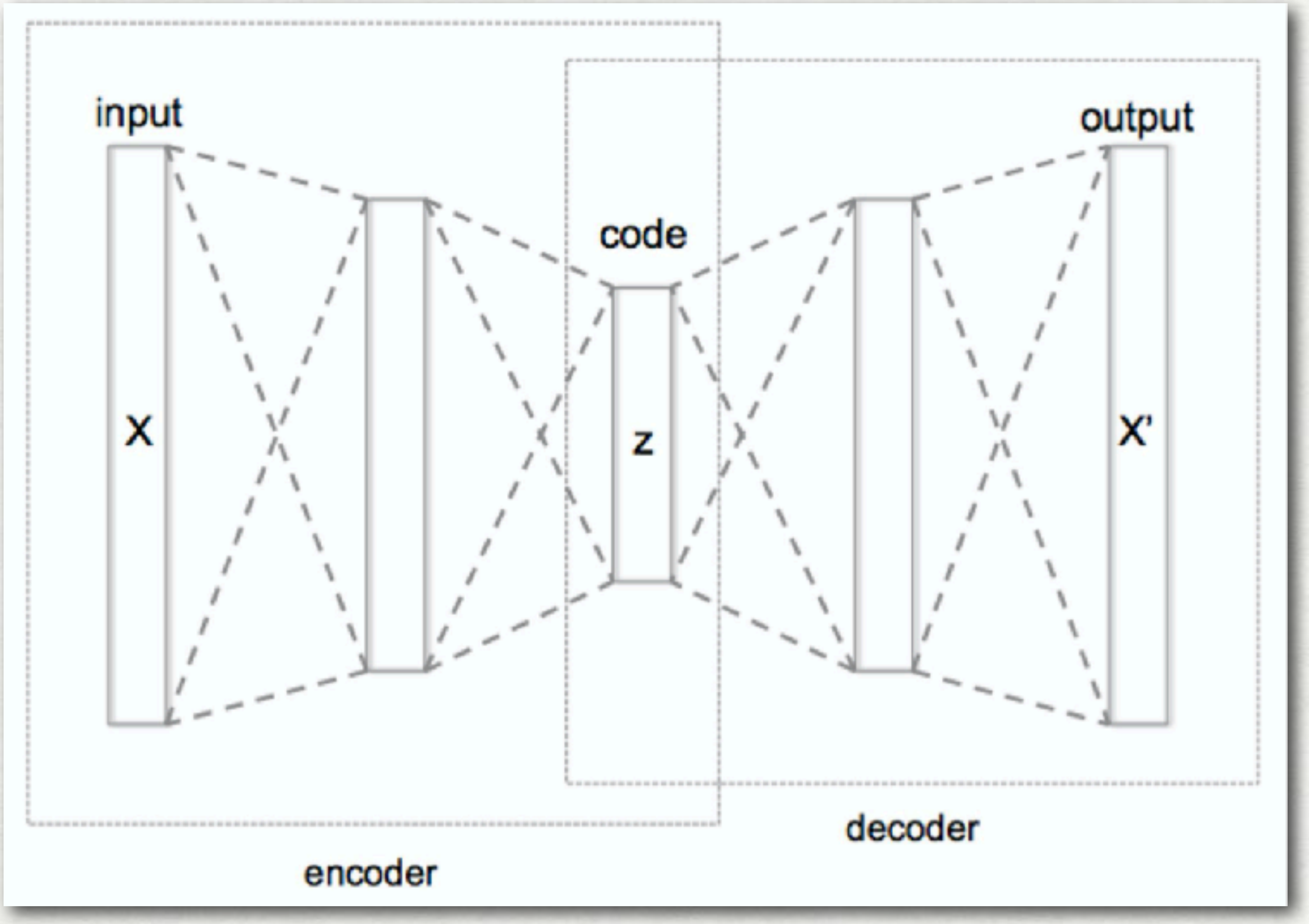


Still, similar words do not share parameters



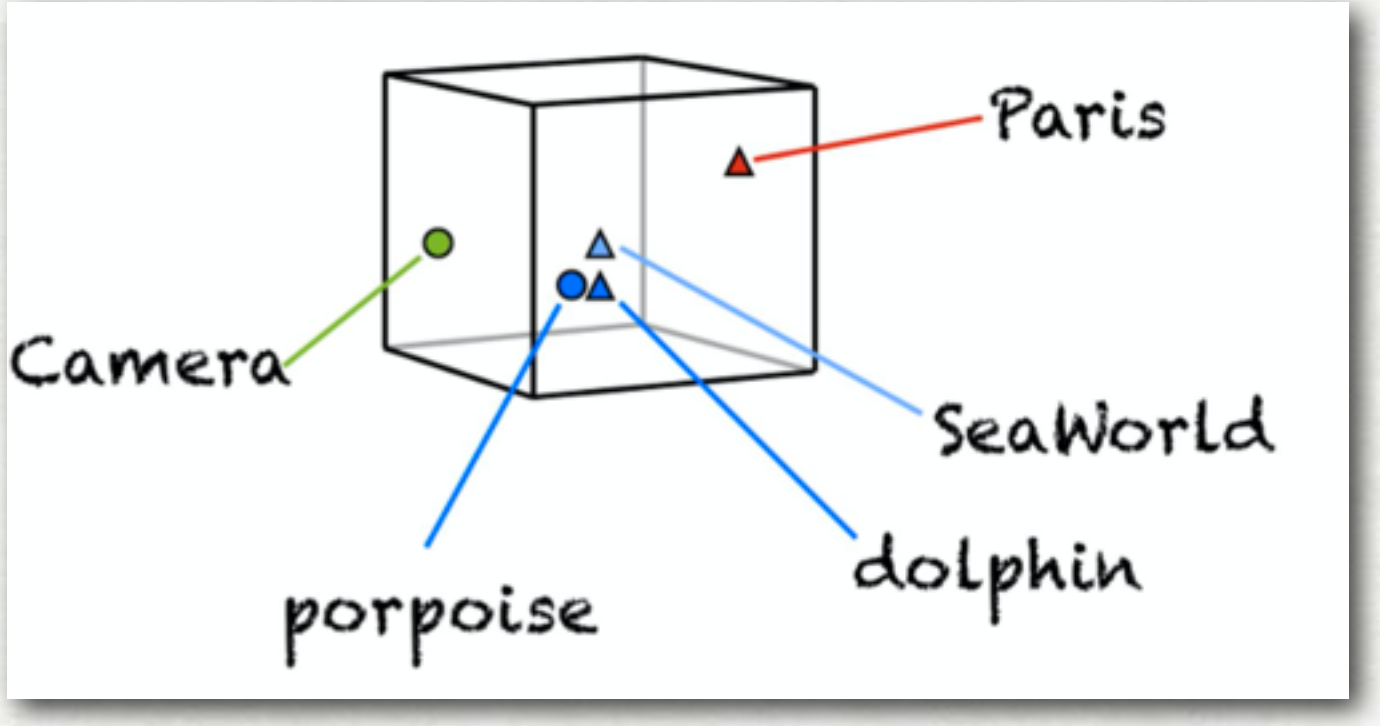
NEURAL NETWORK V2.0: REPRESENTATION LEARNING

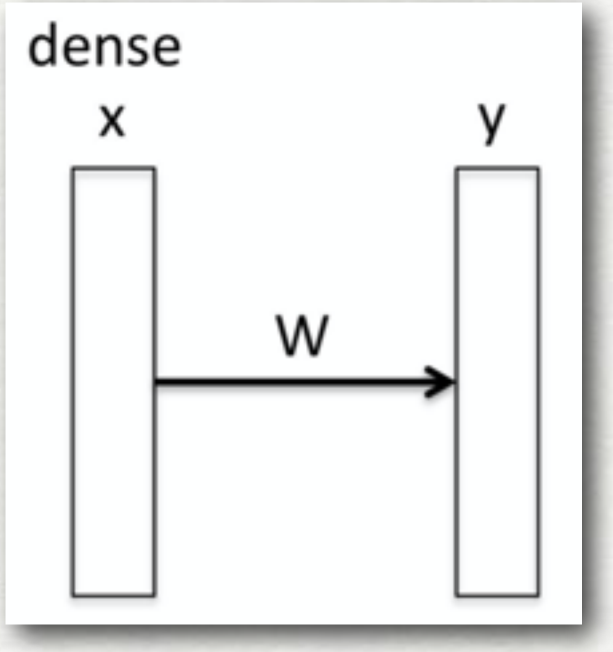
Big idea: induce low-dimensional dense feature representations of high-dimensional objects



NEURAL NETWORK V2.1: REPRESENTATION LEARNING

Big idea: embed words in a **dense** vector space and use the word embeddings as dense features

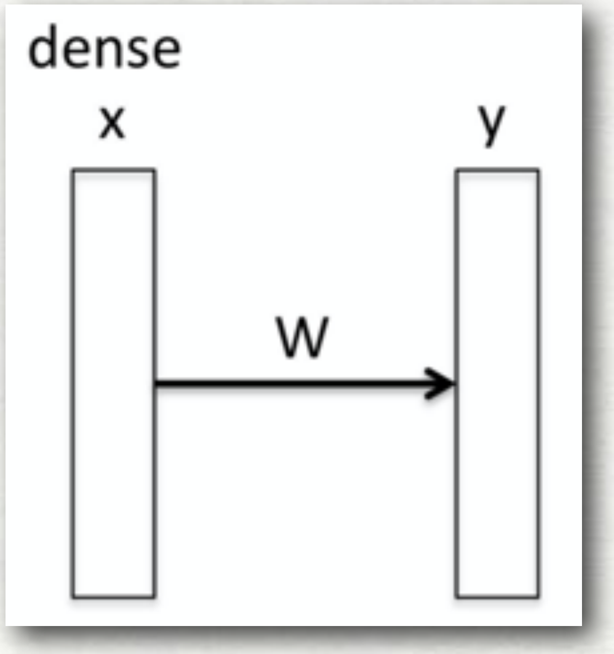


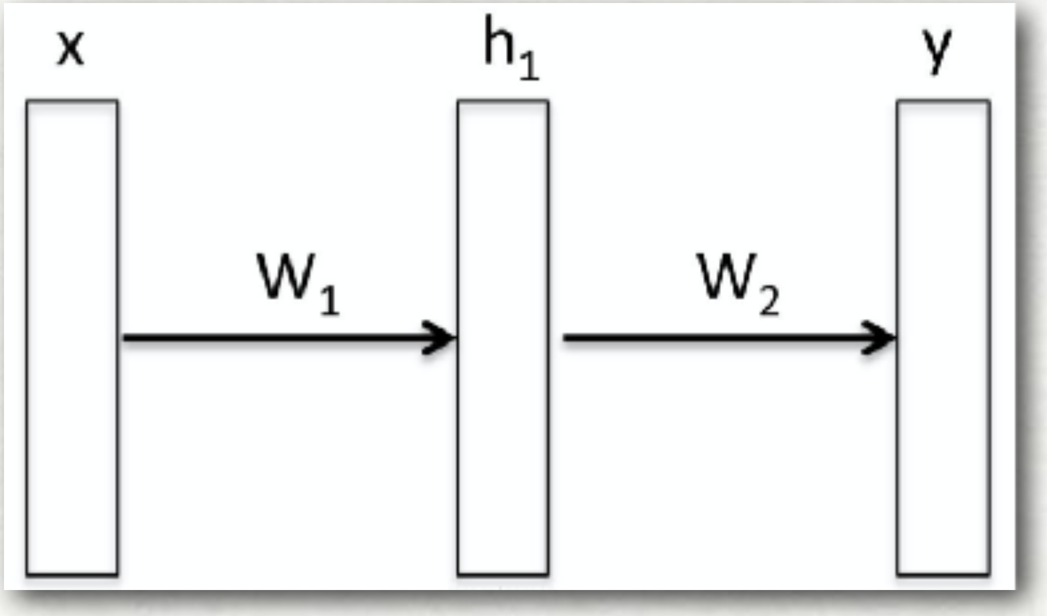


Does this really solve the problem?

NEURAL NETWORK V3.0: COMPLEX FUNCTIONS

Big idea: define more complex functions by adding a hidden layer





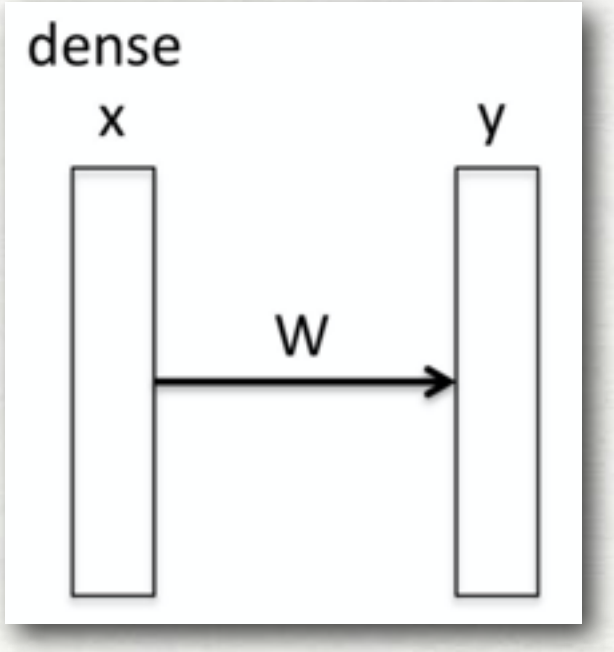
$$y = Wx$$

$$y = W_2 h_1 = W_2(W_1 x) = Wx$$

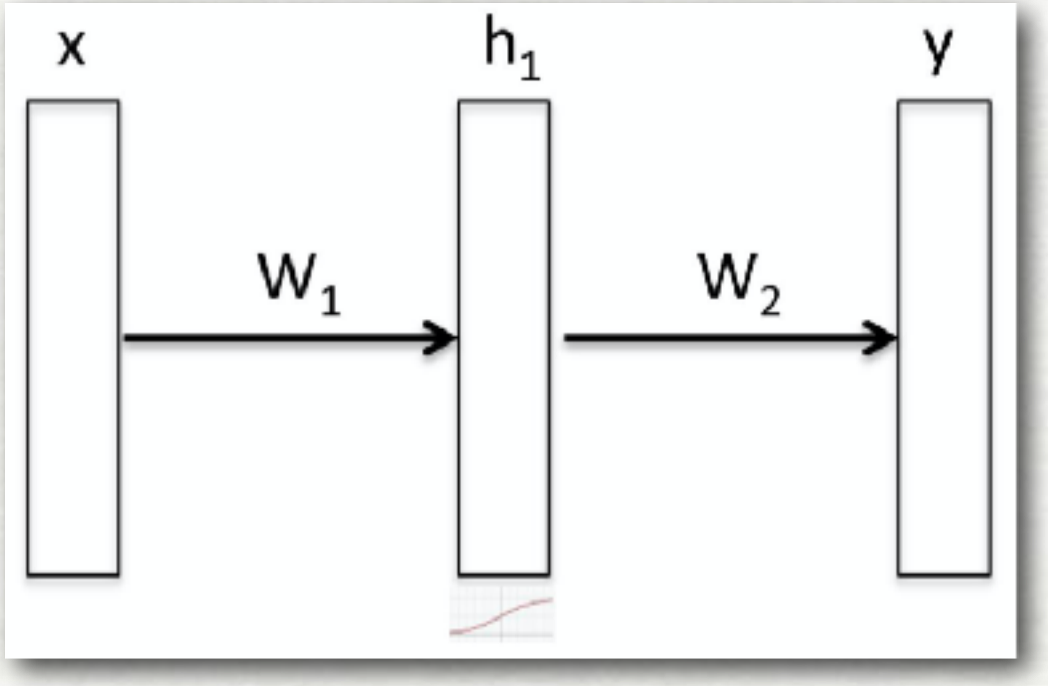
?!?!?

NEURAL NETWORK V3.0: COMPLEX FUNCTIONS

Big idea: define more complex functions by adding a hidden layer



$$y = Wx$$

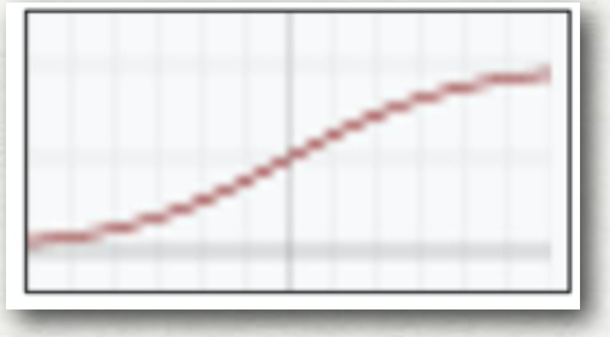


$$y = W_2 h_1 = W_2 \boxed{a_1(W_1 x)}$$

Induced features

Non-linear functions,
e.g. logistic function

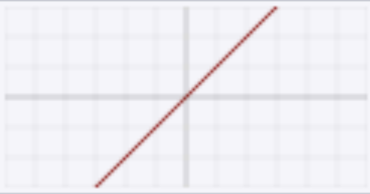
$$a_1(z) = \frac{1}{1 + e^{-z}}$$



Universal approximation theorem
Cybenko., G. (1989)

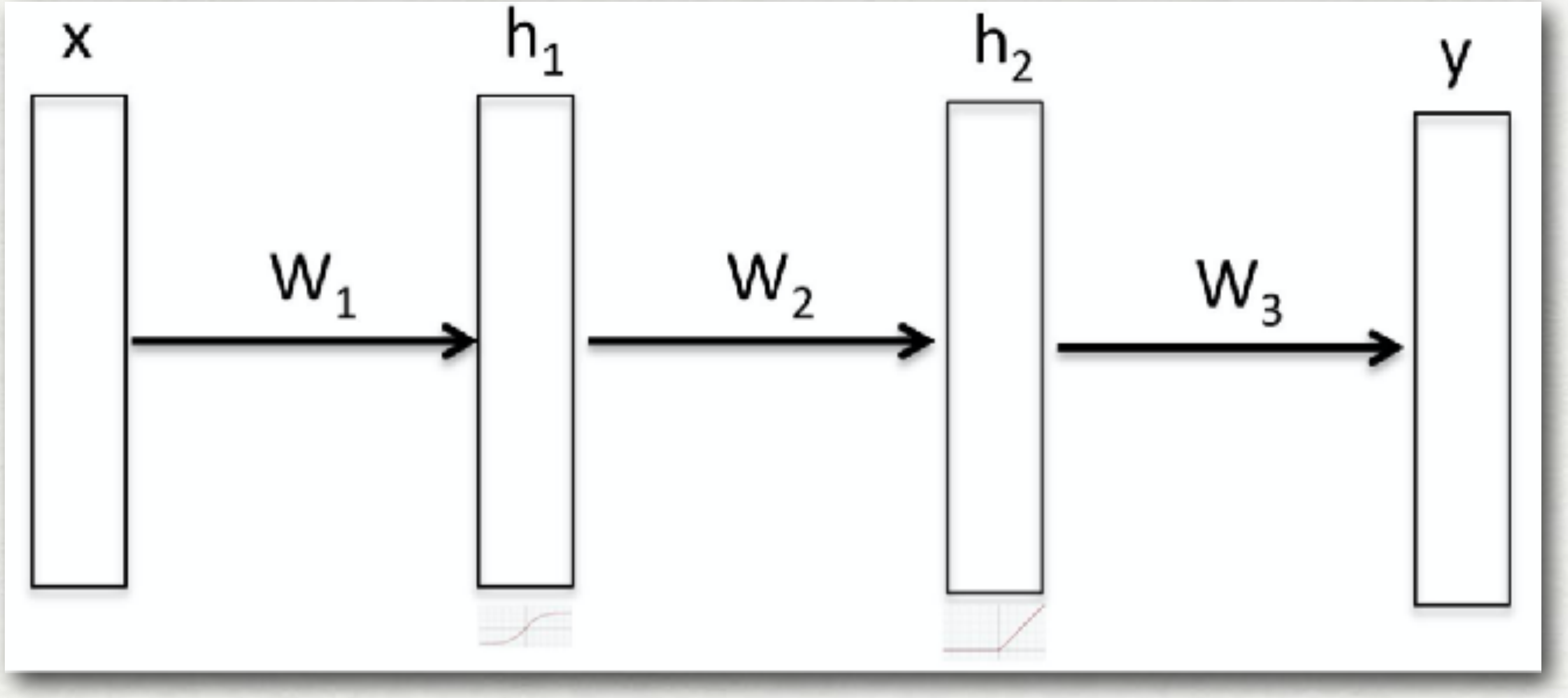
NEURAL NETWORK V3.0: COMPLEX FUNCTIONS

Popular activation/transfer/non-linear functions:

Name	Plot	Function, $f(x)$	Derivative of f, $f'(x)$	Range
Identity		x	1	$(-\infty, \infty)$
Binary step		$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	$\begin{cases} 0 & \text{if } x \neq 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	$\{0, 1\}$
Logistic, sigmoid, or soft step		$\sigma(x) = \frac{1}{1+e^{-x}}$ [1]	$f(x)(1-f(x))$	$(0, 1)$
tanh		$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$1 - f(x)^2$	$(-1, 1)$
Rectified linear unit (ReLU)[11]		$\begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases} = \max\{0, x\} = x\mathbf{1}_{x>0}$	$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	$[0, \infty)$

https://en.wikipedia.org/wiki/Activation_function

NEURAL NETWORK V3.5: DEEPER NETWORKS



$$y = W_3 h_2 = W_3 \ a_2(W_2 \ a_1(W_1 x))$$

Wait — why do we need more layers?

FEATURES AND EMBEDDINGS

SAMPLE REPRESENTATION

List of features → Category

Category: “small” finite discrete # of classes

- e.g. languageID, POS tag, Movie genre,

Features: list of real numbers

- All samples must have the same # of features

HOW TO REPRESENT WORDS

Samples are movie reviews:
- A few sentences of text
- a class: 1-5 (1=very bad, 5=very good)

Class: simple int

Features: ???
- encode the first n words (?)

  # of words
  # of sentences
  # of exclamation points!!!!
  Does “good” appear?
  Does “bad” appear?
  …

Representing Classes

Categories to numbers:

Business	[1,0,0]
Sports	[0,1,0]
Entertainment	[0,0,1]

(“One hot” representations)

Usually better than:
- Business → 1
- Sports → 2
- Entertainment → 3

HOW TO REPRESENT WORDS

Decide on vocabulary size + _other_

- Occurrence of word
- Array of vocal size: set to 1 if word appears
  (or set to # of occurrences of word)
- Vocab should be most frequent/relevant words in corpus
  - should we include very high frequency words?
  - only content words?
  - only words appearing more than once?

HOW TO REPRESENT WORDS

One big vector for whole movie review

- Lots of zeros and few ones
- Might be 1000 or 10000 wide (or more)

Often called “bag of words” representation

- not care about word order
- not case about # of occurrences of word
- same length vector independent of length of review

Bag of Words

Reviews are “similar” if vectors are similar

- similar means similar word distribution
- e.g. simple difference, edit diff, cosine similarity, …

BUT:

- Is “I love the film” equally different from
- “I hate the film” or
- “I like the film”?

Word similarity (“love” vs “hate” vs “like”)

- cannot just be a binary representation

Contextual effects (“good” vs “not good”)

- need longer context
- could add bi-gram features to vectors

WORD DIFFERENCES

"like" and "love" more similar than "like" and "hate"

Sparse vectors treat distance as the same

Word Embeddings:

- Dense (not sparse) representations
- Distance metrics are more "meaningful"
- Do dimension in word embeddings mean something?
  (maybe, maybe not)

CHOOSING WORD EMBEDDINGS

Use existing pre-trained library (word2vec, GloVe, ELMo, BERT, …)

Train your own word2vec or skip-gram on your data

Things to consider:

- are your data like others?
- do you have enough training examples?
- are there special meanings in your domain?

How long should the dense vector be? 300? 768? 1000? Floats

- We don't really know
- It's not the size of the space represented → It's *if* the dimensions found are useful

Hard to implicitly control meaning in vectors

- Easy to explicitly do it, e.g. by concatenating word, POS tag, dependency parent, etc

Word32vec and GloVe were standard

"Everything is better with BERT"
[Devlin et al 2019]

Really: *"everything is better taking context into account"*
SOTA performance in several NLP tasks

Still better ones are being developed

SENTENCE/DOCUMENT EMBEDDINGS

We still need a fixed sized vector for the whole document

- so add up all the vectors
- so find the average of all the vectors
- so find the max of each value in vectors
- or do something else:
    - Learn a representation from sequence of embeddings
    - Train a model on all (whole) documents

TOO MANY WORDS/FEATURES

Words

Contextualized word embeddings
- care about some context

Could once previous and next word vectors

But, it gets very big very quickly
- even with case folding

POS is more limited size
- e.g. 45ish tags (PTB), smaller representation
- smaller number of contexts

Features

If you have too many features
- each sample has some unique combination
-training works well, but no generalization

How much is too much/too little?
- depends
- retraining is good (usually, if in similar domain)
- Ask yourself if the system has the features ***you*** think are important for the task

SUMMARY

Features (must) be numeric

Convert discrete features to one-hot

Sparse vs Dense word representations

Bag of Words (bi-grams/tri-grams)

Word Embeddings (dense)
- pre-trained vs trained from scratch

Are your features enough/not enough?

Does it work? When does it fail? WHY?

NEXT CLASS PREVIEW

Modeling the *output* space with Conditional Random Fields