Natural Language Processing

AI51701/CSE71001

Lecture 5

09/12/2023

Instructor: Taehwan Kim

Language Modeling

Probabilistic Language Models

- Language modeling: assign probabilities to sentences
- Why?
  - machine translation:
    - P(**high** winds tonite) > P(**large** winds tonite)
  - spelling correction:
    - The office is about fifteen **minuets** from my house
  - speech recognition:
    - P(I saw a van) >> P(eyes awe of an)
  - summarization, question answering, etc.!

Probabilistic Language Modeling

- Goal: compute the probability of a sequence of words:

$$P(\boldsymbol{w}) = P(w_1, w_2, ..., w_n)$$

- Related task: probability of next word:

$$P(w_4 \mid w_1, w_2, w_3)$$

- A model that computes either of these:

$$P(\boldsymbol{w}) \quad \text{or} \quad P(w_k \mid w_1, w_2, ..., w_{k-1})$$

is called a **language model (LM)**

Chain Rule for computing joint probability of words in sentence

$$P(w_1, w_2, ..., w_n) = \prod_i P(w_i \mid w_1, w_2, ..., w_{i-1})$$

P(“*its water is so transparent*”) =

P(*its*) × P(*water* | *its*) × P(*is* | *its water*)

× P(*so* | *its water is*) × P(*transparent* | *its water is so*)

How to estimate these probabilities

- Could we just count and divide?

$$P(\text{the} \mid \text{its water is so transparent that}) = \frac{Count(\text{its water is so transparent that the})}{Count(\text{its water is so transparent that})}$$

- No! too many possible sentences!

- We'll never see enough data for estimating these

Markov Assumption

- Simplifying assumption:

$$P(\text{the} \mid \text{its water is so transparent that}) \approx P(\text{the} \mid \text{that})$$

- Or maybe:

$$P(\text{the} \mid \text{its water is so transparent that}) \approx P(\text{the} \mid \text{transparent that})$$

- I.e., we approximate each component in the product:

$$P(w_i \mid w_1, ..., w_{i-2}, w_{i-1}) \approx P(w_i \mid w_{i-k}, ..., w_{i-2}, w_{i-1})$$

Simplest case: Unigram model

$$P(w_1, w_2, ..., w_n) = \prod_i P(w_i)$$

- Automatically generated sentences from a unigram model:

```
fifth an of futures the an incorporated a a the
inflation most dollars quarter in is mass

thrift did eighty said hard 'm july bullish

that or limited the
```

Bigram model

- condition on the previous word:

$$P(w_1, w_2, \ldots, w_n) = \prod_i P(w_i \mid w_{i-1})$$

- Automatically generated sentences from a unigram model:

```
texaco rose one in this issue is pursuing growth in a boiler
house said mr. gurria mexico 's motion control proposal
without permission from five hundred fifty five yen

outside new car parking lot of the agreement reached

this would be a record november
```

n-gram models

- We can extend to trigrams, 4-grams, 5-grams
- In general this is an insufficient model of language
  - because language has **long-distance dependencies**:

    "The computer which I had just put into the machine room on the fi6h floor crashed."

- But we can often get away with n-gram models

Estimating bigram probabilities

- The maximum likelihood estimate (MLE)

$$P(w_i \mid w_{i-1}) = \frac{\text{count}(w_{i-1}, w_i)}{\text{count}(w_{i-1})}$$

An example

$$P(w_i \mid w_{i-1}) = \frac{\text{count}(w_{i-1}, w_i)}{\text{count}(w_{i-1})}$$

<s> I am Sam </s>
<s> Sam I am </s>
<s> I do not like green eggs and ham </s>

$P(\texttt{I} \mid \texttt{<s>}) = \frac{2}{3} = .67$ $P(\texttt{Sam} \mid \texttt{<s>}) = \frac{1}{3} = .33$ $P(\texttt{am} \mid \texttt{I}) = \frac{2}{3} = .67$

$P(\texttt{</s>} \mid \texttt{Sam}) = \frac{1}{2} = 0.5$ $P(\texttt{Sam} \mid \texttt{am}) = \frac{1}{2} = .5$ $P(\texttt{do} \mid \texttt{I}) = \frac{1}{3} = .33$

More examples:
Berkeley Restaurant Project sentences

- Can you tell me about any good Cantonese restaurants close by
- Mid priced thai food is what i'm looking for
- Can you give me a listing of the kinds of food that are available

Raw bigram counts

- Counts from 9,222 sentences
- E.g., *"i want"* occurs 827 times

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Raw bigram probabilities

- normalize by unigram counts:

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

- bigram probabilities

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

J&M/SLP3

Bigram estimates of sentence probabilities

$$
\begin{aligned}
&P(\langle s\rangle\ I\ want\ english\ food\ \langle /s\rangle) = \\
&\quad P(I \mid \langle s\rangle) \\
&\quad \times\ P(want \mid I) \\
&\quad \times\ P(english \mid want) \\
&\quad \times\ P(food \mid english) \\
&\quad \times\ P(\langle /s\rangle \mid food) \\
&\qquad =\ .000031
\end{aligned}
$$

Practical Issues

- we do everything in log space
  - avoid underflow
  - also adding is faster than multiplying

$$\log(p_1 \times p_2 \times p_3) = \log p_1 + \log p_2 + \log p_3$$

Language Modeling Toolkits

- SRILM
  - http://www.speech.sri.com/projects/srilm/
- KenLM
  - https://kheafield.com/code/kenlm/

Evaluation: How good is our model?

- Does our language model prefer good sentences to bad ones?
  - assign higher probability to “real” or “frequently observed” sentences
    - than “ungrammatical” or “rarely observed” sentences?

Extrinsic evaluation of N-gram models

- Best evaluation for comparing models A and B
  - put each model in a task
    - spelling corrector, speech recognizer, MT system
  - run the task, get an accuracy for A and for B
    - how many misspelled words corrected properly
    - how many words translated correctly
  - compare accuracy for A and B

Intuition of Perplexity

- The Shannon Game:
  - how well can we predict the next word?

I always order pizza with cheese and ____

The 33rd President of the US was ____

I saw a ____

mushrooms 0.1

pepperoni 0.1

anchovies 0.01

...

fried rice 0.0001

...

and 1e-100

  - unigrams are terrible at this game (why?)
- A better model of a text is one which assigns a higher probability to the word that actually occurs

Probability of Held-out Data

- probability of held-out sentences:

$$\prod_i P(\boldsymbol{w}^{(i)})$$

- let's work with log-probabilities:

$$\log_2 \prod_i P(\boldsymbol{w}^{(i)}) = \sum_i \log_2 P(\boldsymbol{w}^{(i)})$$

- divide by number of words *M* in held-out sentences:

$$\frac{1}{M} \sum_i \log_2 P(\boldsymbol{w}^{(i)})$$

Probability -> Perplexity

- average log-probability of held-out words:

$$\ell = \frac{1}{M} \sum_i \log_2 P(\boldsymbol{w}^{(i)})$$

- perplexity:

$$\mathrm{PP} = 2^{-\ell}$$

Lower perplexity = better model

- Train: 38 million words
- Test: 1.5 million words

n-gram order:	unigram	bigram	trigram
perplexity:	962	170	109

Journal of Machine Learning Research 3 (2003) 1137–1155 Submitted 4/02; Published 2/03

A Neural Probabilistic Language Model

Yoshua Bengio BENGIOY@IRO.UMONTREAL.CA
Réjean Ducharme DUCHARME@IRO.UMONTREAL.CA
Pascal Vincent VINCENTP@IRO.UMONTREAL.CA
Christian Jauvin JAUVINC@IRO.UMONTREAL.CA
Département d'Informatique et Recherche Opérationnelle
Centre de Recherche Mathématiques
Université de Montréal, Montréal, Québec, Canada

- Idea: use a neural network for n-gram language modeling:

$$P_{\boldsymbol{\theta}}(w_t \mid w_{t-n+1}, \ldots, w_{t-2}, w_{t-1})$$

Journal of Machine Learning Research 3 (2003) 1137–1155 Submitted 4/02; Published 2/03

A Neural Probabilistic Language Model

Yoshua Bengio BENGIOY@IRO.UMONTREAL.CA
Réjean Ducharme DUCHARME@IRO.UMONTREAL.CA
Pascal Vincent VINCENTP@IRO.UMONTREAL.CA
Christian Jauvin JAUVINC@IRO.UMONTREAL.CA
Département d'Informatique et Recherche Opérationnelle
Centre de Recherche Mathématiques
Université de Montréal, Montréal, Québec, Canada

- This is not the earliest paper on using neural networks for n-gram language models, but it's the most well-known and first to scale up
- See paper for citations of earlier work

Neural Probabilistic Language Models (Bengio et al., 2003)

1.1 Fighting the Curse of Dimensionality with Distributed Representations

In a nutshell, the idea of the proposed approach can be summarized as follows:

1. associate with each word in the vocabulary a distributed *word feature vector* (a real-valued vector in $\mathbb{R}^m$),
2. express the joint *probability function* of word sequences in terms of the feature vectors of these words in the sequence, and
3. learn simultaneously the *word feature vectors* and the parameters of that *probability function*.

What is a neural network?

- Just think of a neural network as a function
- It has inputs and outputs
- “neural” typically means one type of functional building block (“neural layers”), but the term has broadened
- Neural modeling is now better thought of as a modeling strategy (leveraging “distributed representations” or “representation learning”), or a family of related methods

Notation

$\mathbf{u} =$ a vector

$u_i =$ entry *i* in the vector

$\mathbf{W} =$ a matrix

$w_{ij} =$ entry (*i*,*j*) in the matrix

$\boldsymbol{x} =$ a structured object

$x_i =$ entry *i* in the structured object

neural layer = affine transform + nonlinearity

$$\mathbf{z}^{(1)} = g\left(\underbrace{\mathbf{U}^{(0)}\mathbf{x} + \mathbf{b}^{(0)}}_{\text{affine transform}}\right)$$

nonlinearity

- This is a single “layer” of a neural network
- Input vector is $\mathbf{x}$
- $\mathbf{U}^{(0)}$ and $\mathbf{b}^{(0)}$ are parameters

neural layer = affine transform + nonlinearity

$$\mathbf{z}^{(1)} = g\left(\mathbf{U}^{(0)}\mathbf{x} + \mathbf{b}^{(0)}\right)$$

"hidden units"

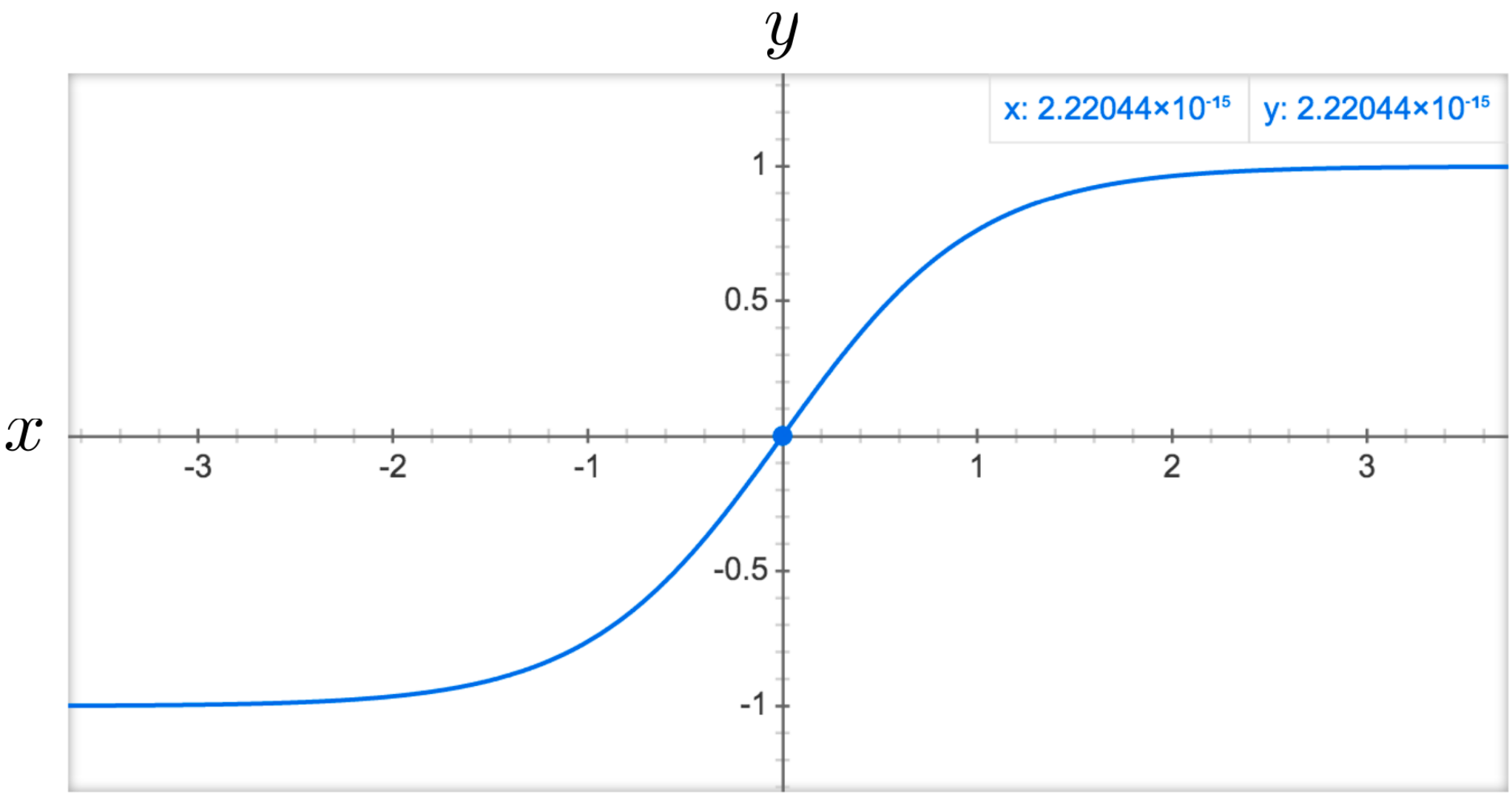
- Vector of "hidden units" is $\mathbf{z}^{(1)}$
- Think of these as features computed from $\mathbf{x}$

Nonlinearities

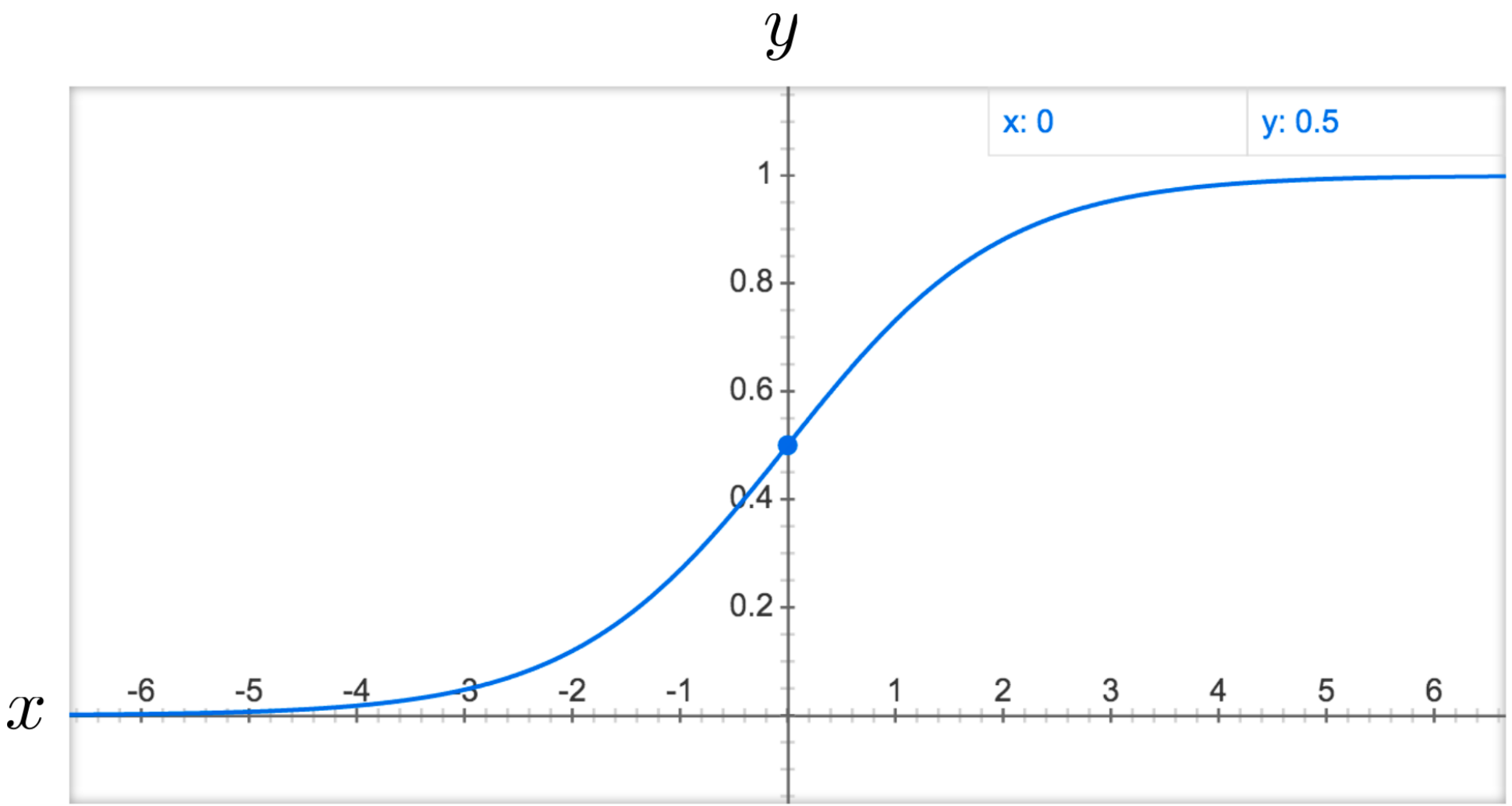
$$\mathbf{z}^{(1)} = g\left(\mathbf{U}^{(0)}\mathbf{x} + \mathbf{b}^{(0)}\right)$$

- Most common: elementwise application of *g* function to each entry in vector
- Examples...

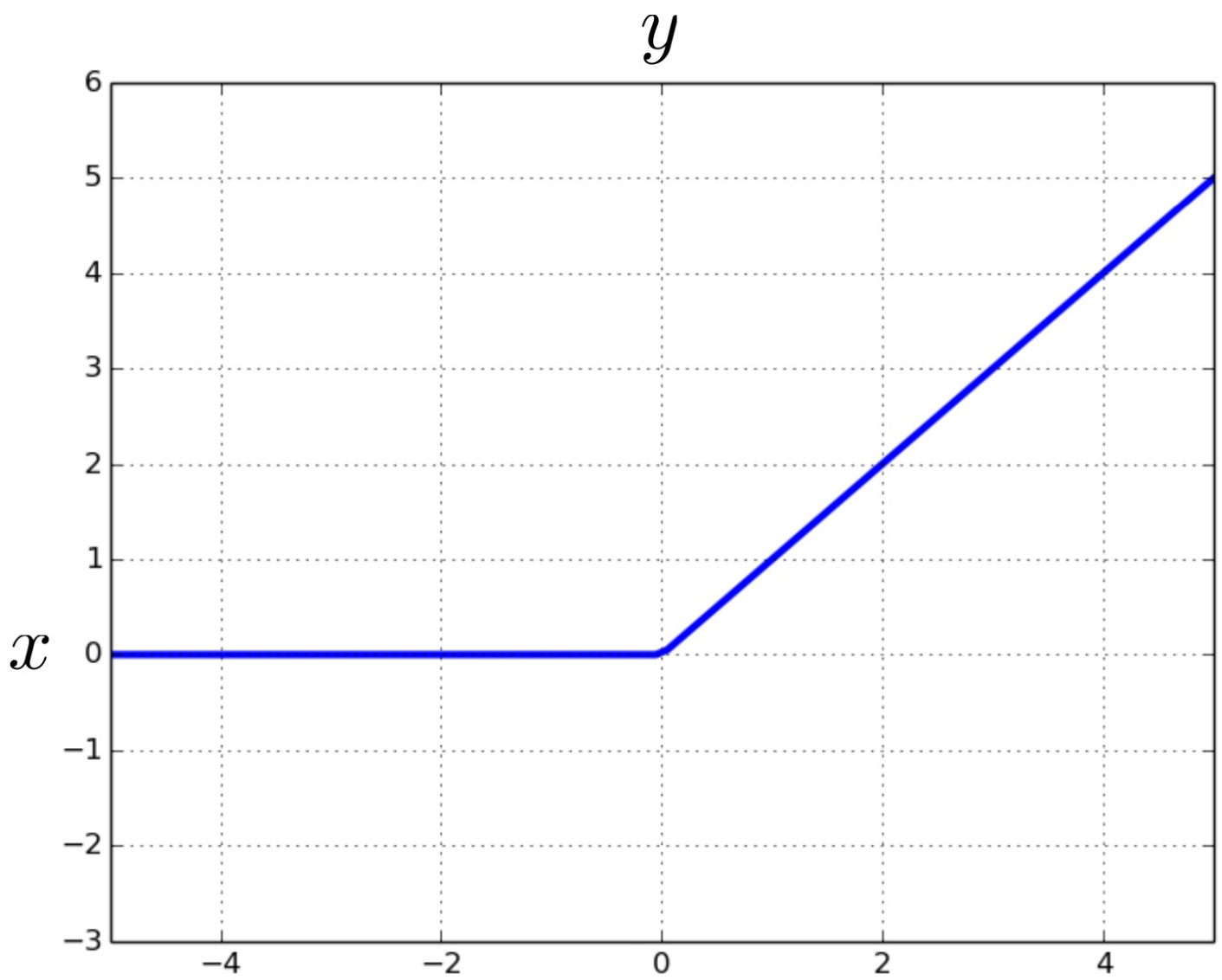
tanh: $y = \tanh(x)$



(logistic) sigmoid: $$y = \frac{1}{1 + \exp\{-x\}}$$



rectified linear unit (ReLU): $y = \max(0, x)$



Adding layers...

$$\mathbf{z}^{(1)} = g\left(\mathbf{U}^{(0)}\mathbf{x} + \mathbf{b}^{(0)}\right)$$

$$\mathbf{z}^{(2)} = g\left(\mathbf{U}^{(1)}\mathbf{z}^{(1)} + \mathbf{b}^{(1)}\right)$$

$$\ldots$$

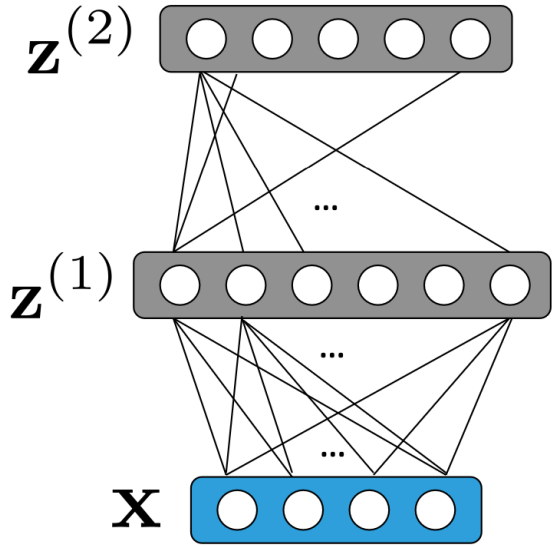
- Use output of one layer as input to next

Adding layers...

$$\mathbf{z}^{(1)} = g\left(\mathbf{U}^{(0)}\mathbf{x} + \mathbf{b}^{(0)}\right)$$

$$\mathbf{z}^{(2)} = g\left(\mathbf{U}^{(1)}\mathbf{z}^{(1)} + \mathbf{b}^{(1)}\right)$$

$$\ldots$$



- Use output of one layer as input to next
- "feed-forward" and/or "fully-connected" layers

Neural Network for Sentiment Classification

$$\mathbf{z}^{(1)} = g\left(\mathbf{U}^{(0)}\mathbf{x} + \mathbf{b}^{(0)}\right)$$

$$\mathbf{s} = \mathbf{U}^{(1)}\mathbf{z}^{(1)} + \mathbf{b}^{(1)}$$

vector of label scores

Neural Network for Sentiment Classification

$$\mathbf{z}^{(1)} = g\left(\mathbf{U}^{(0)}\mathbf{x} + \mathbf{b}^{(0)}\right)$$

$$\mathbf{s} = \mathbf{U}^{(1)}\mathbf{z}^{(1)} + \mathbf{b}^{(1)}$$

$$\mathbf{s} = \begin{bmatrix} \text{score}(\boldsymbol{x}, \text{positive}, \boldsymbol{w}) \\ \text{score}(\boldsymbol{x}, \text{negative}, \boldsymbol{w}) \end{bmatrix}$$

Use softmax function to convert scores into probabilities

$$\mathbf{s} = \begin{bmatrix} \text{score}(\boldsymbol{x}, \text{positive}, \boldsymbol{w}) \\ \text{score}(\boldsymbol{x}, \text{negative}, \boldsymbol{w}) \end{bmatrix}$$

$$\mathbf{p} = \text{softmax}(\mathbf{s}) = \begin{bmatrix} \frac{\exp\{\text{score}(\boldsymbol{x}, \text{positive}, \boldsymbol{w})\}}{Z} \\ \frac{\exp\{\text{score}(\boldsymbol{x}, \text{negative}, \boldsymbol{w})\}}{Z} \end{bmatrix}$$

$$Z = \exp\{\text{score}(\boldsymbol{x}, \text{positive}, \boldsymbol{w})\} + \exp\{\text{score}(\boldsymbol{x}, \text{negative}, \boldsymbol{w})\}$$

Why nonlinearities?

network with 1 hidden layer:

$$\mathbf{z}^{(1)} = g\left(\mathbf{U}^{(0)}\mathbf{x} + \mathbf{b}^{(0)}\right)$$

$$\mathbf{s} = \mathbf{U}^{(1)}\mathbf{z}^{(1)} + \mathbf{b}^{(1)}$$

- If g is linear, then we can rewrite the above as a single affine transform
- Can you prove this? (use distributivity of matrix multiplication)

Learning with Neural Networks

$$\mathbf{z}^{(1)} = g\left(\mathbf{U}^{(0)}\mathbf{x} + \mathbf{b}^{(0)}\right)$$

$$\mathbf{s} = \mathbf{U}^{(1)}\mathbf{z}^{(1)} + \mathbf{b}^{(1)}$$

$$\mathbf{s} = \begin{bmatrix} \text{score}(\boldsymbol{x}, \text{positive}, \boldsymbol{w}) \\ \text{score}(\boldsymbol{x}, \text{negative}, \boldsymbol{w}) \end{bmatrix}$$

$$\text{classify}(\boldsymbol{x}, \boldsymbol{w}) = \underset{y}{\text{argmax}} \;\; \text{score}(\boldsymbol{x}, y, \boldsymbol{w})$$

- We can use any of our loss functions from before, as long as we can compute (sub)gradients
- Algorithm for doing this efficiently: **backpropagation**
- Basically just the chain rule of derivatives

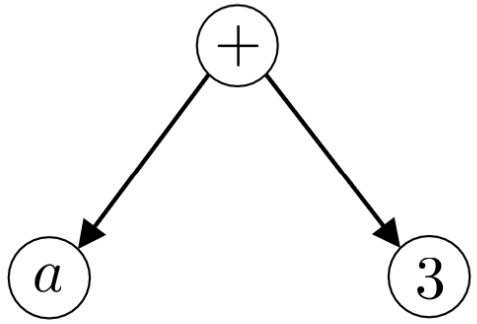
Computation Graphs

- A useful way to represent the computations performed by a neural model (or any model!)
- Why useful? makes it easy to implement automatic differentiation (backpropagation)
- Many neural net toolkits let you define your model in terms of computation graphs (PyTorch, TensorFlow, DyNet, Theano, etc.)

Backpropagation

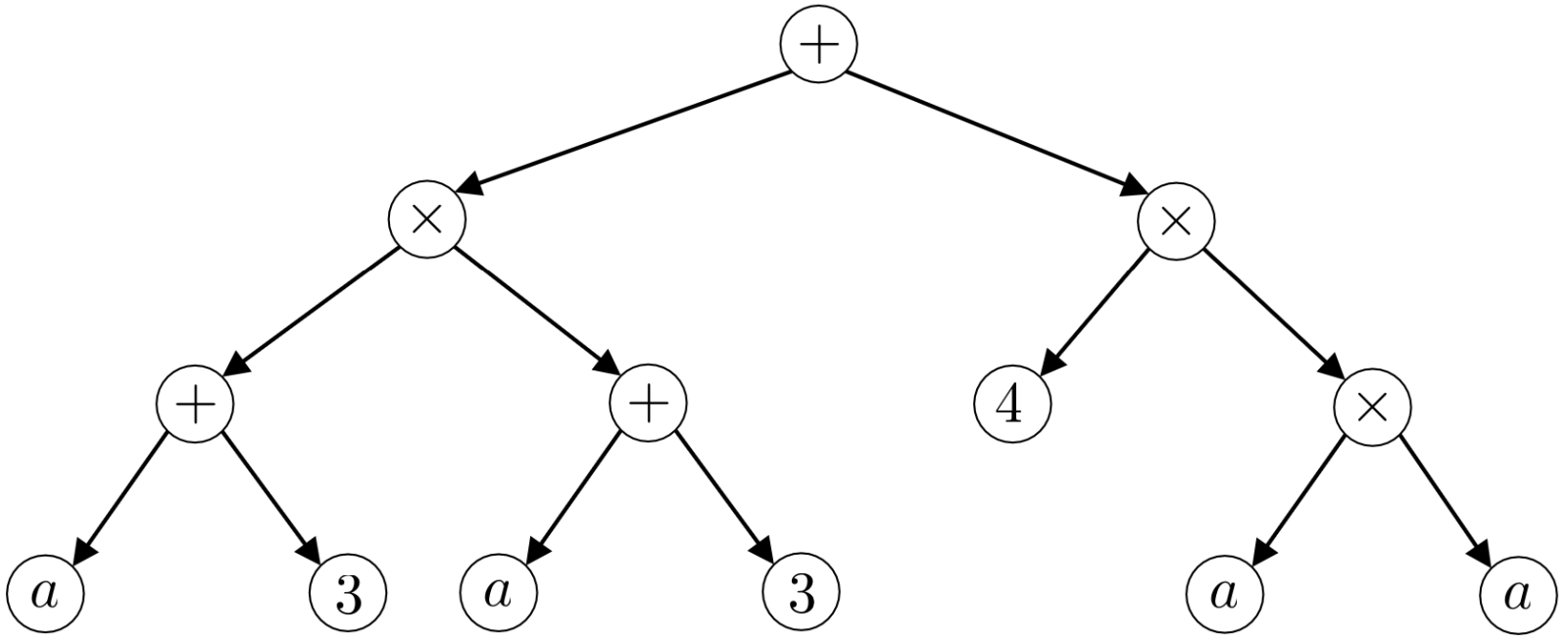
- Backpropagation has become associated with neural networks, but it's much more general
- Can be also used to compute gradients in **linear** models for structured prediction

A simple computation graph:



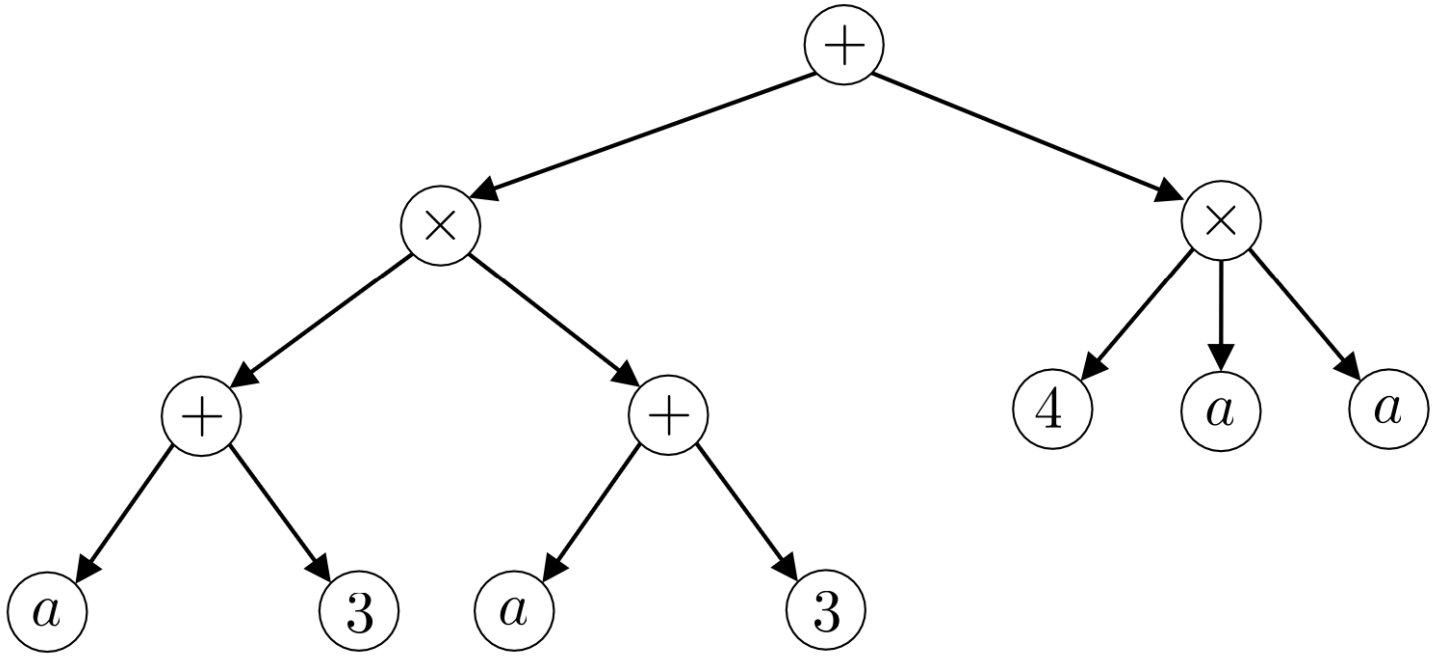
- Represents expression “a + 3”

A slightly bigger computation graph:

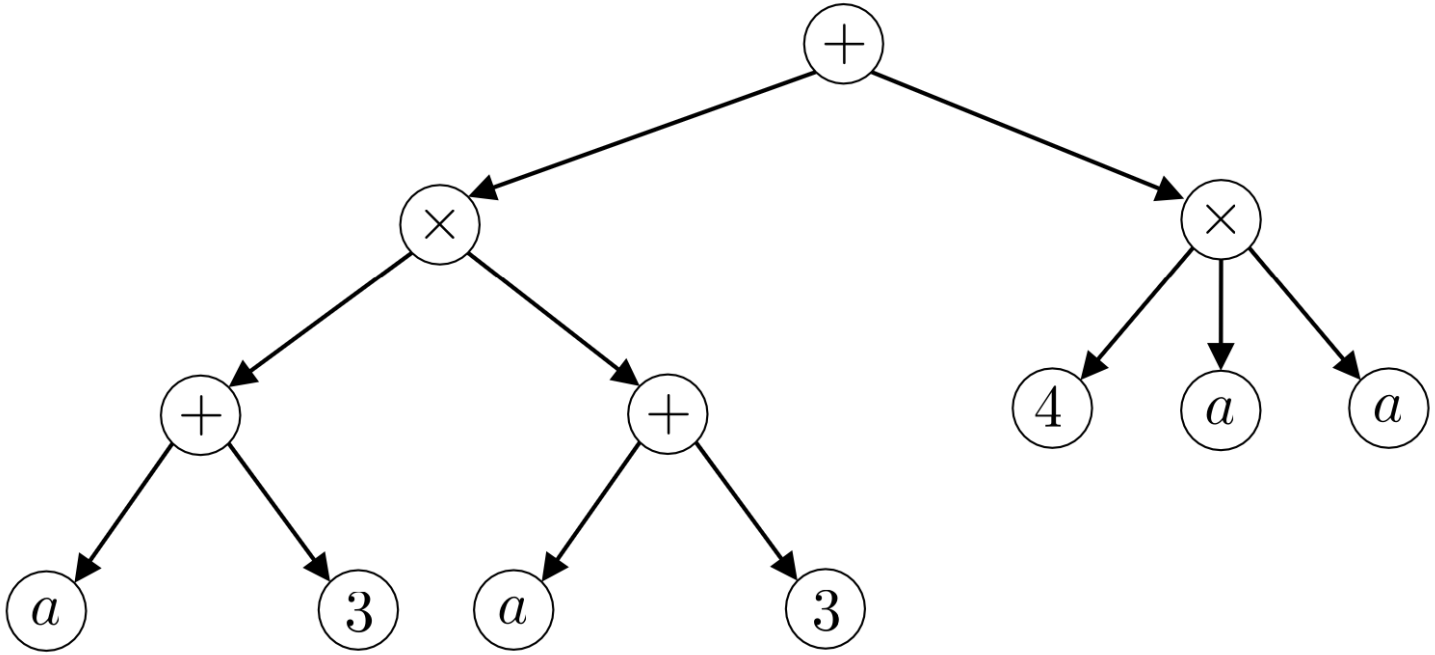


- Represents expression "$(a + 3)^2 + 4a^2$"

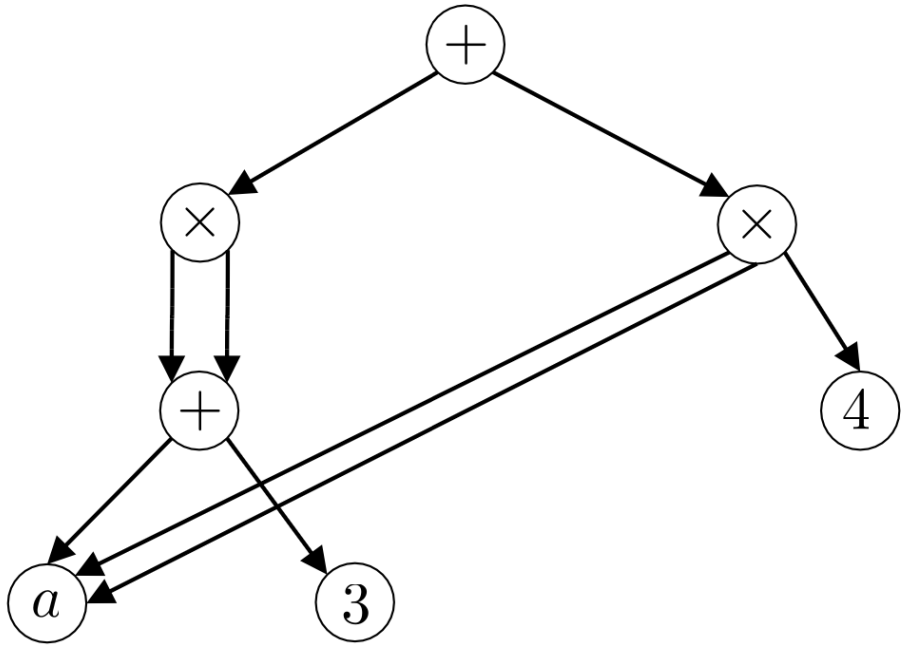
Operators can have more than 2 operands:



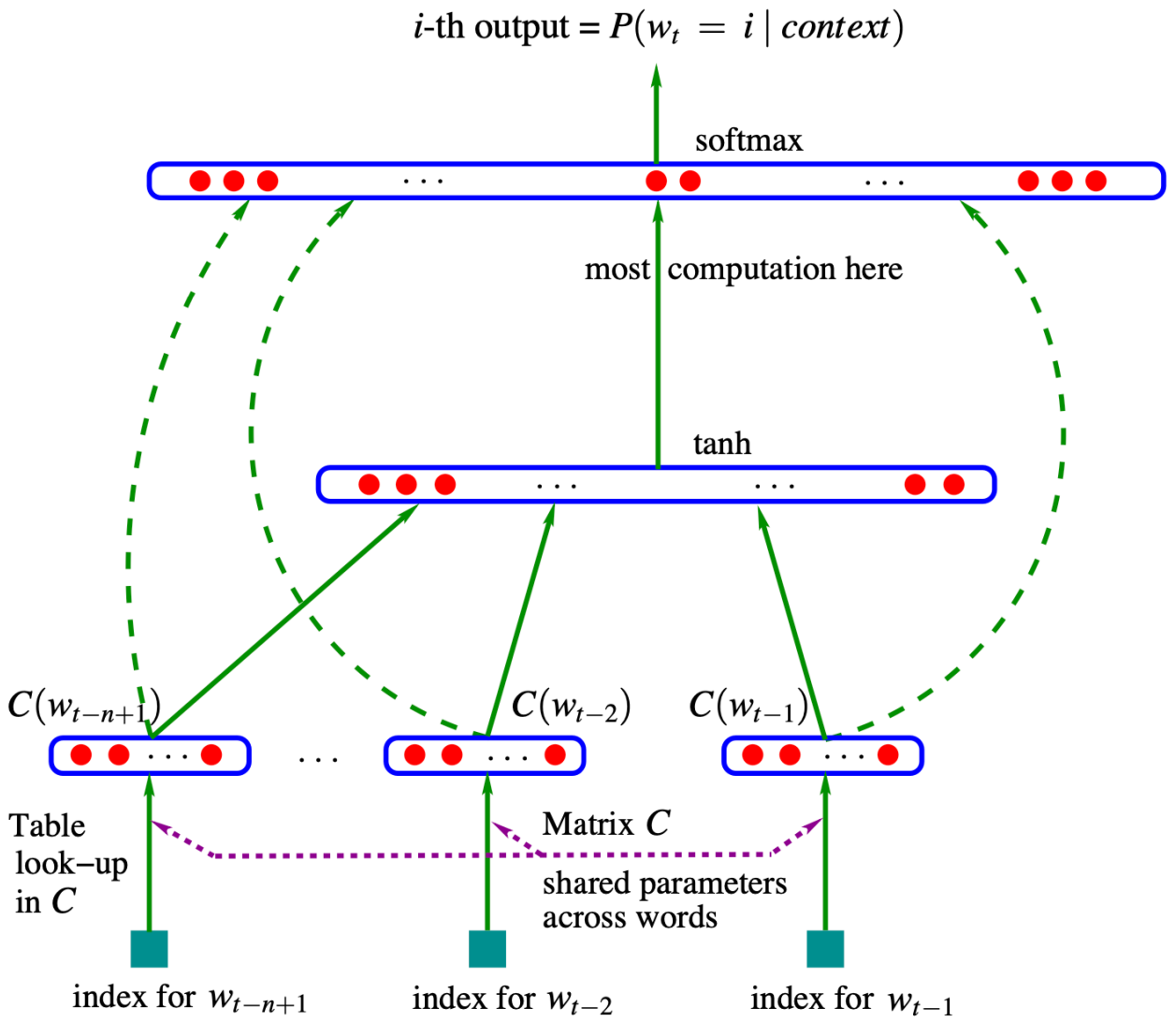
- Still represents expression "$(a + 3)^2 + 4a^2$"



- More concise:

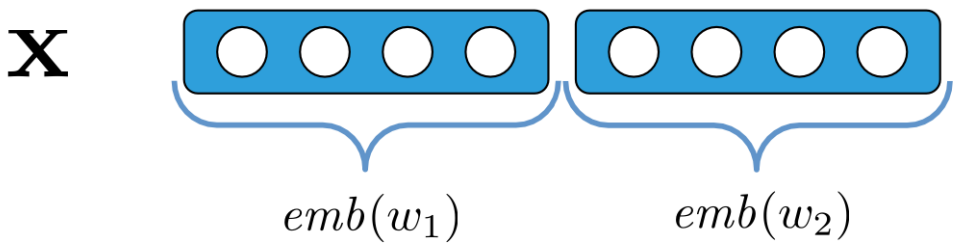


Model (Bengio et al., 2003)



A Simple Neural Trigram Language Model

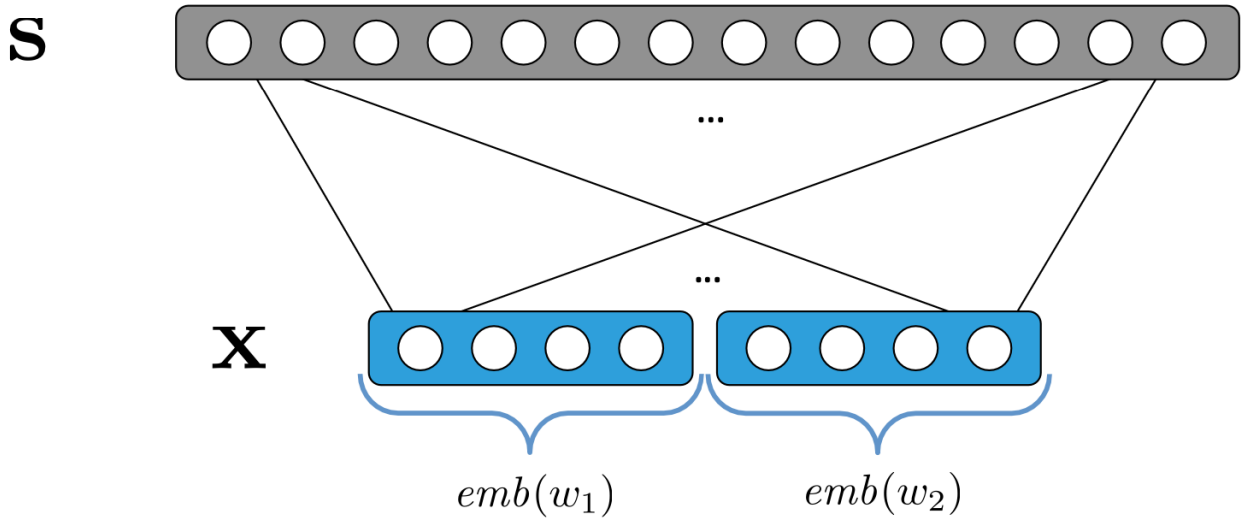
- Given previous words w1 and w2, predict next word
- Input is concatenation of vectors (embeddings) of previous words:



$$\mathbf{x} = cat(emb(w_1), emb(w_2))$$

A Simple Neural Trigram Language Model

- Output vector contains scores of possible next words:



$$emb(w) \in \mathbb{R}^d$$

$$\mathbf{x} \in \mathbb{R}^{2d}$$

$$\mathbf{s} \in \mathbb{R}^{|\mathcal{V}|}$$

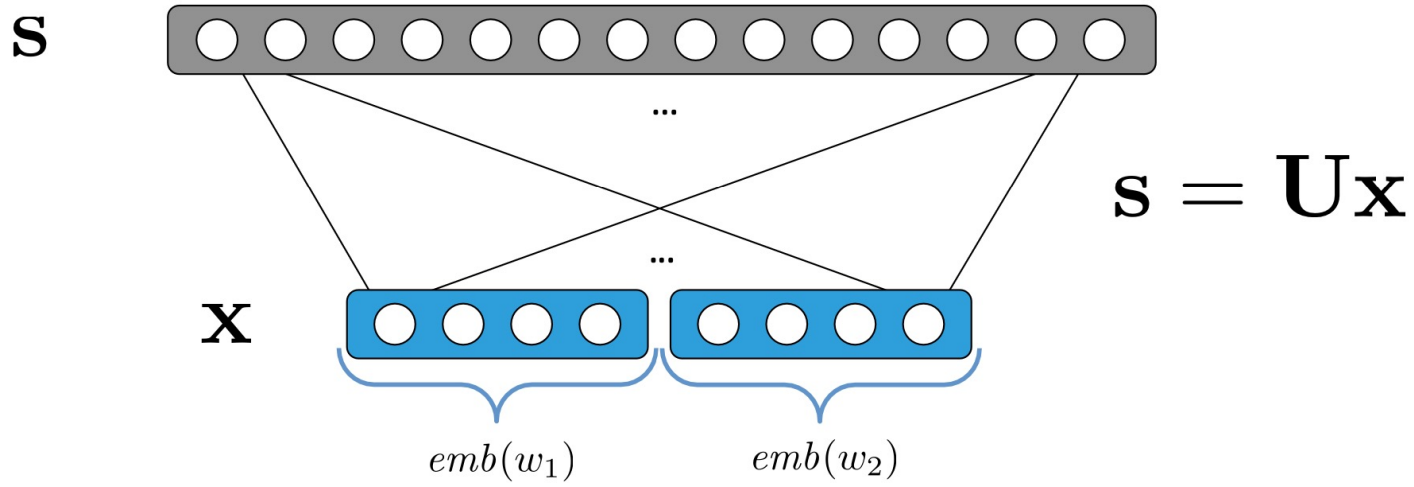
$$\mathbf{U} \in \mathbb{R}^{|\mathcal{V}| \times 2d}$$

$$\mathbf{s} = \mathbf{U}\mathbf{x}$$

$$s_i = \text{score}(\mathbf{x}, w_i, \mathbf{U})$$

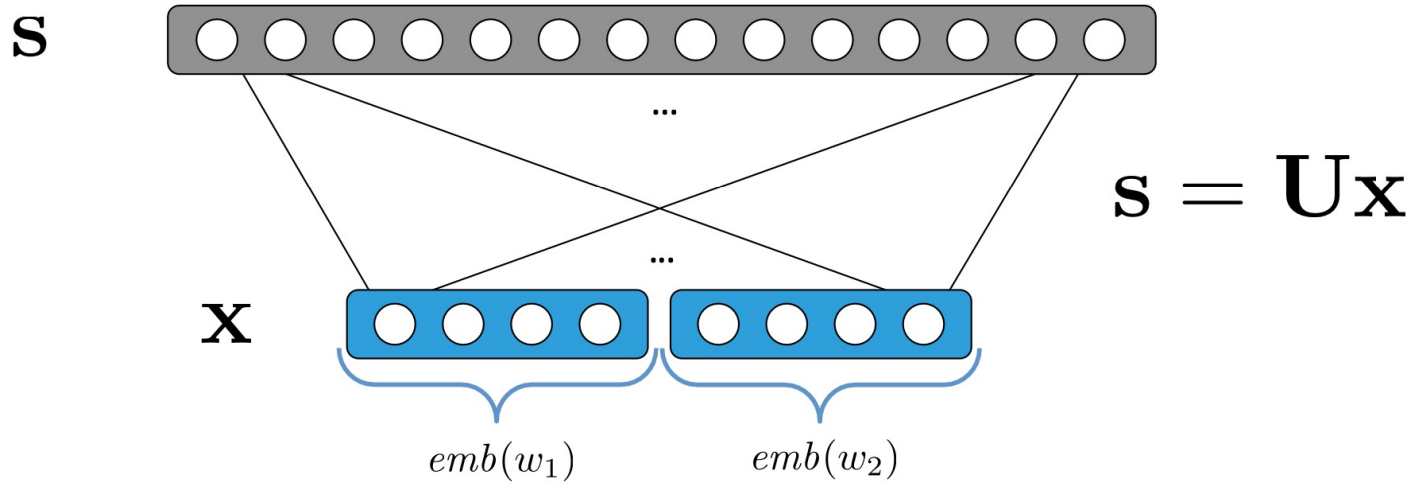
$$\text{score}(\mathbf{x}, w_i, \mathbf{U}) = \mathbf{x}^\top \mathbf{U}_{i,1:d}$$

A Simple Neural Trigram Language Model



- How many parameters are in this model? $|\mathcal{V}| \times 3d$

A Simple Neural Trigram Language Model

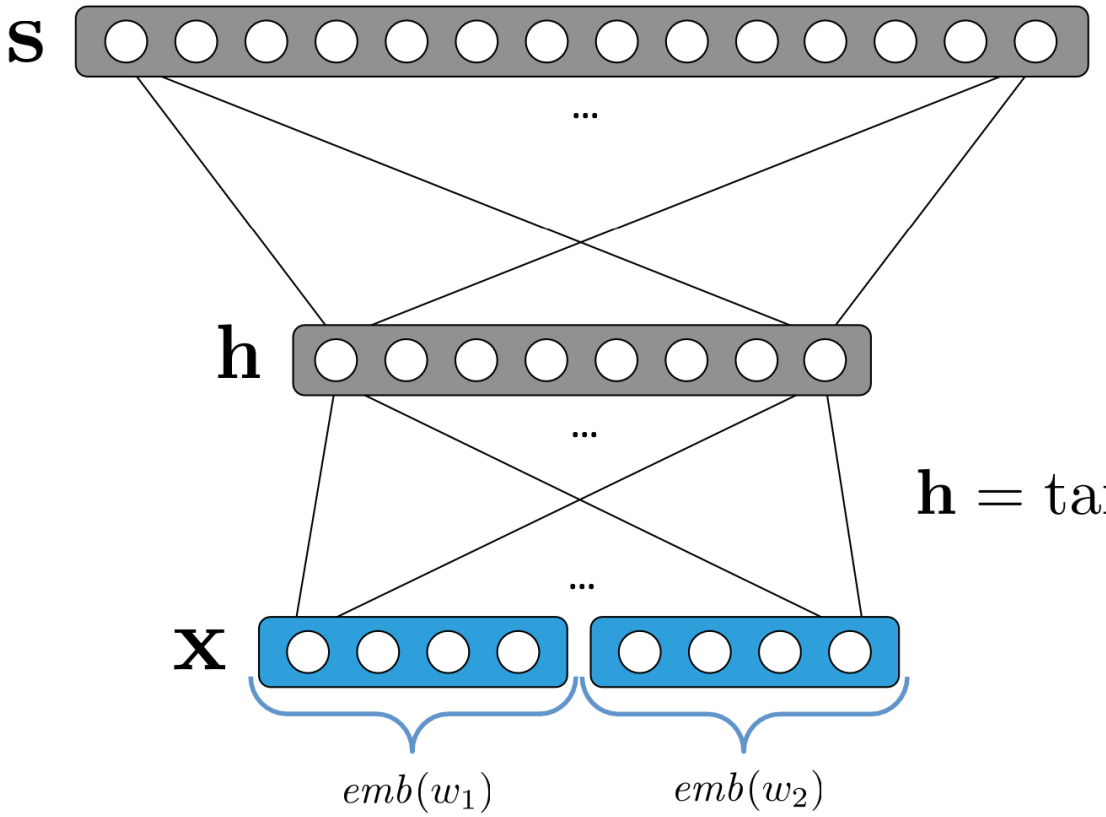


- Most common way: log loss

$$\text{loss}_{\log}(\langle w_1, w_2 \rangle, w_3, \boldsymbol{\theta}) = -\log p_{\boldsymbol{\theta}}(w_3 \mid \langle w_1, w_2 \rangle)$$

$$p_{\boldsymbol{\theta}}(w_3 \mid \langle w_1, w_2 \rangle) \propto \exp\{\text{score}(cat(emb(w_1), emb(w_2)), w_3, \mathbf{U})\}$$

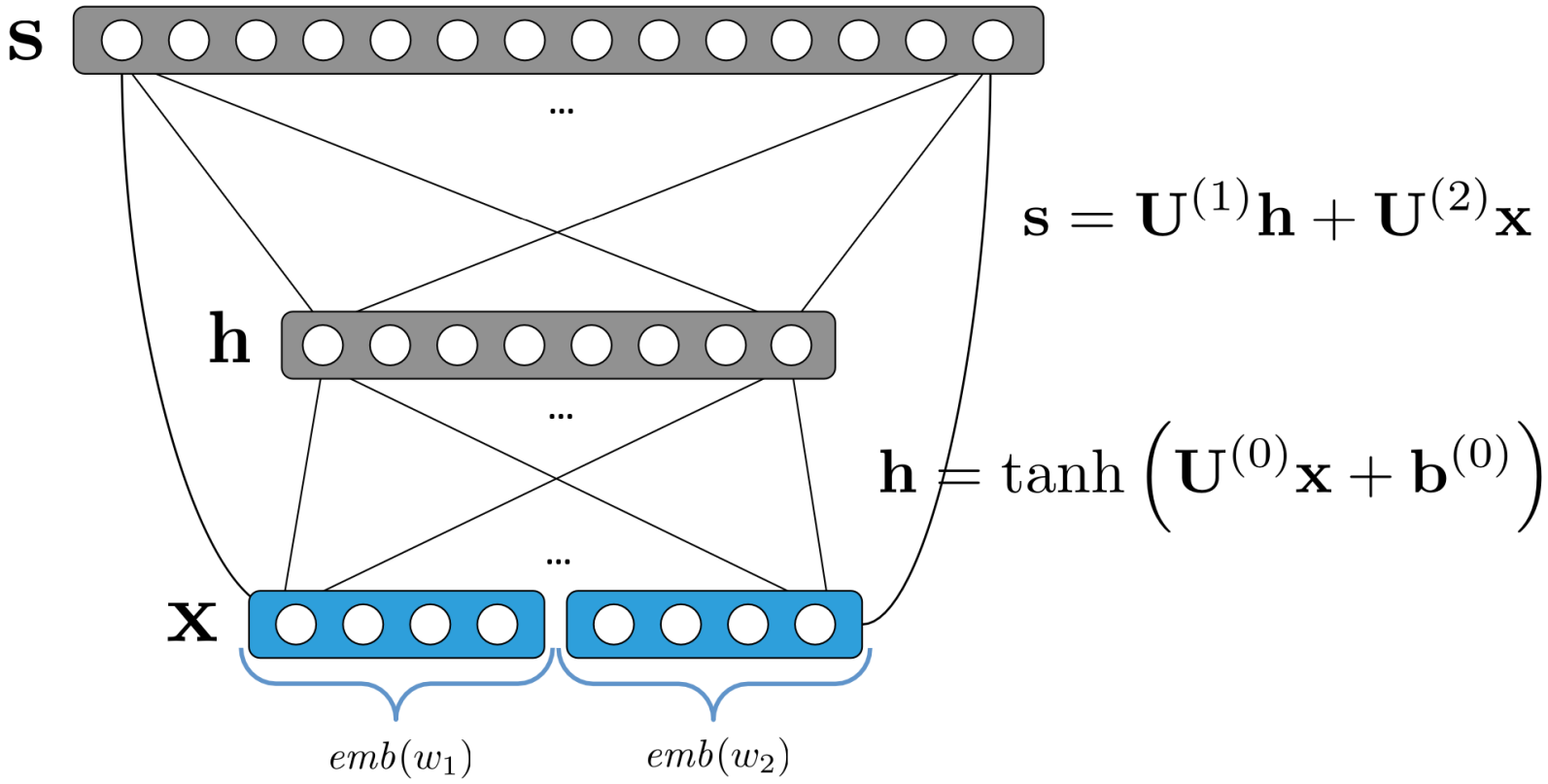
Adding a Hidden Layer



$$\mathbf{s} = \mathbf{U}^{(1)}\mathbf{h}$$

$$\mathbf{h} = \tanh\left(\mathbf{U}^{(0)}\mathbf{x} + \mathbf{b}^{(0)}\right)$$

Adding Connections



Bengio et al. (2003)

- Experiments:
  - feed-forward neural network
  - they minimized log loss of next word conditioned on a fixed number of previous words
  - ~800k training tokens, vocab size of 17k
  - they trained for 5 epochs, which took 3 weeks on 40 CPUs!

Experiments (Bengio et al., 2003)

	n	c	h	m	direct	mix	train.	valid.	test.
MLP1	5		50	60	yes	no	182	284	268
MLP2	5		50	60	yes	yes		275	257
MLP3	5		0	60	yes	no	201	327	310
MLP4	5		0	60	yes	yes		286	272
MLP5	5		50	30	yes	no	209	296	279
MLP6	5		50	30	yes	yes		273	259
MLP7	3		50	30	yes	no	210	309	293
MLP8	3		50	30	yes	yes		284	270
MLP9	5		100	30	no	no	175	280	276
MLP10	5		100	30	no	yes		265	**252**

classes). n : order of the model. c : number of word classes in class-based n-grams. h : number of hidden units. m : number of word features for MLPs, number of classes for class-based n-grams. *direct*: whether there are direct connections from word features to outputs. *mix*: whether the output probabilities of the neural network are mixed with the output of the trigram (with a weight of 0.5 on each). The last three columns give perplexity on the training, validation and test sets.

	n	c	h	m	direct	mix	train.	valid.	test.
MLP1	5		50	60	yes	no	182	284	268
MLP2	5		50	60	yes	yes		275	257
MLP3	5		0	60	yes	no	201	327	310
MLP4	5		0	60	yes	yes		286	272
MLP5	5		50	30	yes	no	209	296	279
MLP6	5		50	30	yes	yes		273	259
MLP7	3		50	30	yes	no	210	309	293
MLP8	3		50	30	yes	yes		284	270
MLP9	5		100	30	no	no	175	280	276
MLP10	5		100	30	no	yes		265	**252**

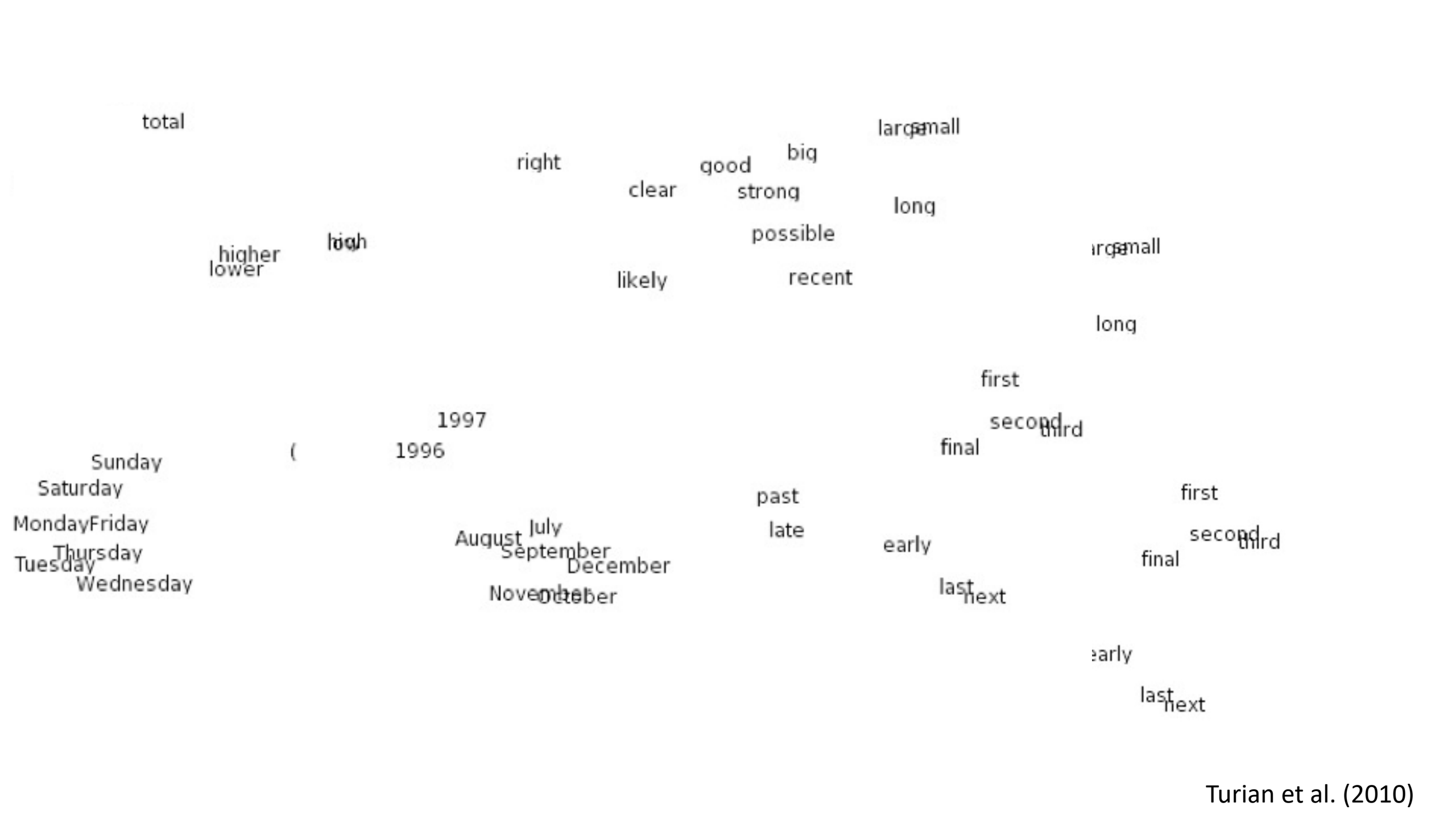
- Observations:
  - hidden layer (h > 0) helps
  - interpolating with n-gram model ("mix") helps
  - using higher word embedding dimensionality helps
  - 5-gram model better than trigram

Experiments

	n	c	h	m	direct	mix	train.	valid.	test.
MLP1	5		50	60	yes	no	182	284	268
MLP2	5		50	60	yes	yes		275	257
MLP3	5		0	60	yes	no	201	327	310
MLP4	5		0	60	yes	yes		286	272
MLP5	5		50	30	yes	no	209	296	279
MLP6	5		50	30	yes	yes		273	259
MLP7	3		50	30	yes	no	210	309	293
MLP8	3		50	30	yes	yes		284	270
MLP9	5		100	30	no	no	175	280	276
MLP10	5		100	30	no	yes		265	**252**
Del. Int.	3						31	352	336
Kneser-Ney back-off	3							334	323
Kneser-Ney back-off	4							332	321
Kneser-Ney back-off	5							332	321
class-based back-off	3	150						348	334
class-based back-off	3	200						354	340
class-based back-off	3	500						326	**312**
class-based back-off	3	1000						335	319

Bengio et al. (2003)

- They discuss how the word embedding space might be interesting to examine but they don't do this
- They suggest that a good way to visualize/ interpret word embeddings would be to use 2 dimensions
- They discussed handling polysemous words, unknown words, inference speed-ups, etc.



Turian et al. (2010)

Collobert et al. (2011)

Journal of Machine Learning Research 12 (2011) 2493-2537 Submitted 1/10; Revised 11/10; Published 8/11

Natural Language Processing (Almost) from Scratch

Ronan Collobert* RONAN@COLLOBERT.COM
Jason Weston† JWESTON@GOOGLE.COM
Léon Bottou‡ LEON@BOTTOU.ORG
Michael Karlen MICHAEL.KARLEN@GMAIL.COM
Koray Kavukcuoglu§ KORAY@CS.NYU.EDU
Pavel Kuksa¶ PKUKSA@CS.RUTGERS.EDU
NEC Laboratories America
4 Independence Way
Princeton, NJ 08540

Collobert et al. (2011)

- 631M word tokens, 100k vocab size, 11-word input window, 4 weeks of training
- They didn’t care about getting good perplexities, just good word embeddings for their downstream NLP tasks
- So they used a pairwise ranking loss (make an observed 11-word window have higher score than an unobserved 11-word window)

Collobert et al. (2011)

FRANCE	JESUS	XBOX	REDDISH	SCRATCHED	MEGABITS
454	1973	6909	11724	29869	87025
AUSTRIA	GOD	AMIGA	GREENISH	NAILED	OCTETS
BELGIUM	SATI	PLAYSTATION	BLUISH	SMASHED	MB/S
GERMANY	CHRIST	MSX	PINKISH	PUNCHED	BIT/S
ITALY	SATAN	IPOD	PURPLISH	POPPED	BAUD
GREECE	KALI	SEGA	BROWNISH	CRIMPED	CARATS
SWEDEN	INDRA	PSNUMBER	GREYISH	SCRAPED	KBIT/S
NORWAY	VISHNU	HD	GRAYISH	SCREWED	MEGAHERTZ
EUROPE	ANANDA	DREAMCAST	WHITISH	SECTIONED	MEGAPIXELS
HUNGARY	PARVATI	GEFORCE	SILVERY	SLASHED	GBIT/S
SWITZERLAND	GRACE	CAPCOM	YELLOWISH	RIPPED	AMPERES

Table 7: Word embeddings in the word lookup table of the language model neural network LM1 trained with a dictionary of size 100,000. For each column the queried word is followed by its index in the dictionary (higher means more rare) and its 10 nearest neighbors (using the Euclidean metric, which was chosen arbitrarily).

word2vec (Mikolov et al., 2013a)

Efficient Estimation of Word Representations in Vector Space

Tomas Mikolov
Google Inc., Mountain View, CA
tmikolov@google.com

Kai Chen
Google Inc., Mountain View, CA
kaichen@google.com

Greg Corrado
Google Inc., Mountain View, CA
gcorrado@google.com

Jeffrey Dean
Google Inc., Mountain View, CA
jeff@google.com

word2vec (Mikolov et al., 2013b)

Distributed Representations of Words and Phrases and their Compositionality

Tomas Mikolov
Google Inc.
Mountain View
mikolov@google.com

Ilya Sutskever
Google Inc.
Mountain View
ilyasu@google.com

Kai Chen
Google Inc.
Mountain View
kai@google.com

Greg Corrado
Google Inc.
Mountain View
gcorrado@google.com

Jeffrey Dean
Google Inc.
Mountain View
jeff@google.com

Learning word vectors

- Let’s use our classification framework
- We want to use unlabeled text to train the vectors
- We can convert our unlabeled text into a classification problem!
- How? (there are many possibilities)

skip-gram training data (window size = 5)

- skip-gram: predict context ("outside") words (position independent) given center word
- Corpus (English Wikipedia):

agriculture is the traditional mainstay of the cambodian economy .

but benares has been destroyed by an earthquake .

…

inputs (*x*)	outputs (*y*)
agriculture	<s>
agriculture	is
agriculture	the
is	<s>
is	agriculture
is	the
is	traditional
the	is
…	…

CBOW training data (window size = 5)

- Continuous Bag of Words (CBOW): predict center word from (bag of) context words
- corpus (English Wikipedia):

agriculture is the traditional mainstay of the cambodian economy .

but benares has been destroyed by an earthquake .

…

inputs (*x*)	outputs (*y*)
{<s>, is, the, traditional}	agriculture
{<s>, agriculture, the, traditional}	is
{agriculture, is, traditional, mainstay}	the
{is, the, mainstay, of}	traditional
{the, traditional, of, the}	mainstay
{traditional, mainstay, the, cambodian}	of
{mainstay, of, cambodian, economy}	the
…	…

skip-gram model

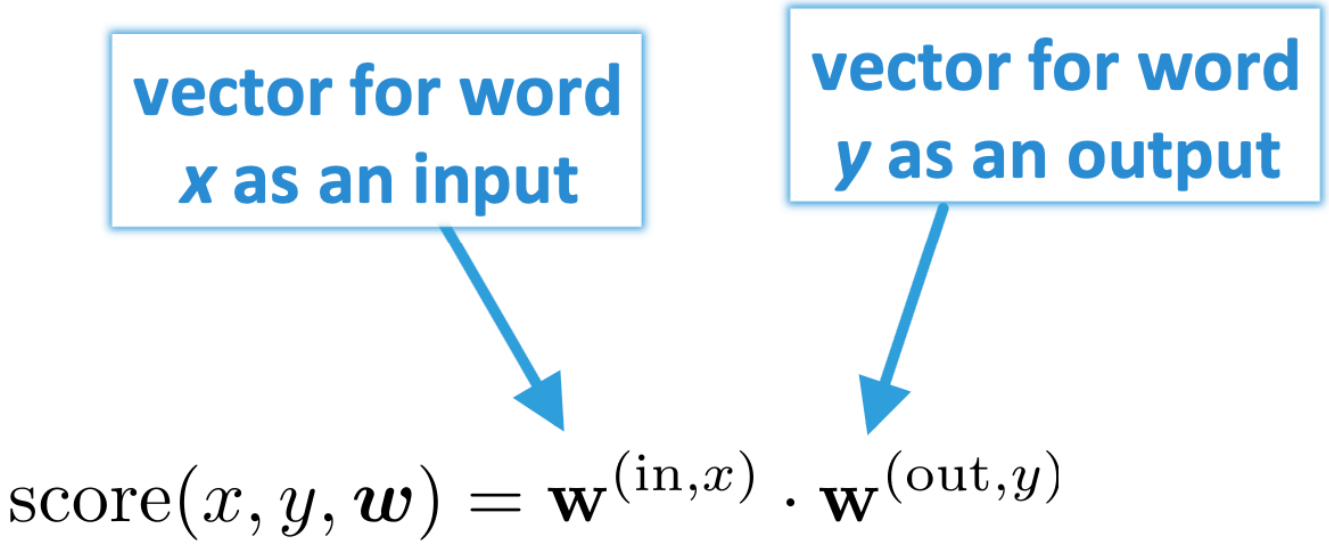
$$\text{classify}(x, \boldsymbol{w}) = \underset{y}{\text{argmax}}\ \text{score}(x, y, \boldsymbol{w})$$

- Here's our data:

inputs (*x*)	outputs (*y*)
agriculture	<s>
agriculture	is
agriculture	the
is	<s>
...	...

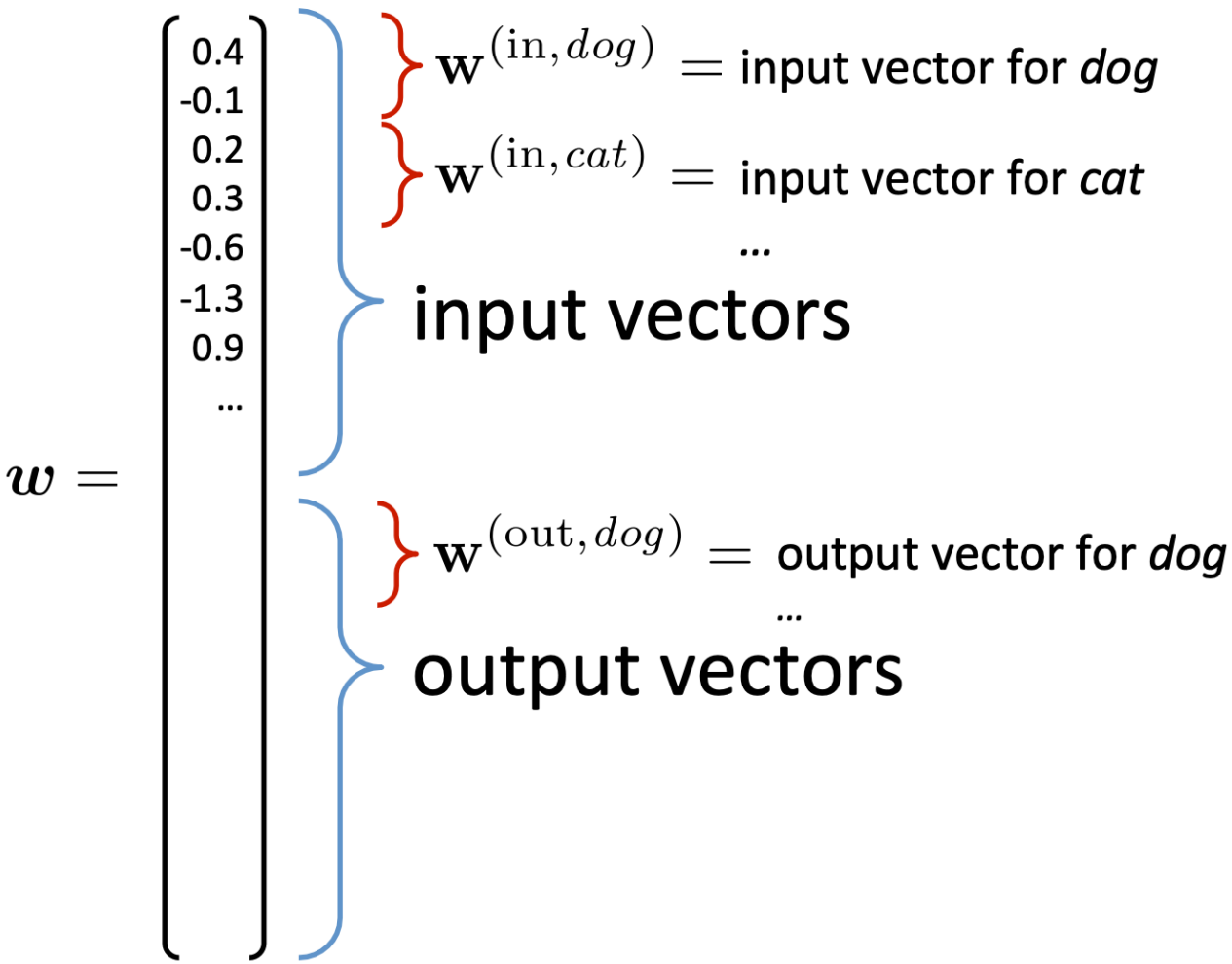
- How should we define the score function?

skip-gram score function: dot product



- Dot product of two vectors, one for each word
- Subtlety: different vector spaces for input and output
- No interpretation to vector dimensions (a **priori**)

skip-gram parameterization



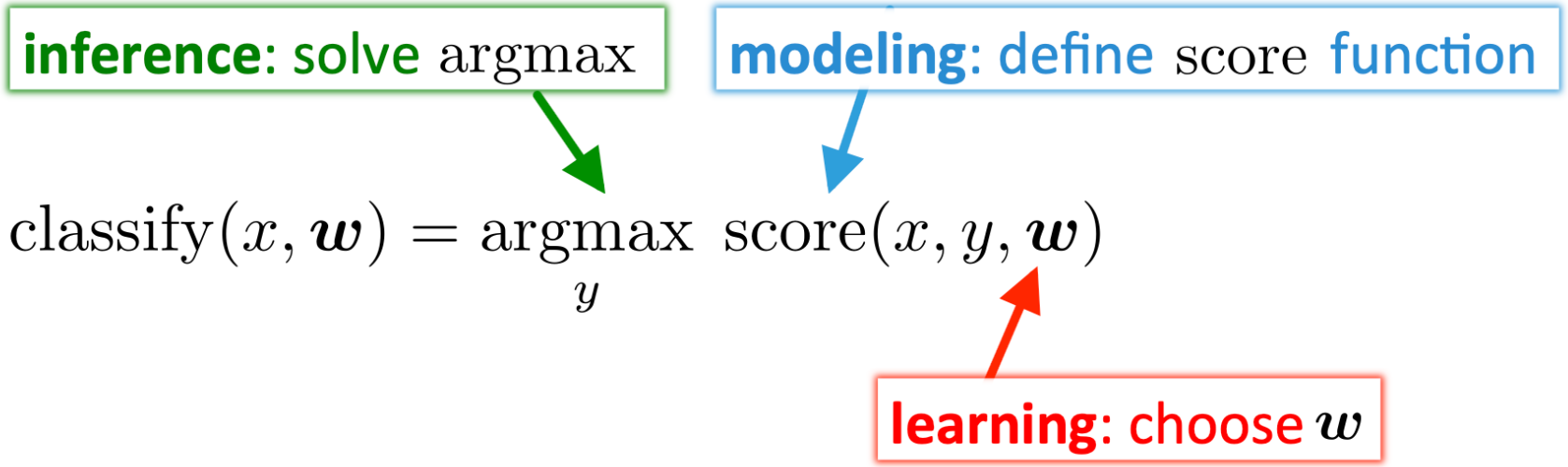
What will the skip-gram model learn?

- Corpus:
  an earthquake destroyed the city
  the town was destroyed by a tornado
- Sample of training pairs:

inputs (*x*)	outputs (*y*)
destroyed	earthquake
earthquake	destroyed
destroyed	tornado
tornado	destroyed
…	…

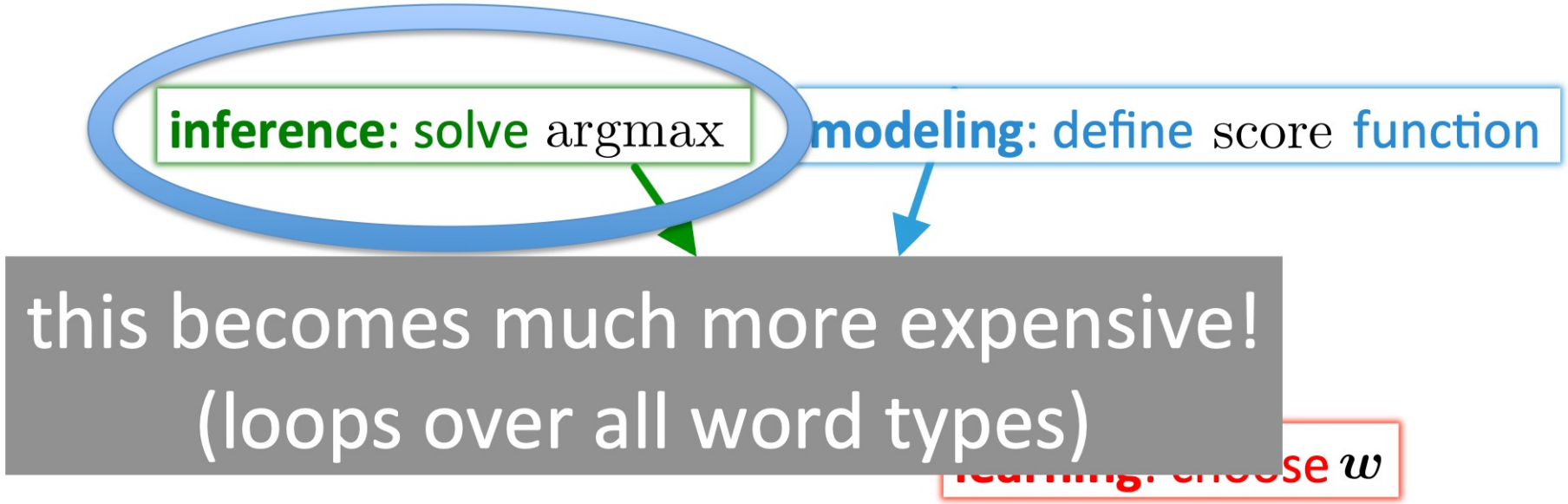
- Output vector for *destroyed* encouraged to be similar to input vectors of *earthquake* and *tornado*

Modeling, Inference, and Learning for Word Vectors



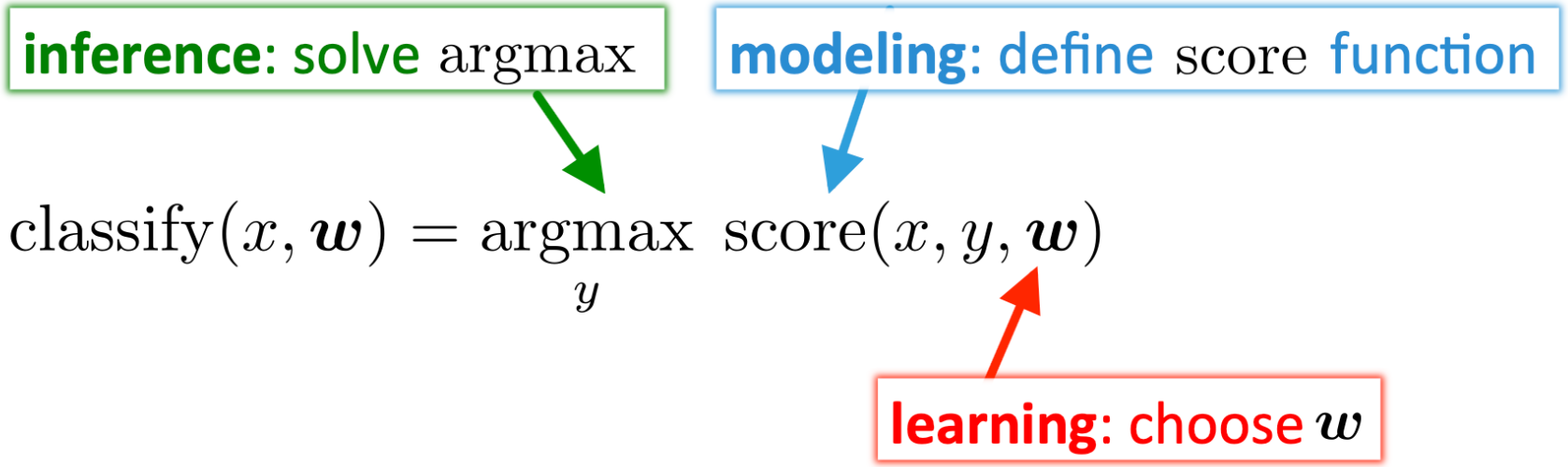
- **Inference**: How do we efficiently search over the space of all outputs?

Modeling, Inference, and Learning for Word Vectors



- **Inference**: How do we efficiently search over the space of all outputs?

Modeling, Inference, and Learning for Word Vectors



- **Learning**: How do we choose the weights w?

skip-gram

- Skip-gram objective: log loss

$$\min_{\boldsymbol{w}} \sum_{1 \le t \le |\mathcal{T}|} \sum_{-c \le j \le c, j \neq 0} -\log P_{\boldsymbol{w}}(x_{t+j} \mid x_t)$$

sum over positions in corpus

sum over context words in window

skip-gram

$$\min_{\boldsymbol{w}} \sum_{1 \le t \le |\mathcal{T}|} \sum_{-c \le j \le c, j \ne 0} -\log P_{\boldsymbol{w}}(x_{t+j} \mid x_t)$$

- From score to probability:

$$P_{\boldsymbol{w}}(y \mid x) \propto \exp\{\text{score}(x, y, \boldsymbol{w})\}$$

$$P_{\boldsymbol{w}}(y \mid x) \propto \exp\{\mathbf{w}^{(\text{in},x)} \cdot \mathbf{w}^{(\text{out},y)}\}$$

skip-gram

$$\min_{\boldsymbol{w}} \sum_{1 \le t \le |\mathcal{T}|} \sum_{-c \le j \le c, j \neq 0} -\log P_{\boldsymbol{w}}(x_{t+j} \mid x_t)$$

- Normalization requires sum over entire vocabulary:

$$P_{\boldsymbol{w}}(y \mid x) = \frac{\exp\{\mathbf{w}^{(\text{in},x)} \cdot \mathbf{w}^{(\text{out},y)}\}}{\sum_{y'} \exp\{\mathbf{w}^{(\text{in},x)} \cdot \mathbf{w}^{(\text{out},y')}\}}$$

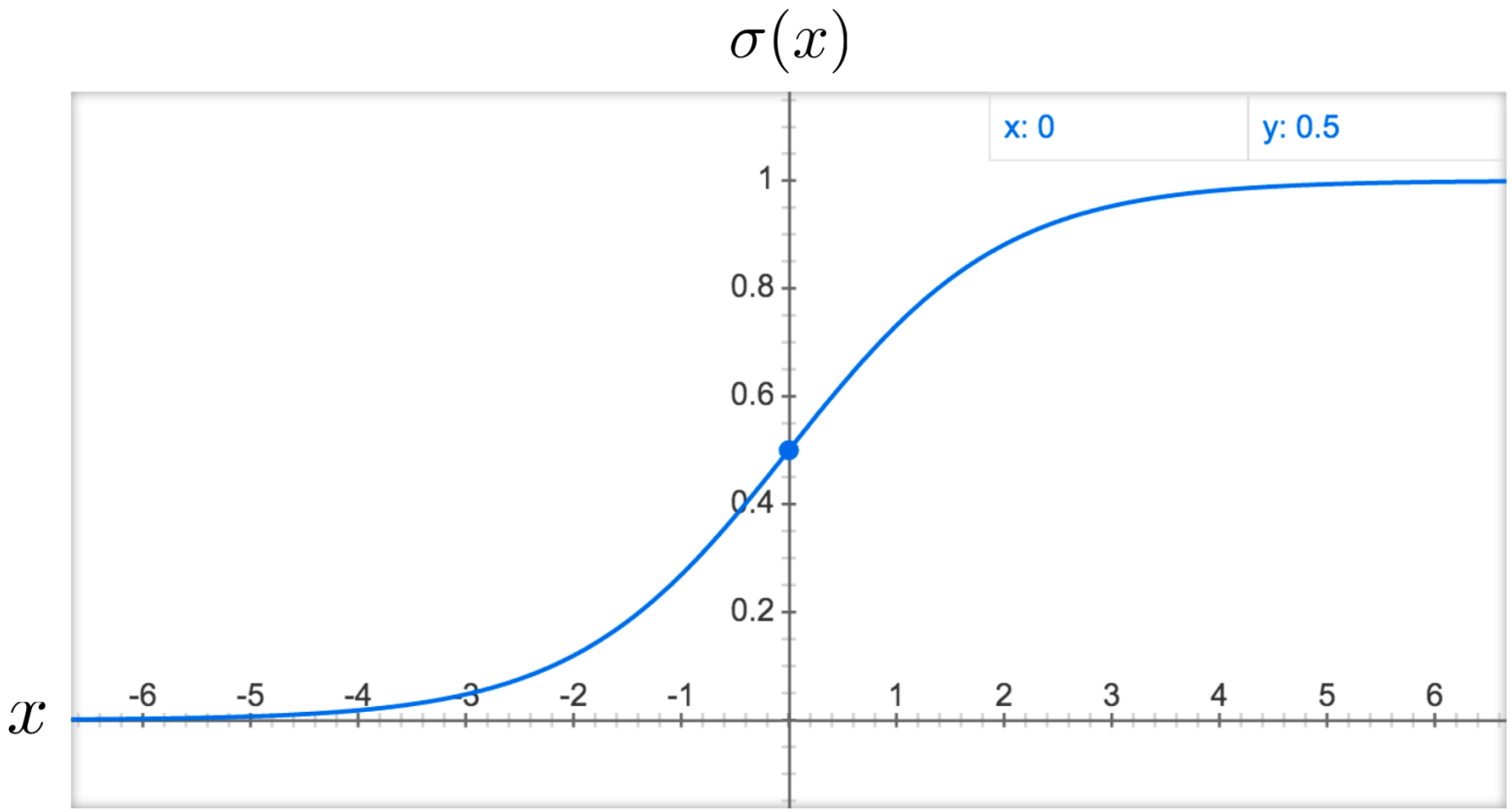
Negative Sampling (Mikolov et al., 2013)

- Rather than sum over entire vocabulary, generate samples and sum over them
- Main idea: train binary logistic regressions for a true pair (center word and a word in its context window) versus several "noise" pairs (the center word paired with a random word)

$$\min_{\boldsymbol{w}} \sum_{1 \leq t \leq |\mathcal{T}|} \sum_{-c \leq j \leq c, j \neq 0} -\log \sigma(\text{score}(x_t, x_{t+j}, \boldsymbol{w})) + \sum_{x \in \text{NEG}} \log \sigma(\text{score}(x_t, x, \boldsymbol{w}))$$

- Where sigma is logistic sigmoid function (see next slide)

(logistic) sigmoid: $\sigma(x) = \dfrac{1}{1 + \exp\{-x\}}$



- $\sigma(\text{score})$ often used to turn a score function into a probabilistic binary classifier, because its outputs range from 0 to 1

Negative Sampling (Mikolov et al., 2013)

$$\min_{\boldsymbol{w}} \sum_{1 \le t \le |\mathcal{T}|} \sum_{-c \le j \le c, j \ne 0} -\log \sigma(\text{score}(x_t, x_{t+j}, \boldsymbol{w})) + \sum_{x \in \text{NEG}} \log \sigma(\text{score}(x_t, x, \boldsymbol{w}))$$

- Maximize probability that real outside word appears;
- Minimize probability that random words appear around center word
- NEG contains 2-20 words sampled from some distribution
  - e.g., uniform, unigram, or smoothed unigram
  - smoothed: raise probabilities to power 3⁄4, renormalize to get a distribution
  - .g., sample with $P(w)=U(w)^{3/4}/Z$, the unigram distribution U(w) raised to the 3/4 power.
  - The power makes less frequent words be sampled more often

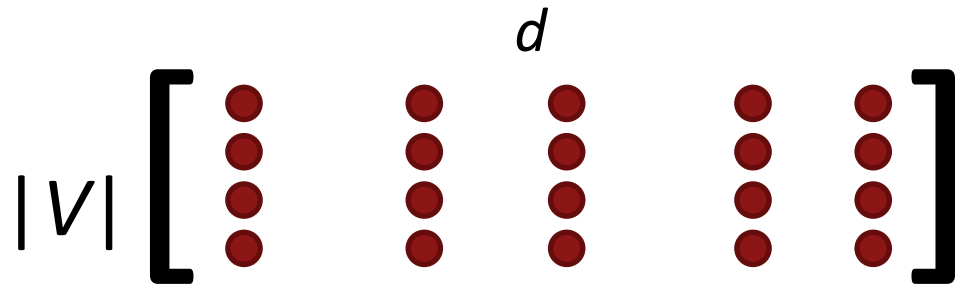
Stochastic gradients with negative sampling

- We iteratively take gradients at each window for SGD
- In each window, we only have at most *2m + 1* words plus *2km* negative words with negative sampling, so $\nabla_\theta J_t(\theta)$ is very sparse!

$$\nabla_\theta J_t(\theta) = \begin{bmatrix} 0 \\ \vdots \\ \nabla_{v_{like}} \\ \vdots \\ 0 \\ \nabla_{u_I} \\ \vdots \\ \nabla_{u_{learning}} \\ \vdots \end{bmatrix} \in \mathbb{R}^{2dV}$$

Stochastic gradients with negative sampling

- We might only update the word vectors that actually appear!
- Solution: either you need sparse matrix update operations to only update certain **rows** of full embedding matrices *U* and *V*, or you need to keep around a hash for word vectors



- If you have millions of word vectors and do distributed computing, it is important to not have to send gigantic updates around!

Two Ways to Represent Word Embeddings

- $\mathcal{V}$ = vocabulary, $|\mathcal{V}|$ = size of vocab
- 1: create $|\mathcal{V}|$-dimensional "one-hot" vector for each word, multiply by word embedding matrix:

$$emb(x) = \mathbf{W}\text{onehot}(\mathcal{V}, x)$$

- 2: store embeddings in a hash/dictionary data structure, do lookup to find embedding for word:

$$emb(x) = \text{lookup}(\mathbf{W}, x)$$

- These are equivalent, second can be much faster (though first can be fast if using sparse operations)

- We went through skip-gram in detail
- word2vec contains two models: skip-gram and continuous bag of words (CBOW)
- For CBOW: we can use the same loss and inference tricks as skip-gram, so we will just focus on the CBOW scoring function

word2vec Score Functions

- skip-gram:

$$\text{score}(x, y, \boldsymbol{w}) = \mathbf{w}^{(\text{in},x)} \cdot \mathbf{w}^{(\text{out},y)}$$

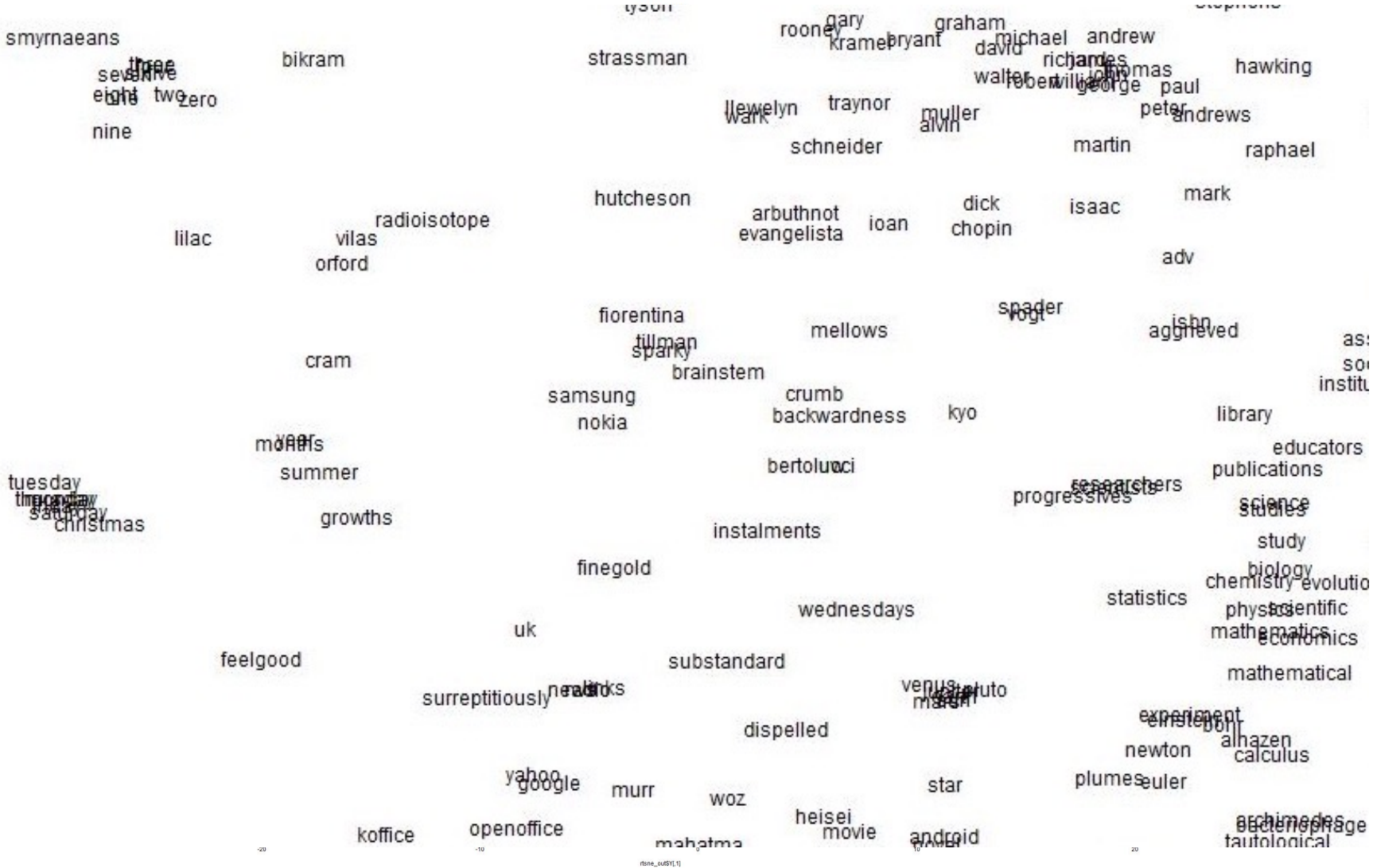
inputs (*x*)	outputs (*y*)
agriculture	<s>
agriculture	is
agriculture	the

- CBOW:

$$\text{score}(\boldsymbol{x}, y, \boldsymbol{w}) = \left(\frac{1}{|\boldsymbol{x}|} \sum_i \mathbf{w}^{(\text{in},x_i)} \right) \cdot \mathbf{w}^{(\text{out},y)}$$

inputs (*x*)	outputs (*y*)
{<s>, is, the, traditional}	agriculture
{<s>, agriculture, the, traditional}	is
{agriculture, is, traditional, mainstay}	the

Word2vec maximizes objective function by putting similar words nearby in space



word2vec

- word2vec toolkit implements training for skip- gram and CBOW models: https://code.google.com/archive/p/word2vec/
- Very fast to train, even on large corpora
- Pretrained embeddings available

A simple way to investigate the learned representations is to find the closest words for a user-specified word. The *distance* tool serves that purpose. For example, if you enter 'france', *distance* will display the most similar words and their distances to 'france', which should look like:

```
                Word       Cosine distance
------------------------------------------
               spain              0.678515
             belgium              0.665923
         netherlands              0.652428
               italy              0.633130
         switzerland              0.622323
          luxembourg              0.610033
            portugal              0.577154
              russia              0.571507
             germany              0.563291
           catalonia              0.534176
```