TTIC 31190: Natural Language Processing

Kevin Gimpel
Spring 2018

Lecture 8: Neural Language Models and Word Embeddings

Other Naturally-Occurring Data

- quality of scientific journalism:

What Makes Writing Great? First Experiments on Article Quality Prediction in the Science Journalism Domain

Annie Louis
University of Pennsylvania
Philadelphia, PA 19104
lannie@seas.upenn.edu

Ani Nenkova
University of Pennsylvania
Philadelphia, PA 19104
nenkova@seas.upenn.edu

Abstract

Great writing is rare and highly admired. Readers seek out articles that are beautifully written, informative and entertaining. Yet information-access technologies lack capabilities for predicting article quality at this level. In this paper we present first experiments on article quality prediction in the science journalism domain. We introduce a corpus of great pieces of science journalism, along with typical articles from the genre. We imple-

done before. The fawn, known as Dewey, was developing normally and seemed to be healthy. He had no mother, just a surrogate who had carried his fetus to term. He had no father, just a "donor" of all his chromosomes. He was the genetic duplicate of a certain trophy buck out of south Texas whose skin cells had been cultured in a laboratory. One of those cells furnished a nucleus that, transplanted and rejiggered, became the DNA core of an egg cell, which became an embryo, which in time became Dewey. So he was wildlife, in a sense, and in another sense elaborately synthetic. This is the sort of news

Other Naturally-Occurring Data

- memorability of quotations:

You had me at hello: How phrasing affects memorability

Cristian Danescu-Niculescu-Mizil **Justin Cheng** **Jon Kleinberg** **Lillian Lee**
Department of Computer Science
Cornell University
cristian@cs.cornell.edu, jc882@cornell.edu, kleinber@cs.cornell.edu, llee@cs.cornell.edu

Abstract

Understanding the ways in which information achieves widespread public awareness is a research question of significant interest. We consider whether, and how, the way in which the information is phrased — the choice of words and sentence structure — can affect this process. To this end, we develop an analysis framework and build a corpus of movie quotes, annotated with memorability information, in which we are able to control for both the speaker and the setting of the quotes.

Building on a foundation in the sociology of diffusion [27, 31], researchers have explored the ways in which network structure affects the way information spreads, with domains of interest including blogs [1, 11], email [37], on-line commerce [22], and social media [2, 28, 33, 38]. There has also been recent research addressing temporal aspects of how different media sources convey information [23, 30, 39] and ways in which people react differently to information on different topics [28, 36].

Beyond all these factors, however, one's everyday

Other Naturally-Occurring Data

- satire detection (legitimate news outlets vs. The Onion or other satirical sites):

Automatic Satire Detection: Are You Having a Laugh?

Clint Burfoot
CSSE
University of Melbourne
VIC 3010 Australia
cburfoot@csse.unimelb.edu.au

Timothy Baldwin
CSSE
University of Melbourne
VIC 3010 Australia
tim@csse.unimelb.edu.au

Abstract

We introduce the novel task of determining whether a newswire article is "true" or satirical. We experiment with SVMs, feature scaling, and a number of lexical and semantic feature types, and achieve promising results over the task.

Satire classification is a novel task to computational linguistics. It is somewhat similar to the more widely-researched text classification tasks of spam filtering (Androutsopoulos et al., 2000) and sentiment classification (Pang and Lee, 2008), in that: (a) it is a binary classification task, and (b) it is an intrinsically semantic task, i.e. satire news articles are recognisable as such through interpretation and cross-comparison to world knowledge

Other Naturally-Occurring Data

- predicting novel success from text of novels:

Success with Style: Using Writing Style to Predict the Success of Novels

Vikas Ganjigunte Ashok **Song Feng** **Yejin Choi**
Department of Computer Science
Stony Brook University
Stony Brook, NY 11794-4400
vganjiguntea, songfeng, ychoi@cs.stonybrook.edu

Abstract

Predicting the success of literary works is a curious question among publishers and aspiring writers alike. We examine the quantitative connection, if any, between *writing style* and successful literature. Based on novels over several different genres, we probe the predictive power of statistical stylometry in discriminating successful literary works, and identify characteristic stylistic elements that are more prominent in successful writings. Our study

fore they are picked up by a publisher.[1]

Perhaps due to its obvious complexity of the problem, there has been little previous work that attempts to build statistical models that predict the success of literary works based on their intrinsic content and quality. Some previous studies do touch on the notion of stylistic aspects in successful literature, e.g., extensive studies in Literature discuss literary styles of significant authors (e.g., Ellegård (1962), McGann (1998)), while others consider content characteristics such as plots, characteristics of charac-

- I posted some hints for assignment 2

Roadmap

- words, morphology, lexical semantics
- text classification
- language modeling
- word embeddings
- recurrent/recursive/convolutional networks in NLP
- sequence labeling, HMMs, dynamic programming
- syntax and syntactic parsing
- semantics, compositionality, semantic parsing
- machine translation and other NLP tasks

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)**

J&M/SLP3

Probability -> Perplexity

- average log-probability of held-out words:

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

- perplexity:

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

Perplexity as branching factor

- given a sentence consisting of random digits
- perplexity of this sentence under a model that gives probability 1/10 to each digit?

$$\ell = \frac{1}{M} \log_2 P(w_1, w_2, ..., w_M)$$
$$= \frac{1}{M} \log_2 \prod_{i=1}^{M} \frac{1}{10}$$

Perplexity as branching factor

- given a sentence consisting of random digits
- perplexity of this sentence under a model that gives probability 1/10 to each digit?

$$\begin{aligned}\ell &= \frac{1}{M}\log_2 P(w_1, w_2, ..., w_M) \\ &= \frac{1}{M}\log_2 \prod_{i=1}^{M}\frac{1}{10} \\ &= \frac{1}{M}\log_2 \left(\frac{1}{10}\right)^M \\ &= \frac{1}{M} M \log_2 \frac{1}{10}\end{aligned}$$

$$\mathrm{PP} = 2^{-\ell} = 10$$

Lower perplexity = better model

- train: 38 million words
- test: 1.5 million words

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

"Add-1" estimation

- just add 1 to all counts
- MLE estimate:

$$P_{\mathrm{MLE}}(w_i \mid w_{i-1}) = \frac{\mathrm{count}(w_{i-1}, w_i)}{\mathrm{count}(w_{i-1})}$$

- Add-1 estimate:

$$P_{\mathrm{add-1}}(w_i \mid w_{i-1}) = \frac{\mathrm{count}(w_{i-1}, w_i) + 1}{\mathrm{count}(w_{i-1}) + |\mathcal{V}|}$$

J&M/SLP3

Absolute Discounting

Bigram count in training set	Bigram count in heldout set
0	0.0000270
1	0.448
2	1.25
3	2.24
4	3.23
5	4.21
6	5.23
7	6.21
8	7.21
9	8.26

Figure 4.8 For all bigrams in 22 million words of AP newswire of count 0, 1, 2,...,9, the counts of these bigrams in a held-out corpus also of 22 million words.

Absolute Discounting

Bigram count in training set	Bigram count in heldout set
0	0.0000270
1	0.448
2	1.25
3	2.24
4	3.23
5	4.21
6	5.23
7	6.21
8	7.21

observed bigrams have counts that are **overestimated**

unobserved bigrams have counts that are **underestimated**

J&M/SLP3

Absolute Discounting

- subtract *d* from each numerator count
- use the original counts for the denominator

$$P_{\mathrm{AbsDisc}}(w \mid w') = \frac{\max(0, \mathrm{count}(w', w) - d)}{\sum_v \mathrm{count}(w', v)} + \lambda(w')P(w)$$

- so there's some "missing probability mass"
- lambda function is defined to make things normalize correctly

J&M/SLP3

Kneser-Ney Smoothing

- Shannon game: *I can't see without my reading*______________?
  - "*Francisco*" is more common than "*glasses*"
  - … but "*Francisco*" always follows "*San*"
- unigram is most useful when we haven't seen bigram!
- so instead of unigram $P(w)$ ("How likely is w?")
- use $P_{\text{continuation}}(w)$ ("How likely is w to appear as a novel continuation?")

J&M/SLP3

Kneser-Ney Smoothing

- how many times is *w* a novel continuation?

$$P_{\text{continuation}}(w) \propto \underbrace{|\{w' : \text{count}(w', w) > 0\}|}_{\text{number of unique words that appeared before } w}$$

Kneser-Ney Smoothing

- how many times is *w* a novel continuation?

$$P_{\text{continuation}}(w) \propto |\{w' : \text{count}(w', w) > 0\}|$$

- normalize by total number of word bigram types:

$$P_{\text{continuation}}(w) = \frac{|\{w' : \text{count}(w', w) > 0\}|}{|\{\langle w', w'' \rangle : \text{count}(w', w'') > 0\}|}$$

Kneser-Ney Smoothing

- Interpolated Kneser-Ney:

$$P_{\mathrm{KN}}(w \mid w') = \frac{\max(0, \mathrm{count}(w', w) - d)}{\sum_v \mathrm{count}(w', v)} + \lambda(w') P_{\mathrm{continuation}}(w)$$

- again, lambda function is defined to make things normalize correctly

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:

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

Classifier Framework

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

- linear model score function:

$$\text{score}(\boldsymbol{x}, y, \mathbf{w}) = \mathbf{w}^\top \mathbf{f}(\boldsymbol{x}, y) = \sum_i w_i f_i(\boldsymbol{x}, y)$$

- we can also use a neural network for the score function!

neural layer = affine transform + nonlinearity

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

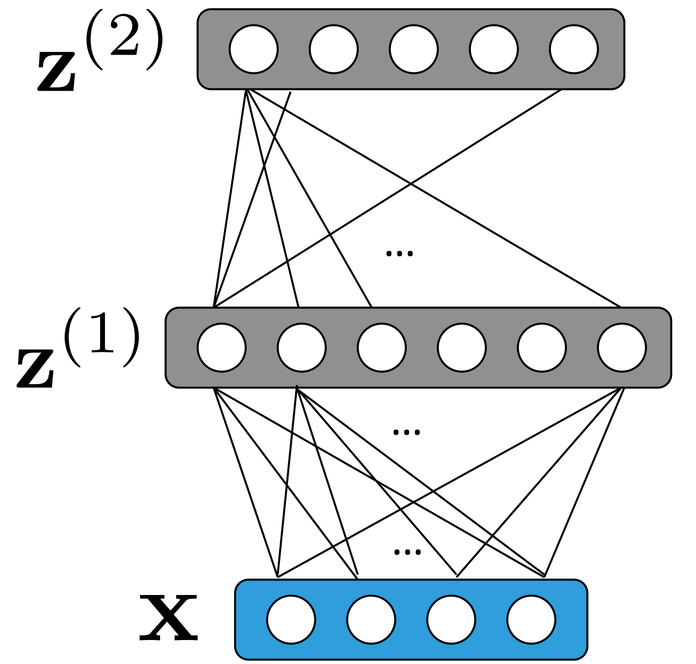
- this is a single "layer" of a neural network
- input vector is $\mathbf{x}$
- $\mathbf{U}^{(0)}$ and $\mathbf{b}^{(0)}$ are parameters

Neural Networks

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

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

$$\cdots$$



- 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

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

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

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

Learning with Neural Networks

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

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

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

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

- 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

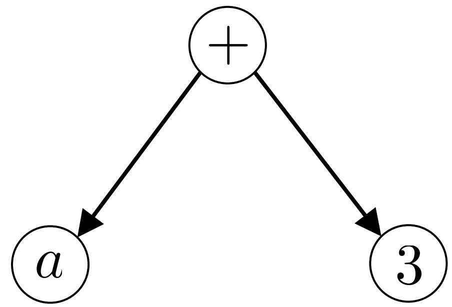
Computation Graphs

- 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

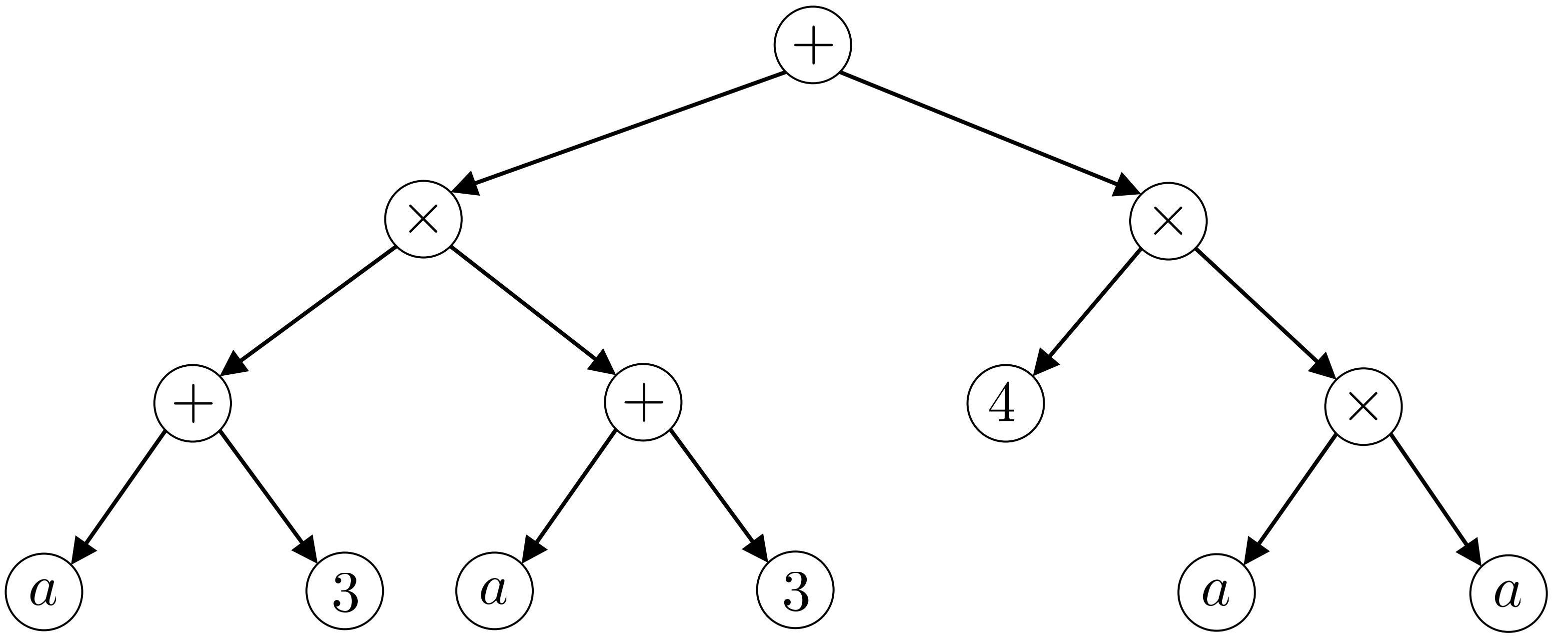
- backpropagation has become associated with neural networks, but it’s much more general
- I also use backpropagation to compute gradients in **linear** models for structured prediction

A simple computation graph:



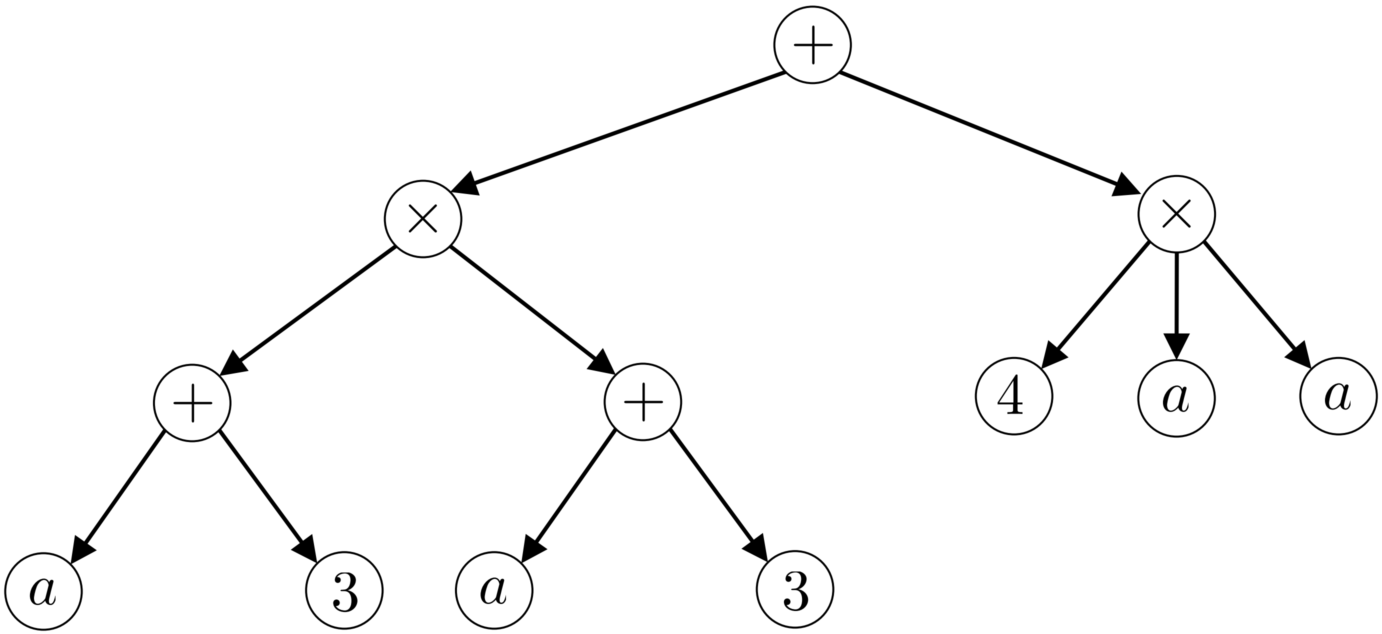
- represents expression “a + 3”

A slightly bigger computation graph:

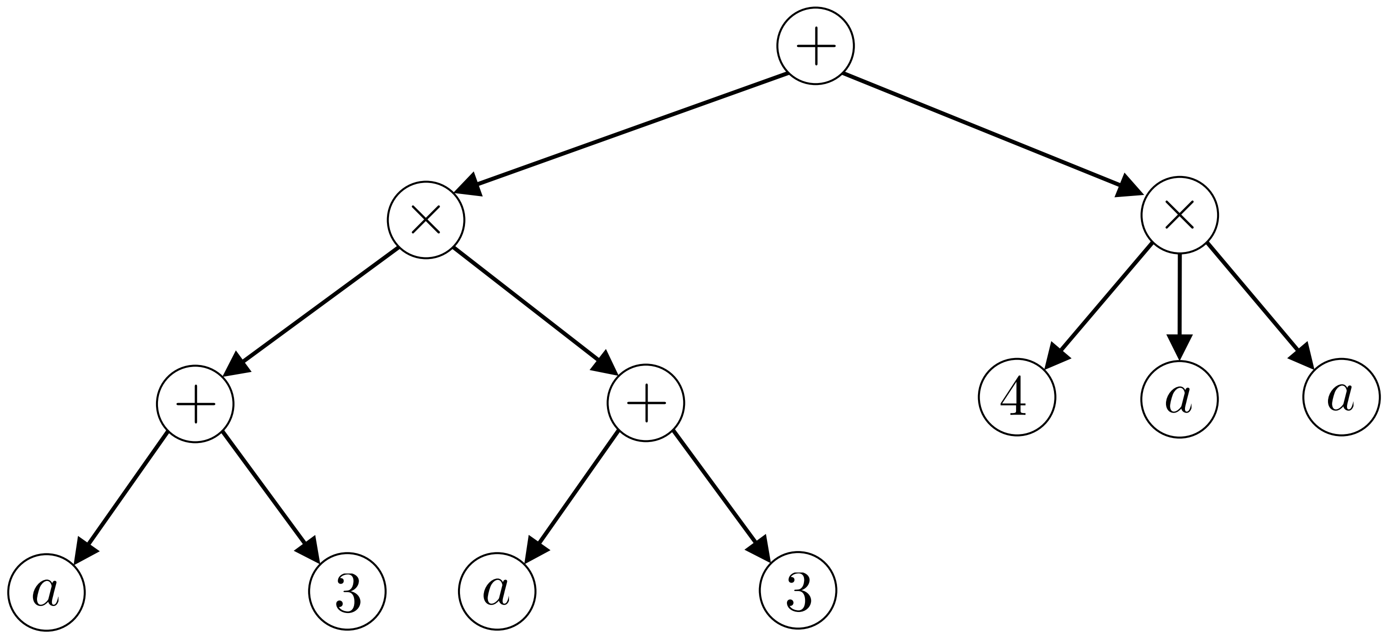


- represents expression “(a + 3)^2 + 4a^2”

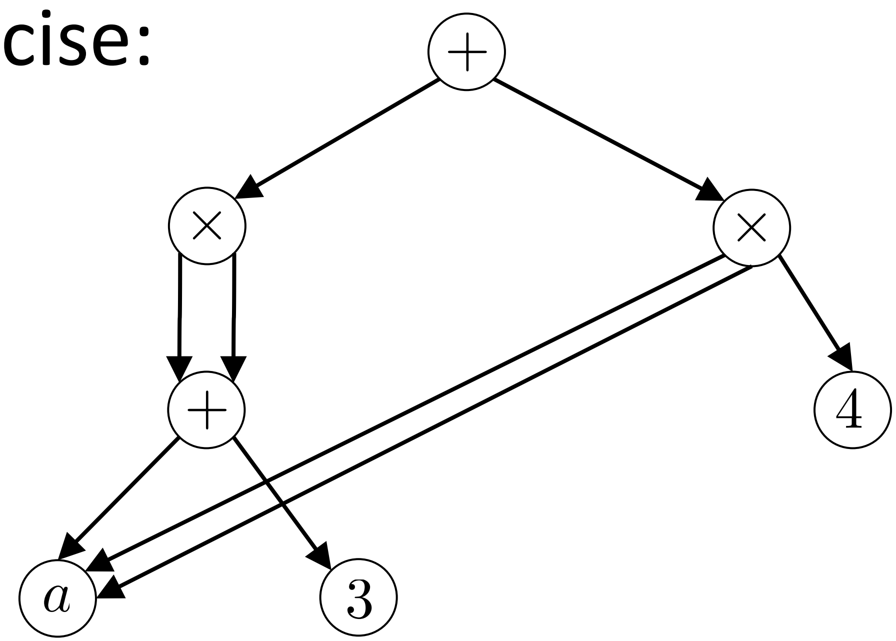
Operators can have more than 2 operands:



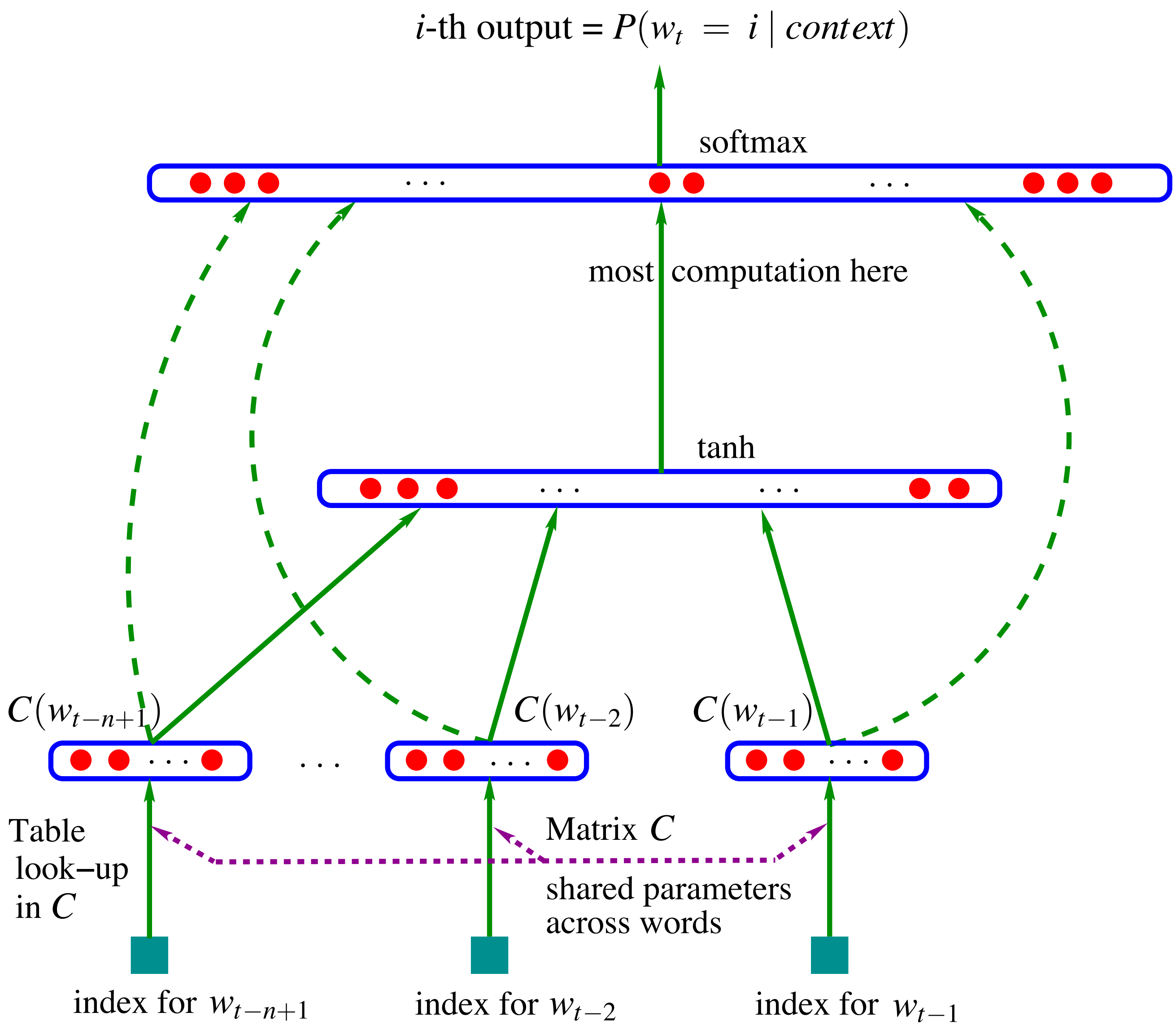
- still represents expression "$(a + 3)^2 + 4a^2$"



- more concise:



Model (Bengio et al., 2003)

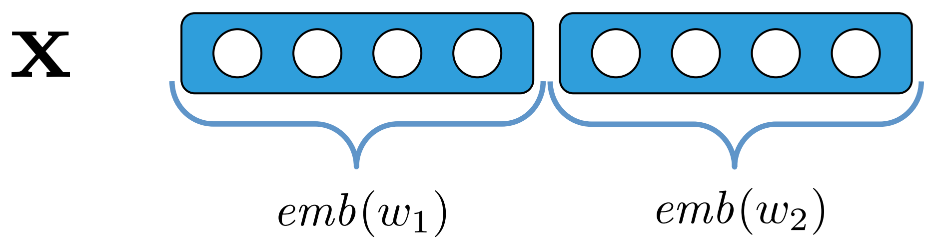


A Simple Neural Trigram Language Model

- given previous words w_1 and w_2, predict next word

A Simple Neural Trigram Language Model

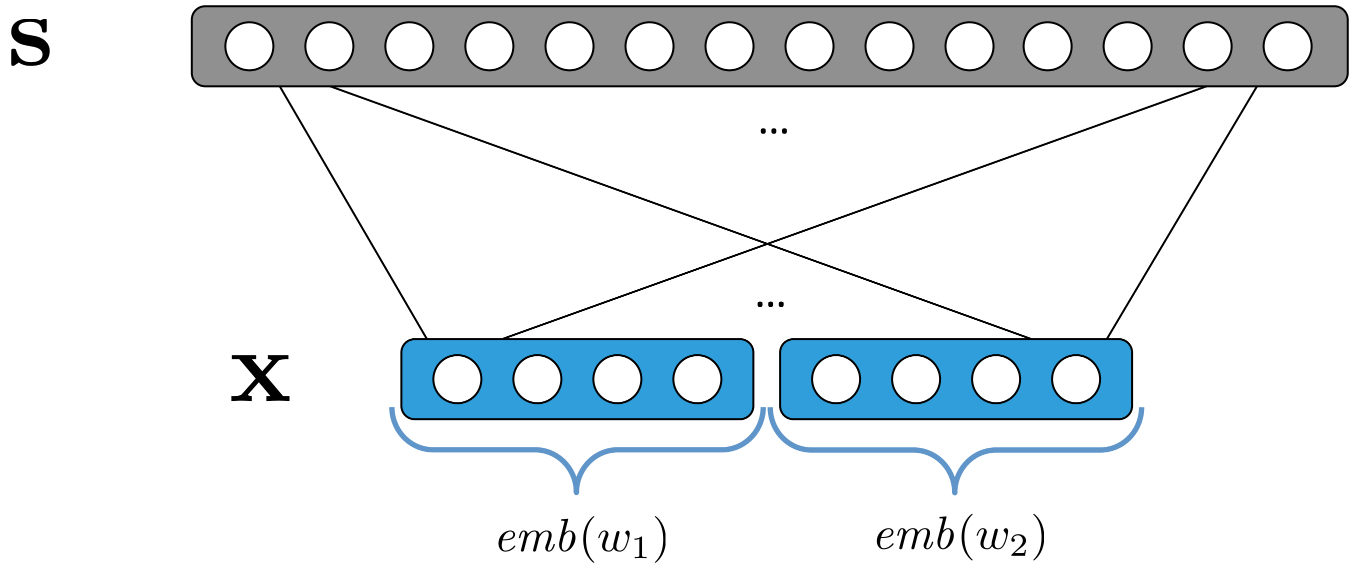
- given previous words w_1 and w_2, predict next word
- input is concatenation of vectors (embeddings) of previous words:



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

A Simple Neural Trigram Language Model

- output vector contains scores of possible next words:



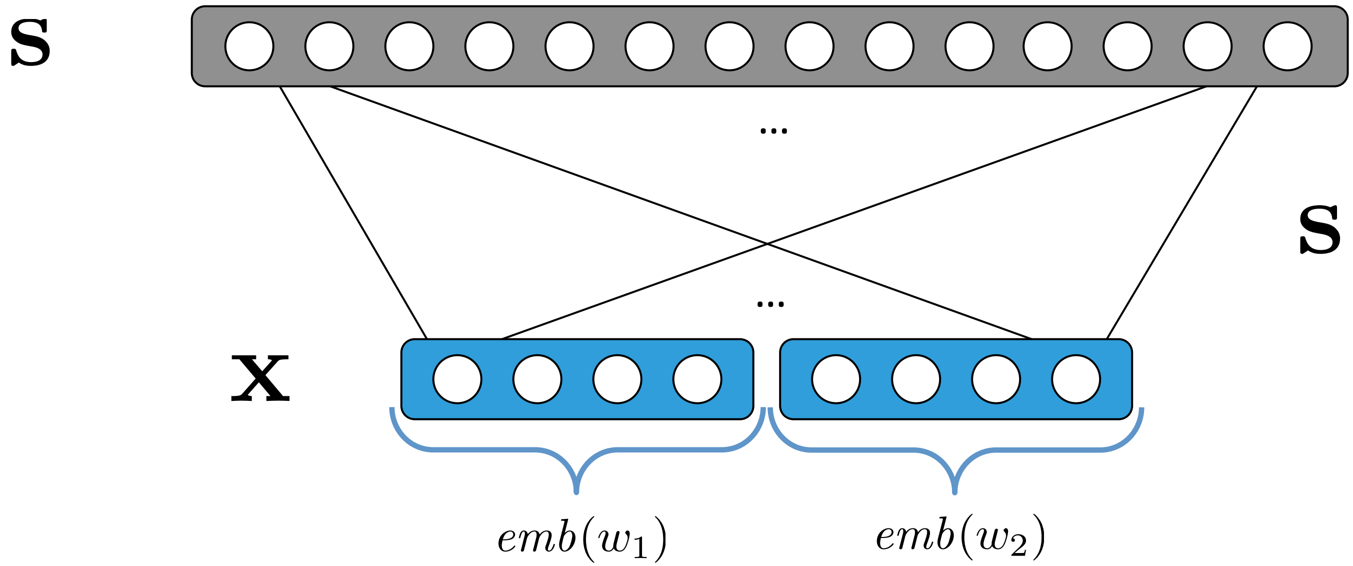
$$\mathbf{s} = \mathbf{U}\mathbf{x}$$

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

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

A Simple Neural Trigram Language Model

- output vector contains scores of possible next words:



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

- dimensionalities?

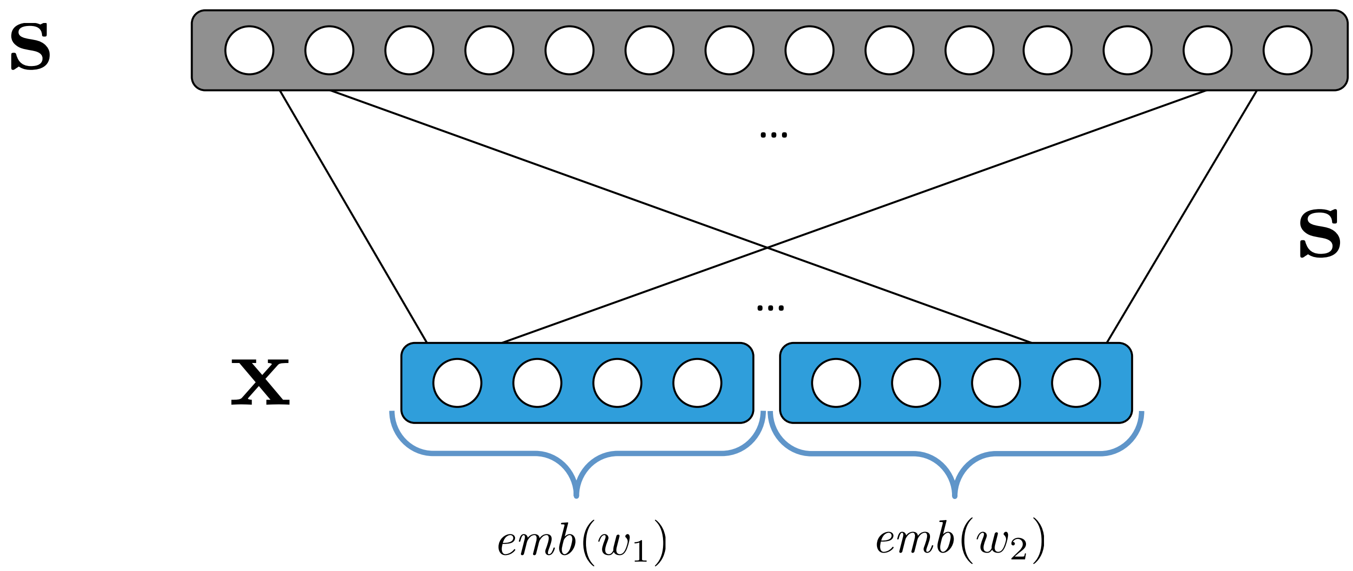
$$emb(w) \in \mathbb{R}^d$$

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

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

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

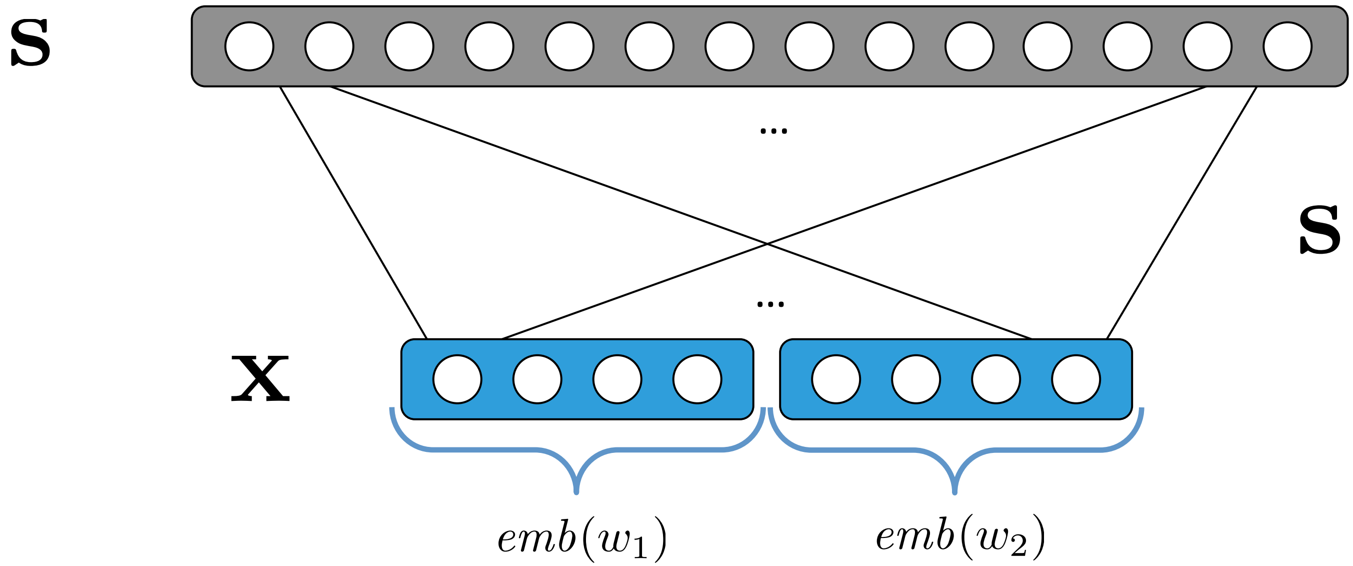
A Simple Neural Trigram Language Model



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

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

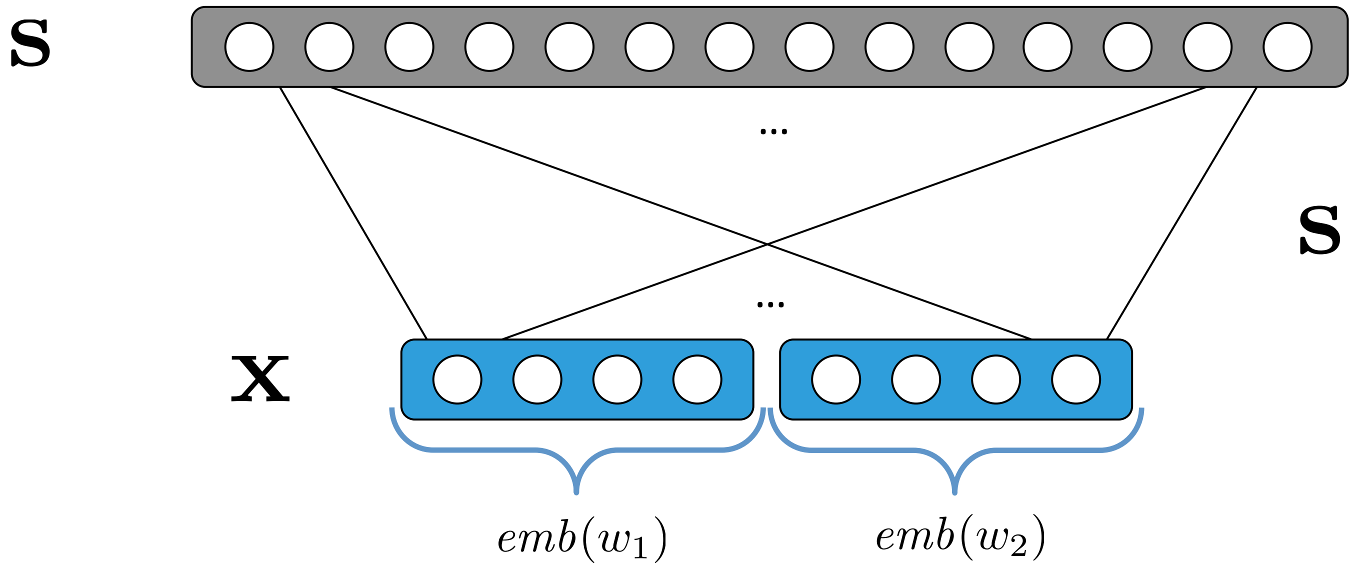
A Simple Neural Trigram Language Model



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

- how should we train this model?
- we have lots of training examples (just collect trigrams)
- we can use any of our classification losses!

A Simple Neural Trigram Language Model



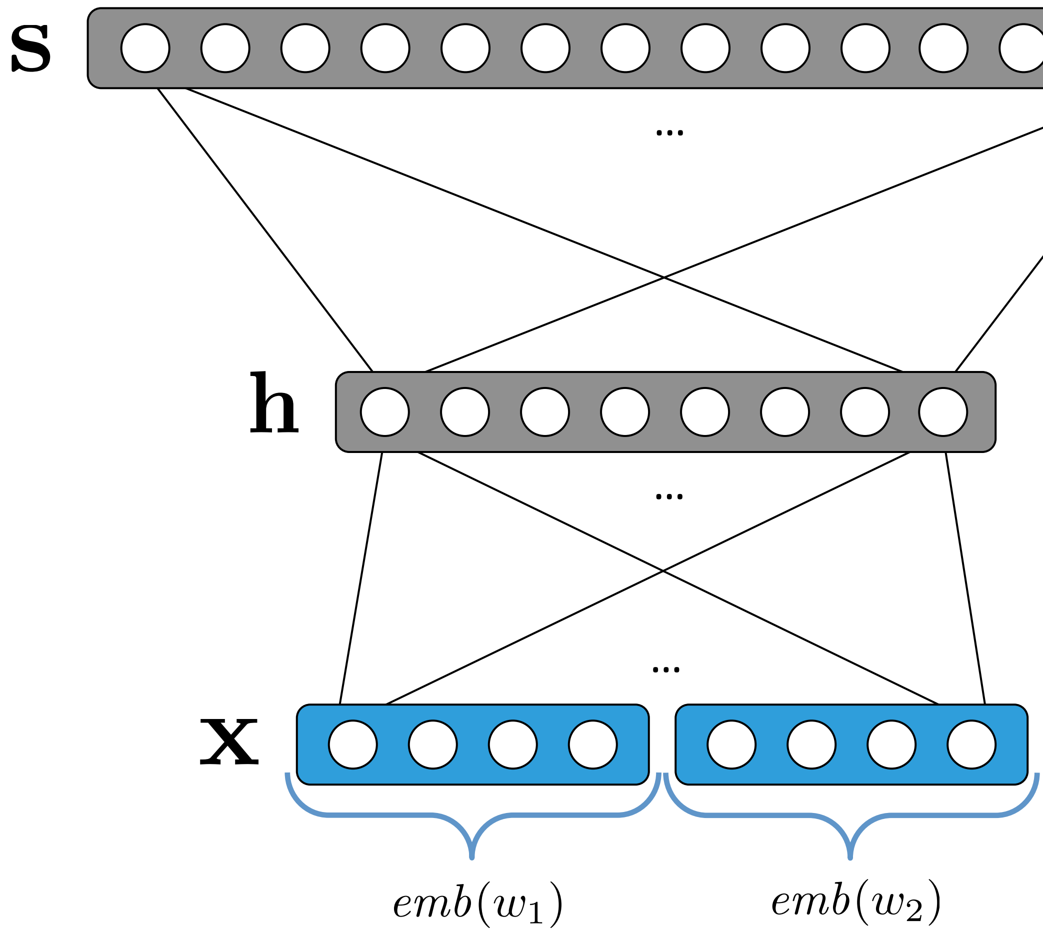
$$\mathbf{s} = \mathbf{U}\mathbf{x}$$

- most common way: log loss

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

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

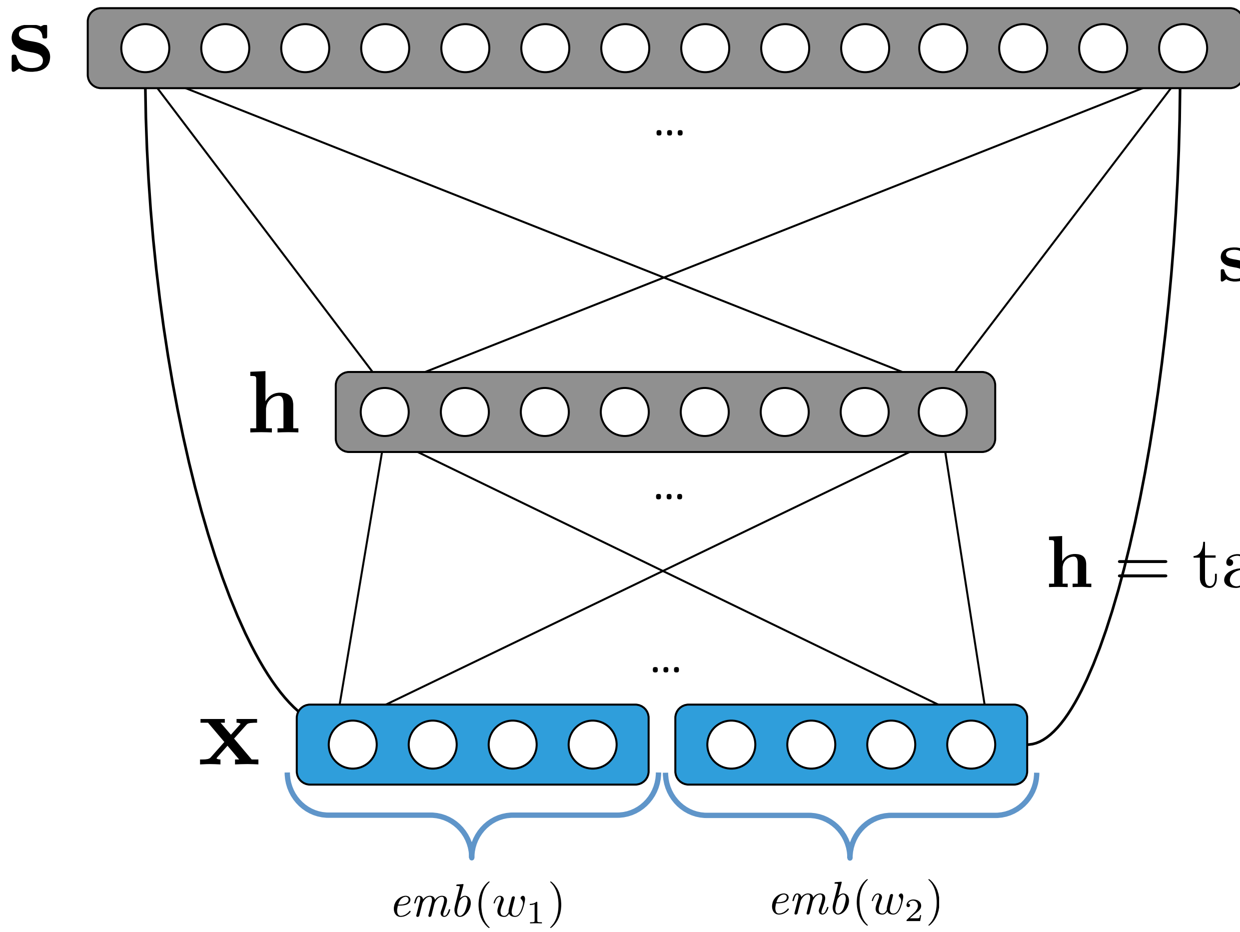
Adding a Hidden Layer



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

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

Adding Connections



$$\mathbf{s} = \mathbf{U}^{(1)}\mathbf{h} + \mathbf{U}^{(2)}\mathbf{x}$$

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

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.

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**

- 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

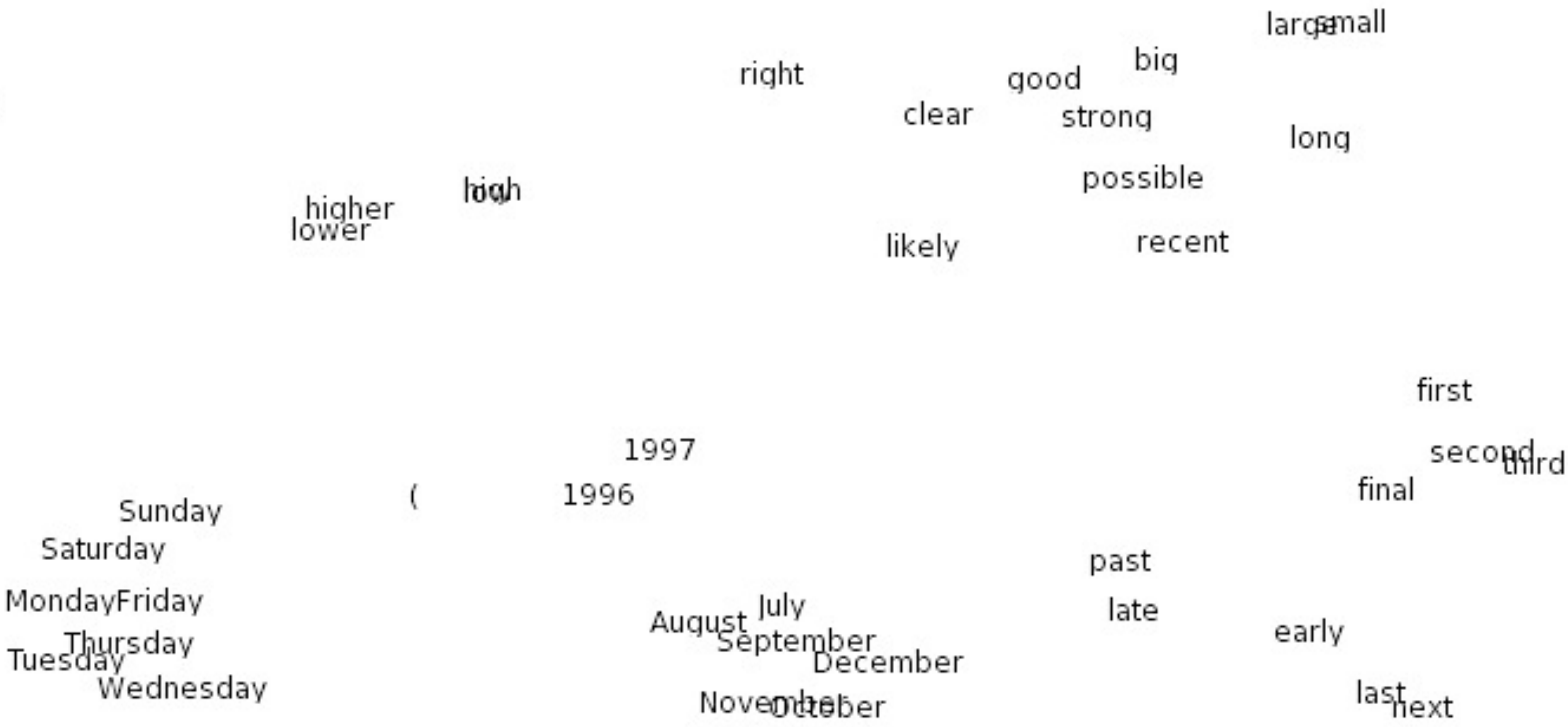
Experiments

	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.

Word Embeddings



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)

- word embedding nearest neighbors:

FRANCE 454	JESUS 1973	XBOX 6909	REDDISH 11724	SCRATCHED 29869	MEGABITS 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)

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)

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
…	…