TTIC 31190: Natural Language Processing

Kevin Gimpel

Spring 2018

Lecture 9: Word Embeddings

Assignment 1

Assignment 2 due in one week

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

Classifier Framework

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

- linear model score function:

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

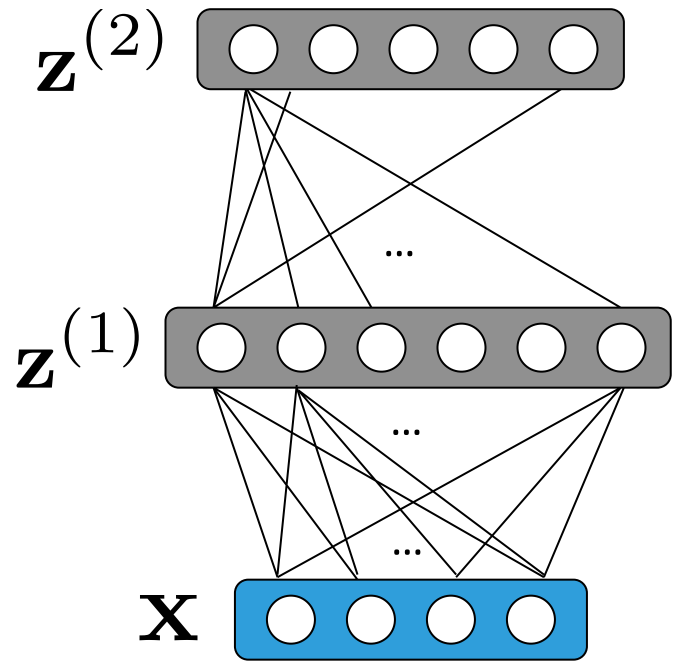
- we can also use a neural network for the score function!

Neural Networks

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

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

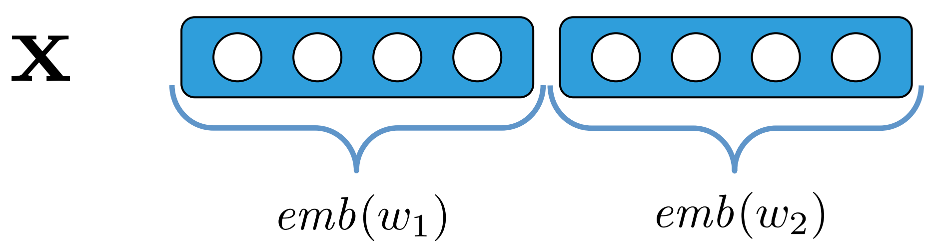
$$\ldots$$



- use output of one layer as input to next
- “feed-forward” and/or “fully-connected” layers

A Simple Neural Trigram Language Model

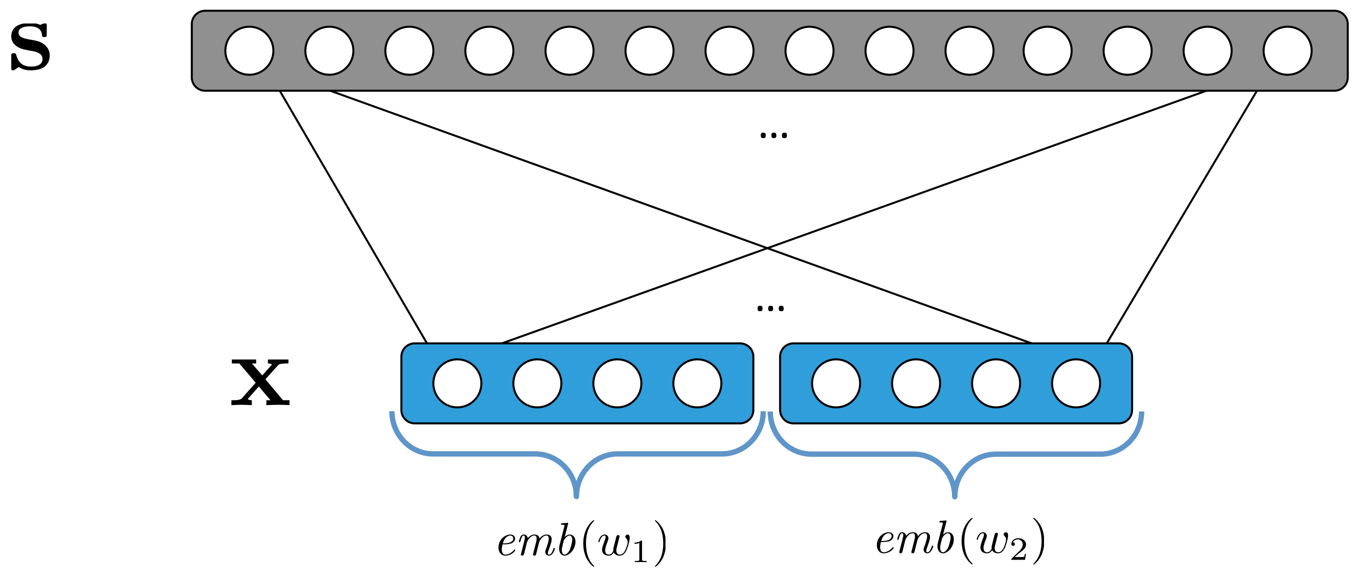
- given previous words w_1 and w_2, predict next word
- input is concatenation of vectors (embeddings) of previous words:



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

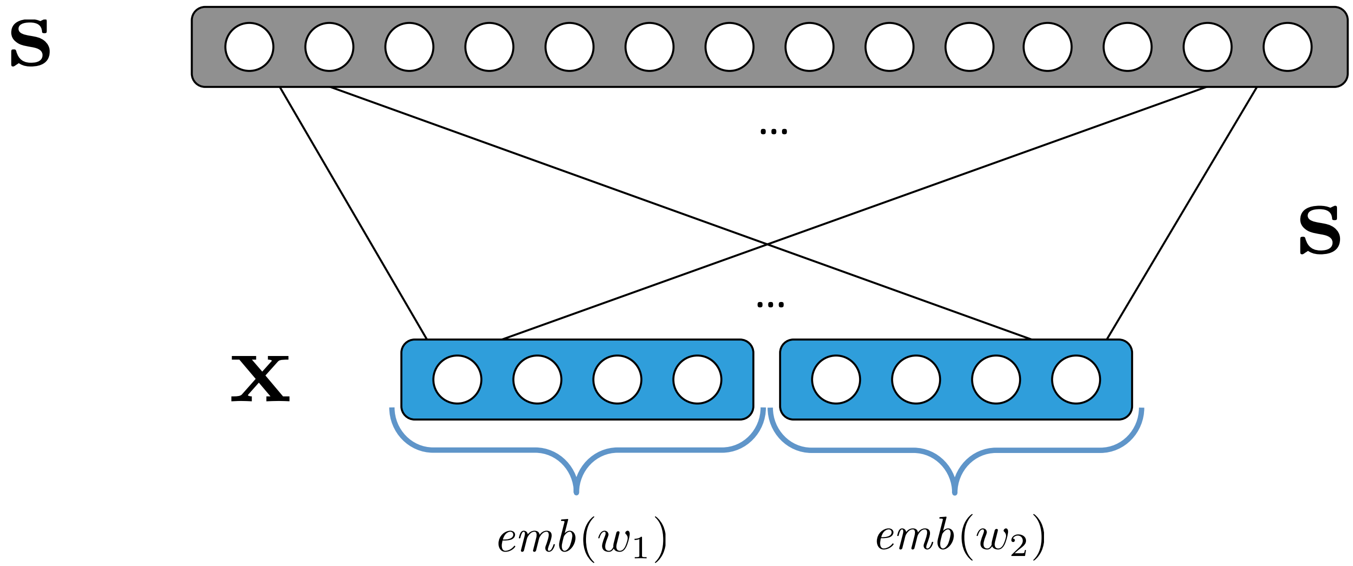
A Simple Neural Trigram Language Model

- output vector contains scores of possible next words:



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

A Simple Neural Trigram Language Model



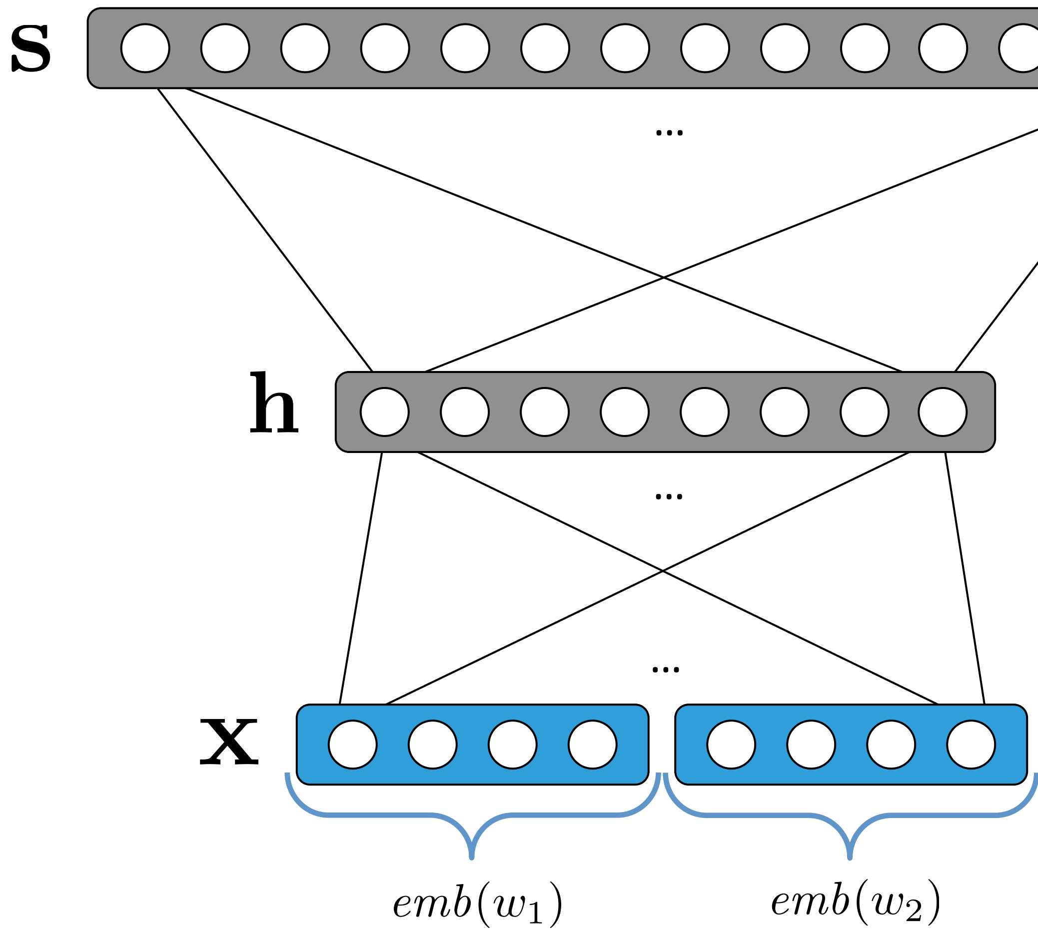
$$\mathbf{s} = \mathbf{U}\mathbf{x}$$

- training: log loss

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

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

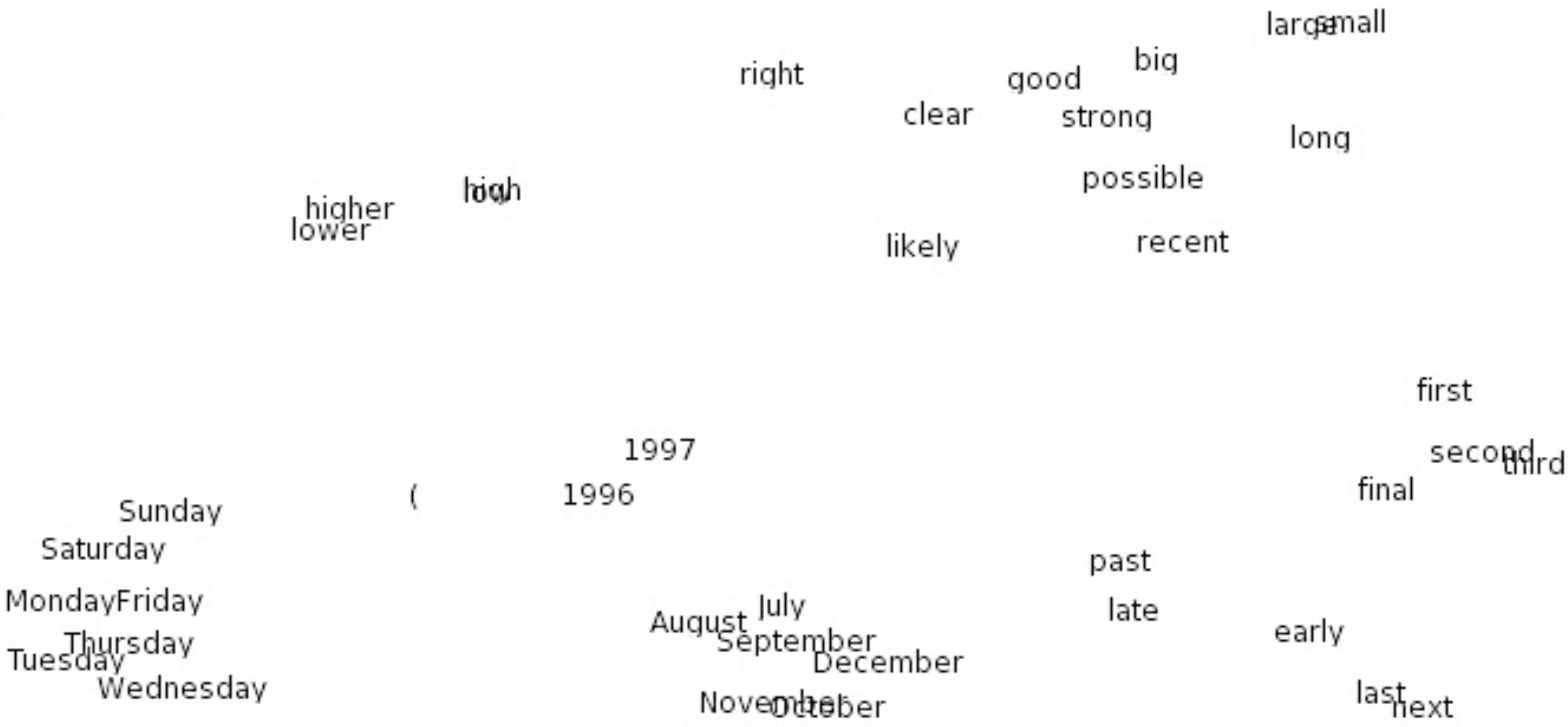
Adding a Hidden Layer



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

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

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

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
…	…

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

vector for word *x* as an input

vector for word *y* as an output

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

- 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

$$\boldsymbol{w} = \begin{bmatrix} 0.4 \\ -0.1 \\ 0.2 \\ 0.3 \\ -0.6 \\ -1.3 \\ 0.9 \\ \dots \end{bmatrix}$$

$\mathbf{w}^{(\mathrm{in}, dog)} =$ input vector for *dog*

$\mathbf{w}^{(\mathrm{in}, cat)} =$ input vector for *cat*

…

input vectors

$\mathbf{w}^{(\mathrm{out}, dog)} =$ output vector for *dog*

…

output vectors

skip-gram score function

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

- why use different vector spaces for input and output?
- also, what should we use as our final word embeddings?

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: solve argmax

modeling: define score function

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

learning: choose $\boldsymbol{w}$

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

Modeling, Inference, and Learning for Word Vectors

inference: solve argmax

modeling: define score function

this becomes much more expensive!
(loops over all word types)

learning: choose $\boldsymbol{w}$

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

Modeling, Inference, and Learning for Word Vectors

inference: solve argmax

modeling: define score function

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

learning: choose $\boldsymbol{w}$

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

skip-gram

- skip-gram objective: log loss

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

sum over positions in corpus

sum over context words in window

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

from score to probability:

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

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

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

normalization requires sum over what?

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

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

normalization requires sum over entire vocabulary:

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

Hierarchical Softmax
(Morin and Bengio, 2005)

- based on a new generative story for the probability $P_{\boldsymbol{w}}(y \mid x)$
- but the generative story is so simple!
  – just draw from the conditional distribution
- how can we make it more efficient?
  – see paper or advanced NLP course for details

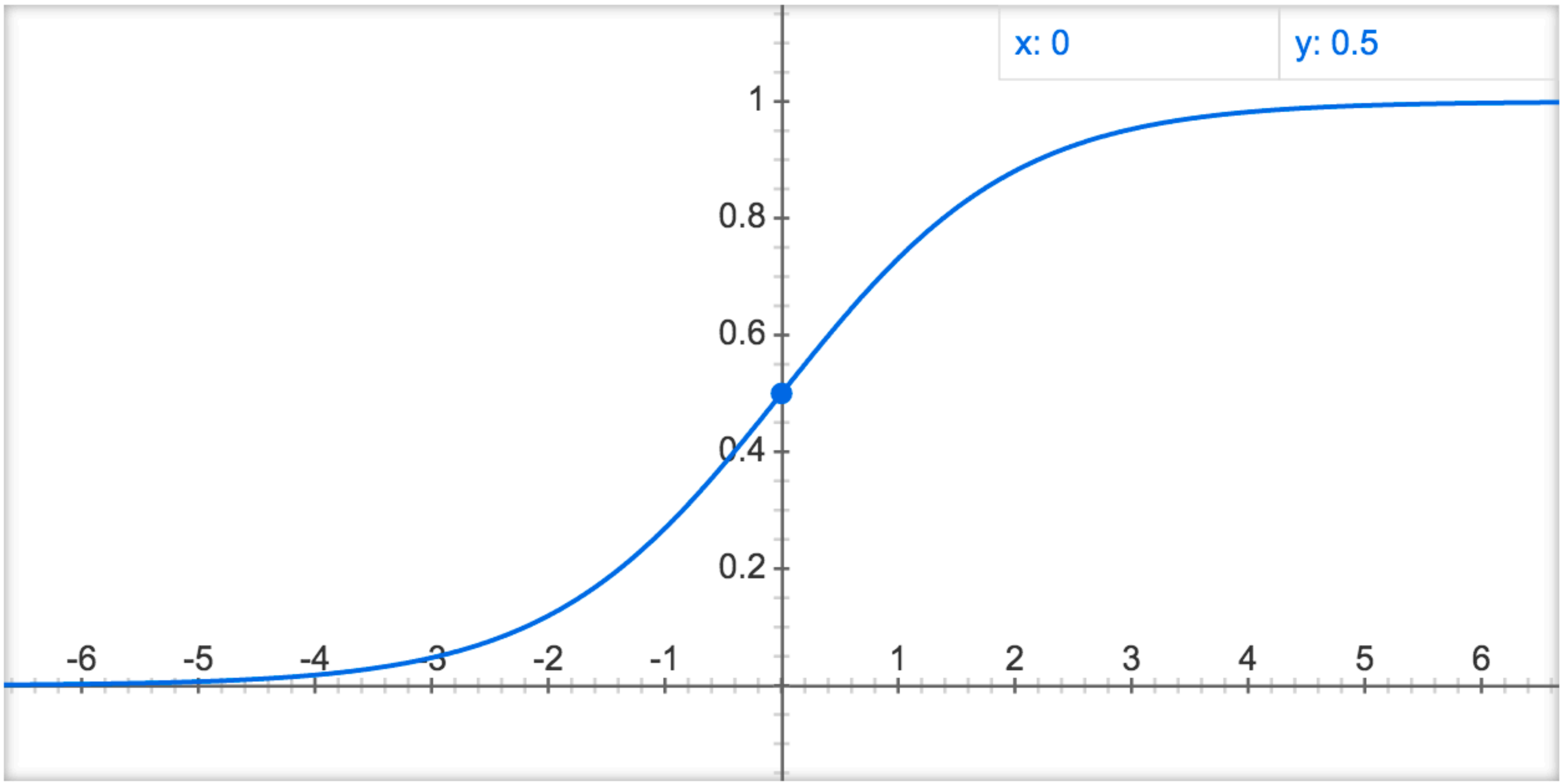
Negative Sampling
(Mikolov et al., 2013)

- rather than sum over entire vocabulary, generate samples and sum over them
- instead of a multiclass classifier, use a binary classifier:

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

- where sigma is logistic sigmoid function (see next slide) (logistic) sigmoid: $\sigma(x) = \dfrac{1}{1 + \exp\{-x\}}$

$\sigma(x)$



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)

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

- NEG contains 2-20 words sampled from some distribution
  - e.g., uniform, unigram, or smoothed unigram
  - smoothed: raise probabilities to power ¾, renormalize to get a distribution

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:

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

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:

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

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

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

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:

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

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

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

- 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

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
…	…

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:

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

- `word2vec` toolkit implements training for skip-gram and CBOW models
- very fast to train, even on large corpora
- pretrained embeddings available

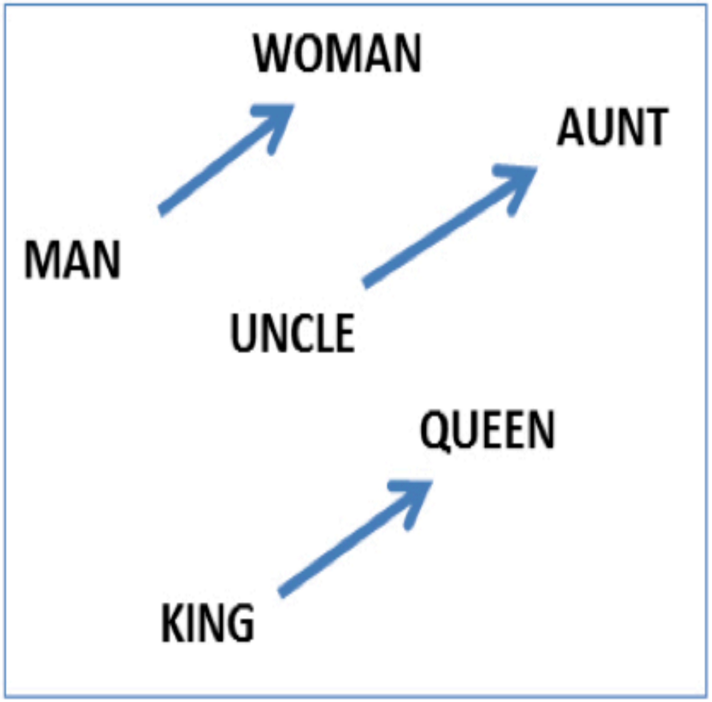
A simple way to investigate the learned representations is to find the closest words for a user-specified word. The *distance* tool serves that purpose. For example, if you enter 'france', *distance* will display the most similar words and their distances to 'france', which should look like:

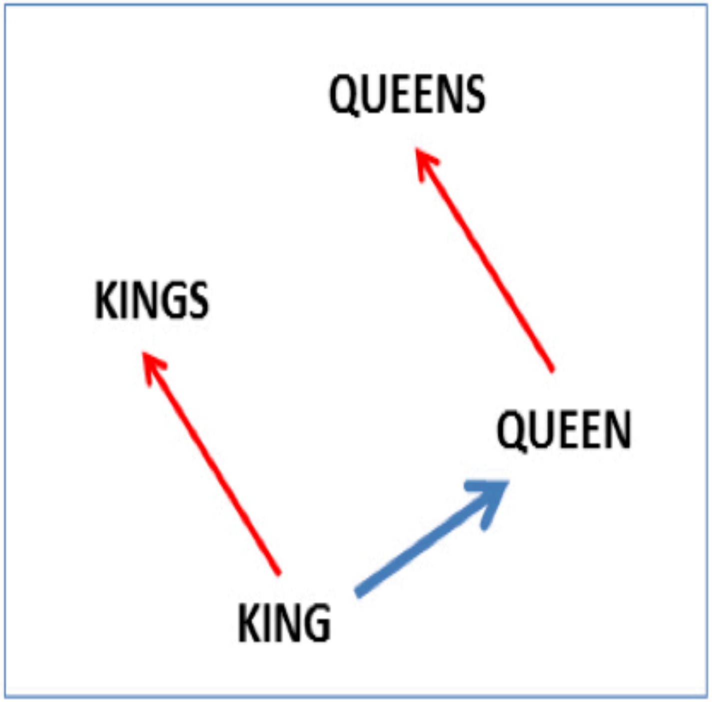
```
                 Word       Cosine distance
------------------------------------------
                spain              0.678515
              belgium              0.665923
          netherlands              0.652428
                italy              0.633130
          switzerland            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

Embeddings capture relational meaning!

vector(*king*) – vector(*man*) + vector(*woman*) ≈ vector(*queen*)

vector(*Paris*) – vector(*France*) + vector(*Italy*) ≈ vector(*Rome*)





J&M/SLP3

GloVe
(Pennington et al., 2014)

GloVe: Global Vectors for Word Representation

Jeffrey Pennington, Richard Socher, Christopher D. Manning
Computer Science Department, Stanford University, Stanford, CA 94305
jpennin@stanford.edu, richard@socher.org, manning@stanford.edu

Other Work on Word Embeddings

- active research area (probably too active)
- other directions:
  - multiple embeddings for a single word corresponding to different word senses
  - using subword information (e.g., characters) in word embeddings
  - tailoring embeddings for different NLP tasks

Other ways to learn word vectors

- aside: any labeled dataset can be used to learn word vectors (depending on model/features)
- how could you use your assignment 2 classifiers to produce word vectors?
- learned feature weights for a 5-way sentiment classifier (binary unigram features), for two words:

feel-good

label	weight
strongly positive	0.025
positive	0.035
neutral	-0.045
negative	0
strongly negative	-0.015

dull

label	weight
strongly positive	0
positive	0
neutral	-0.04
negative	0.015
strongly negative	0.025

Task-Driven Word Embeddings

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

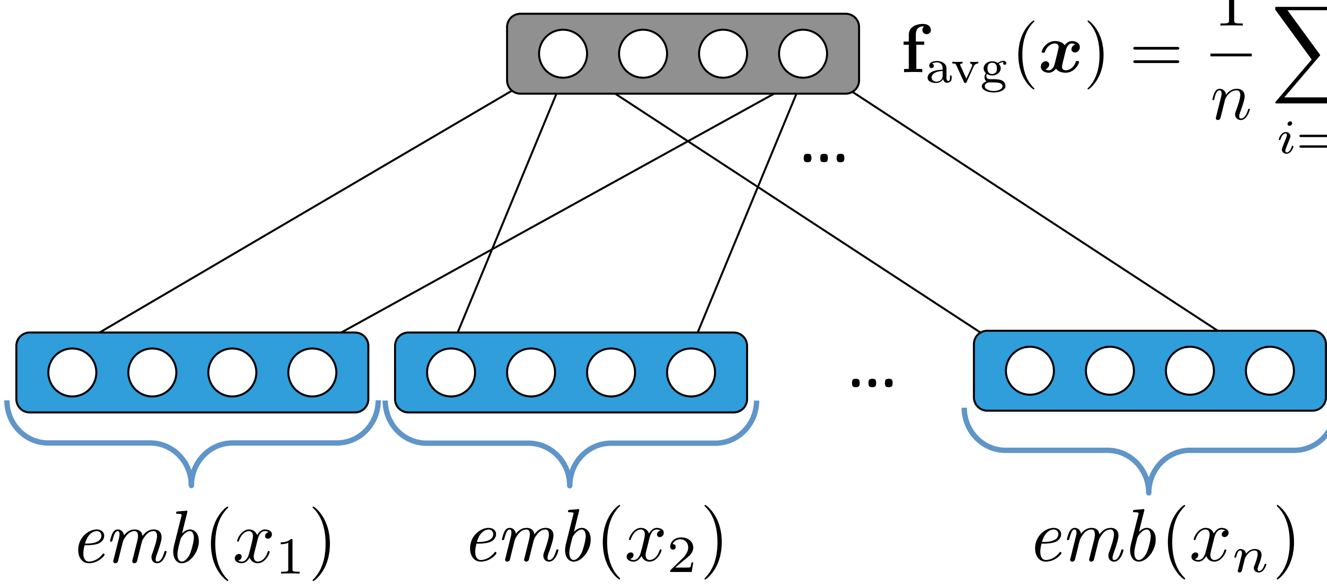
$$\mathbf{z}^{(1)} = g\left(\mathbf{U}^{(0)}\mathbf{x} + \mathbf{b}^{(0)}\right)$$

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

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

A Simple Neural Text Classification Model

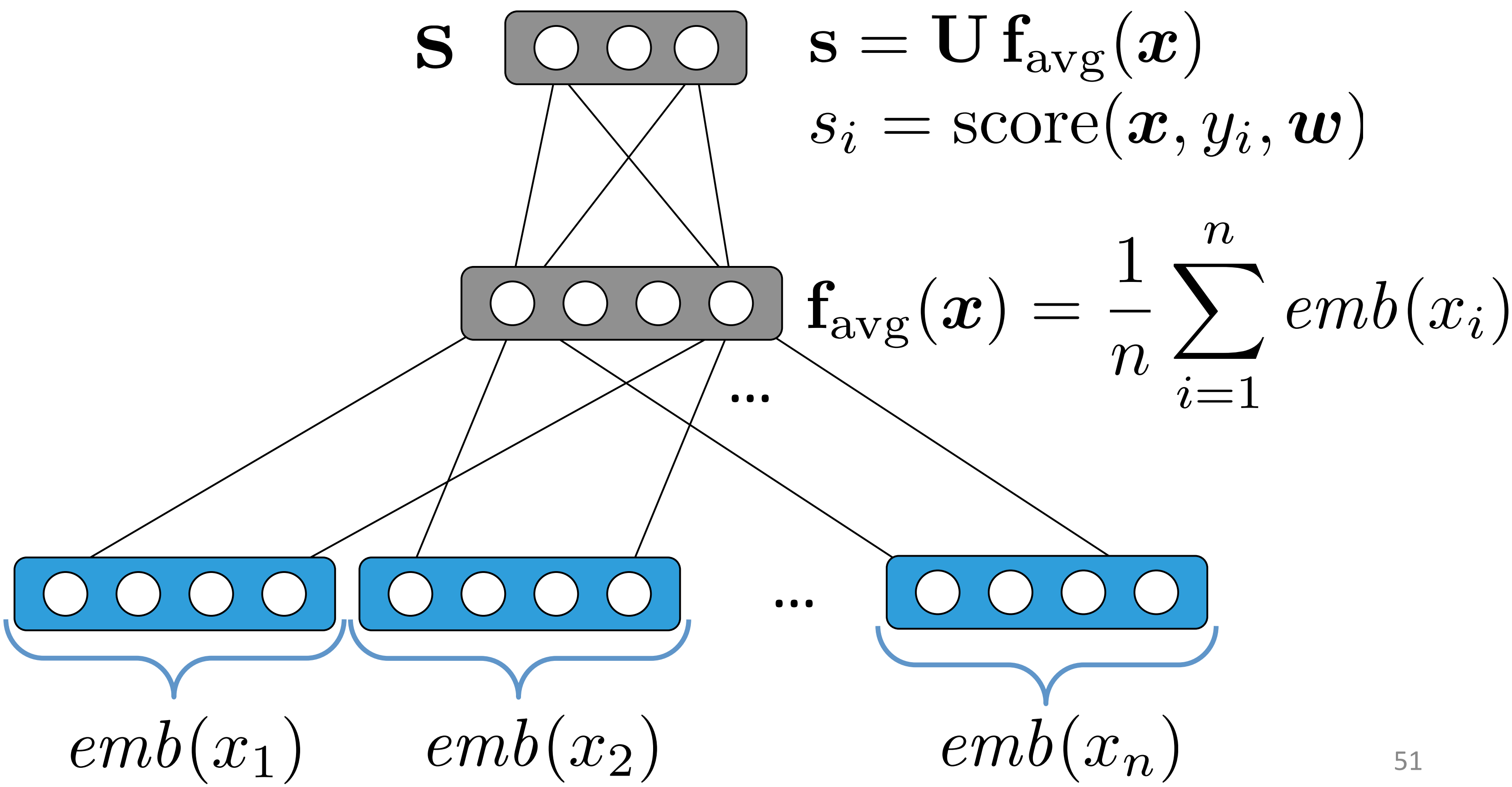
- given a word sequence ***x***, predict its label
- represent ***x*** by averaging its word embeddings:



$$\mathbf{f}_{\text{avg}}(\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n} emb(x_i)$$

A Simple Neural Text Classification Model

- represent ***x*** by averaging its word embeddings
- output is a score vector over all possible labels:



$$\mathbf{s} = \mathbf{U}\,\mathbf{f}_{\text{avg}}(\boldsymbol{x})$$

$$s_i = \text{score}(\boldsymbol{x}, y_i, \boldsymbol{w})$$

$$\mathbf{f}_{\text{avg}}(\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n} emb(x_i)$$

Averaging Word Embeddings

- effective encoder for text classification and many other tasks
- sometimes called a neural bag of words (NBOW) model (Kalchbrenner et al., 2014)
- or a deep averaging network (DAN), especially if hidden layers are used (Iyyer et al., 2015)

Encoders

- encoder: a function to represent a word sequence as a vector
- simplest: average word embeddings:

$$\mathbf{f}_{\mathrm{avg}}(\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n} emb(x_i)$$

- many other functions possible!
- lots of recent work on developing better ways to encode word sequences

Attention

- attention is a useful generic tool
- often used to replace a sum or average with an attention-weighted sum

Attention

- attention is a useful generic tool
- often used to replace a sum or average with an attention-weighted sum
- e.g., for a word averaging encoder:

$$\mathbf{f}_{\text{att}}(\boldsymbol{x}) = \sum_{i=1}^{n} \underbrace{att(x_i, i, \boldsymbol{x})}_{\text{“attention” function, returns a scalar}} emb(x_i)$$

Attention

- attention is a useful generic tool
- often used to replace a sum or average with an attention-weighted sum
- e.g., for a word averaging encoder:

$$\mathbf{f}_{\text{att}}(\boldsymbol{x}) = \sum_{i=1}^{n} att(x_i, i, \boldsymbol{x}) emb(x_i)$$

$$\sum_{i=1}^{n} att(x_i, i, \boldsymbol{x}) = 1$$

- many attention functions are possible!

Encoders

- many neural network architectures have been designed for encoding sequences

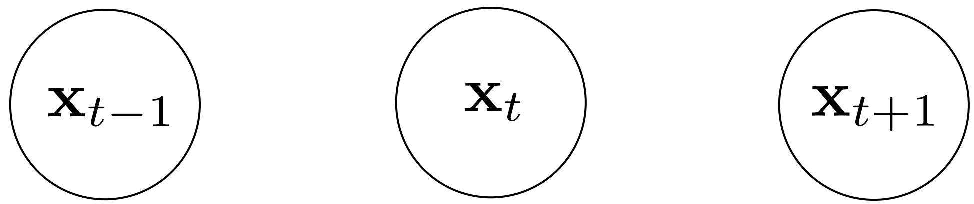
Recurrent Neural Networks

Input is a sequence:

x_{t-1}	x_t	x_{t+1}
not	*too*	*bad*

Recurrent Neural Networks

Input is a sequence:

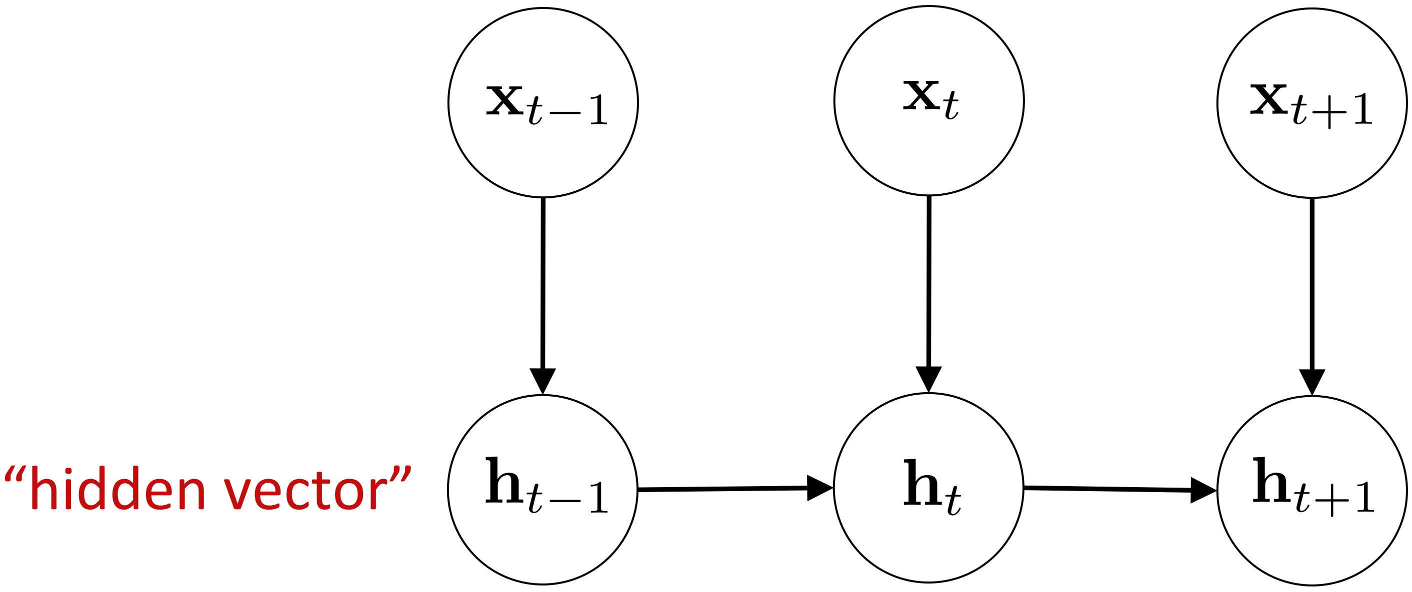


x_{t-1} x_t x_{t+1}

not *too* *bad*

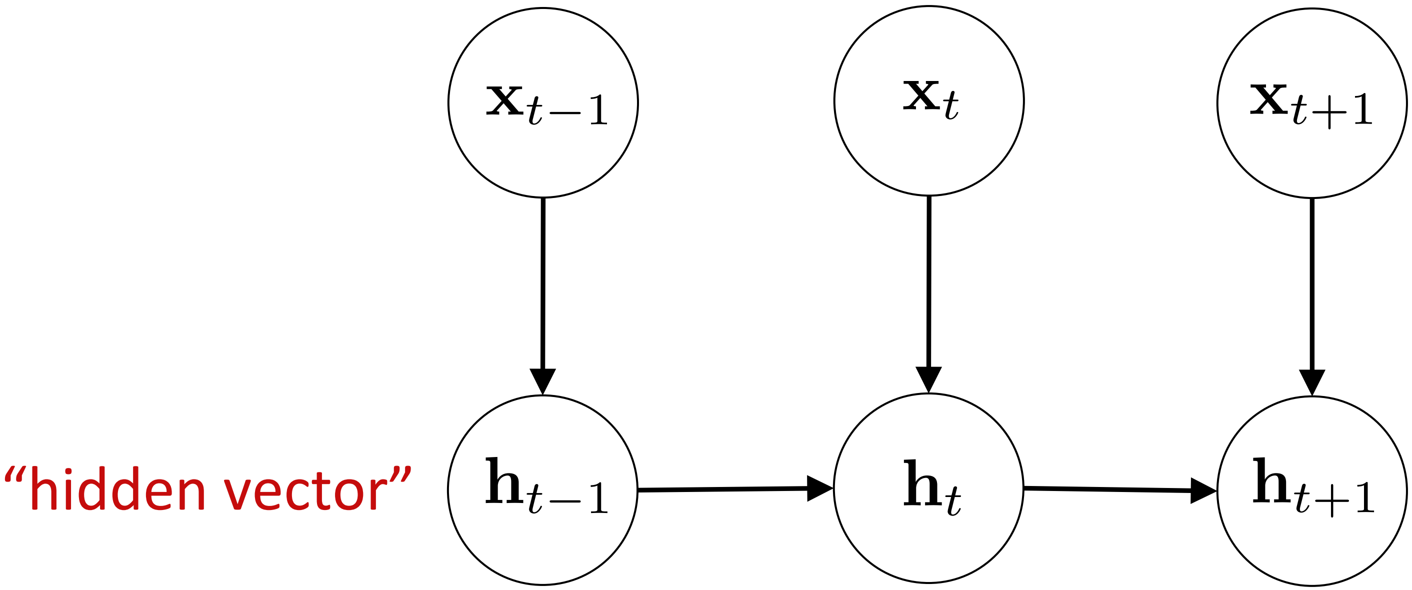
Recurrent Neural Networks

Input is a sequence:



Recurrent Neural Networks

$$\mathbf{h}_t = \tanh\left(\mathbf{W}^{(x)}\mathbf{x}_t + \mathbf{W}^{(h)}\mathbf{h}_{t-1} + \mathbf{b}\right)$$



Disclaimer

- these diagrams are often useful for helping us understand and communicate neural network architectures
- but they rarely have any sort of formal semantics (unlike graphical models)
- they are more like cartoons