TTIC 31210:
Advanced Natural Language Processing

Kevin Gimpel

Spring 2019

Lecture 2:
Elements of Neural NLP

Assignment 1

- Assignment 1 has been posted; due April 16

Course Web Page

https://ttic.uchicago.edu/~kgimpel/teaching/31210-s19/index.html

TTIC 31210: Advanced Natural Language Processing

lectures assignments

This is the course webpage for the Spring 2019 version of TTIC 31210: Advanced Natural Language Processing.
For the Spring 2017 course, go **here**.

Quarter: Spring 2019
Time: Monday/Wednesday 1:30-2:50pm
Location: Room 526 (fifth floor), TTIC

Instructor: Kevin Gimpel
Instructor Office Hours: Mondays 2:50-3:15pm, Wednesdays 2:50-4pm, Room 531
Teaching Assistant: Mingda Chen
Teaching Assistant Office Hours: Mondays 3-4pm, TTIC Library (fourth floor)

Prerequisites: TTIC 31190 or permission of the instructor.

Contents:

Roadmap

- intro (1 lecture)
- **deep learning for NLP (5 lectures)**
- structured prediction: sequence labeling, syntactic and semantic parsing, dynamic programming (4 lectures)
- generative models, latent variables, unsupervised learning, variational autoencoders (2 lectures)
- Bayesian methods in NLP (2 lectures)
- Bayesian nonparametrics in NLP (2 lectures)
- review & other topics (1 lecture)

Today

- brief review of neural language modeling
- neural similarity modeling
- loss functions for similarity modeling

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)**

Markov Assumption

- only use the last *k* words to predict the next:

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

Bigram model

condition on the previous word:

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

automatically generated sentences from a bigram 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
```

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:

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

What is a neural network?

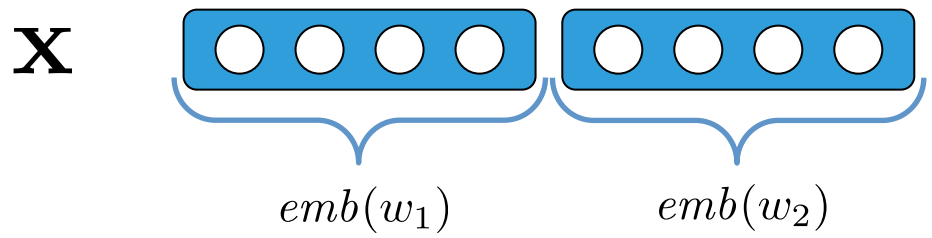
- just think of a neural network as a function
- it has inputs and outputs
- “neural” typically means one type of functional building block (“neural layers”), but the term has broadened
- neural modeling is now better thought of as a modeling strategy (leveraging “distributed representations” or “representation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

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) representing previous words:



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

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

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

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:

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

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

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

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:

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

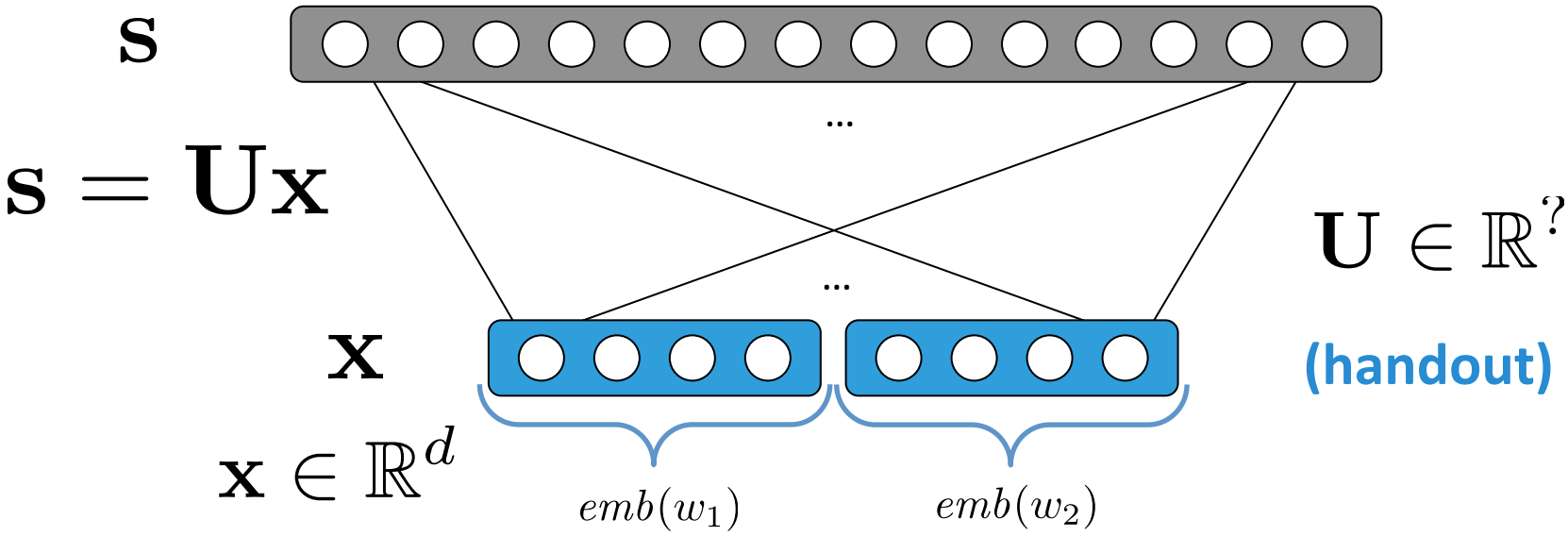
- 2: store embeddings in a hash/dictionary data structure, do lookup to find embedding for word:

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

- these are equivalent; second may be much faster (first can be fast with sparse operations)

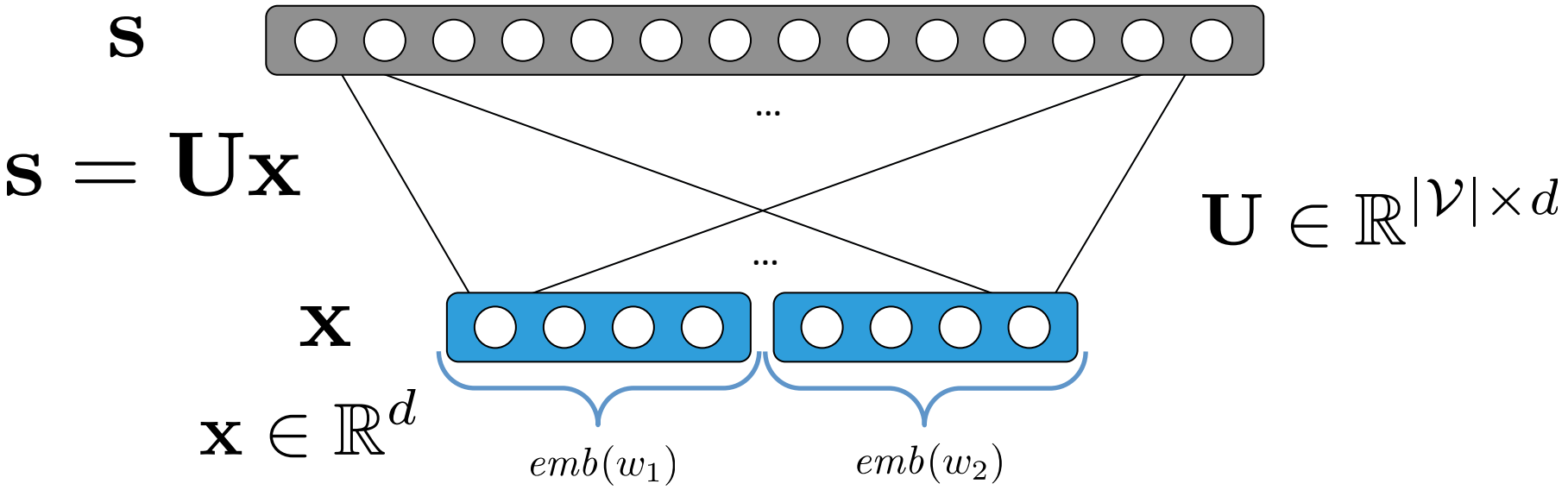
A Simple Neural Trigram Language Model

- output vector contains scores of possible next words:



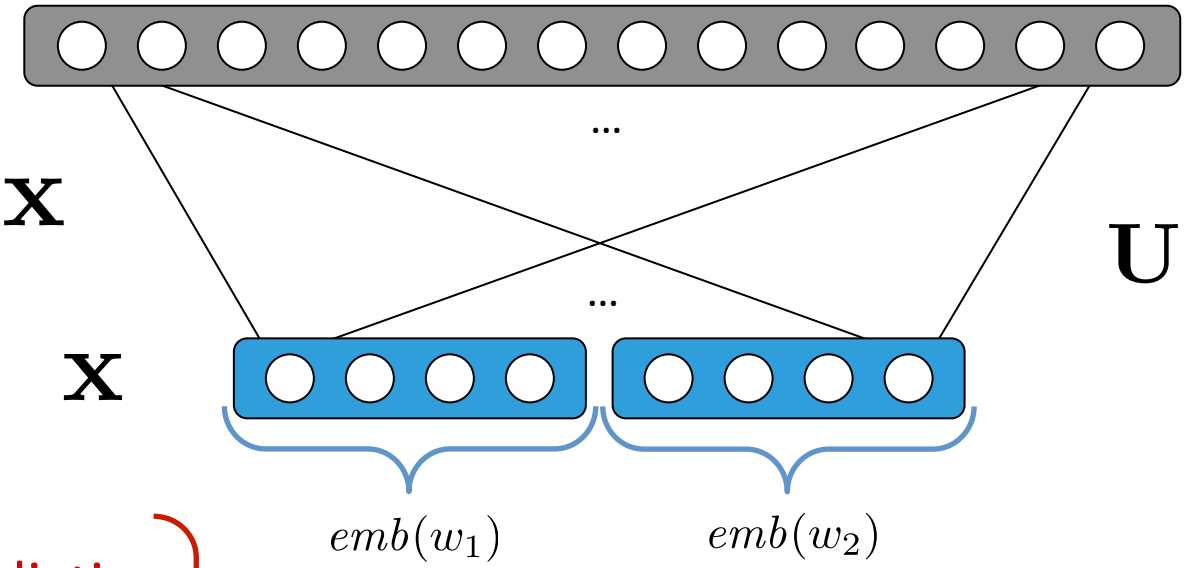
A Simple Neural Trigram Language Model

- output vector contains scores of possible next words:



A Simple Neural Trigram Language Model

- output vector contains scores of possible next words:



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

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

score of predicting word type *i* given previous words w_1 and w_2

$$s_i = \text{score}(\langle w_1, w_2 \rangle, i, \boldsymbol{\theta})$$

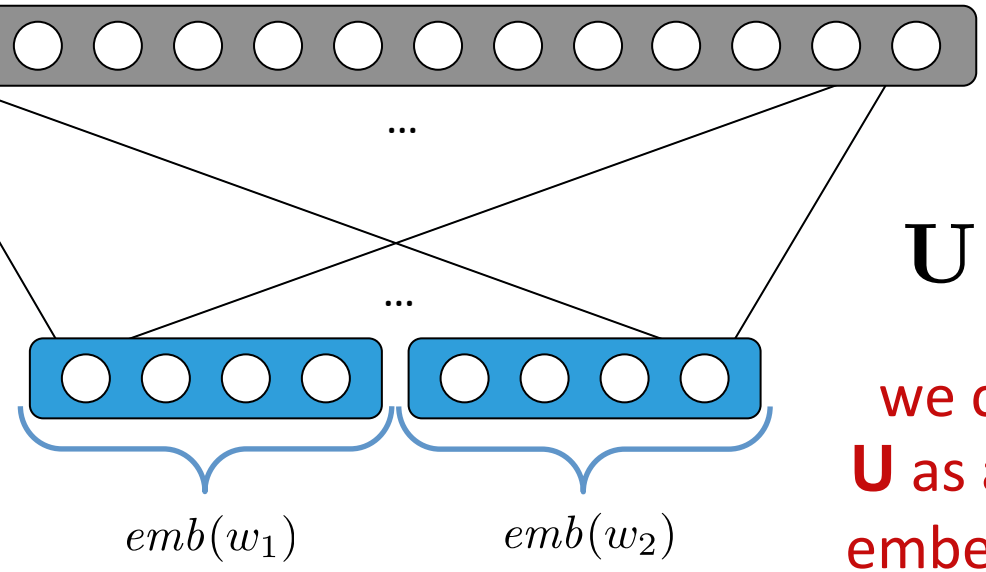
$$\text{score}(\langle w_1, w_2 \rangle, i, \boldsymbol{\theta}) = \mathbf{U}_{i,1:d}\,\mathbf{x}$$

A Simple Neural Trigram Language Model

- output vector contains scores of possible next words:

$\mathbf{s}$

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



$\mathbf{x}$

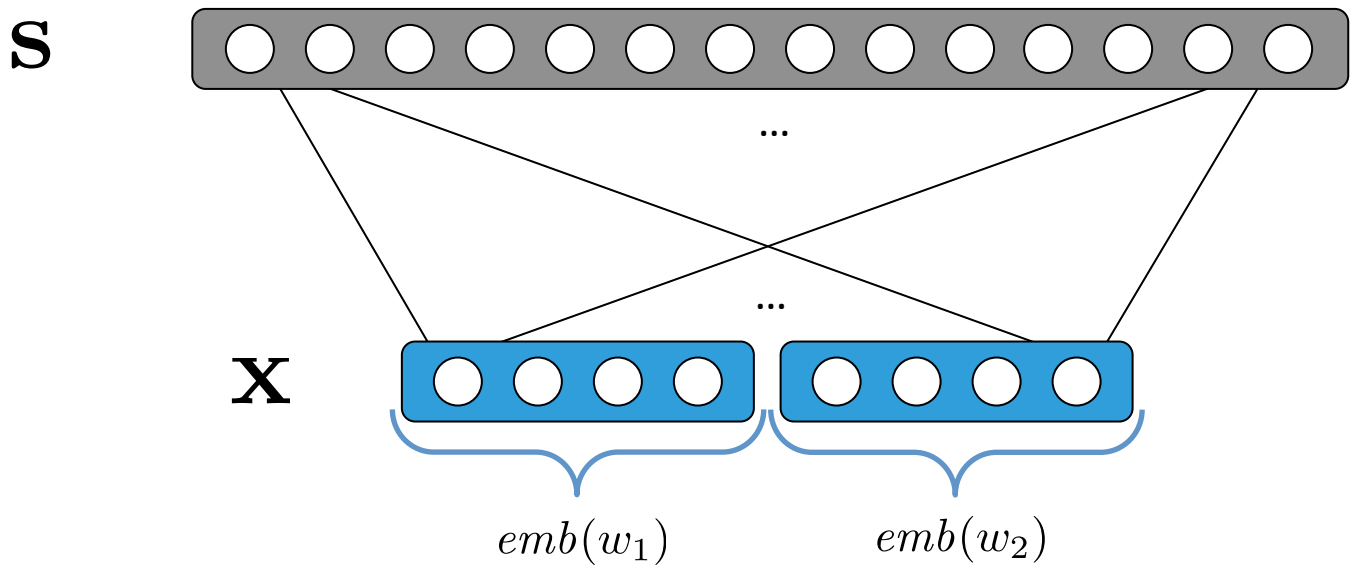
$\mathbf{U} \in \mathbb{R}^{|\mathcal{V}| \times d}$

we can interpret **U** as another word embedding matrix!

$$s_i = \text{score}(\langle w_1, w_2 \rangle, i, \boldsymbol{\theta})$$

$$\text{score}(\langle w_1, w_2 \rangle, i, \boldsymbol{\theta}) = \mathbf{U}_{i,1:d}\,\mathbf{x}$$

Training for Neural Language Models

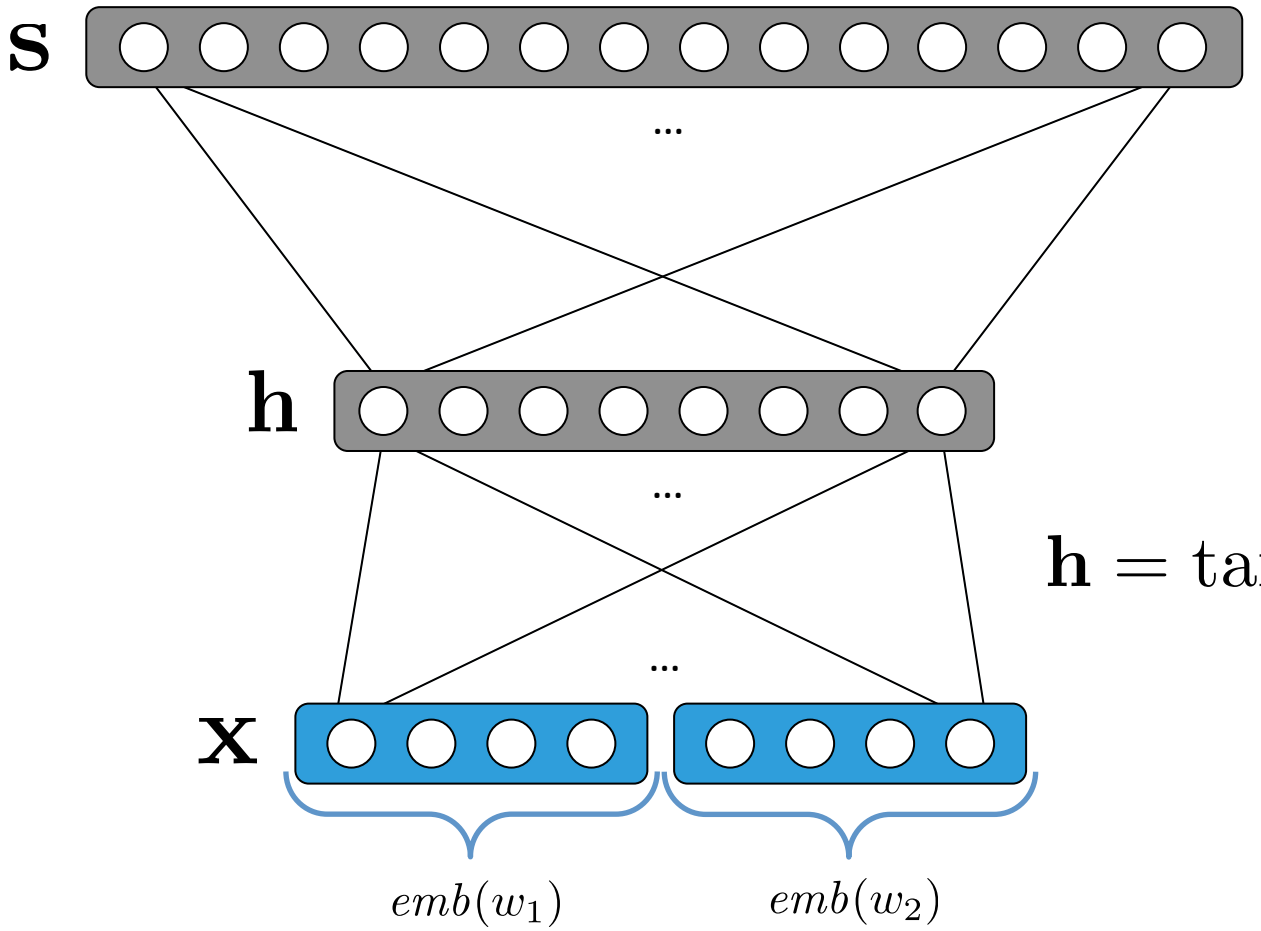


- most common way to train: log loss

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

$$p_{\boldsymbol{\theta}}(i \mid \langle w_1, w_2 \rangle) \propto \exp\{\text{score}(\langle w_1, w_2 \rangle, i, \boldsymbol{\theta})\}$$

Adding a Hidden Layer



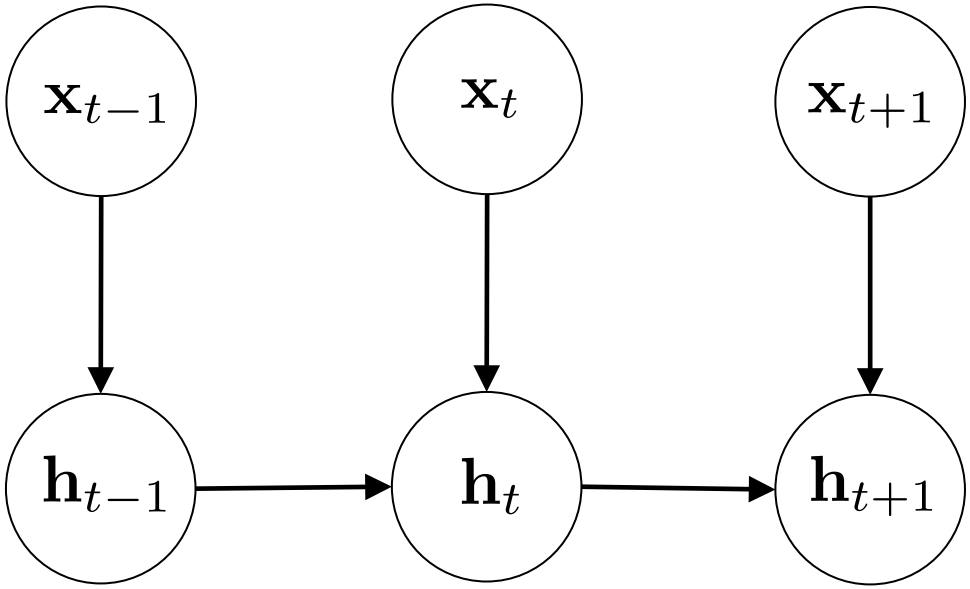
$$\mathbf{s} = \mathbf{U}^{(1)}\mathbf{h}$$

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

Recurrent Neural Networks

Input is a sequence:

$$\mathbf{x}_t = emb(w_t)$$



"hidden vector"

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

Notation

$\mathbf{u} =$ a vector

$u_i =$ entry *i* in the vector

$\mathbf{W} =$ a matrix

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

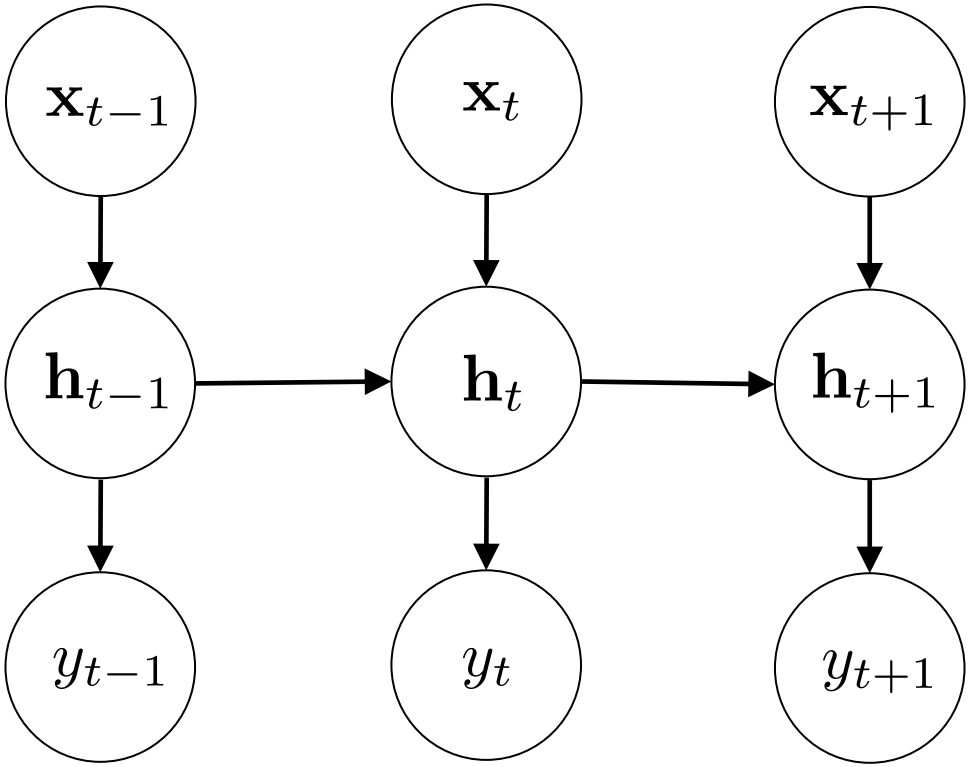
- we often use RNNs to encode sentences or longer text sequences

- common for sentence classification, machine translation, question answering, etc.

- but RNNs are also frequently used for **generating** sequences

“Output” Recurrent Neural Networks

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



“hidden vector”

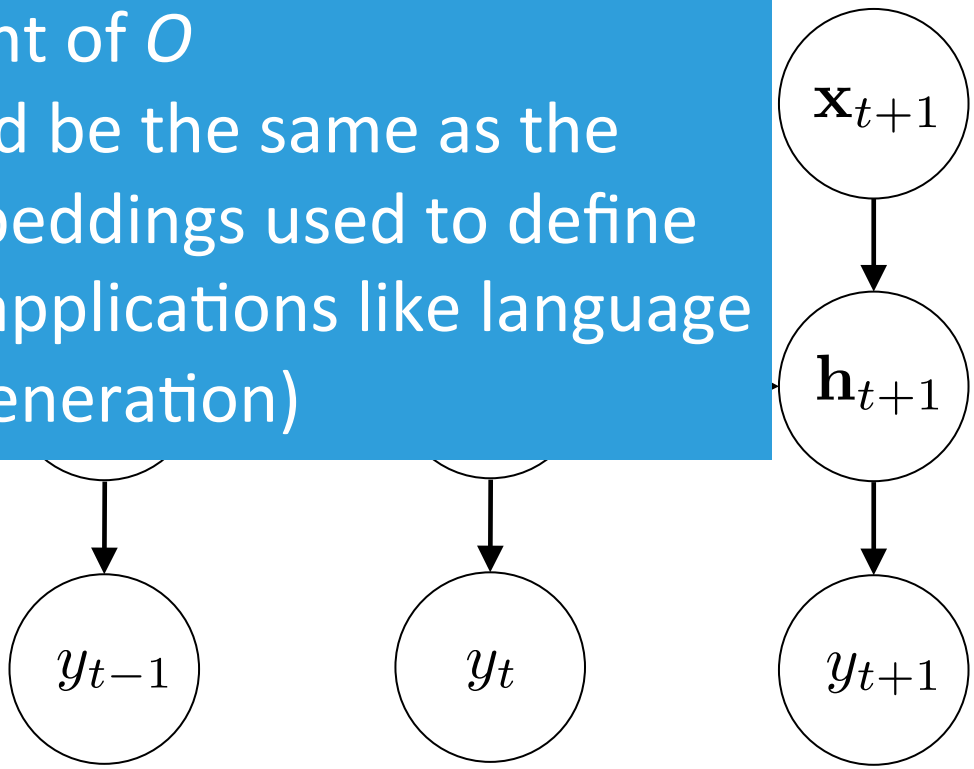
“output symbol”

$$y_t = \underset{y \in \mathcal{O}}{\operatorname{argmax}} \; emb(y)^\top \mathbf{h}_t$$

- *y* is a symbol, not a vector
- *O* is the "output" vocabulary
- we have new parameters *emb(y)* for each element of *O*
- *emb(y)* could be the same as the "input" embeddings used to define each **x** (for applications like language modeling/generation)

Networks

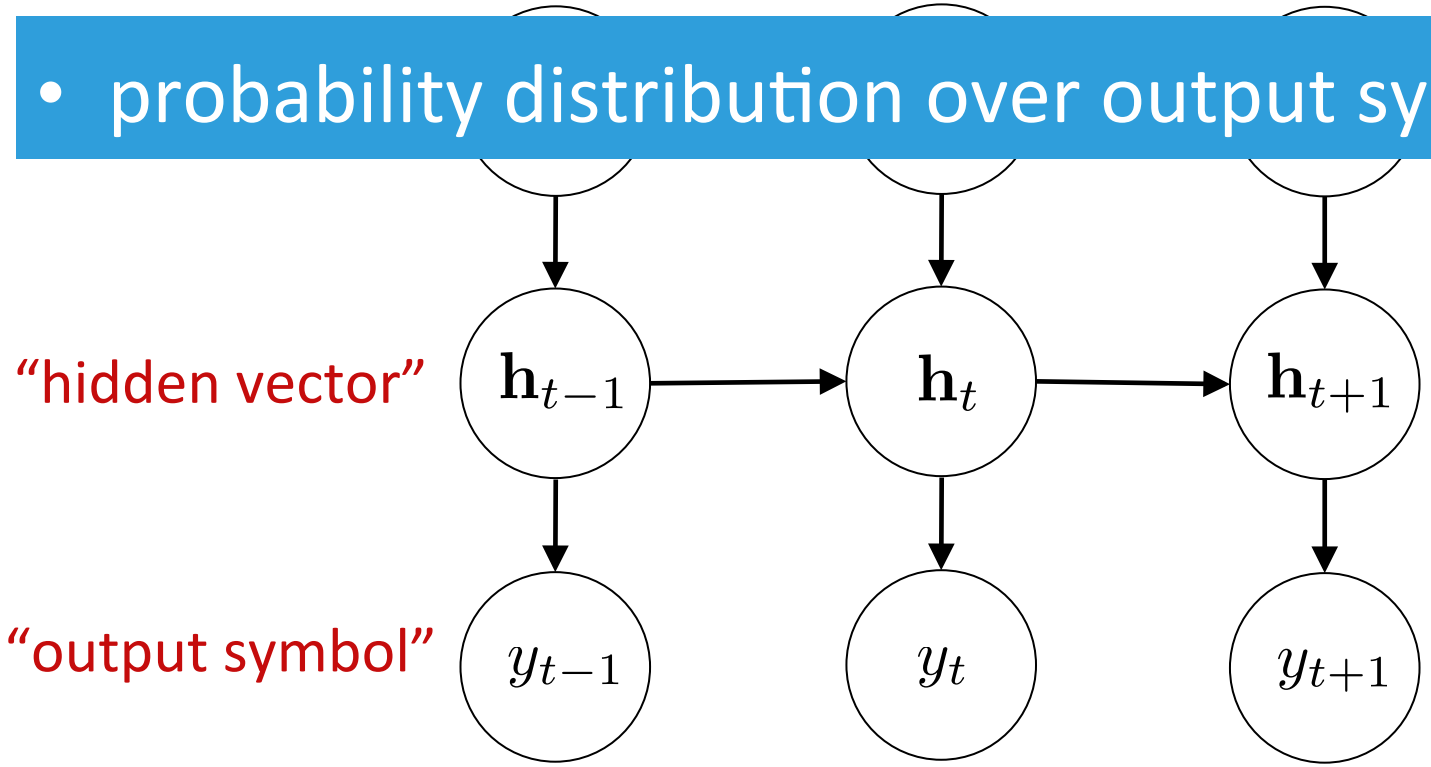
$$\mathbf{h}_{t-1} + \mathbf{b}\Big)$$



"output symbol"

$$y_t = \underset{y \in \mathcal{O}}{\operatorname{argmax}} \; emb(y)^\top \mathbf{h}_t$$

- probability distribution over output symbols?



$$y_t = \underset{y \in \mathcal{O}}{\text{argmax}} \quad emb(y)^\top \mathbf{h}_t$$

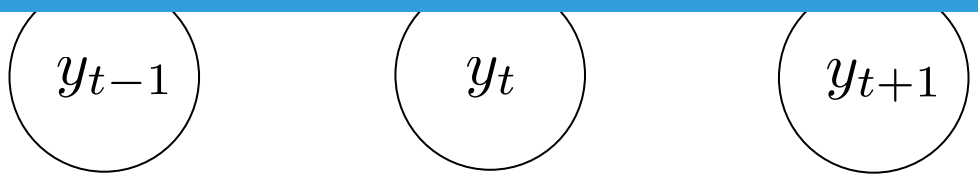
$$P(Y_t) = \text{softmax}\left(\mathbf{W}\mathbf{h}_t\right)$$

$$\mathbf{W} = \left[emb(y_1)^\top; emb(y_2)^\top; ...; emb(y_{|\mathcal{O}|})^\top\right]$$

For language modeling:

- *emb(y)* are often same as input embeddings
- but could be entirely new parameters that are learned (as in your first assignment)

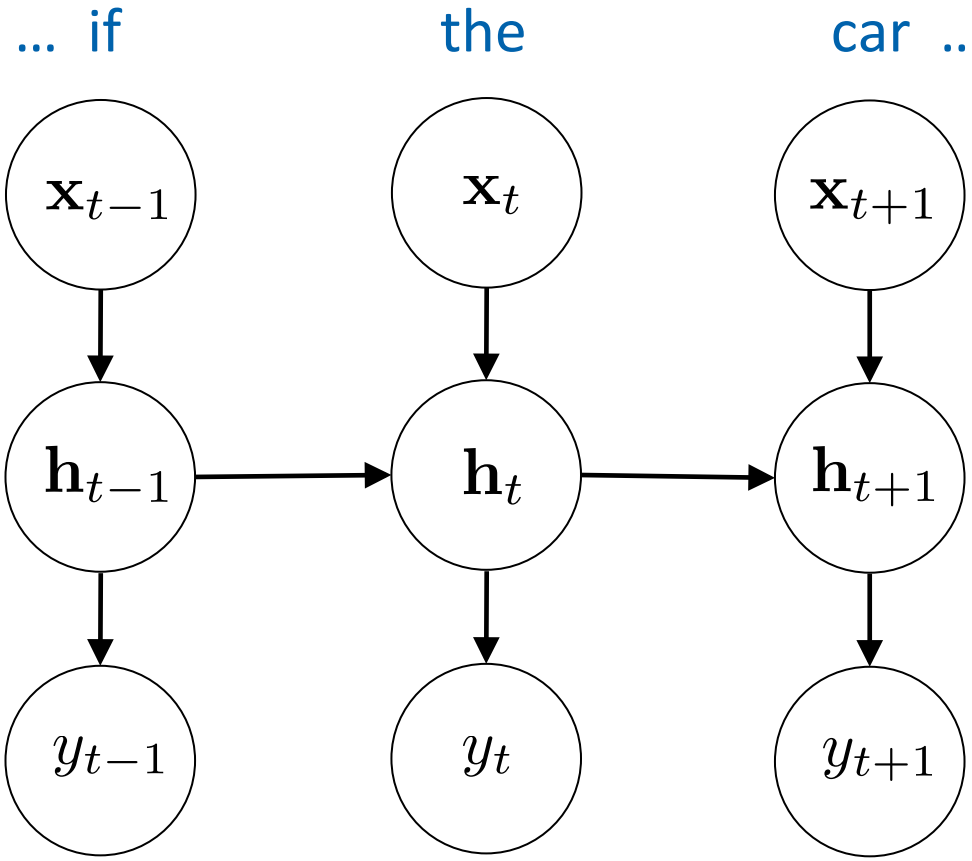
"output symbol"



$$P(Y_t) = \text{softmax}\left(\mathbf{W}\mathbf{h}_t\right)$$

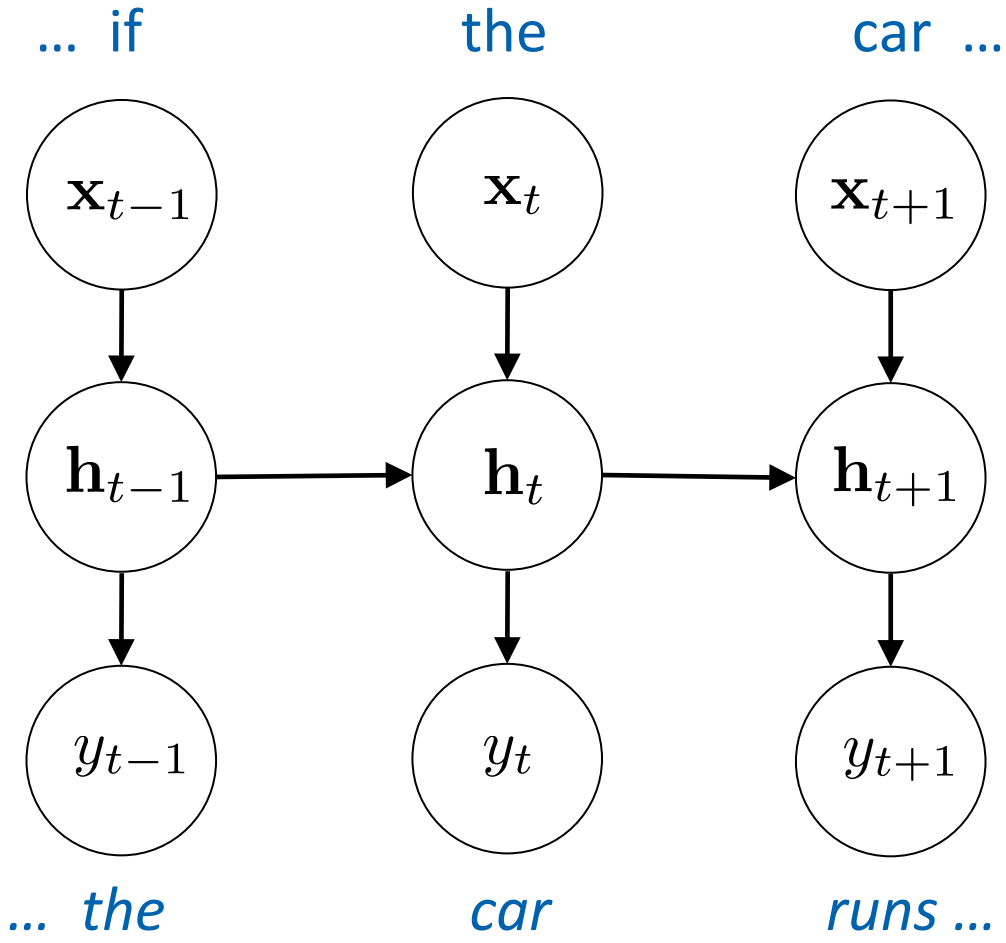
$$\mathbf{W} = \left[emb(y_1)^\top; emb(y_2)^\top; ...; emb(y_{|\mathcal{O}|})^\top\right]$$

Language Modeling



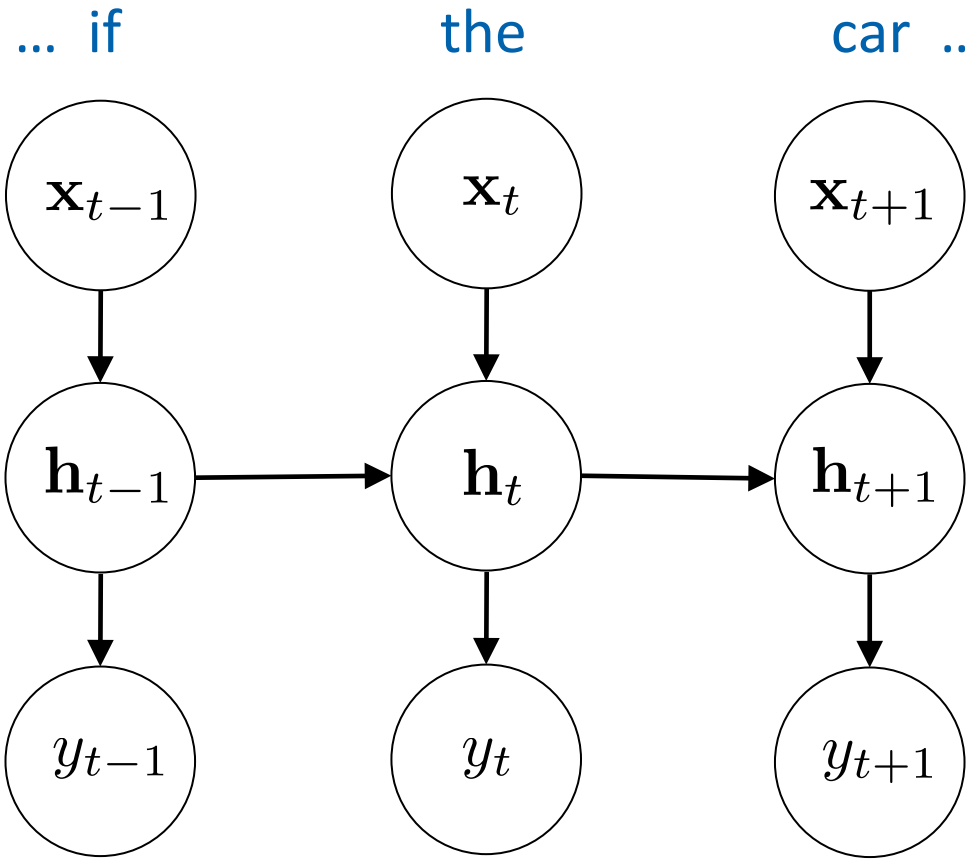
- input: a word sequence
- output?

Language Modeling



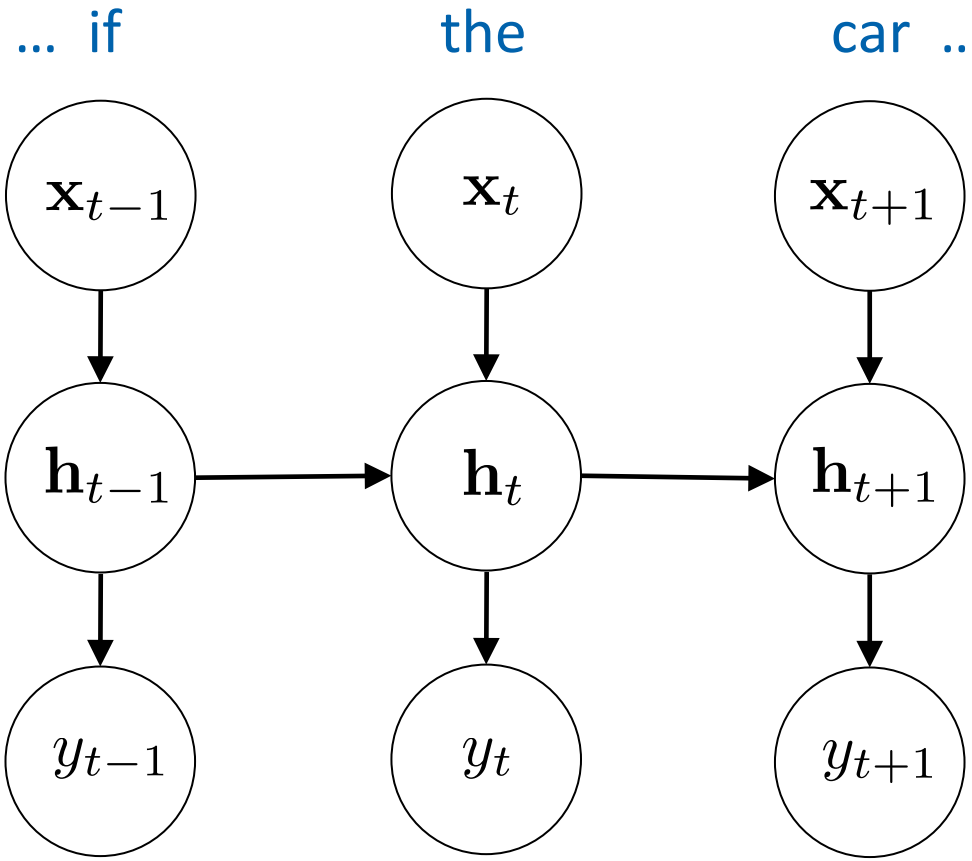
- target output at each position: next word in the sequence

Language Modeling: Training



$$-\log P(Y_{t-1} = ?)$$

Language Modeling: Training



$$-\log P(Y_{t-1} = \text{“the”}) \quad -\log P(Y_t = \text{“car”}) \quad \ldots$$

- while we showed this for simple RNNs, it's easy to instead use LSTMs, GRUs, etc.

- LSTMs/GRUs still produce a hidden vector at each position in the sequence, just like RNNs

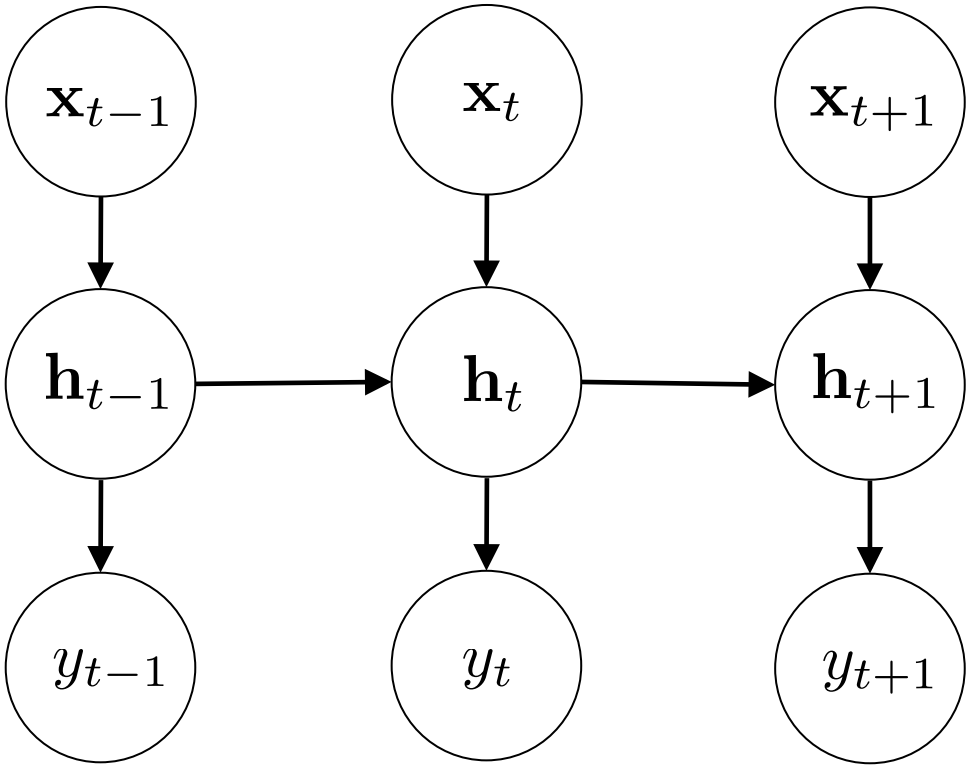
- it's common to discuss models using RNN abstraction that produces a hidden vector at each time step, but you can always substitute in GRU, LSTM, etc.

Neural Similarity Learning

- A common need:

  compute similarity/affinity of two things
  - maybe two things of the same type,
  - two things with different types being mapped to same space, or
  - two things with different types being mapped to different spaces, but being compared with a learned similarity function
- Examples?

Example: Classification



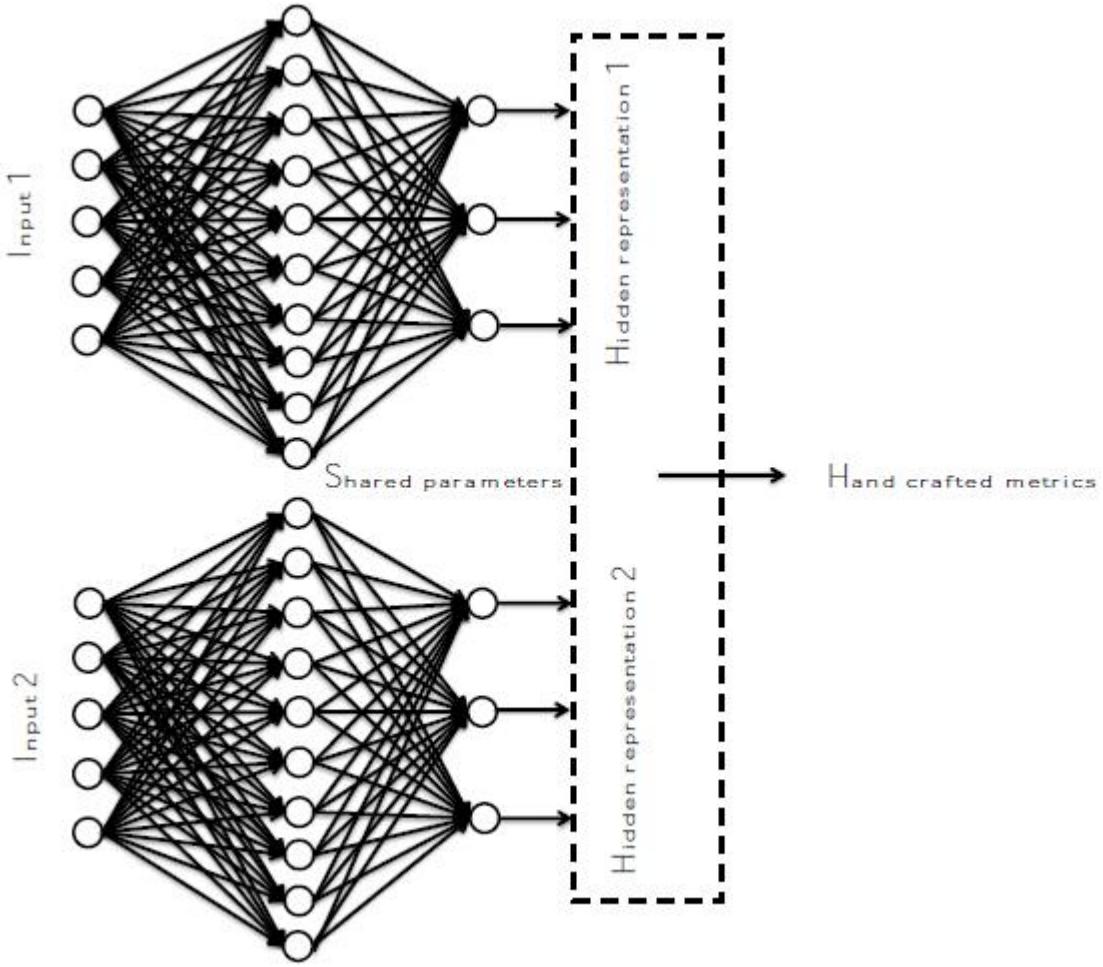
$$y_t = \underset{y \in \mathcal{O}}{\text{argmax}} \quad emb(y)^\top \mathbf{h}_t$$

$$P(Y_t) = \text{softmax}\left(\mathbf{W}\mathbf{h}_t\right)$$

similarity between hidden vector and output symbol embedding (using dot product)

Neural Similarity Modeling

- "Siamese networks" (Bromley et al., 1993)
  - these typically share parameters



Similarity Modeling

- Siamese networks typically share parameters across the two networks

- but it's also common to not share parameters when the items have different types, but we still want to relate them in some way
  - whether map to same space + compute similarity or map each to some other space + compute similarity

Synonym Pairs

- from WordNet, paraphrase resources like the Paraphrase Database, etc.
- Faruqui et al. (2014), Wieting et al. (2015), *inter alia*

contamination	pollution
converged	convergence
captioned	subtitled
outwit	thwart
bad	villain
broad	general
permanent	permanently
bed	sack
carefree	reckless
absolutely	urgently
…	…

Lexical Translation Pairs

- from bilingual dictionaries, automatically extracted from bitext, etc.
- Haghighi et al. (2008), Mikolov et al. (2013), Faruqui and Dyer (2014)

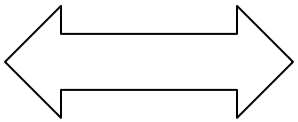
dog	hund
man	mann
woman	frau
city	stadt
person	man
the	der
the	die
the	das
…	…

Paraphrase Pairs

this was also true for pompeii , where the temple of jupiter that was already there was enlarged and made more roman when the romans took over .

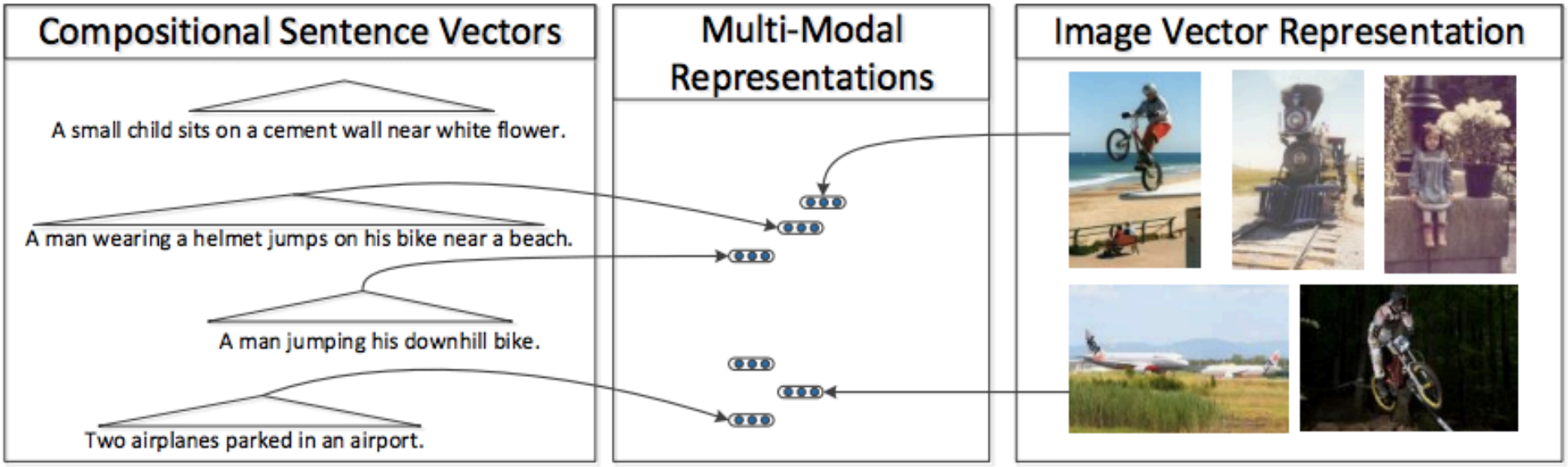
this held true for pompeii , where the previously existing temple of jupiter was enlarged and romanized upon conquest .

WIKIPEDIA
The Free Encyclopedia



WIKIPEDIA
Simple English

Images and Captions



Socher et al. (2014): *Grounded Compositional Semantics For Finding And Describing Images With Sentences*

Images and Captions

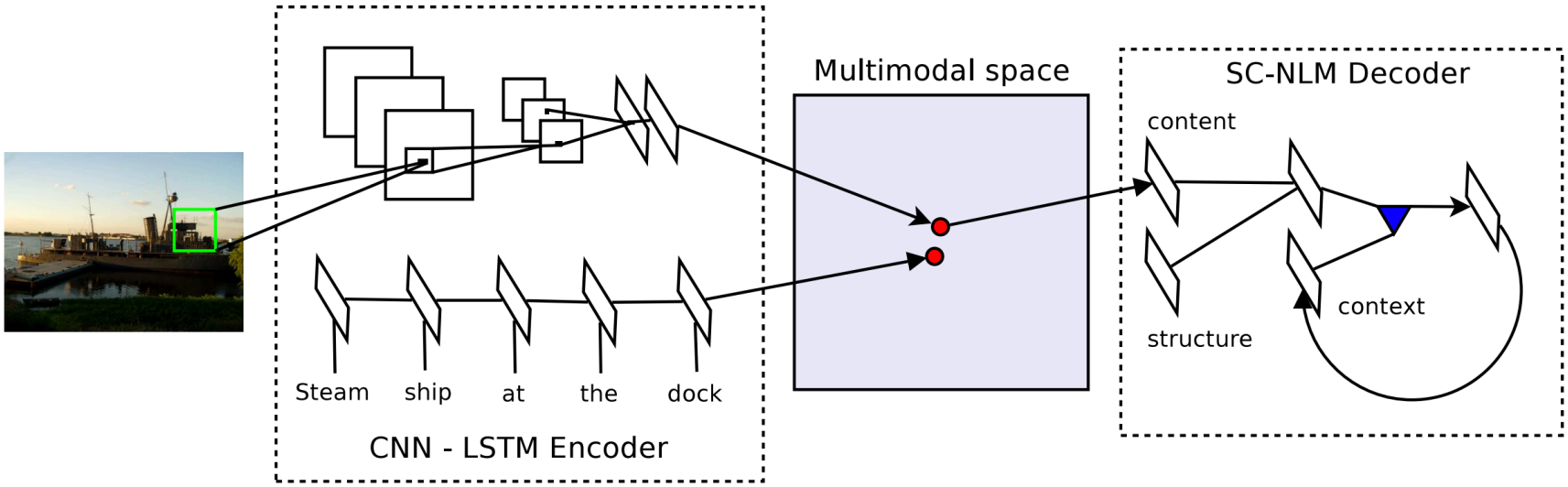
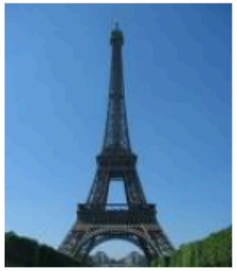


Figure 2: **Encoder:** A deep convolutional network (CNN) and long short-term memory recurrent network (LSTM) for learning a joint image-sentence embedding. **Decoder:** A new neural language model that combines structure and content vectors for generating words one at a time in sequence.

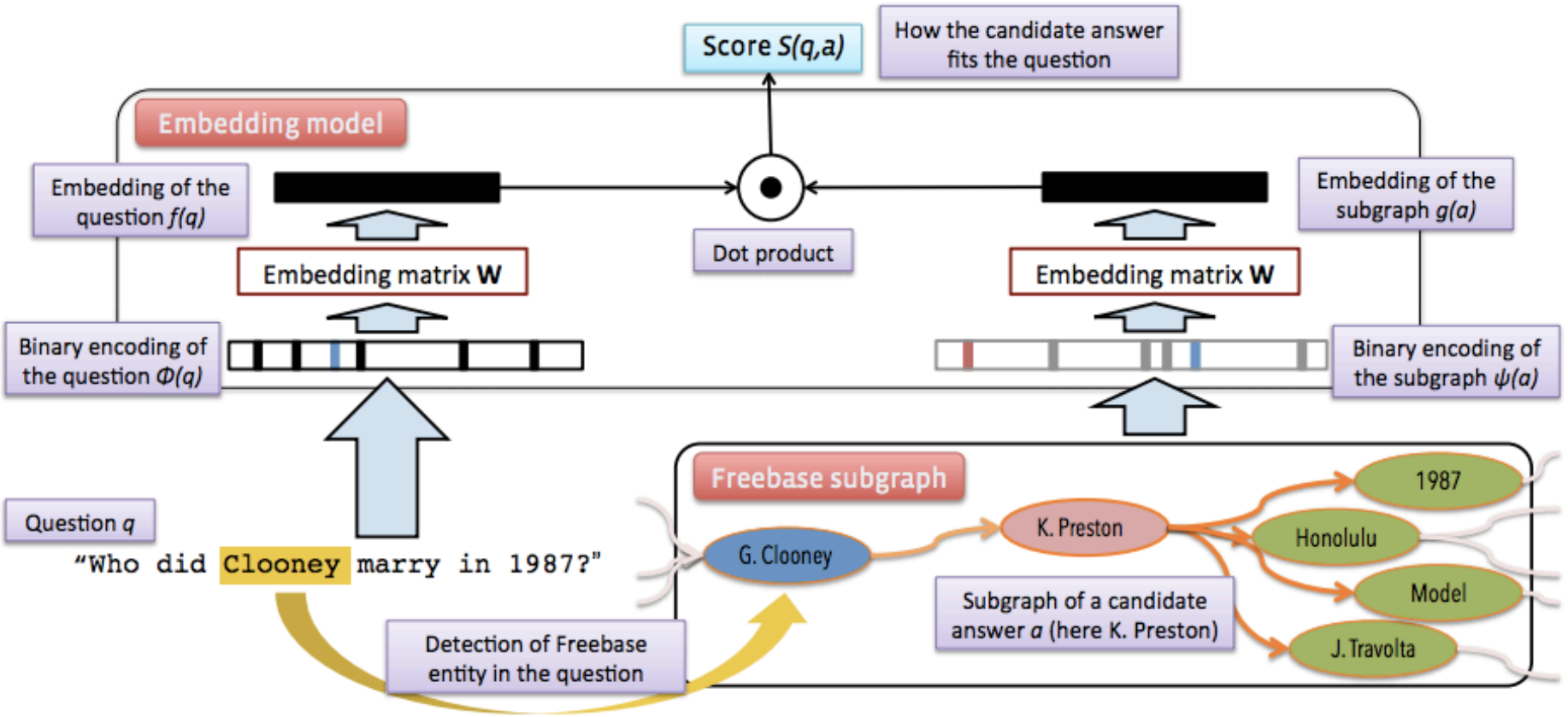
Kiros et al. (2014): *Unifying Visual-Semantic Embeddings with Multimodal Neural Language Models*

Images and Annotations

Image	One-vs-Rest	WSABIE
	surf, bora, belize, sea world, balena, wale, tahiti, delfini, surfing, mahi mahi	delfini, orca, **dolphin**, mar, delfin, dauphin, whale, cancun, killer whale, sea world
	**eiffel tower**, tour eiffel, snowboard, blue sky, empire state building, luxor, eiffel, lighthouse, jump, adventure	**eiffel tower**, statue, eiffel, mole antoneliana, la tour eiffel, londra, cctv tower, big ben, calatrava, tokyo tower
	falco, barack, daniel craig, **obama**, barack obama, kanye west, pharrell williams, 50 cent, barrack obama, bono	barrack obama, barack obama, barack hussein obama, barack obama, james marsden, jay z, **obama**, nelly, falco, barack

Weston et al. (2011): *WSABIE: scaling up to large vocabulary image annotation*

Questions and Answers



Bordes et al. (2014): *Question Answering with Subgraph Embeddings*

Iyyer et al. (2014): *A Neural Network for Factoid Question Answering over Paragraphs*

Entities in Knowledge Bases

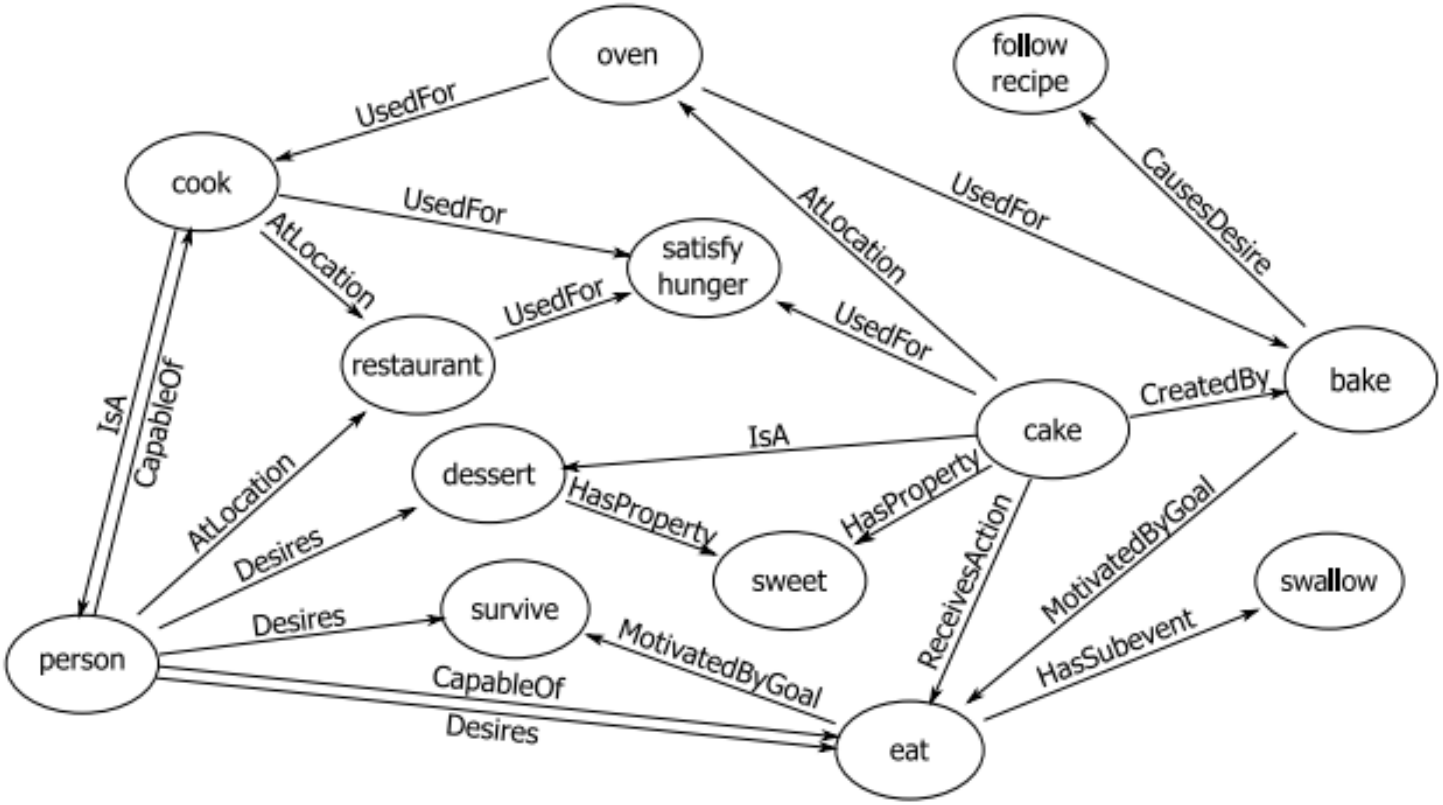
Table 5: **Example predictions** on the FB15k test set using TransE. **Bold** indicates the test triplet's true tail and *italics* other true tails present in the training set.

INPUT (HEAD AND LABEL)	PREDICTED TAILS
J. K. Rowling `influenced by`	*G. K. Chesterton*, J. R. R. Tolkien, *C. S. Lewis*, **Lloyd Alexander**, Terry Pratchett, Roald Dahl, Jorge Luis Borges, *Stephen King*, Ian Fleming
Anthony LaPaglia `performed in`	*Lantana*, *Summer of Sam*, *Happy Feet*, *The House of Mirth*, Unfaithful, **Legend of the Guardians**, Naked Lunch, X-Men, The Namesake
Camden County `adjoins`	**Burlington County**, *Atlantic County*, *Gloucester County*, Union County, Essex County, New Jersey, Passaic County, Ocean County, Bucks County
The 40-Year-Old Virgin `nominated for`	*MTV Movie Award for Best Comedic Performance*, *BFCA Critics' Choice Award for Best Comedy*, *MTV Movie Award for Best On-Screen Duo*, MTV Movie Award for Best Breakthrough Performance, **MTV Movie Award for Best Movie**, MTV Movie Award for Best Kiss, D. F. Zanuck Producer of the Year Award in Theatrical Motion Pictures, Screen Actors Guild Award for Best Actor - Motion Picture
Costa Rica football team `has position`	*Forward*, *Defender*, *Midfielder*, **Goalkeepers**, Pitchers, Infielder, Outfielder, Center, Defenseman
Lil Wayne `born in`	**New Orleans**, Atlanta, Austin, St. Louis, Toronto, New York City, Wellington, Dallas, Puerto Rico
WALL-E `has the genre`	Animations, Computer Animation, *Comedy film*, *Adventure film*, *Science Fiction*, **Fantasy**, Stop motion, *Satire*, Drama

Bordes et al. (2013): *Translating Embeddings for Modeling Multi-relational Data*

Entities in Commonsense Knowledge Bases

given: (*cake*, UsedFor, *satisfy hunger*) and (*cake*, IsA, *dessert*)
predict: (*dessert*, UsedFor, *satisfy hunger*)



Li et al. (2016): *Commonsense Knowledge Base Completion*

Stories and Endings

- story:

  Jennifer has a big exam tomorrow. She got so stressed, she pulled an all-nighter. She went into class the next day, weary as can be. Her teacher stated that the test is postponed for next week.

- ending:

  Jennifer felt bittersweet about it.

Source: ROC Story Corpus (Mostafazadeh et al., 2016)

- other examples/applications?
- sometimes direction matters, sometimes not
- sometimes there is a particular kind of relation being named for each pair, sometimes not (i.e., just one kind)

- why don't we just view these tasks as classification? **(Q2 on handout)**
  - we are only given examples of similar things
  - may be infeasible to iterate over the space of dissimilar things ("negative examples") because the space is unbounded or very large
  - there may be multiple things that are similar to something but may not be in our dataset

contamination	pollution
converged	convergence
captioned	subtitled
broad	general
permanent	permanently
…	…

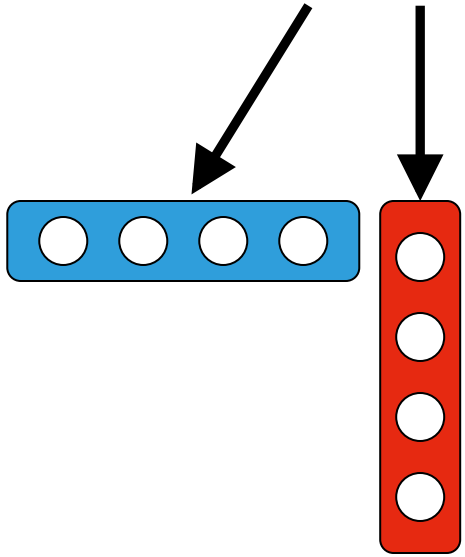
- why don't we just view these tasks as generation? (i.e., given something, generate the other thing)
  - generation is difficult to model and may be more time-consuming to train
  - we may not need to generate anything at test time; we may only need to compute similarities

Similarity Functions

- many choices for similarity functions
- we'll go over some in the next few slides
- throughout, keep in mind:
  - output range
  - symmetric? $sim(\mathbf{x}, \mathbf{y}) = sim(\mathbf{y}, \mathbf{x})$
  - introduces new parameters?
  - connections among similarity functions?
  - notes/tips on using these

Dot Product

$$sim(\mathbf{x}, \mathbf{y}) = \mathbf{x}^\top \mathbf{y}$$



range? $\mathbb{R}$

symmetric or asymmetric? **symmetric**

introduces parameters? **no**

Cosine Similarity

$$sim(\mathbf{x}, \mathbf{y}) = \cos(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x}^\top \mathbf{y}}{\|\mathbf{x}\| \, \|\mathbf{y}\|}$$

$$\|\mathbf{x}\| = \sqrt{\sum_i x_i^2}$$

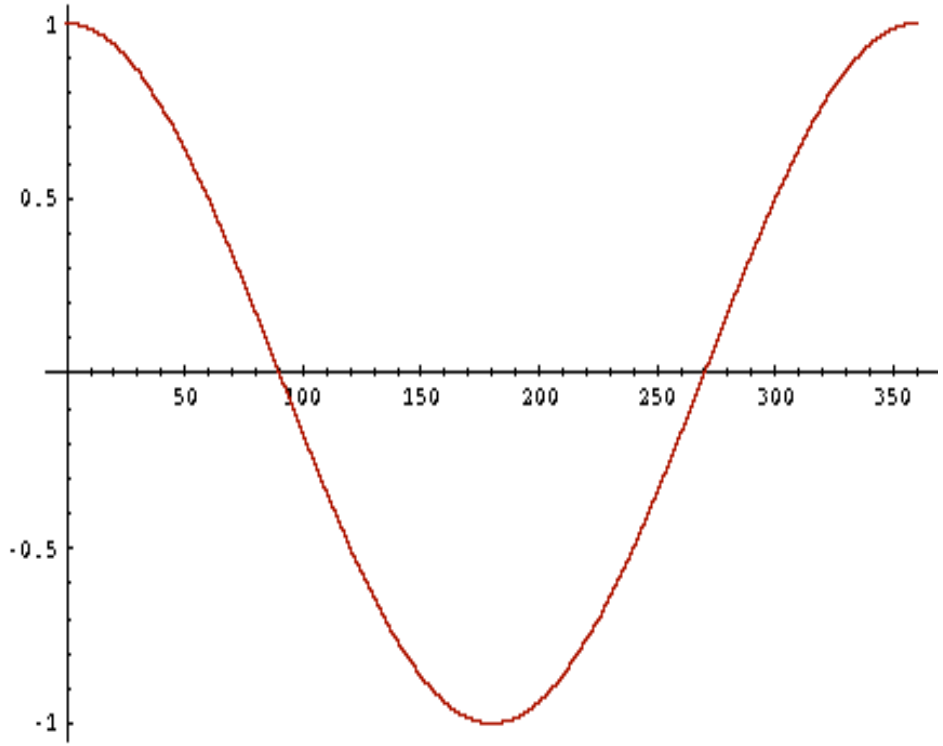
range? $[-1, 1]$

symmetric or asymmetric? **symmetric**

introduces parameters? **no**

generalizes anything? **dot product when vectors have norm 1**

Cosine Similarity = Cosine of angle between vectors

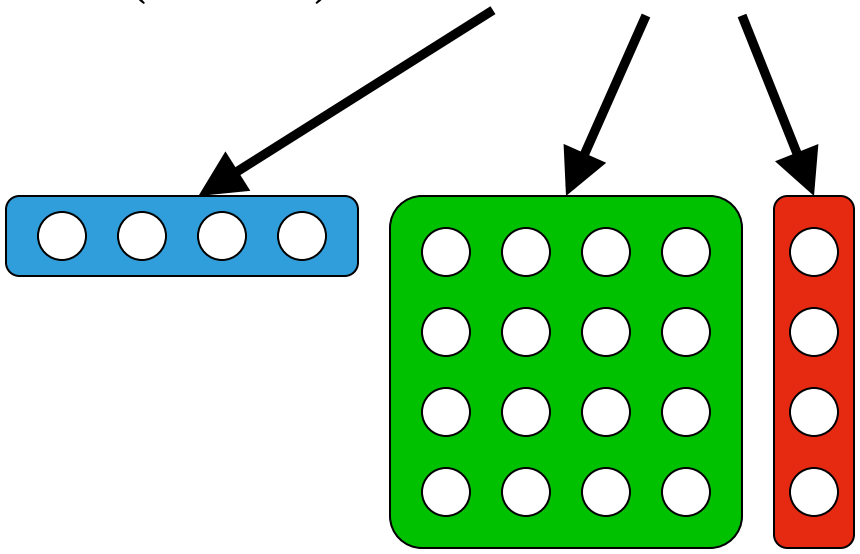


- -1: vectors point in opposite directions
- +1: vectors point in same directions
- 0: vectors are orthogonal

J&M/SLP3

Bilinear Function

$$sim(\mathbf{x}, \mathbf{y}) = \mathbf{x}^\top \mathbf{W} \mathbf{y}$$



range? $\mathbb{R}$

symmetric or asymmetric? **asymmetric in general**

introduces parameters? **yes**

generalizes anything? **dot product if $\mathbf{W}$ is identity**

Notes on Using Bilinear Functions

$$sim(\mathbf{x}, \mathbf{y}) = \mathbf{x}^\top \mathbf{W} \mathbf{y}$$

similarity can depend on relation by using different bilinear weight matrices for different relations:

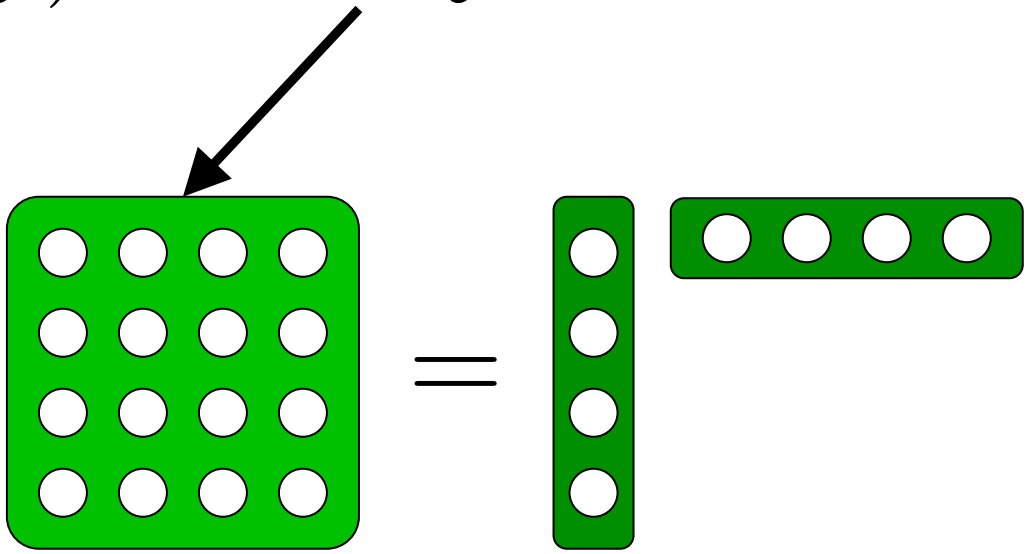
$$sim(\mathbf{x}, r, \mathbf{y}) = \mathbf{x}^\top \mathbf{W}_r \, \mathbf{y}$$

potential issue: $\mathbf{W}$ might have a lot of parameters

- with bilinear similarity functions, there may be too many parameters to learn in $\mathbf{W}$, especially when using large dimensionalities and separate matrices for each relation
- how should we handle this? **(Q3 on handout)**
  - initialize $\mathbf{W}$ to the identity matrix and regularize back to it
  - use a low-rank matrix factorization parameterization (next slide)

Concise Parameterizations of the Bilinear Matrix

$$sim(\mathbf{x}, \mathbf{y}) = \mathbf{x}^\top \mathbf{W} \mathbf{y}$$



e.g., we can parameterize $\mathbf{W}$ as the outer product of two vectors and only learn the two vectors

(Negative) Squared L2 Distance

$$sim(\mathbf{x}, \mathbf{y}) = -\|\mathbf{x} - \mathbf{y}\|_2^2$$

$$\|\mathbf{x}\|_2 = \|\mathbf{x}\| = \sqrt{\sum_i x_i^2}$$

range? $\mathbb{R}_{\leq 0}$

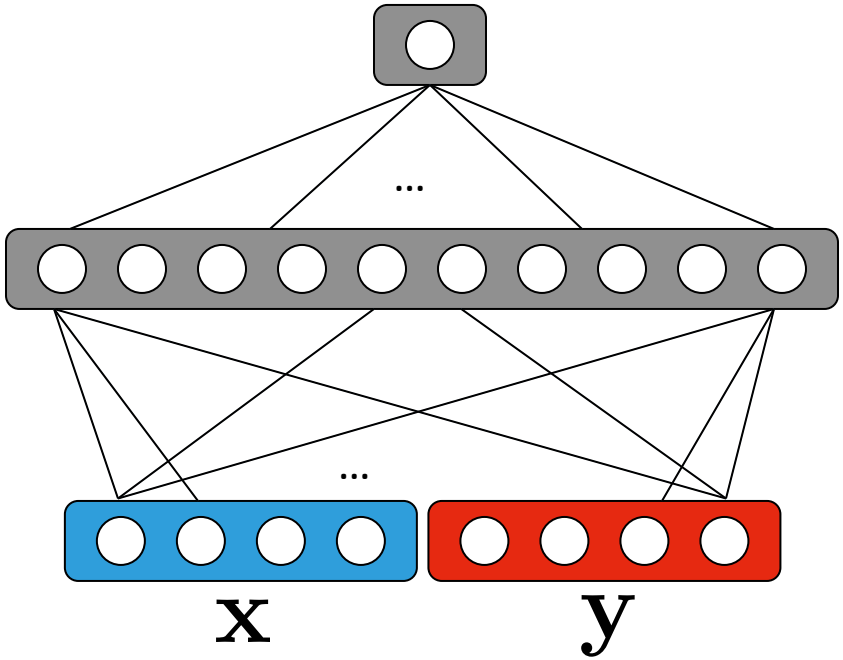
symmetric or asymmetric? **symmetric**

introduce parameters? **no**

Multi-Layer Perceptron (MLP)

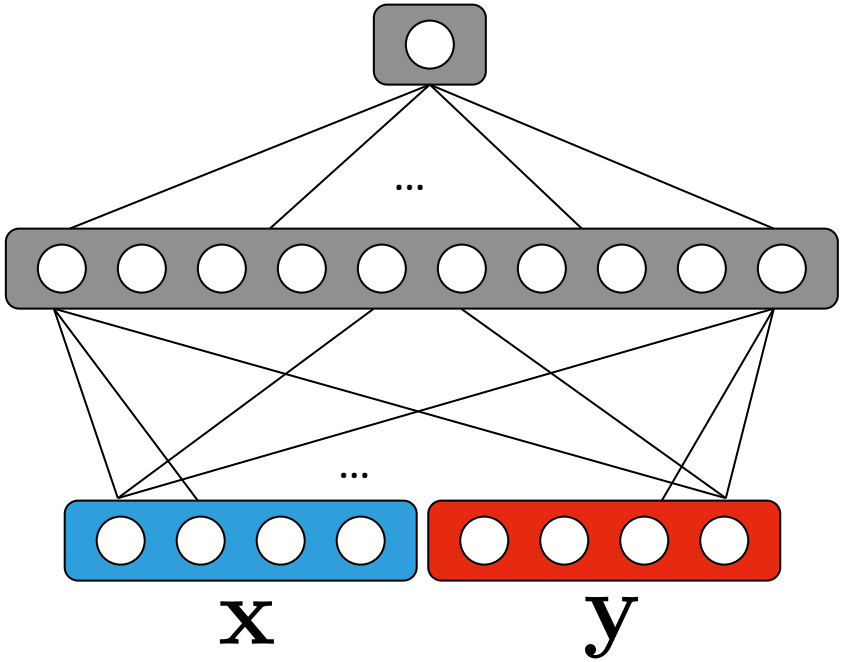
$$sim(\mathbf{x}, \mathbf{y}) = MLP(\text{cat}(\mathbf{x}, \mathbf{y}))$$

concatenate vectors, pass to MLP, use scalar for final output:



Multi-Layer Perceptron

$$sim(\mathbf{x}, \mathbf{y}) = MLP(\text{cat}(\mathbf{x}, \mathbf{y}))$$



range? **depends on nonlinearity**

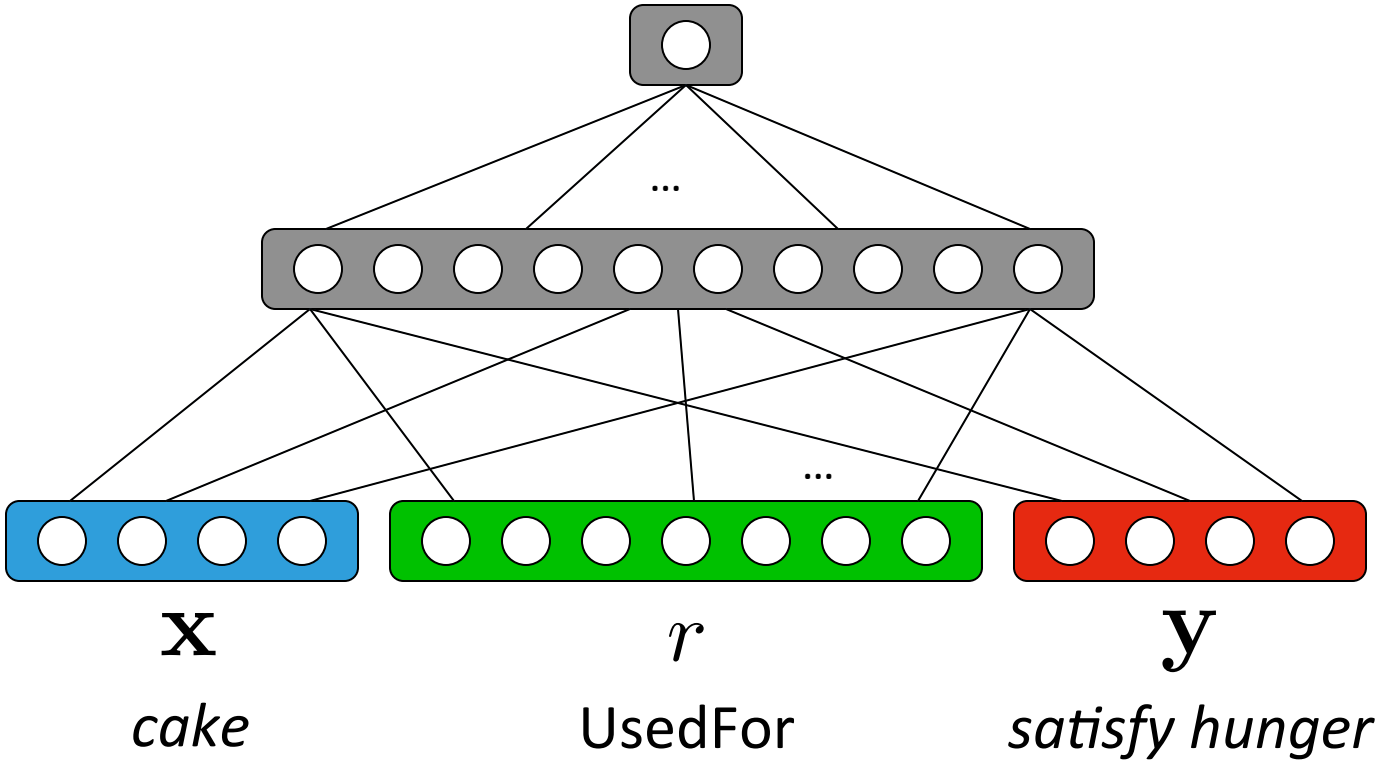
symmetric? **asymmetric**

introduces parameters? **yes**

generalizes anything? **yes, can represent any function!**

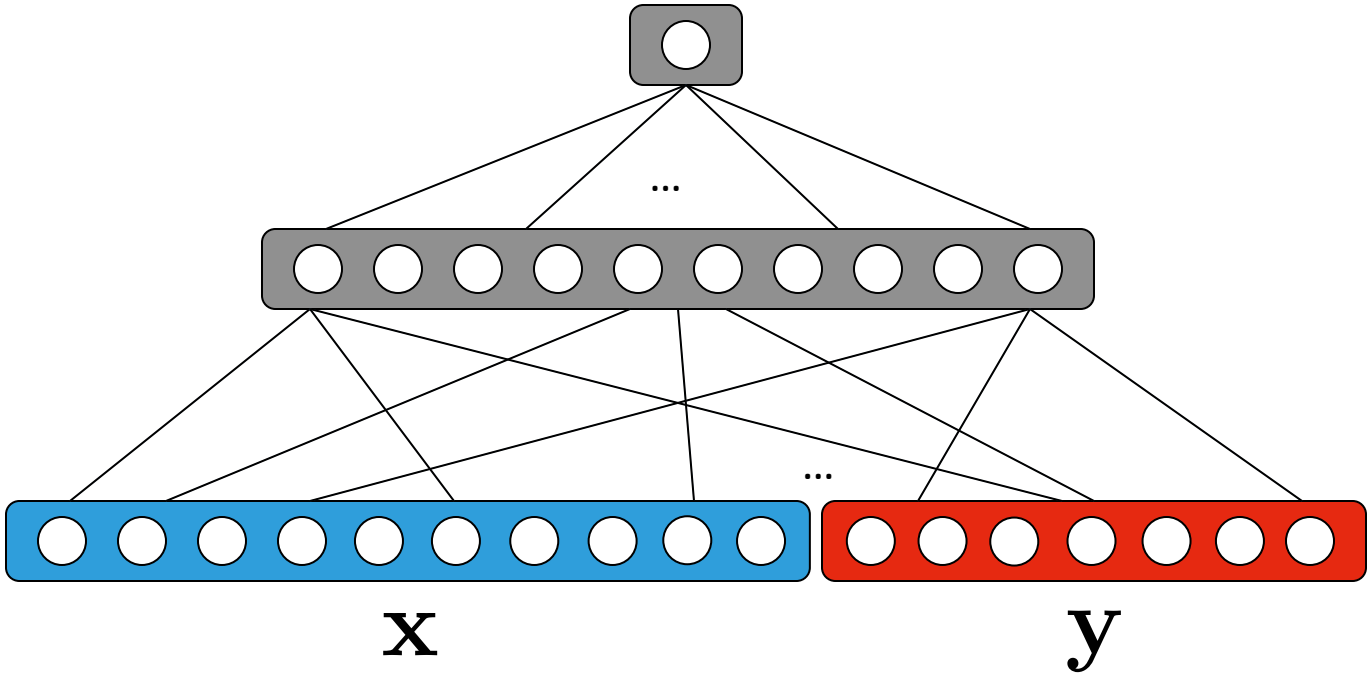
Notes on MLP Similarity Functions

similarity can easily depend on relation:



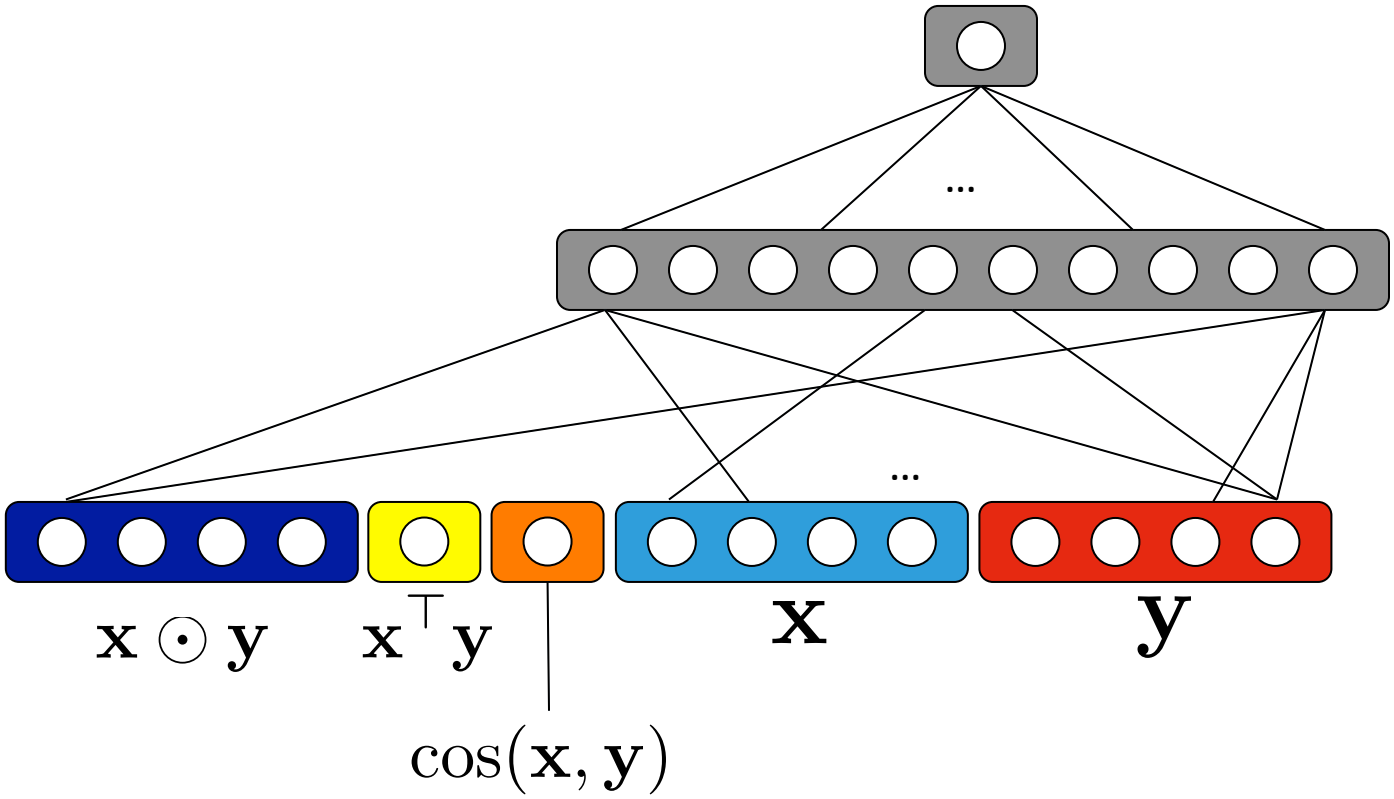
Notes on MLP Similarity Functions

can handle different dimensionalities of the two items:



Notes on MLP Similarity Functions

since MLPs are so expressive and not constrained, things could go horribly wrong. in practice, often pass additional quantities:

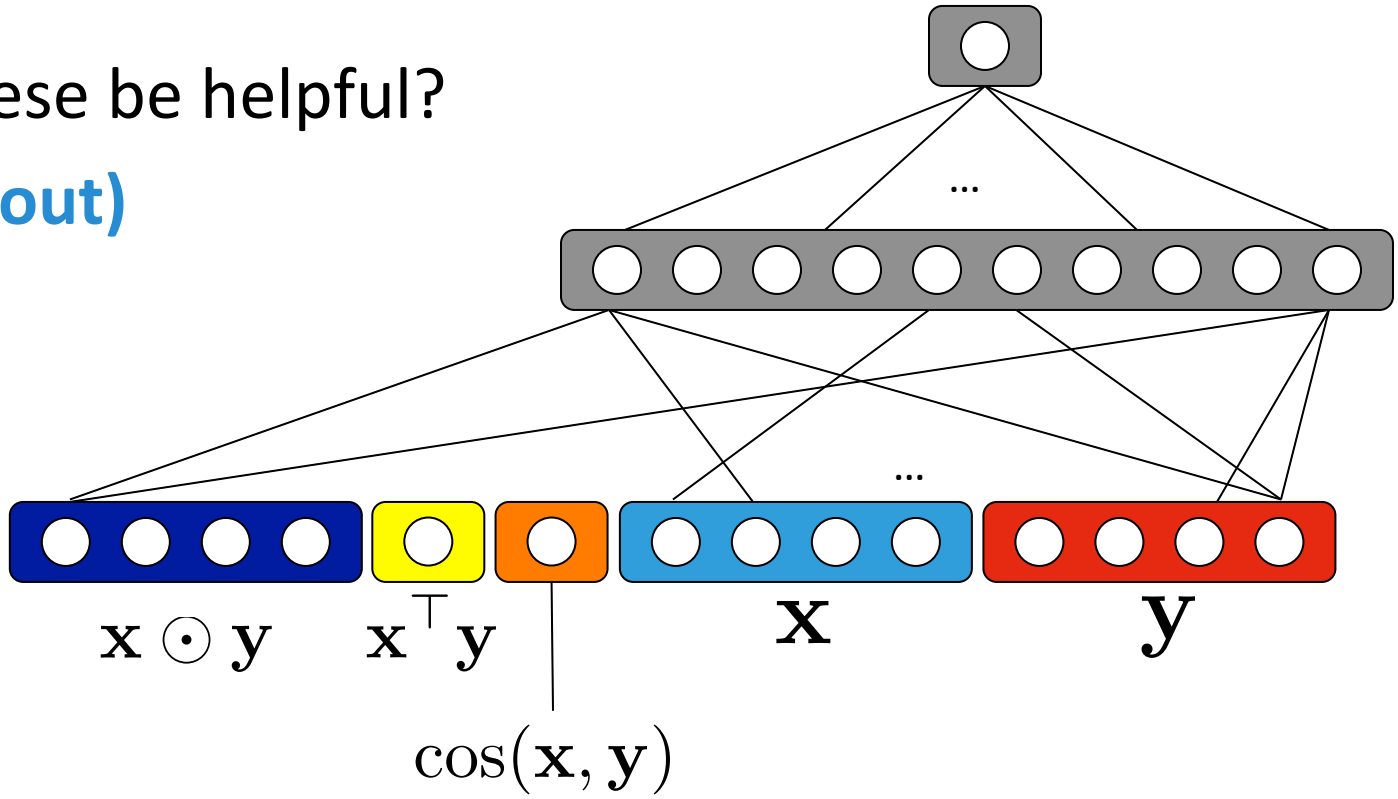


Notes on MLP Similarity Functions

since MLPs are so expressive and not constrained, things could go horribly wrong. in practice, often pass additional quantities:

when will these be helpful?

(Q4 on handout)



When are these terms helpful?

- helpful:
  - when we have identical or tied parameters for the two encoders
  - when we want them to be in the same space

- not necessarily helpful if:
  - no tied parameters in the encoders of the two items
  - the two items have different dimensionalities
  - we don't care if they are in the same space

- we talked about similarity functions and functional architectures
- now let's move on to **learning**
- we have pairs of structured objects

$$\langle \boldsymbol{x}_1, \boldsymbol{x}_2 \rangle$$

Notation

$\mathbf{u} =$ a vector

$u_i =$ entry *i* in the vector

$\mathbf{W} =$ a matrix

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

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

- we talked about similarity functions and functional architectures
- now let's move on to **learning**
- we have pairs of structured objects

$$\langle \boldsymbol{x}_1, \boldsymbol{x}_2 \rangle$$

Learning for Similarity

- We want to learn input representation function $f_{\boldsymbol{\theta}}$ as well as any parameters of similarity function
- We'll just write all these parameters as $\boldsymbol{\theta}$
- How about this loss? **(Q5 on handout)**

$$\min_{\boldsymbol{\theta}} \sum_{\langle \boldsymbol{x}_1, \boldsymbol{x}_2 \rangle \in \mathcal{T}} -sim(f_{\boldsymbol{\theta}}(\boldsymbol{x}_1), f_{\boldsymbol{\theta}}(\boldsymbol{x}_2))$$

- Any potential problems with this?