Intro to Word Embeddings/Language Model

Tf-idf, One hot encodings are sparse representations

tf-idf vectors are

- **long** (length |V|= 20,000 to 50,000)
- **sparse** (most elements are zero)

Slide Credit: Dan Jurafsky and James H.

Sparse versus dense vectors

Why dense vectors?

- Short vectors may be easier to use as **features** in machine learning (less weights to tune)
- Dense vectors may **generalize** better than storing explicit counts
- They may do better at capturing synonymy:
  - *car* and *automobile* are synonyms; but are distinct dimensions
    - a word with *car* as a neighbor and a word with *automobile* as a neighbor should be similar, but aren't
- **In practice, they work better**

Slide Credit: Dan Jurafsky and James H. Martin

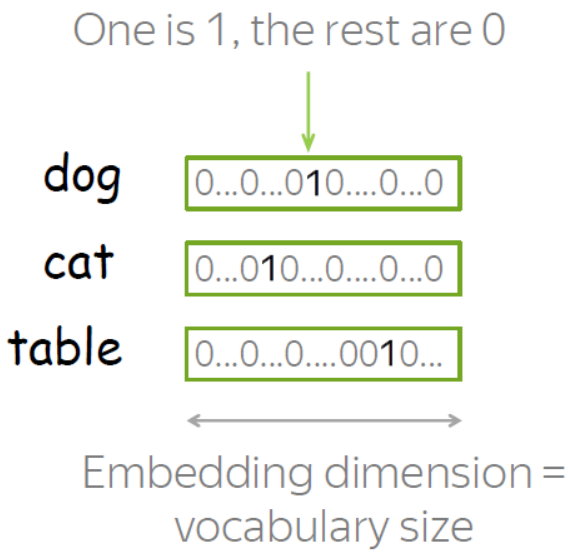
Dense embeddings you can download!

Word2vec (Mikolov et al.)
https://code.google.com/archive/p/word2vec/

Fasttext http://www.fasttext.cc/

Glove (Pennington, Socher, Manning)
http://nlp.stanford.edu/projects/glove/

One-Hot Vectors: Represent Words as Discrete Symbols



Problems:

- Vector size is too large
- Vectors know nothing about meaning

e.g., **cat** is as close to **dog** as it is to **table**!

What is meaning?

Slide Credit : Leena Voita https://drive.google.com/file/d/1y2GKIKBzie7l8iycBO6gTKGiTTfJc4Dr/view

What is meaning?

Do you know what the word tezgüino means ?

(We hope you do not)

Slide Credit : Leena Voita https://drive.google.com/file/d/1y2GKIKBzie7l8iycBO6gTKGiTTfJc4Dr/view

What is meaning?

Now look how this word is used in different contexts:

A bottle of tezgüino is on the table.

Everyone likes tezgüino.

Tezgüino makes you drunk.

We make tezgüino out of corn.

Can you understand what tezgüino means ?

Slide Credit : Leena Voita https://drive.google.com/file/d/1y2GKIKBzie7l8iycBO6gTKGiTTfJc4Dr/view

What is meaning?

Now look how this word is used in different contexts:

A bottle of tezgüino is on the table.
Everyone likes tezgüino.
Tezgüino makes you drunk.
We make tezgüino out of corn.

→ Tezgüino is a kind of alcoholic beverage made from corn.

With context, you can understand the meaning!

Slide Credit : Leena Voita https://drive.google.com/file/d/1y2GKIKBzie7l8iycBO6gTKGiTTfJc4Dr/view

What is meaning?

(1) A bottle of _____ is on the table.

(2) Everyone likes _____ .

(3) _____ makes you drunk.

(4) We make _____ out of corn.

What other words fit into these contexts ?

	(1)	(2)	(3)	(4)	…
tezgüino	1	1	1	1	
loud	0	0	0	0	
motor oil	1	0	0	1	
tortillas	0	1	0	1	
wine	1	1	1	0	

← contexts

← rows show contextual properties: 1 if a word can appear in the context, 0 if not

Slide Credit : Leena Voita https://drive.google.com/file/d/1y2GKIKBzie7l8iycBO6gTKGiTTfJc4Dr/view

What is meaning?

(1) A bottle of _____ is on the table.

(2) Everyone likes _____ .

(3) _____ makes you drunk.

(4) We make _____ out of corn.

	(1)	(2)	(3)	(4)	...
tezgüino	1	1	1	1	
loud	0	0	0	0	
motor oil	1	0	0	1	
tortillas	0	1	0	1	
wine	1	1	1	0	

This is the **distributional hypothesis**

rows are similar → meanings of the words are similar

Slide Credit : Leena Voita https://drive.google.com/file/d/1y2GKIKBzie7l8iycBO6gTKGiTTfJc4Dr/view

Distributional Hypothesis

> Words which frequently appear in **similar contexts** have **similar meaning**.

(Harris 1954, Firth 1957)

This can be used in practice to build word vectors!

Slide Credit : Leena Voita https://drive.google.com/file/d/1y2GKIKBzie7l8iycBO6gTKGiTTfJc4Dr/view

Word2vec

Popular embedding method

Very fast to train

Code available on the web

Idea: **predict** rather than **count**

Slide Credit: Dan Jurafsky and James H.

Word2vec

- Instead of **counting** how often each word *w* occurs near "*apricot*"
- Train a classifier on a binary **prediction** task:
  - Is *w* likely to show up near "*apricot*"?

- We don’t actually care about this task
  - But we'll take the learned classifier weights as the word embeddings

Slide Credit: Dan Jurafsky and James H.

Brilliant insight: Use running text as implicitly supervised training data!

- A word *s* near *apricot*
  - Acts as gold ‘correct answer’ to the question
  - “Is word *w* likely to show up near *apricot*?”
- No need for hand-labeled supervision

Slide Credit: Dan Jurafsky and James H.

Word2Vec: **Skip-Gram** Task

Word2vec provides a variety of options. Let's do

- "skip-gram with negative sampling" (SGNS)

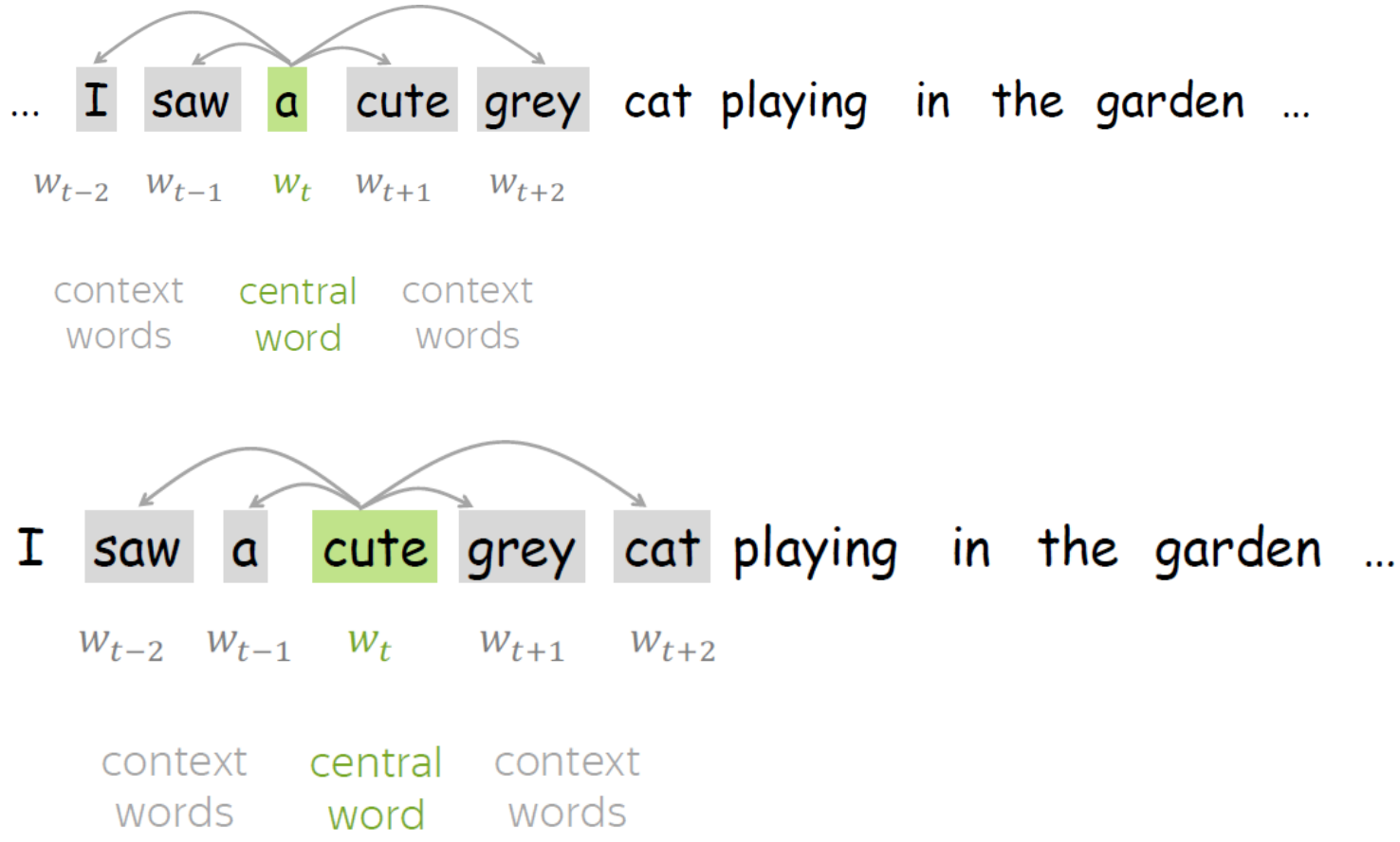
Slide Credit: Dan Jurafsky and James H.

Skip-gram algorithm

1. Treat the target word and a neighboring context word as positive examples.
2. Randomly sample other words in the lexicon to get negative samples
3. Use logistic regression to train a classifier to distinguish those two cases
4. Use the weights as the embeddings

Slide Credit: Dan Jurafsky and James H.

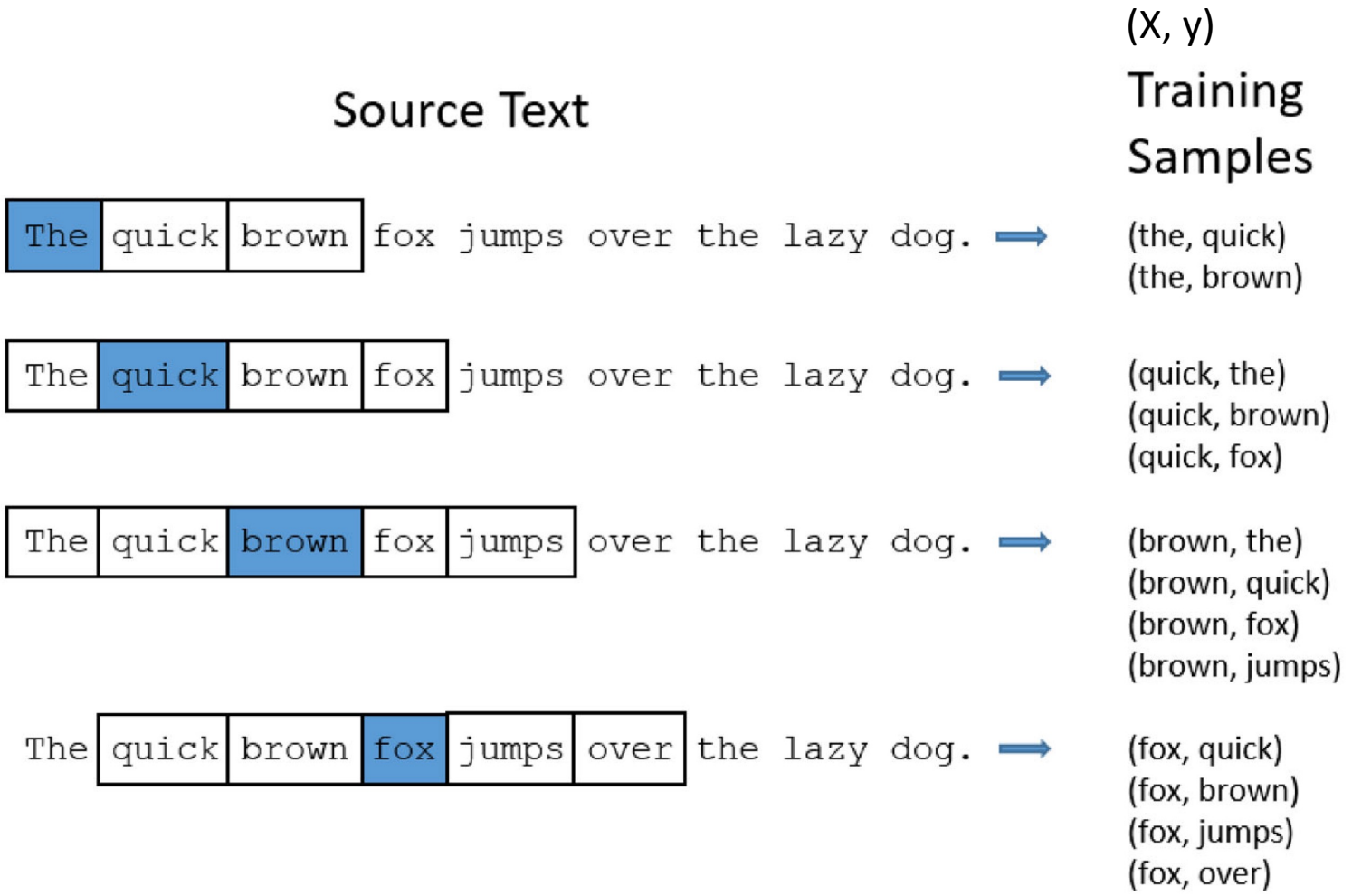
Skip-Gram Training Data

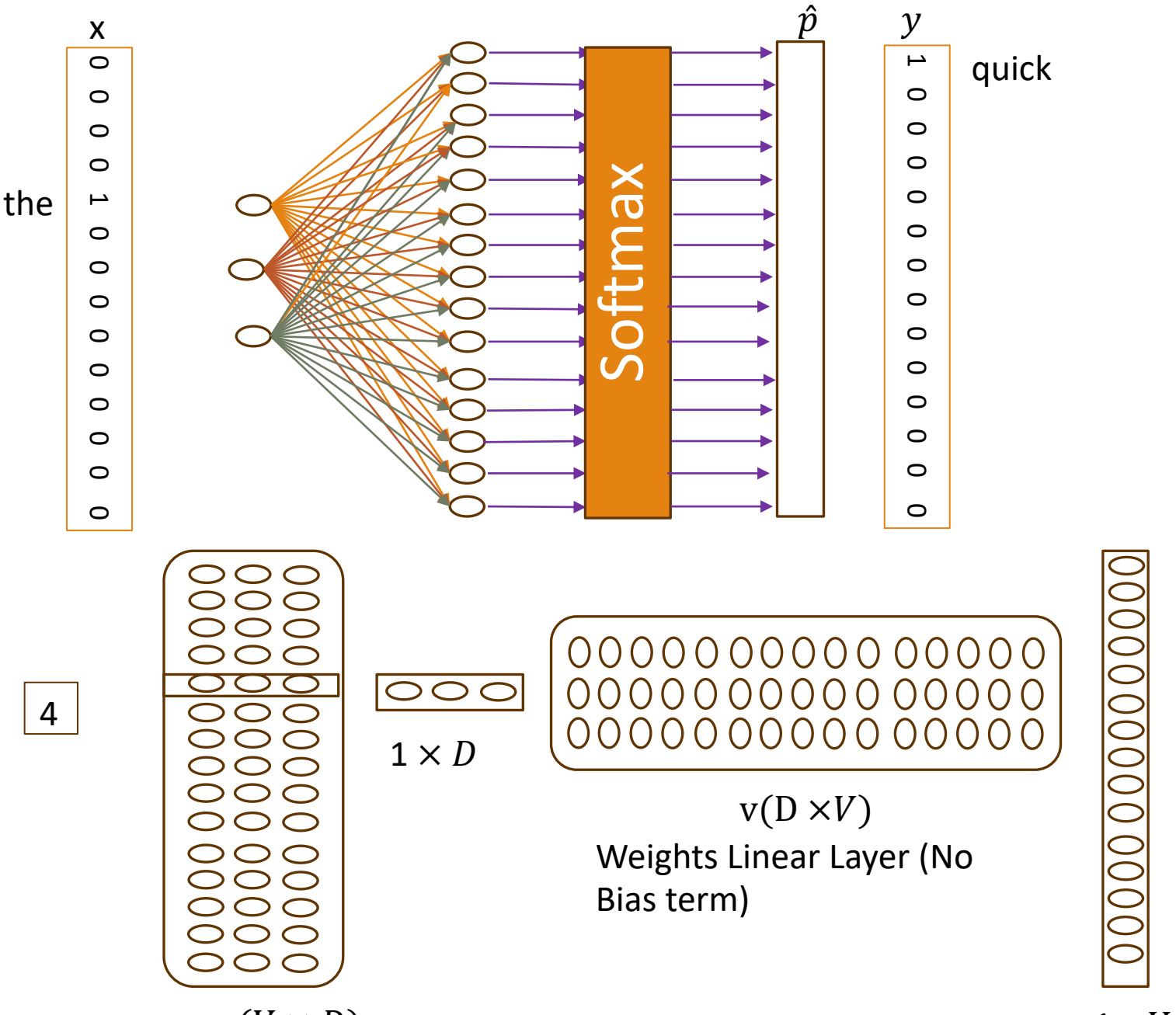


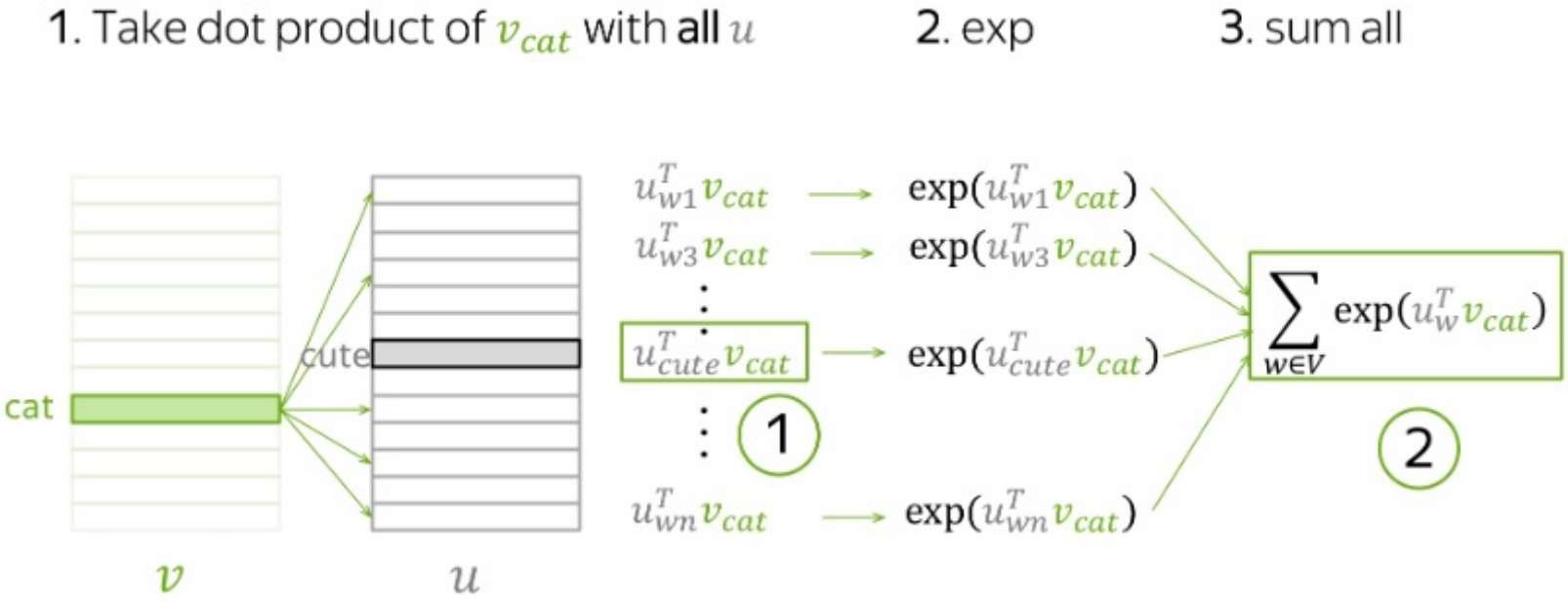
Given a central word (X), predict context words (y).

Slide Credit : Leena Voita https://drive.google.com/file/d/1y2GKIKBzie7l8iycBO6gTKGiTTfJc4Dr/view

Skip-Gram Training Data




x
the
Softmax
$\hat{p}$
y
quick
4
$1 \times D$
$u\ (V \times D)$
Weights Embedding Layer
input
$\mathrm{v}(\mathrm{D} \times V)$
Weights Linear Layer (No Bias term)
$1 \times V$



Softmax: -(log(num/den) = -(log(num) – log(den)= -log(num) + log(den)

4. get loss (for this one step)

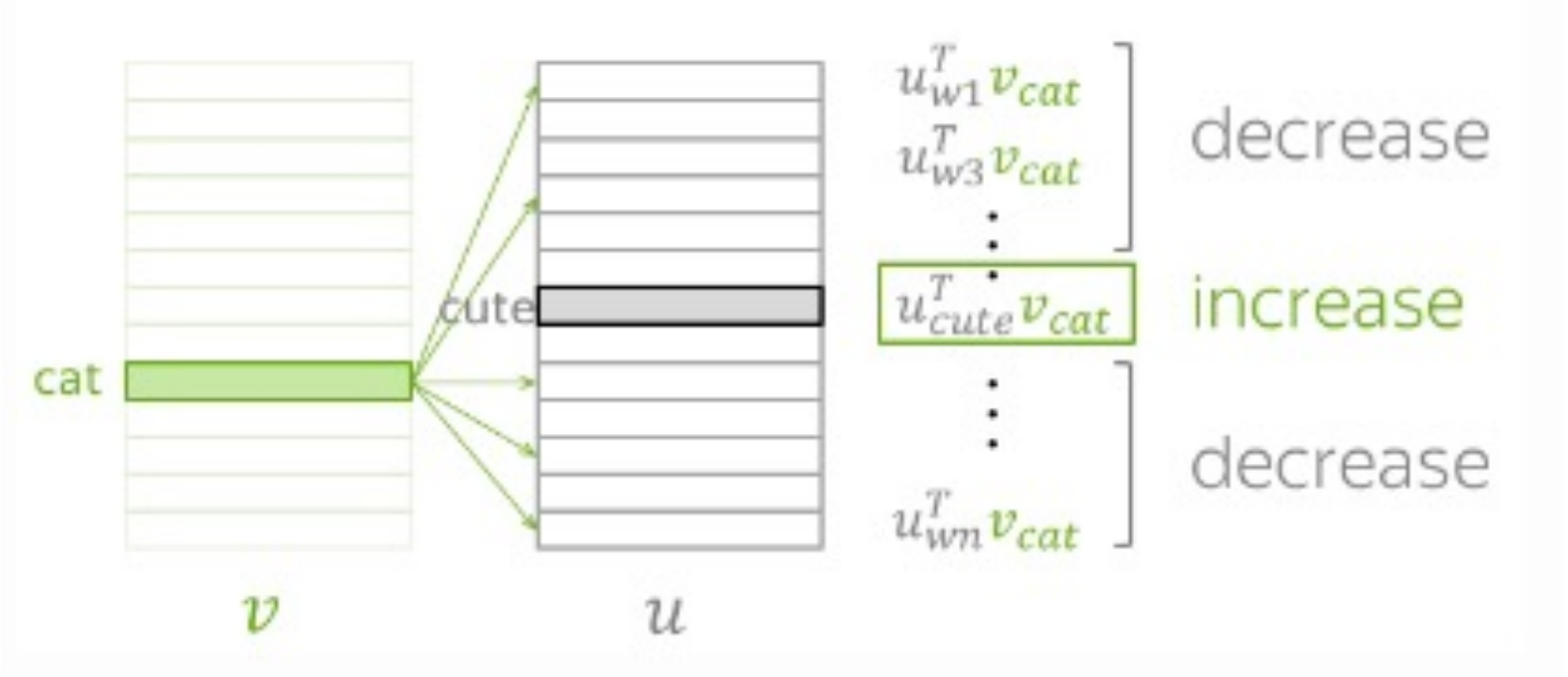
$$J_{t,j}(\theta) = \underbrace{-u_{cute}^T v_{cat}}_{1} + \underbrace{\log \sum_{w \in V} \exp(u_w^T v_{cat})}_{2}$$

5. evaluate the gradient, make an update

$$v_{cat} := v_{cat} - \alpha \frac{\partial J_{t,j}(\theta)}{\partial v_{cat}}$$

$$u_w := u_w - \alpha \frac{\partial J_{t,j}(\theta)}{\partial u_w} \quad \forall w \in V$$

Slide Credit : Leena Voita https://drive.google.com/file/d/1y2GKIKBzie7l8iycBO6gTKGiTTfJc4Dr/view

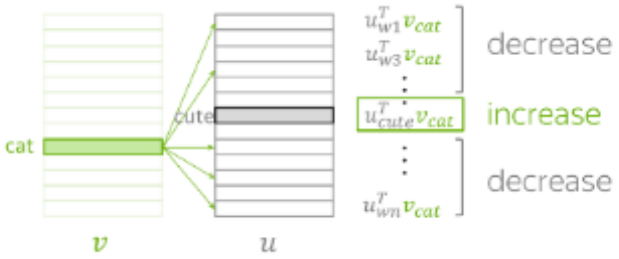


- The doc product increases as the vectors become similar
- The words which appear in a similar context, the embeddings become similar

Slide Credit : Leena Voita https://drive.google.com/file/d/1y2GKIKBzie7l8iycBO6gTKGiTTfJc4Dr/view

Negative Sampling

Dot product of v_{cat}:
- with u_{cute} - increase,
- with **all other** u - decrease



Parameters to be updated:
- v_{cat}
- u_w for all w in the vocabulary — |V| + 1 vectors

Dot product of v_{cat}:
- with u_{cute} - increase,
- with **a subset of other** u - decrease

Negative samples: randomly selected K words

Parameters to be updated:
- v_{cat}
- u_{cute} and u_w for w in K negative examples — K + 2 vectors

Selecting Negative Samples

Could pick w according to their unigram frequency P(w)

More common to chosen then according to $p_\alpha(w)$

$$P_\alpha(w) = \frac{count(w)^\alpha}{\sum_w count(w)^\alpha}$$

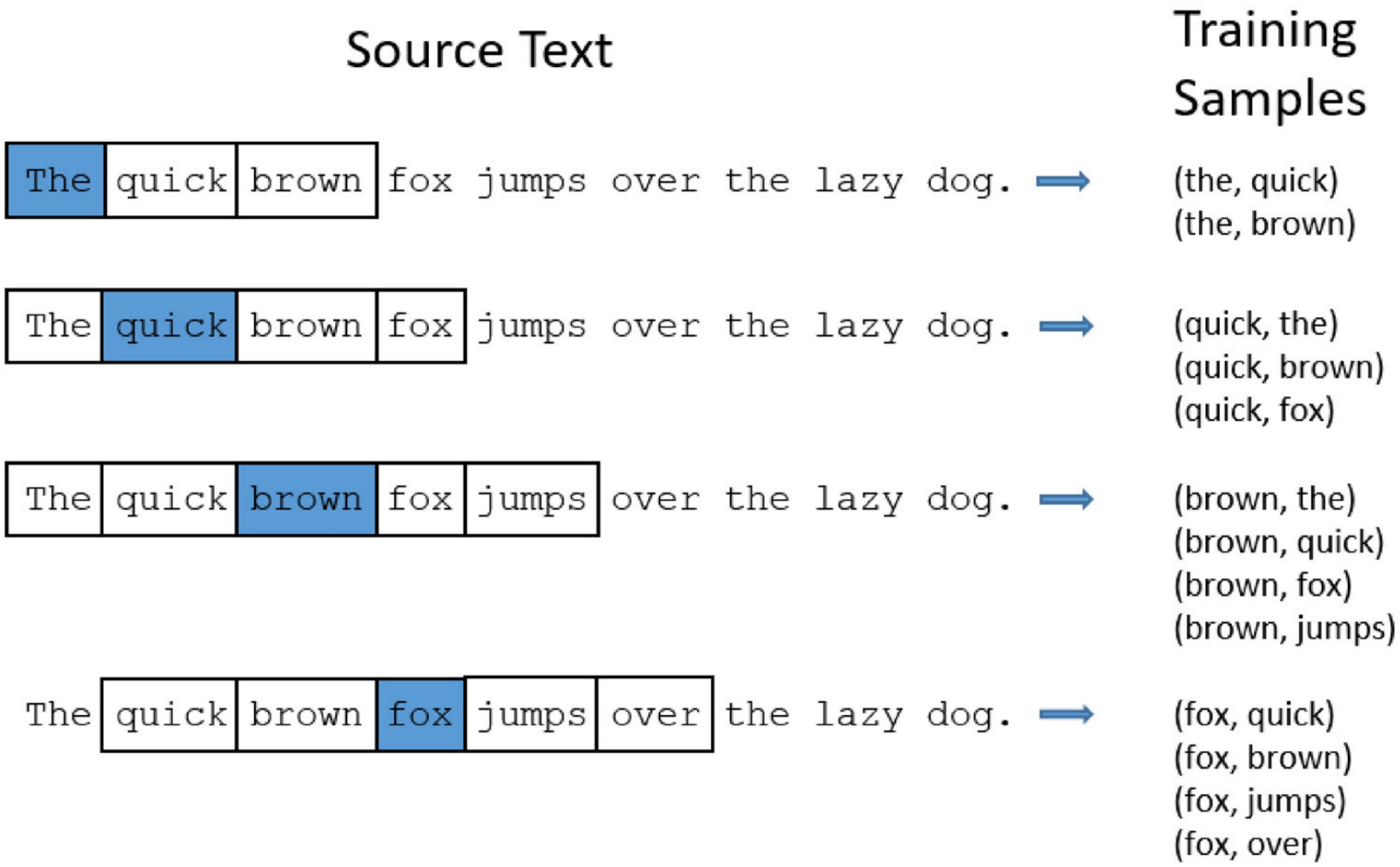
α= ¾ works well because it gives rare noise words slightly higher probability

To show this, imagine two events p(a)=.99 and p(b) = .01:

$$P_\alpha(a) = \frac{.99^{.75}}{.99^{.75} + .01^{.75}} = .97$$

$$P_\alpha(b) = \frac{.01^{.75}}{.99^{.75} + .01^{.75}} = .03$$

Sample


Source Text
Training Samples
The quick brown fox jumps over the lazy dog.
(the, quick)
(the, brown)
The quick brown fox jumps over the lazy dog.
(quick, the)
(quick, brown)
(quick, fox)
The quick brown fox jumps over the lazy dog.
(brown, the)
(brown, quick)
(brown, fox)
(brown, jumps)
The quick brown fox jumps over the lazy dog.
(fox, quick)
(fox, brown)
(fox, jumps)
(fox, over)

Problem – frequent Words

There are two "problems" with common words like "the":

- When looking at word pairs, ("fox", “and the") do not tell us much about the meaning of "fox". "the" appears in the context of pretty much every word.
- We will have many more samples of ("the", ...) than we need to learn a good vector for "the".

Solution – Sub Sampling

If we have a window size of 10, and we remove a specific instance of "the" from our text:

- As we train on the remaining words, "the" will not appear in any of their context windows.
- We'll have 10 fewer training samples where "the" is the input word.

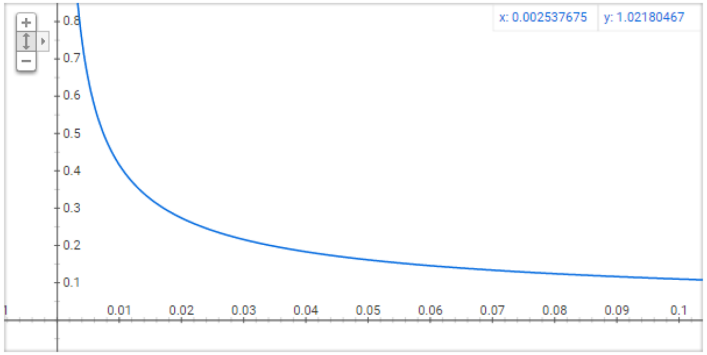
Sampling rate

probability of *keeping* the word:

the fraction of the total words in the corpus that are that word.

$$P(w_i) = (\sqrt{\frac{z(w_i)}{0.001}} + 1) \cdot \frac{0.001}{z(w_i)}$$

Graph for (sqrt(x/0.001)+1)*0.001/x



- $P(w_i) = 1.0 \; if \;\; z(w_i) < -0.0026$
- $P(w_i) = 0.5 \; if \;\; z(w_i) \;\; when \; z(w_i) = 0.00746$
- $P(w_i) = 0.033 \; if \;\; z(w_i) \;\; when \; z(w_i) = 1$
- For Wikipedia only top 27 most frequent words would have any subsampling applied.

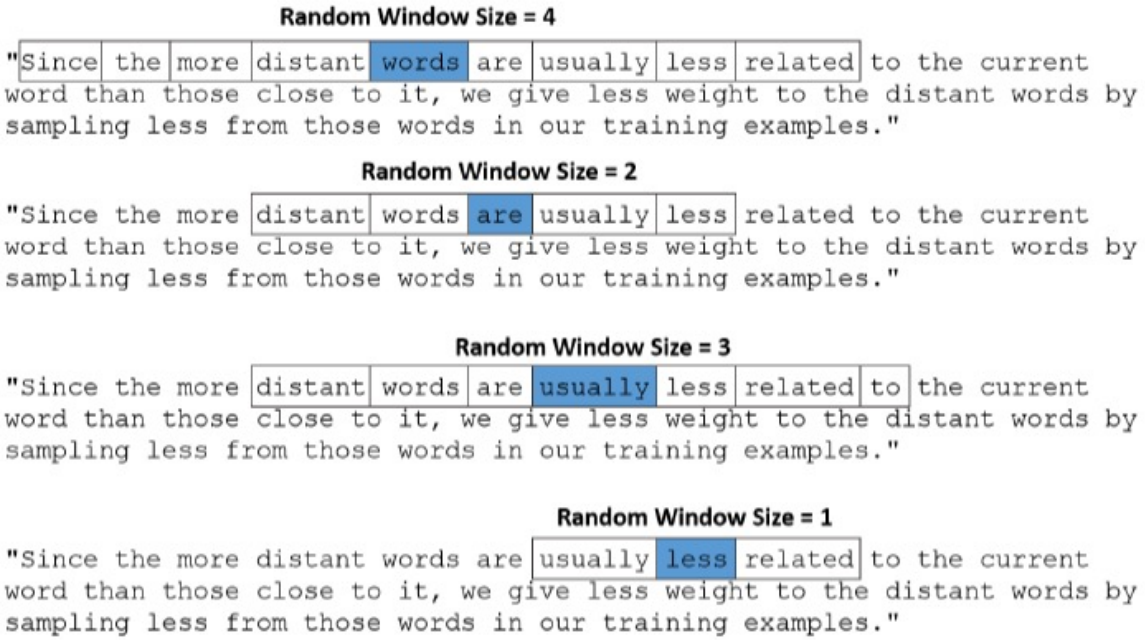
Rank	Word	Portion of Corpus	Probability to Keep
0	the	7.80%	12.60%
1	of	3.71%	19.10%
2	and	3.19%	20.84%

Context Position Weighting

Randomly shrink the size of context window

For example if window size is 5 – This means we can pick any size window with equal probability.

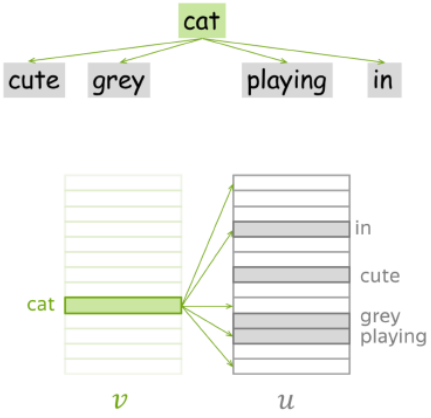
This means that words closest to Target word are more likely to be chosen as positive samples.



Word2Vec Variants: Skip-Gram and CBOW

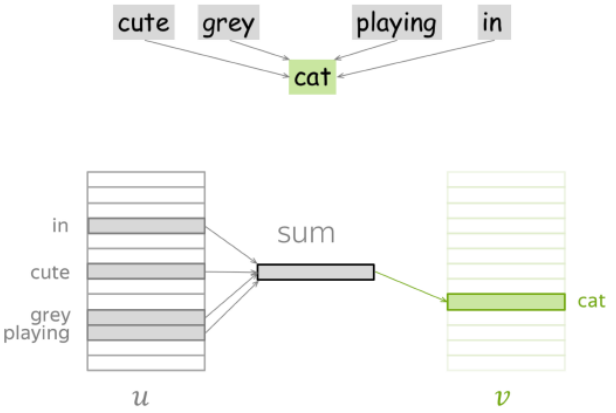
… I saw a cute grey cat playing in the garden …

Skip-Gram: from central predict context (one at a time)



(this is what we did so far)

CBOW: from sum of context predict central



(Continuous Bag of Words)

Slide Credit : Leena Voita https://drive.google.com/file/d/1y2GKIKBzie7l8iycBO6gTKGiTTfJc4Dr/view

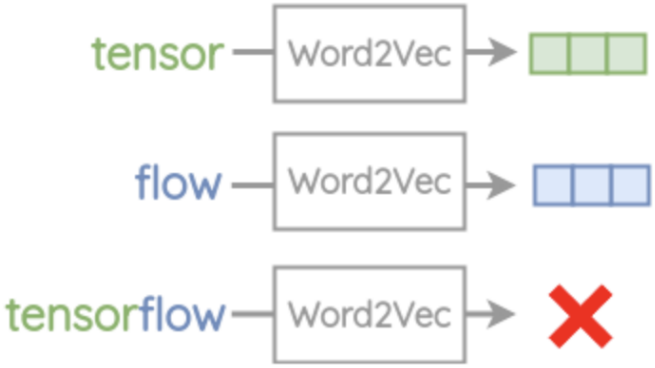
CBOW

Positive Training Samples

Sentence	Skip-Gram	CBOW
The quick brown fox jumps over the lazy dog. →	(the, quick) (the, brown)	((quick, brown), the)
The quick brown fox jumps over the lazy dog. →	(quick, the) (quick, brown) (quick, fox)	((the, brown, fox), quick)
The quick brown fox jumps over the lazy dog. →	(brown, the) (brown, quick) (brown, fox) (brown, jumps)	((the, quick, fox, jumps), brown)
The quick brown fox jumps over the lazy dog. →	(fox, quick) (fox, brown) (fox, jumps) (fox, over)	((quick, brown, jumps, over), fox)

Short Comings of Word2Vec

- Out of Vocabulary Words



- Morphology

Shared radical

eat eats eaten eater eating

Source : https://amitness.com/2020/06/fasttext-embeddings/

FastText – Sub-word Generation

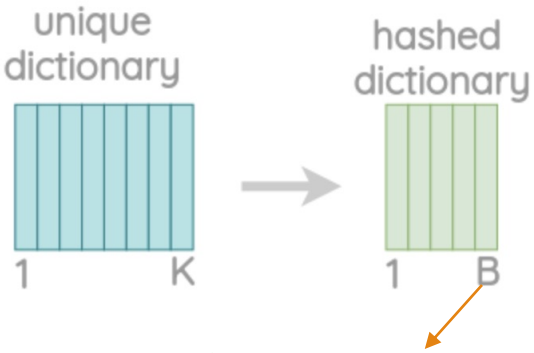
Steps

- (1) Take a word and add angular brackets to denote the beginning and end of a word

- (2) Generate character n-grams of length n

Word	Length(n)	Character n-grams
eating	3	<ea, eat, ati, tin, ing, ng>
eating	4	<eat, eati, atin, ting, ing>
eating	5	<eati, eatin, ating, ting>
eating	6	<eatin, eating, ating>

- 3a. Instead of learning an embedding for each unique n-gram, we learn total B embeddings where B denotes the bucket size



Source : https://amitness.com/2020/06/fasttext-embeddings/

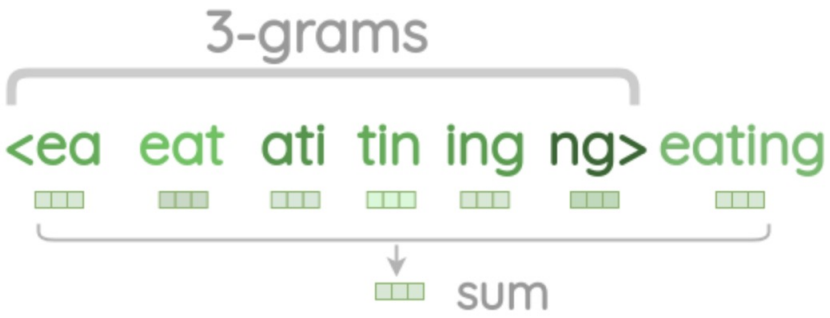
FastText – Embedding

Steps

- 3b. Each character n-gram is hashed to an integer between 1 to B

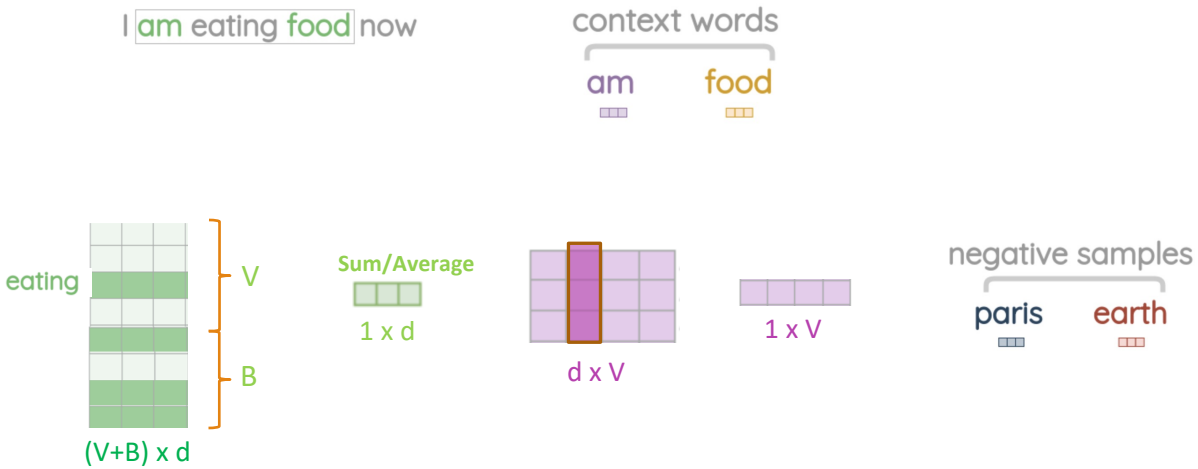
- Many to One Mapping – There could be collisions

- embedding for the center word is calculated by taking a sum of vectors for the character n-grams and the whole word itself

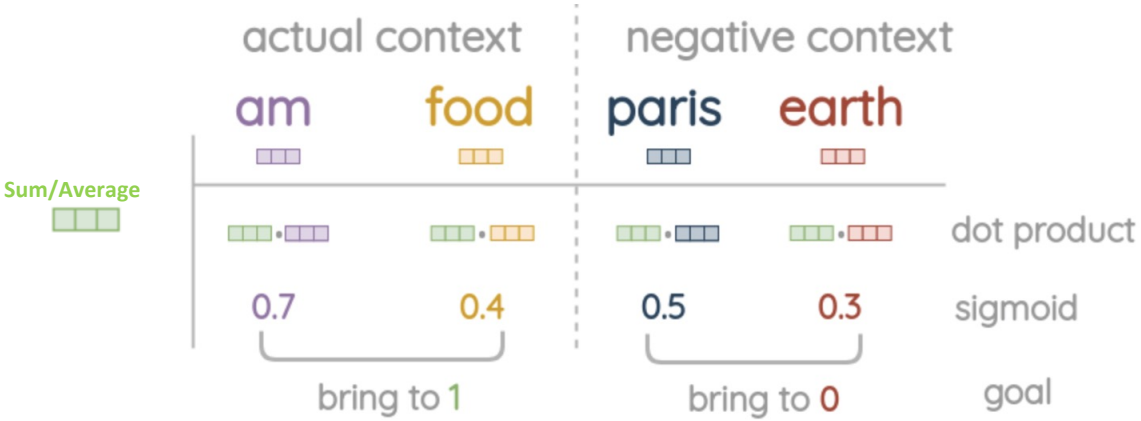


Source : https://amitness.com/2020/06/fasttext-embeddings/

FastText – Training



V= Vocab size, B= Bucket Size, d= embedding size



Source : https://amitness.com/2020/06/fasttext-embeddings/

FastText vs Word2Vec

- FastText – more useful to learn embeddings for rare words and can deal with OOV (Out of off vocabulary) words
- Sub words do not improve representations of common words and may sometimes lead to slightly worse representation
- FastText improves performance on syntactic word analogy tasks significantly for morphologically rich language like Czech and German
- FastText has degraded performance on semantic analogy tasks compared to Word2Vec.
- FastText can be slower to train than regular skipgram

Summary:

- Fasttext is an extension of Word2Vec and has pros and cons.

Source : https://amitness.com/2020/06/fasttext-embeddings/

Capture co-occurrence counts directly?

With a co-occurrence matrix *X*

- 2 options: windows vs. full document
- Window: Similar to word2vec, use window around each word → captures both syntactic (POS) and semantic information
- Word-document co-occurrence matrix will give general topics (all sports terms will have similar entries) leading to "Latent Semantic Analysis"

Source: http://web.stanford.edu/class/cs224n/slides/cs224n-2020-lecture02-wordvecs2.pdf

Window based co-occurrence matrix

- Example corpus:
  - I like deep learning.
  - I like NLP.
  - I enjoy flying

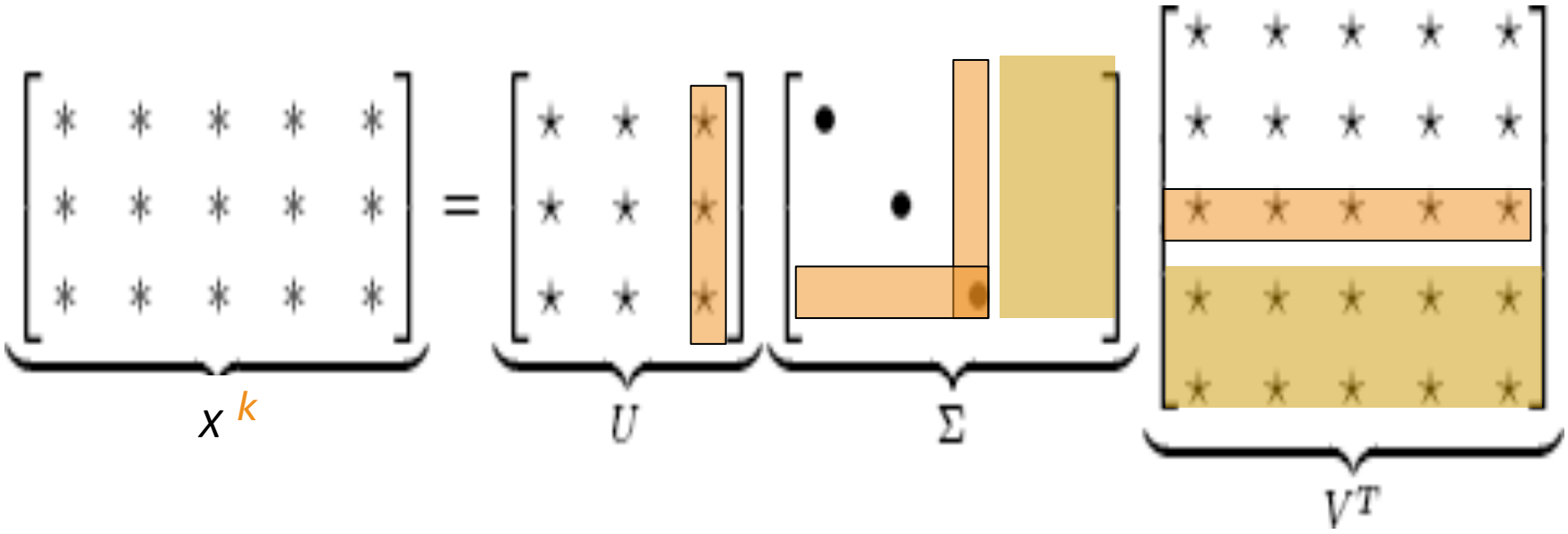
counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

Source: http://web.stanford.edu/class/cs224n/slides/cs224n-2020-lecture02-wordvecs2.pdf

Method: Dimensionality Reduction on X

Singular Value Decomposition of co-occurrence matrix *X*

Factorizes *X* into $U\Sigma V^T$, where *U* and *V* are orthonormal

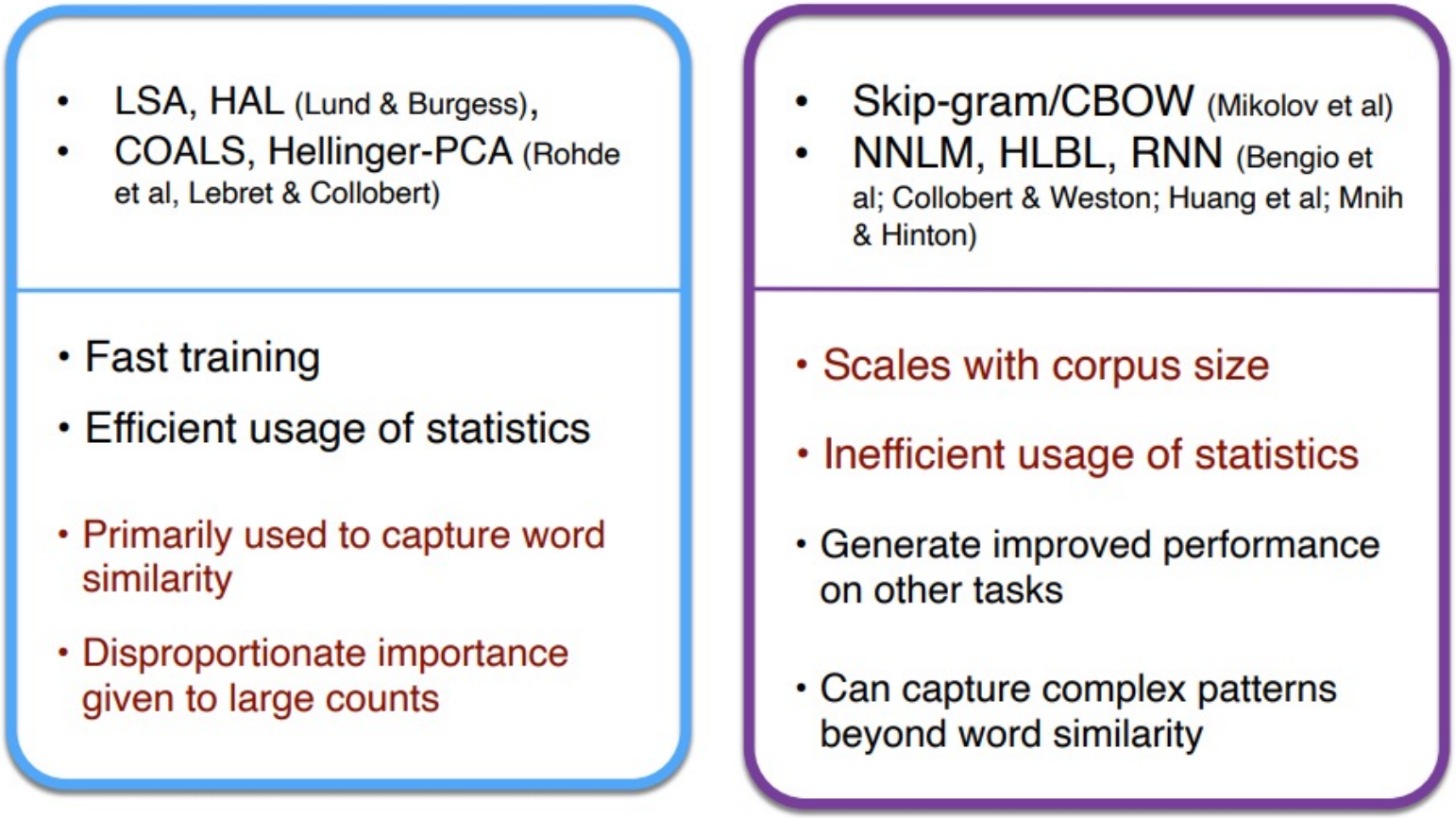


Retain only *k* singular values, in order to generalize.

Expensive to compute for large matrices.

Source: http://web.stanford.edu/class/cs224n/slides/cs224n-2020-lecture02-wordvecs2.pdf

Towards GloVe: Count based vs. direct prediction



Source: http://web.stanford.edu/class/cs224n/slides/cs224n-2020-lecture02-wordvecs2.pdf

GloVe (Global Vectors) model

- Combines the advantages of the two major model families in the literature:
  - global matrix factorization and
  - local context window methods

- It leverages statistical information by training only on the nonzero elements in a word-word co-occurrence matrix, rather than on the entire sparse matrix or on individual context windows in a large corpus.

Original Glove Paper :
https://nlp.stanford.edu/pubs/glove.pdf

Combining the best of both worlds GloVe

[Pennington et al., EMNLP 2014]

$$J = \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2$$

100 3/4

$$f(x) = \begin{cases} (x/x_{\max})^{\alpha} & \text{if } x < x_{\max} \\ 1 & \text{otherwise} \end{cases}$$

- Fast training
- Scalable to huge corpora
- Good performance even with small corpus and small vectors

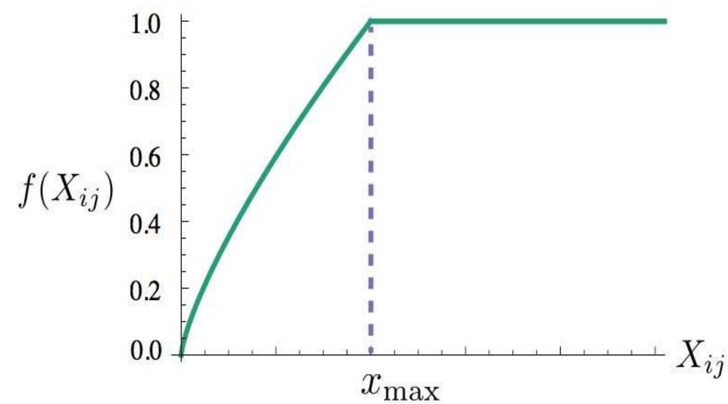


Figure 1: Weighting function f with $\alpha = 3/4$.

Original Glove Paper :
https://nlp.stanford.edu/pubs/glove.pdf

GloVe Results

Nearest words to frog:

1. frogs
2. toad
3. litoria
4. leptodactylidae
5. rana
6. lizard
7. eleutherodactylus

litoria

leptodactylidae

rana

eleutherodactylus

Source: http://web.stanford.edu/class/cs224n/slides/cs224n-2020-lecture02-wordvecs2.pdf

Comparison

Feature/Model	Word2Vec	FastText	GloVe
Core Idea	Predictive model using local context windows	Predictive model using local context windows and subword information	Count-based model using aggregated global word co-occurrence statistics from the entire corpus
Context Window	Fixed size	Fixed size, but includes character n-grams	Uses a fixed-size context window to build the co-occurrence matrix but aggregates statistics across the entire corpus
Handling of OOV Words	Cannot handle OOV words	Can handle OOV words through subword information	Cannot handle OOV words without additional methods
Morphological Richness	Poor (does not consider subword information)	Excellent (captures morphological information)	Poor (does not consider subword information)
Semantic Relationships	Good (especially syntactic analogies)	Good (captures morphological semantics)	Good (captures both global co-occurrence and local context semantics)
Use Cases	General NLP tasks, syntactic relationships	Morphologically rich languages, text classification	Tasks requiring understanding of global word usage patterns
Strengths	Efficient, well-researched, good at capturing syntactic relationships	Handles OOV and morphologically complex words, fast predictions	Captures global statistics efficiently, good for word analogy and similarity
Weaknesses	Struggles with OOV words, does not use global statistics	Larger model size, potential noise from Subwords	Requires significant memory for co-occurrence matrix, less effective with simple morphologies

Evaluating embeddings -Intrinsic

Compare to human scores on word similarity-type tasks:

- WordSim-353 (Finkelstein et al., 2002)
- SimLex-999 (Hill et al., 2015)
- Stanford Contextual Word Similarity (SCWS) dataset (Huang et al., 2012)
- TOEFL dataset: *Levied is closest in meaning to: imposed, believed, requested, correlated*

Source : Chapter 6 of Book: https://web.stanford.edu/~jurafsky/slp3/

Evaluating embeddings -Extrinsic

Use embeddings in Downstream tasks like Classification

Properties of embeddings

Similarity depends on window size C

C = ±2 The nearest words to *Hogwarts:*

- *Sunnydale*
- *Evernight*

C = ±5 The nearest words to *Hogwarts:*

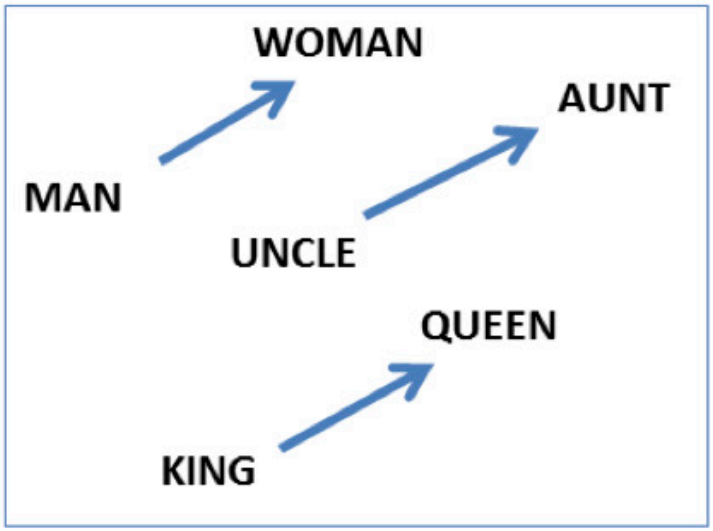
- *Dumbledore*
- *Malfoy*
- *halfblood*

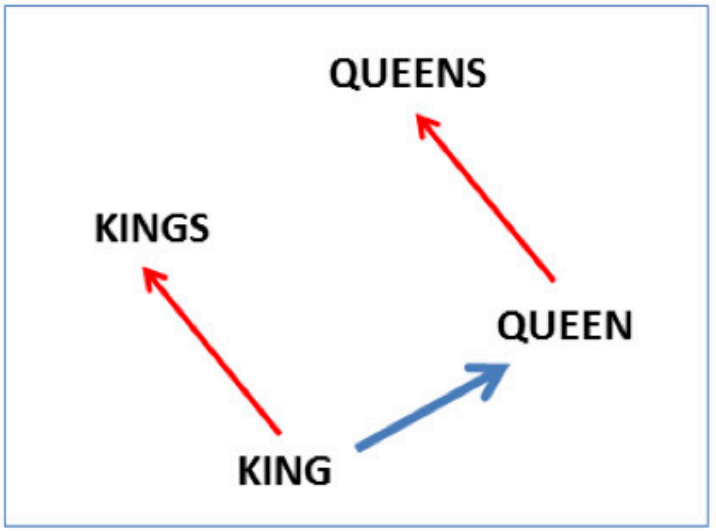
Source : Chapter 6 of Book: https://web.stanford.edu/~jurafsky/slp3/

Analogy: Embeddings capture relational meaning!

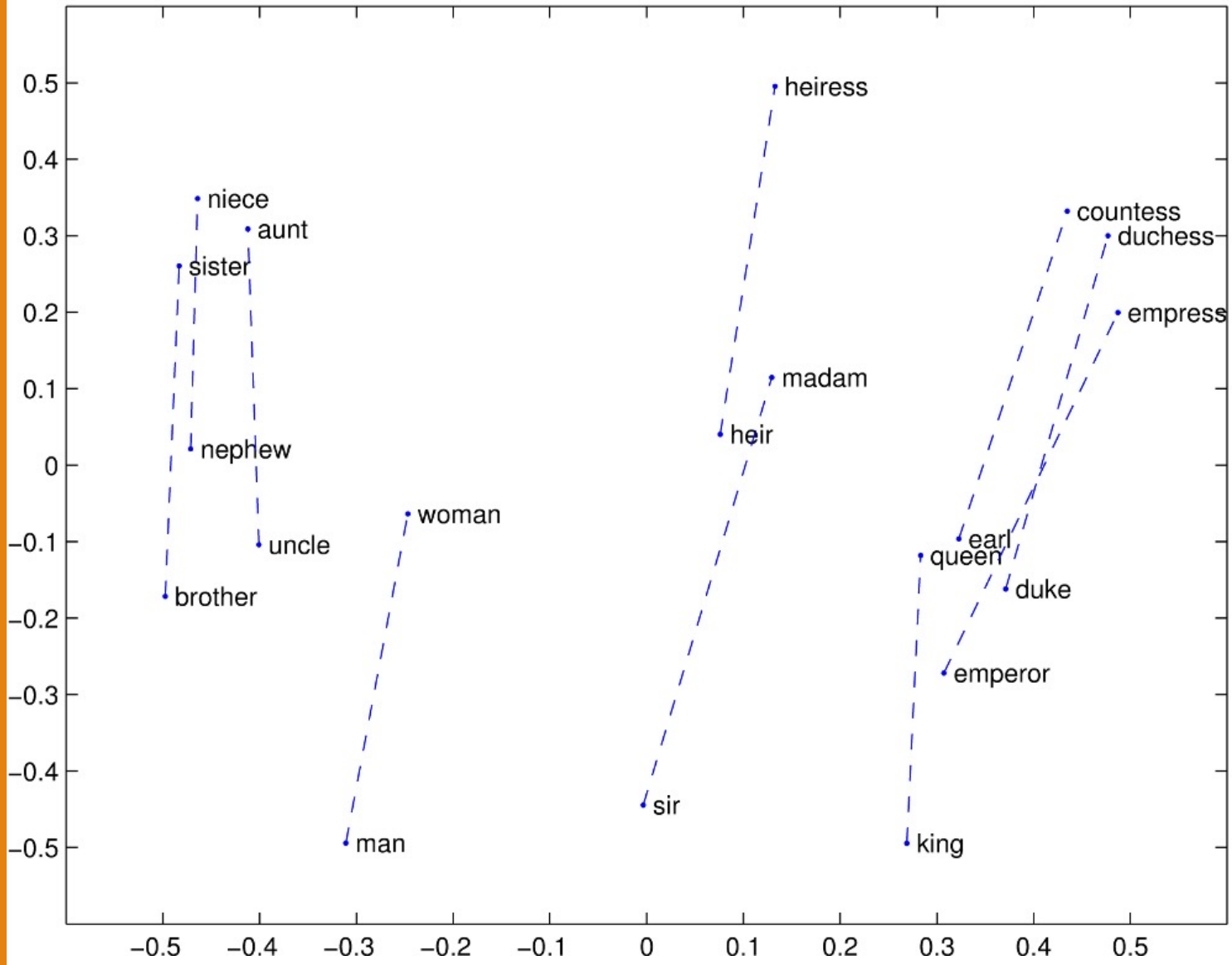
vector(*'king'*) - vector(*'man'*) + vector(*'woman'*) ≈ vector('queen')

vector(*'Paris'*) - vector(*'France'*) + vector(*'Italy'*) ≈ vector('Rome')

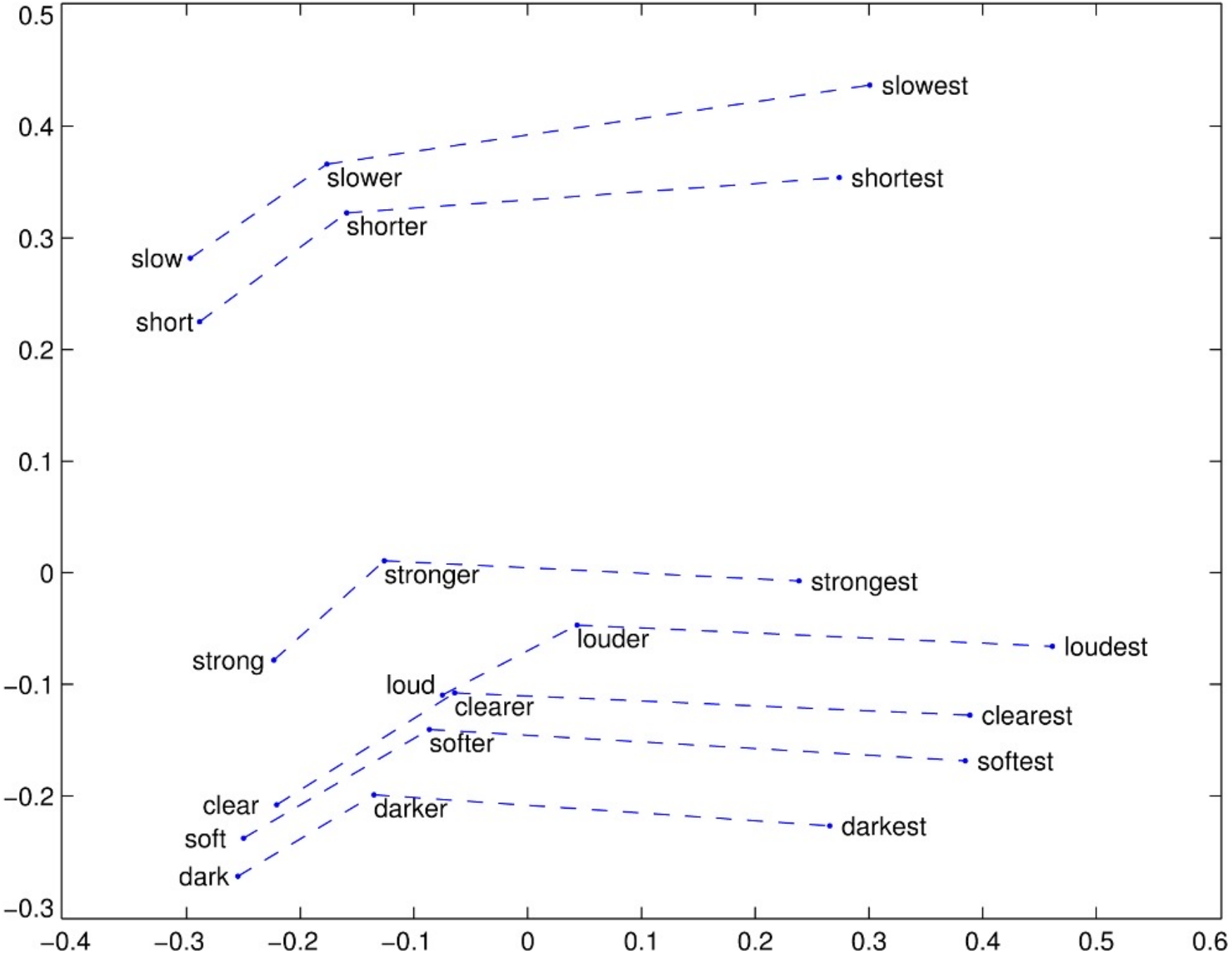




Source : Chapter 6 of Book: https://web.stanford.edu/~jurafsky/slp3/



Source : Chapter 6 of Book: https://web.stanford.edu/~jurafsky/slp3/


slowest
slower
shortest
shorter
slow
short
stronger
strongest
louder
loudest
strong
loud
clearer
clearest
softer
softest
clear
darker
darkest
soft
dark
0.5
0.4
0.3
0.2
0.1
0
−0.1
−0.2
−0.3
−0.4
−0.3
−0.2
−0.1
0
0.1
0.2
0.3
0.4
0.5
0.6

Embeddings and bias

Embeddings reflect cultural bias

Bolukbasi, Tolga, Kai-Wei Chang, James Y. Zou, Venkatesh Saligrama, and Adam T. Kalai. "Man is to computer programmer as woman is to homemaker? debiasing word embeddings." In *Advances in Neural Information Processing Systems*, pp. 4349-4357. 2016.

Ask “Paris : France :: Tokyo : x”

- x = Japan

Ask “father : doctor :: mother : x”

- x = nurse

Ask “man : computer programmer :: woman : x”

- x = homemaker

Source : Chapter 6 of Book: https://web.stanford.edu/~jurafsky/slp3/

Embeddings reflect cultural bias

Caliskan, Aylin, Joanna J. Bruson and Arvind Narayanan. 2017. Semantics derived automatically from language corpora contain human-like biases. Science 356:6334, 183-186.

Implicit Association test (Greenwald et al 1998): How associated are
- concepts (*flowers*, *insects*) & attributes (*pleasantness*, *unpleasantness*)?
- Studied by measuring timing latencies for categorization.

Psychological findings on US participants:
- African-American names are associated with unpleasant words (more than European-American names)
- Male names associated more with math, female names with arts
- Old people's names with unpleasant words, young people with pleasant words.

Caliskan et al. replication with embeddings:
- African-American names (*Leroy, Shaniqua*) had a higher GloVe cosine with unpleasant words (*abuse, stink, ugly*)
- European American names (*Brad, Greg, Courtney*) had a higher cosine with pleasant words (*love, peace, miracle*)

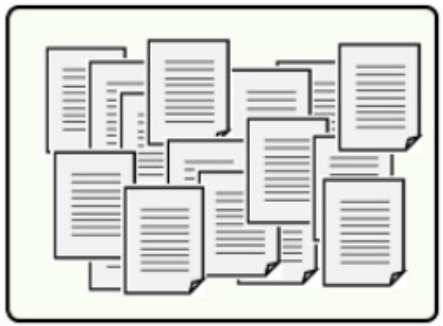
Embeddings reflect and replicate all sorts of pernicious biases.

Source : Chapter 6 of Book: https://web.stanford.edu/~jurafsky/slp3/

Using Dense Embeddings

Training data for text classification (labeled)

- Not huge, or not diverse, or both
- Domain: task-specific



Training data for word embeddings (unlabeled)

- Huge diverse corpus (e.g., Wikipedia)
- Domain: general

Source: https://lena-voita.github.io/nlp_course/transfer_learning.html

Pre-Trained Embeddings

- Train from scratch

What they will know:

May be not enough to learn relationships between words

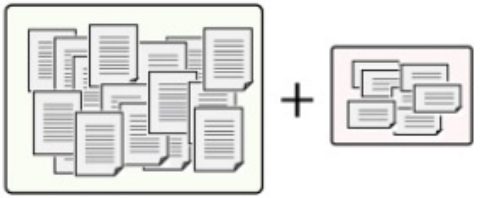
- Take pretrained (Word2Vec, GloVe)

What they will know:

Know relationships between words, but are **not** specific to the task

- Initialize with pretrained, then fine-tune

What they will know:

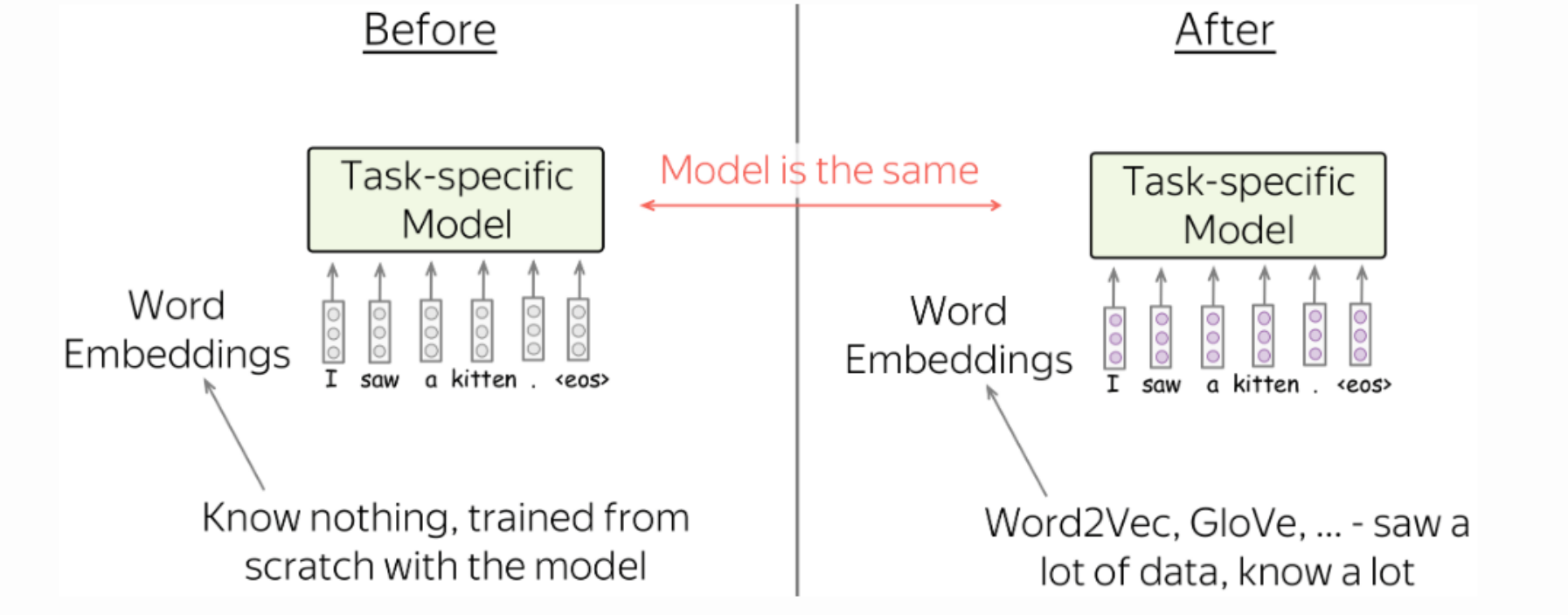


Know relationships between words and adapted for the task

"**Transfer**" knowledge from a huge unlabeled corpus to your task-specific model

Source: https://lena-voita.github.io/nlp_course/transfer_learning.html

Pre-Trained Embeddings



Source: https://lena-voita.github.io/nlp_course/transfer_learning.html

Trained embeddings

- Word2vec
  https://code.google.com/archive/p/word2vec/
- Glove
  http://nlp.stanford.edu/projects/glove/
- Levy/Goldberg dependency embeddings
  https://levyomer.wordpress.com/2014/04/25/dependency-based-word-embeddings/

Source : https://people.ischool.berkeley.edu/~dbamman/info256.html

Improvement over tfidf

- **Semantic Meaning**: Word2Vec and GloVe embeddings capture deep semantic meaning by considering the context in which words appear. This allows words with similar meanings to have similar embeddings, which isn't the case with TF-IDF.

- **Word Relationships**: These models can capture complex relationships between words, such as analogies ("king" is to "queen" as "man" is to "woman"), which TF-IDF cannot do.

- **Dimensionality**: Word2Vec and GloVe embeddings are typically of a much lower dimension than TF-IDF, making them more computationally efficient while retaining more useful information.

- **Handling Synonyms**: By considering word contexts, Word2Vec and GloVe are better at associating synonyms with similar vectors, while TF-IDF treats all words as independent

Limitation still remaining

Word2Vec and GloVe provided significant advancements over TF-IDF embeddings, but there were still issues and limitations that remained even with these more sophisticated models:

- **OOV Words:** Word2Vec and GloVe can't create embeddings for words not in their training data, a challenge for the ever-changing vocabulary of human languages.
- **Polysemy:** Both assign one vector per word, unable to differentiate meanings for words like "bank," which can signify a river's edge or a financial institution.
- **Static Representations:** Word2Vec and GloVe produce fixed embeddings, not adapting to varying word meanings in different contexts.
- **Word Order Sensitivity:** These models miss the word order, so phrases like "free drug" and "drug free" might get similar representations despite contrasting meanings.

Two kinds of "training" data

- The labeled data for a specific task (e.g., labeled sentiment for movie reviews): ~ 2K labels/reviews, ~1.5M words → used to train a supervised model → Supervised training

- General text (Wikipedia, the web, books, etc.), ~ trillions of words → used to train word distributed representations → Self-supervised training (We can train the model on generic text – no manually labelled data required)

Source : https://people.ischool.berkeley.edu/~dbamman/info256.html

Self Supervised training

Self-supervised learning is a type of machine learning that does not require explicit external labeling of data. Instead, it generates its own supervisory signal based on the input data.

Can we think of another such task?

Source : https://people.ischool.berkeley.edu/~dbamman/info256.html

Self Supervised training

Self-supervised learning is a type of machine learning that does not require explicit external labeling of data. Instead, it generates its own supervisory signal based on the input data.

Can we think of other such task?

Language Model

Source : https://people.ischool.berkeley.edu/~dbamman/info256.html

Language Modeling & RNN

Probabilistic Language Models

Goal: assign a probability to a sentence

- Machine Translation:
  - P(**high** winds tonite) > P(**large** winds tonite)
- Spell Correction
  - The office is about fifteen **minuets** from my house
    - P(about fifteen **minutes** from) > P(about fifteen **minuets** from)
- Speech Recognition
  - P(I saw a van) >> P(eyes awe of an)
- + Summarization, question-answering, etc., etc.!!

Why?

Slide Credit: Dan Jurafsky and James H. Martin

Probabilistic Language Modeling

Goal: compute the probability of a sentence or sequence of words:

$P(W) = P(w_1,w_2,w_3,w_4,w_5 \ldots w_n)$

Related task: probability of an upcoming word:

$P(w_5|w_1,w_2,w_3,w_4)$

A model that computes either of these:

$P(W)$ or $P(w_n|w_1,w_2 \ldots w_{n-1})$ is called a **language model**.

Better: **the grammar** But **language model** or **LM** is standard

Slide Credit: Dan Jurafsky and James H. Martin

Why should we care about Language Modeling?

- Language Modeling is a benchmark task that helps us measure our progress on understanding language

- Language Modeling is a subcomponent of many NLP tasks, especially those involving generating text or estimating the probability of text:
  - Predictive typing
  - Speech recognition
  - Handwriting recognition
  - Spelling/grammar correction
  - Authorship identification
  - Machine translation
  - Summarization
  - Dialogue
  - etc.

Slide credit: Christopher Manning , Abigail See and John Hewitt

How to compute P(W)

How to compute this joint probability:

- P(its, water, is, so, transparent, that)

Intuition: let's rely on the Chain Rule of Probability

Slide Credit: Dan Jurafsky and James H. Martin

Reminder: The Chain Rule

Recall the definition of conditional probabilities

$p(B|A) = P(A,B)/P(A)$ Rewriting: **$P(A,B) = P(A)P(B|A)$**

More variables:

$P(A,B,C,D) = P(A)P(B|A)P(C|A,B)P(D|A,B,C)$

The Chain Rule in General

$P(x_1,x_2,x_3,\ldots,x_n) = P(x_1)P(x_2|x_1)P(x_3|x_1,x_2)\ldots P(x_n|x_1,\ldots,x_{n-1})$

Slide Credit: Dan Jurafsky and James H. Martin

The Chain Rule applied to compute joint probability of words in sentence

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

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)

Slide Credit: Dan Jurafsky and James H. Martin

How to estimate these probabilities

$$P(w_n|w_1, w_2, \cdots, w_{n-1}) = \frac{N(w_1, w_2, \ldots . w_n)}{N(w_1, w_2, \ldots . w_{n-1})}$$

where $N(w_1, w_2, \ldots . w_n)$ is the number of times $(w_1, w_2, \ldots . w_n)$ ocuurs in the corpus

These are too long, and many of the counts will be zero

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

Slide Credit: lena Voita, Dan Jurafsky and James H. Martin

Markov Assumption

> The probability of a word only depends on a **fixed** number of previous words.

For an n-gram model,

$$P(y_t \mid \underbrace{y_1, y_2, \ldots, y_{t-1}}) = P(y_t \mid \underbrace{y_{t-n+1}, \ldots, y_{t-1}})$$

all previous tokens — n-1 previous token

For example,

- n=3 (trigram model): $P(y_t \mid y_1, y_2, \ldots, y_{t-1}) = P(y_t \mid y_{t-2}, y_{t-1})$
- n=2 (bigram model): $P(y_t \mid y_1, y_2, \ldots, y_{t-1}) = P(y_t \mid y_{t-1})$
- n=1 (unigram model): $P(y_t \mid y_1, y_2, \ldots, y_{t-1}) = P(y_t)$

SLIDE CREDIT: LENA VOITA (GITHUB.IO/NLP_COURSE.HTML#PREVIEW_LANG_MODELS)

Markov Assumption: Example

P(I saw a cat on a mat) =?

Before

P(I saw a cat on a mat) =

P(I)
· P(saw | I)
· P(a | I saw)
· P(cat | I saw a)
· P(on | I saw a cat)
· P(a | I saw a cat on)
· P(mat | I saw a cat on a)

After (3-gram)

P(I saw a cat on a mat) =

P(I) → P(I)
· P(saw | I) → · P(saw | I)
· P(a | I saw) → · P(a | I saw)
· P(cat | ~~I~~ saw a) → · P(cat | saw a)
· P(on | ~~I saw~~ a cat) → · P(on | a cat)
· P(a | ~~I saw a~~ cat on) → · P(a | cat on)
· P(mat | ~~I saw a cat~~ on a) → · P(mat | on a)

ignore use

SLIDE CREDIT: LENA VOITA (GITHUB.IO/NLP_COURSE.HTML#PREVIEW_LANG_MODELS)

Estimating Probabilities using bigrams

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

<s> I am Sam </s>

<s> Sam I am </s>

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

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

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

Slide Credit: Dan Jurafsky and James H. Martin

Raw bigram counts

Out of 9222 sentences

	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

Slide Credit: Dan Jurafsky and James H. Martin

Raw bigram probabilities

$$P\left(\frac{food}{chinese}\right) = \frac{count(chinese, food)}{count(\text{chinese})} = \frac{82}{158} = 0.52$$

$$P\left(\frac{w}{chinese}\right) = \frac{count(chinese, w)}{count(\text{chinese})}$$

Normalize by unigrams:

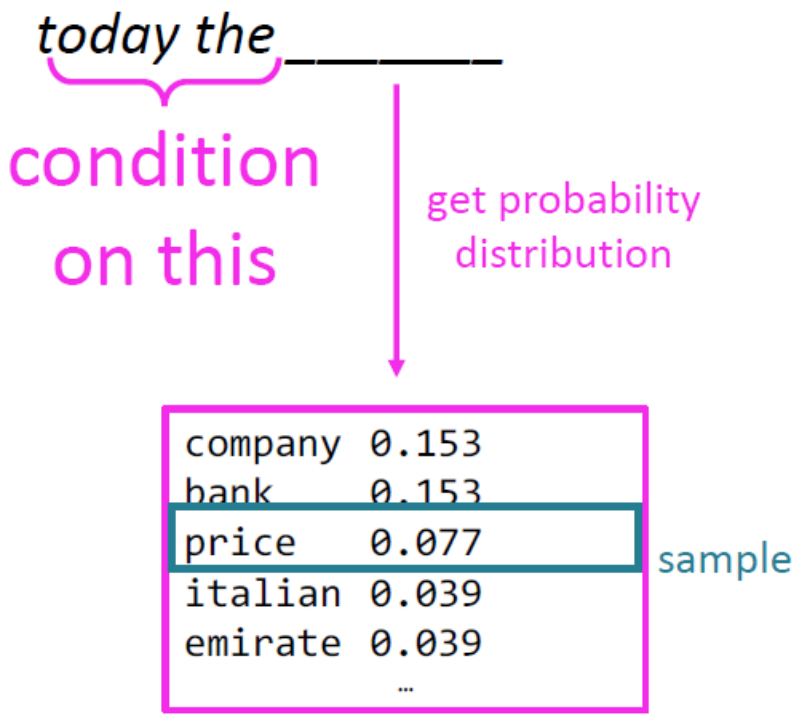
i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

Result:

	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

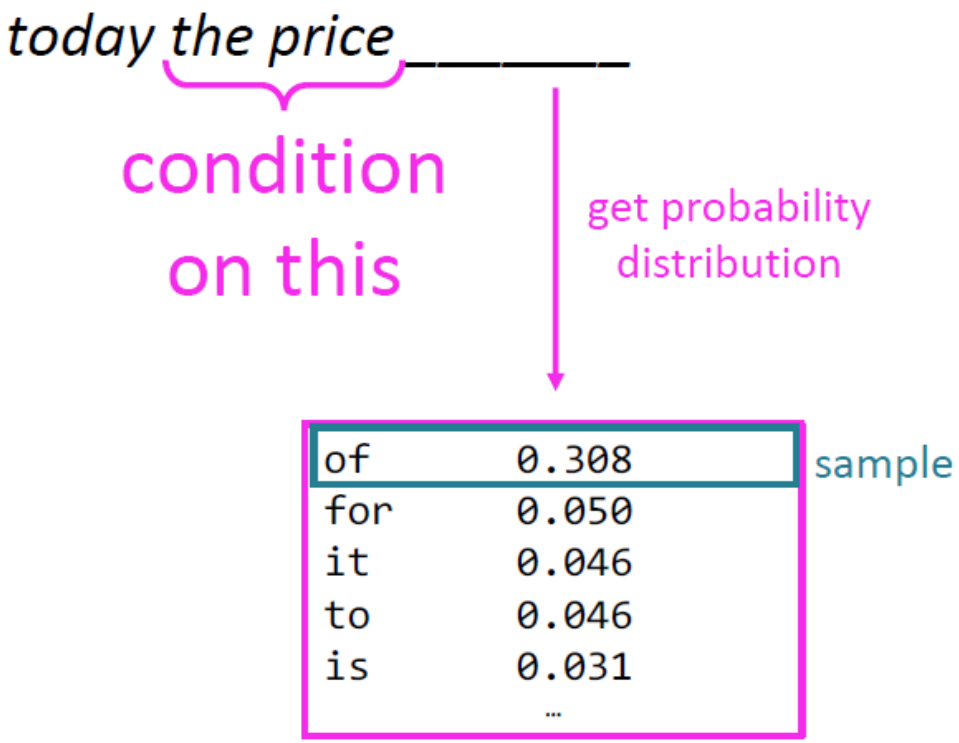
Slide Credit: Dan Jurafsky and James H. Martin

Generating text with n-gram Language Model



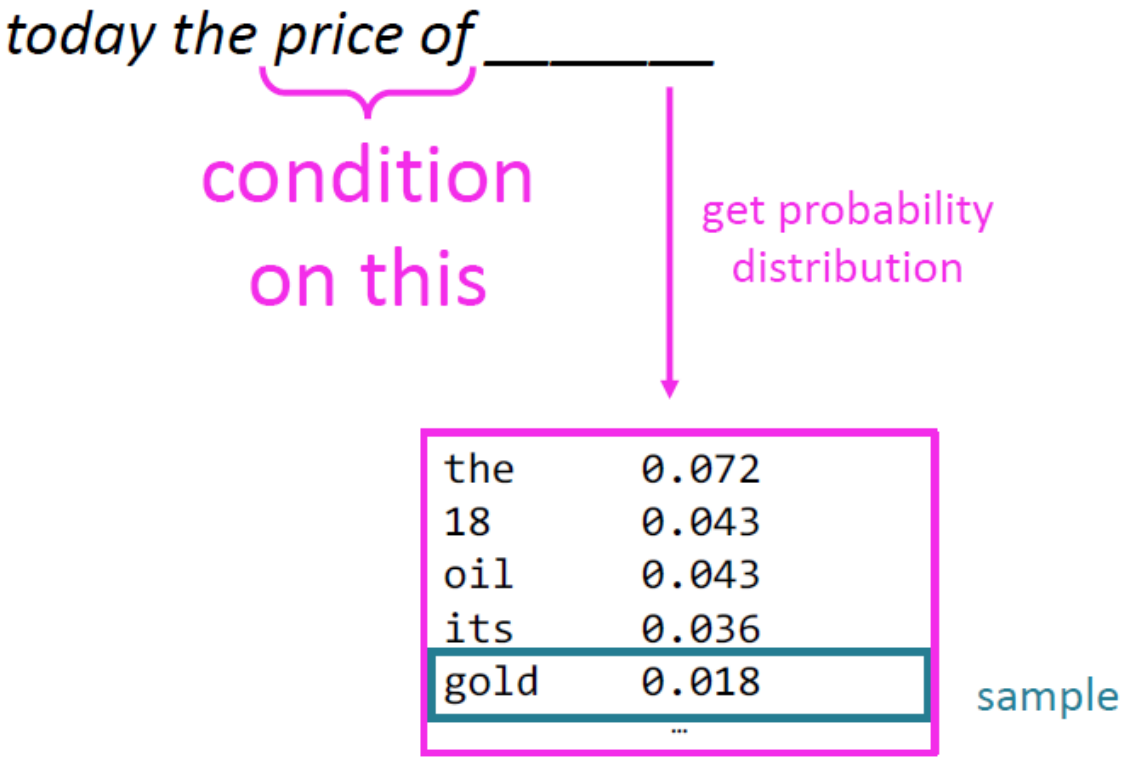
Slide credit: Christopher Manning , Abigail See and John Hewitt

Generating text with n-gram Language Model



Slide credit: Christopher Manning , Abigail See and John Hewitt

Generating text with n-gram Language Model



Slide credit: Abigail See and John Hewitt

Generating text with n-gram Language Model

[Try Yourself: https://nlpforhackers.io/language-models/](https://nlpforhackers.io/language-models/)

Slide credit: Abigail See and John Hewitt

Issues n-gram Language Models

Suppose we are learning a 4-gram Language Model.

~~*as the proctor started the clock, the*~~ *students opened their* _____

discard

condition on this

$$P(\boldsymbol{w}|\text{students opened their}) = \frac{\text{count}(\text{students opened their } \boldsymbol{w})}{\text{count}(\text{students opened their})}$$

For example, suppose that in the corpus:

- "students opened their" occurred 1000 times
- "students opened their books" occurred 400 times
  - → P(books | students opened their) = 0.4
- "students opened their exams" occurred 100 times
  - → P(exams | students opened their) = 0.1

Should we have discarded the "proctor" context?

Can we increase n to improve model ?

Slide credit: Christopher Manning , Abigail See and John Hewitt

Sparsity problem- n-gram Language Models

$$P(\text{mat} \mid \text{I saw a cat on a}) = P(\text{mat} \mid \text{cat on a}) = \frac{N(\text{cat on a mat})}{N(\text{cat on a})}$$

$$P(\text{mat} \mid \text{cat on a}) = \frac{N(\text{cat on a mat})}{\boxed{N(\text{cat on a})}} = ?$$

zero

not good: can not compute the probability

Sparsity Problem 1

$$P(\text{mat} \mid \text{cat on a}) = \frac{\boxed{N(\text{cat on a mat})}}{N(\text{cat on a})} = ?$$

zero

not good: zeros out probability of the sentence

Sparsity Problem 2

Increasing n makes sparsity problem worse

SLIDE CREDIT: LENA VOITA (GITHUB.IO/NLP_COURSE.HTML#PREVIEW_LANG_MODELS)

Storage problem- n-gram Language Models

Storage: Need to store count for all *n*-grams you saw in the corpus.

$$P(\boldsymbol{w}|\text{students opened their}) = \frac{\text{count}(\text{students opened their } \boldsymbol{w})}{\text{count}(\text{students opened their})}$$

Increasing *n* or increasing corpus increases model size!

Slide credit: Christopher Manning , Abigail See and John Hewitt

Fixed Window Neural Language Model

A fixed-window neural Language Model

Fixed Window Neural Language Model

A fixed-window neural Language Model

output distribution

$$\hat{\boldsymbol{y}} = \text{softmax}(\boldsymbol{U}\boldsymbol{h} + \boldsymbol{b}_2) \in \mathbb{R}^{|V|}$$

hidden layer

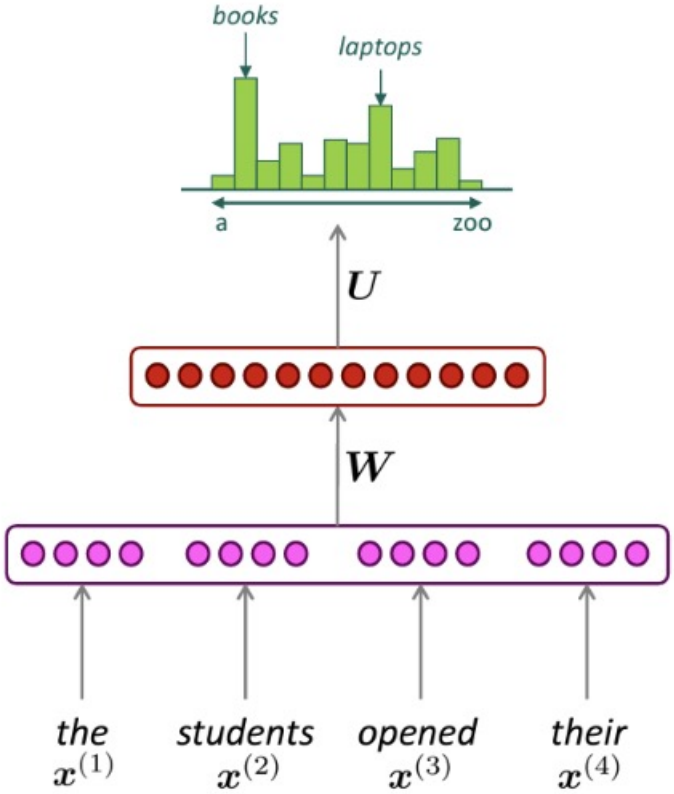
$$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{e} + \boldsymbol{b}_1)$$

concatenated word embeddings

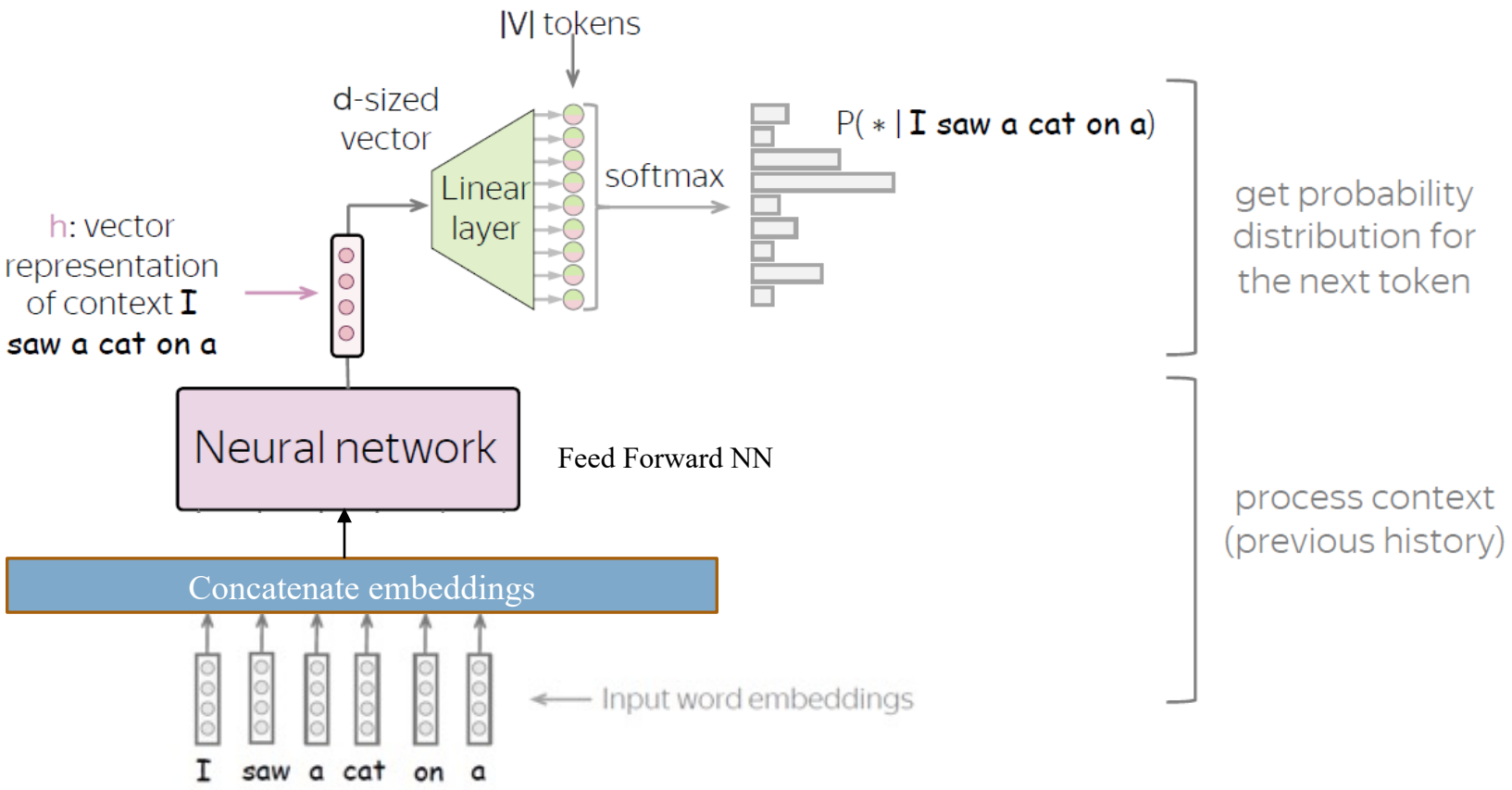
$$\boldsymbol{e} = [\boldsymbol{e}^{(1)}; \boldsymbol{e}^{(2)}; \boldsymbol{e}^{(3)}; \boldsymbol{e}^{(4)}]$$

words / one-hot vectors

$$\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}, \boldsymbol{x}^{(4)}$$

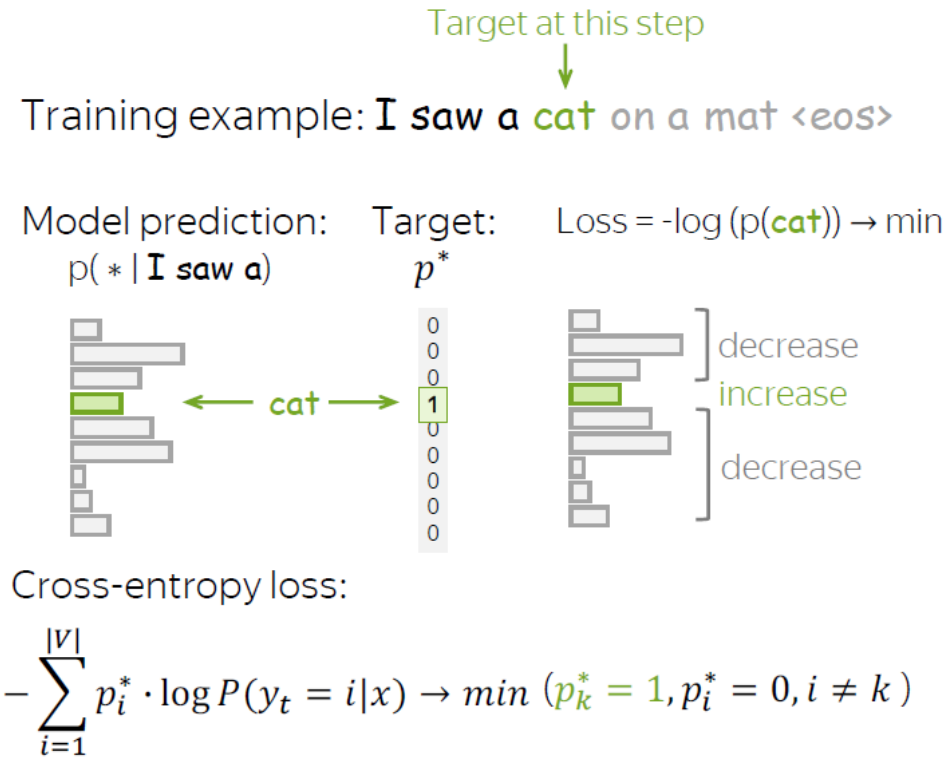


Fixed Window Neural Language Model



SLIDE CREDIT: LENA VOITA (GITHUB.IO/NLP_COURSE.HTML#PREVIEW_LANG_MODELS)

Training: Fixed Window Neural Language Model



Cross-entropy loss:

$$-\sum_{i=1}^{|V|} p_i^* \cdot \log P(y_t = i|x) \rightarrow min \ (p_k^* = 1, p_i^* = 0, i \neq k\ )$$

For one-hot targets, this is equivalent to

$$-\log P(y_t = cat|x) \rightarrow min$$

SLIDE CREDIT: LENA VOITA (GITHUB.IO/NLP_COURSE.HTML#PREVIEW_LANG_MODELS)

Fixed Window Neural Language Model

Improvements: The model does not depend on the counts

- No Sparsity Problem
- No Storage problem

Issues in using Feed Forward Neural Network:

- Fixed window – Window is still small
- Increasing Window size will increase number of parameters to be estimated

We need a network architecture that can process a **sequence** of **any length.**

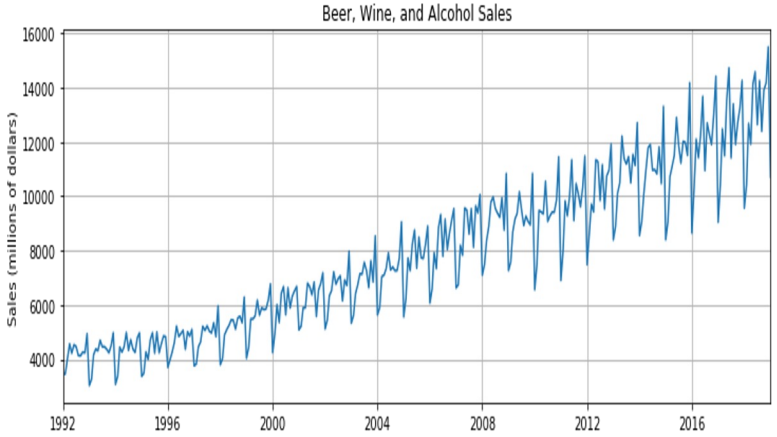
Recurrent Neural Networks

Sequence

We've used Neural Networks to solve Classification and Regression problems, but we still haven't seen how Neural Networks can deal with sequence information.

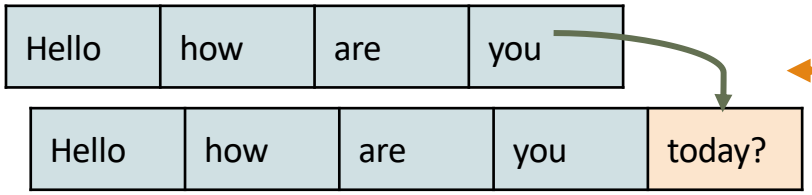
Sequences

Time Series



Examples of Sequences

- Time Series Data (Sales)
- Sentences
- Audio
- Car Trajectories
- Music
- Patient Records



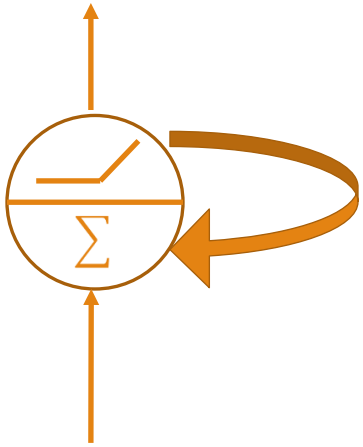
Language Model

RNN - Intution

To process sequence, we need to somehow let the neuron “know” about its previous history of outputs.

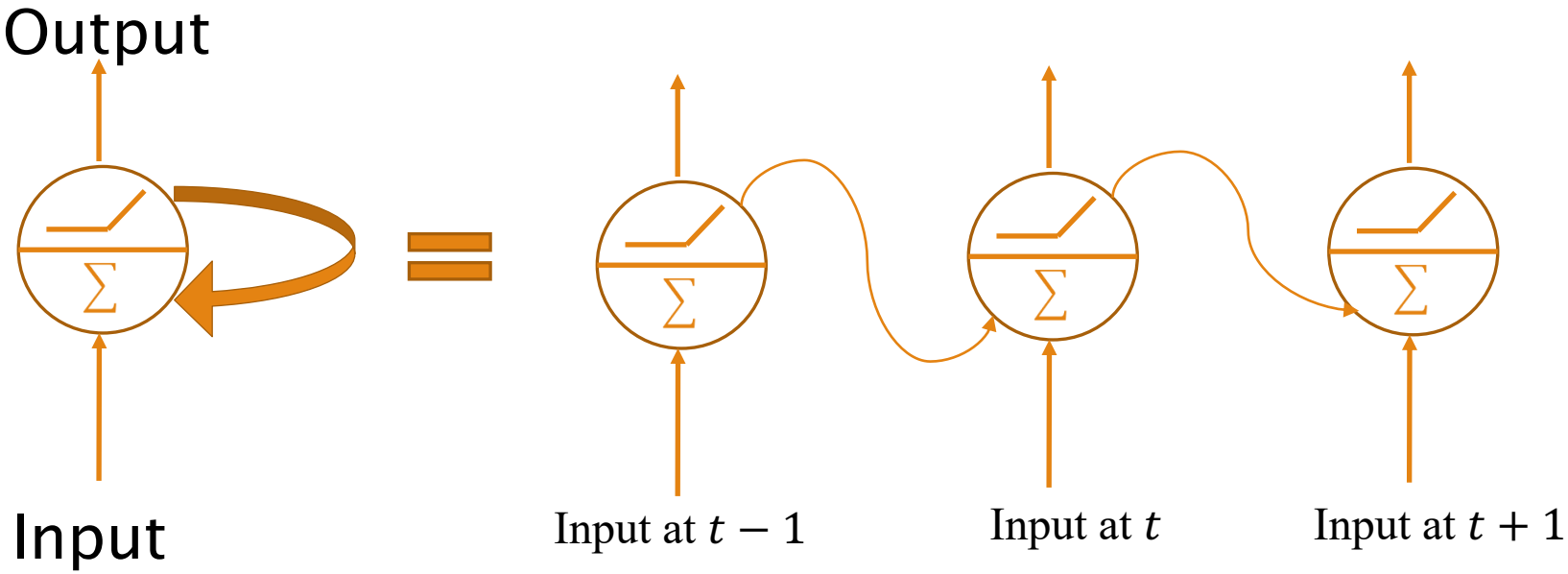
One easy way to do this is to simply feed its output back into itself as an input!

Recurrent Neuron

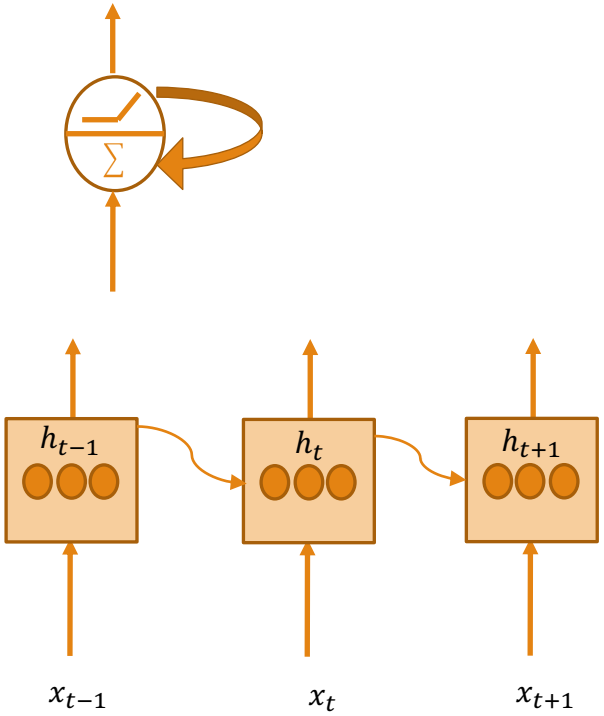


- RNNs have an '*hidden state*' that is updated as the sequence is processed

RNN Intuition



RNN Intuition



We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

$$h_t = f_W(h_{t-1}, x_t)$$

new state

old state

input vector at some time step

some function with parameters W

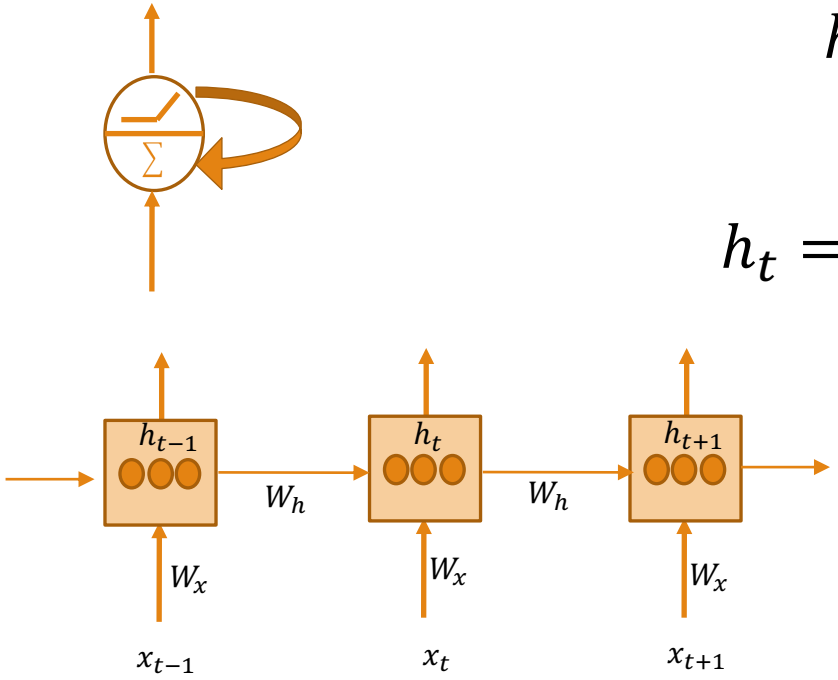
SLIDE CREDIT: JUSTIN JOHNSON (UNIVERSITY OF MICHIGAN)

RNN Intuition

The state consists of a "hidden" vector h:

$$h_t = f_W(h_{t-1}, x_t)$$

$$h_t = \tanh(W_h h_{t-1} + b_h + W_x x_t + b_x)$$



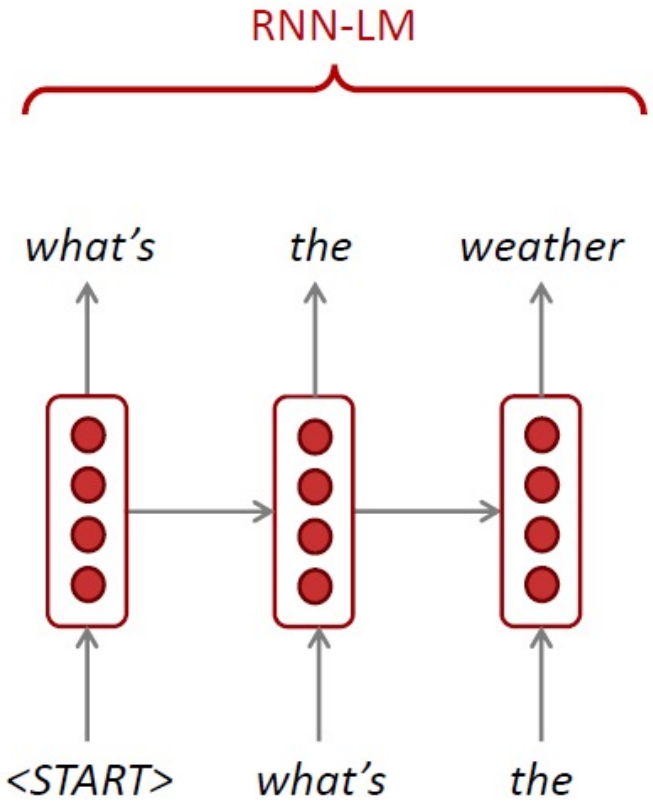
SLIDE CREDIT: JUSTIN JOHNSON (UNIVERSITY OF MICHIGAN)

RNN-LMs can be used to generate text

e.g. speech recognition, machine translation, summarization

Input (audio)

conditioning



Slide credit: Christopher Manning , Abigail See and John Hewitt

Language Modelling with RNN

Language Model with RNN

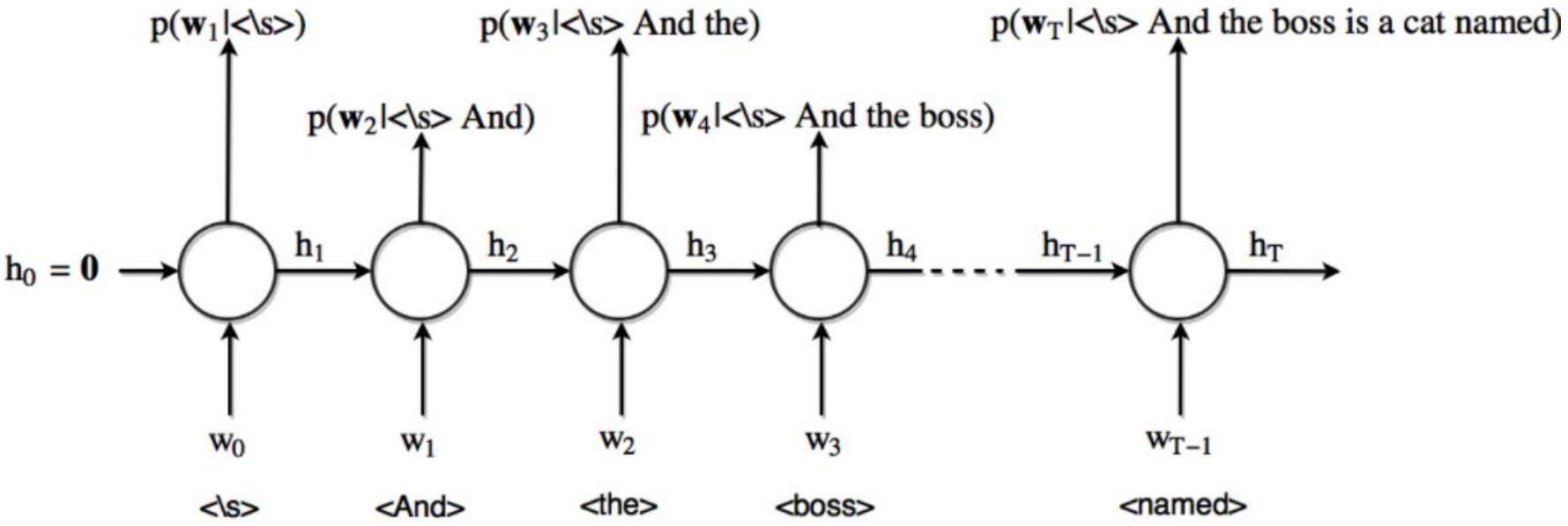
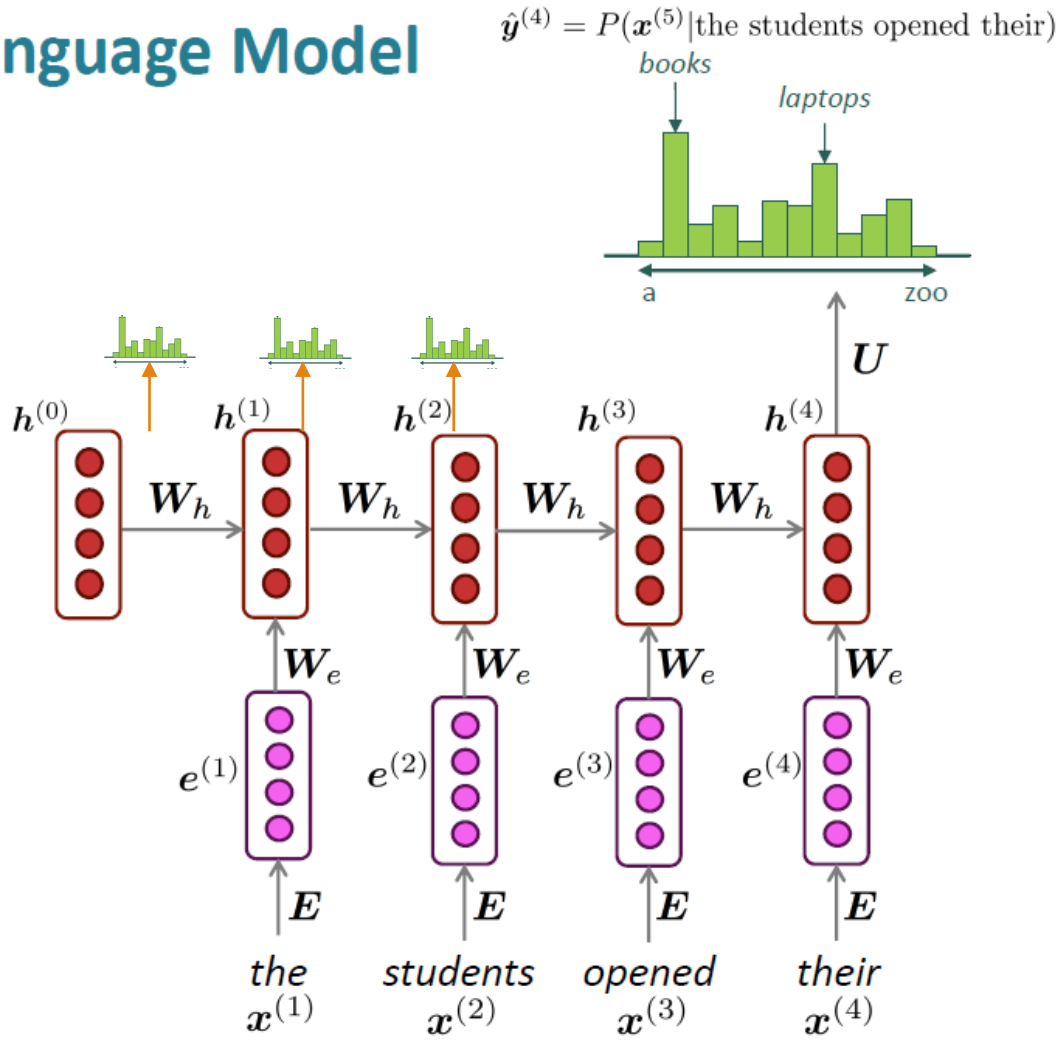


Figure Source https://blog.paperspace.com/recurrent-neural-networks-part-1-2/

A Simple RNN Language Model



output distribution

$$\hat{y}^{(t)} = \text{softmax}\left(Uh^{(t)} + b_2\right) \in \mathbb{R}^{|V|}$$

hidden states

$$h^{(t)} = \sigma\left(W_h h^{(t-1)} + W_e e^{(t)} + b_1\right)$$

$h^{(0)}$ is the initial hidden state

word embeddings

$$e^{(t)} = Ex^{(t)}$$

words / one-hot vectors

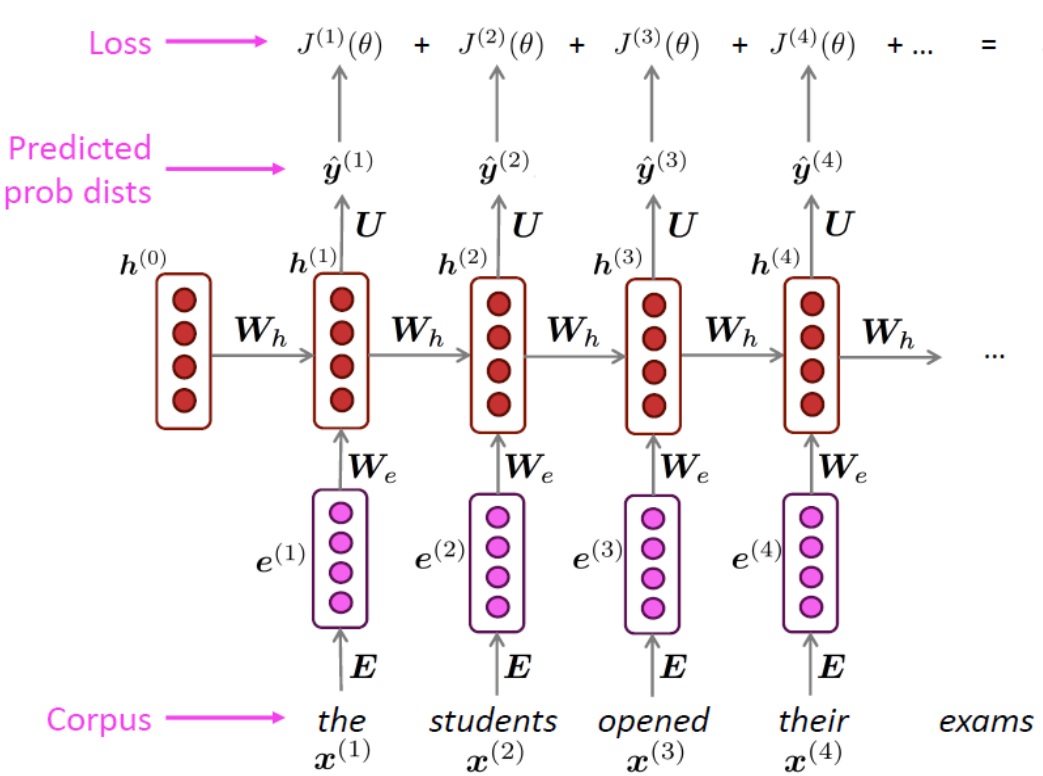
$$x^{(t)} \in \mathbb{R}^{|V|}$$

Note: *this input sequence could be much longer now!*

Slide credit: Christopher Manning , Abigail See and John Hewitt

Training an RNN Language Model

"Teacher forcing"



$J^{(1)}(\theta)$ = negative log prob of "students"

$J^{(2)}(\theta)$ = negative log prob of "opened"

$J^{(3)}(\theta)$ = negative log prob of "their"

$J^{(4)}(\theta)$ = negative log prob of "exams"

Slide credit: Christopher Manning , Abigail See and John Hewitt

Training a RNN Language Model

- However: Computing loss and gradients across entire corpus $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}$ is too expensive!

$$J(\theta) = \frac{1}{T} \sum_{t=1}^{T} J^{(t)}(\theta)$$

- In practice, consider $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}$ as a sentence (or a document)

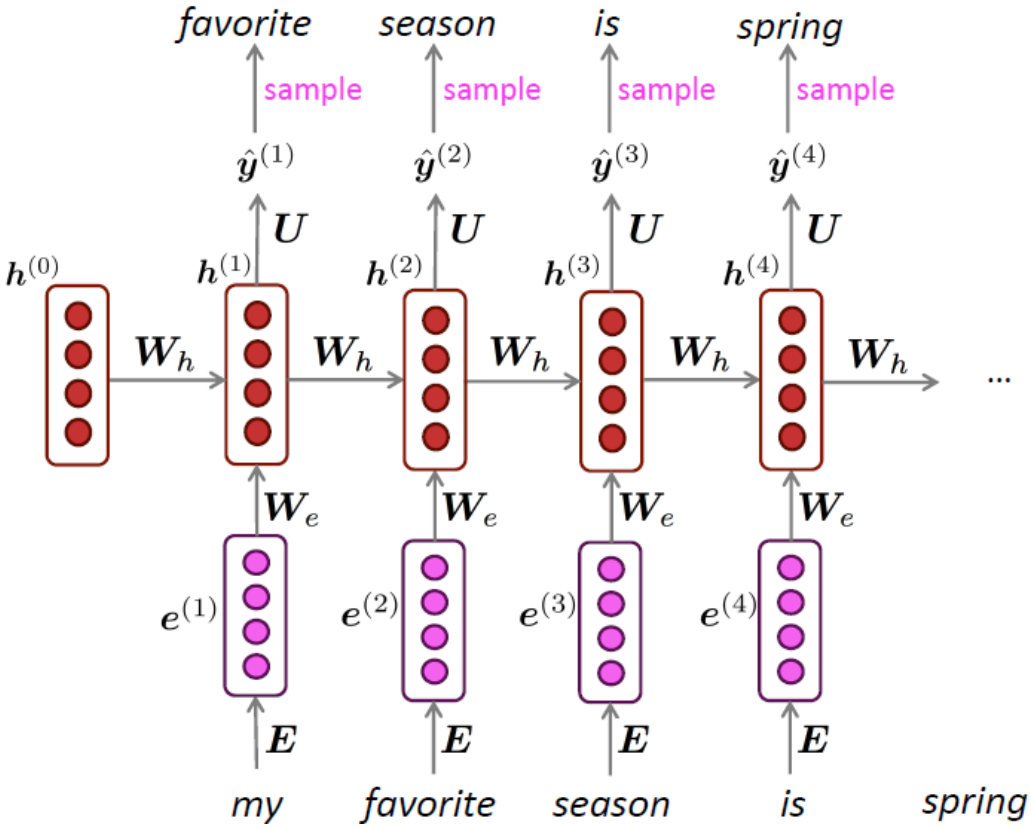
- Recall: Stochastic Gradient Descent allows us to compute loss and gradients for small chunk of data, and update.

- Compute loss $J(\theta)$ for a sentence (actually, a batch of sentences), compute gradients and update weights. Repeat.

Slide credit: Christopher Manning , Abigail See and John Hewitt

Generating text with a RNN Language Model

Just like a n-gram Language Model, you can use a RNN Language Model to generate text by repeated sampling. Sampled output becomes next step's input.



Slide credit: Christopher Manning , Abigail See and John Hewitt

RNN Language Models

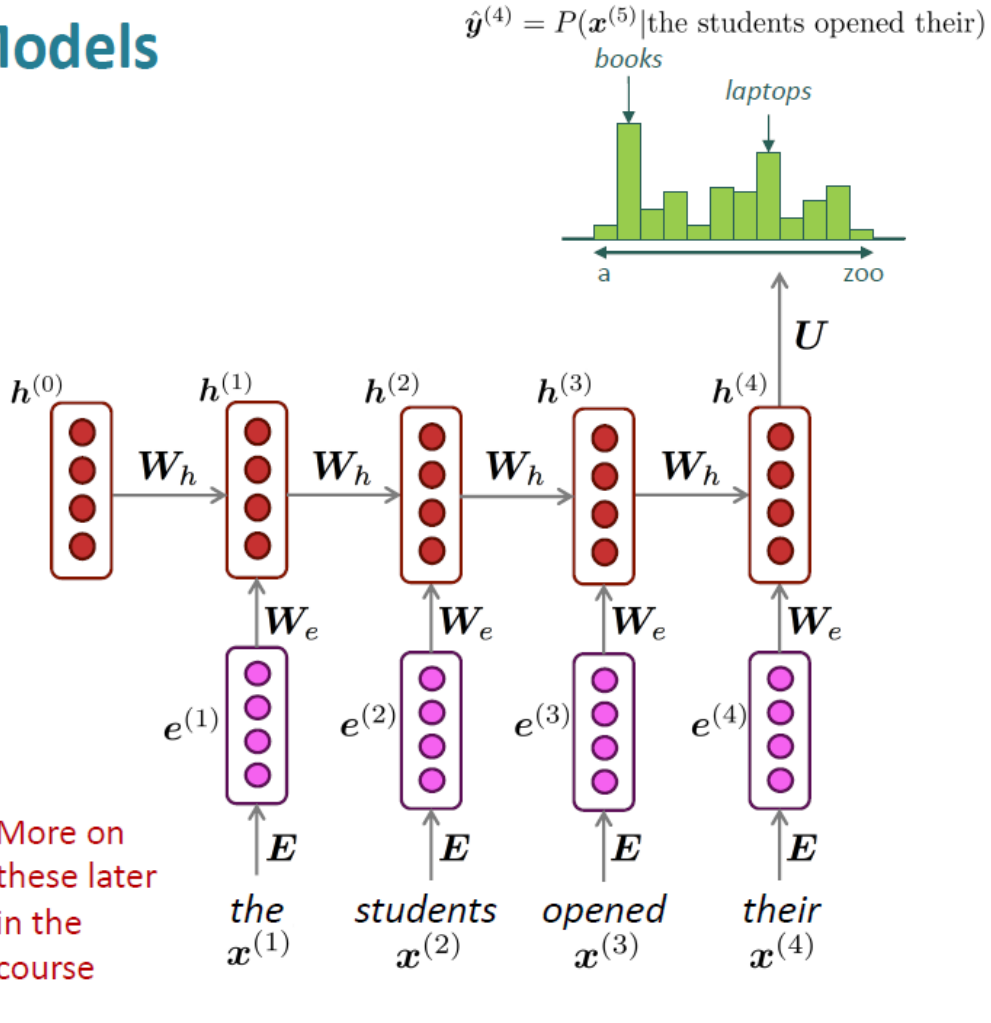
RNN **Advantages**:

- Can process any length input
- Computation for step *t* can (in theory) use information from many steps back
- Model size doesn't increase for longer input context
- Same weights applied on every timestep, so there is symmetry in how inputs are processed.

RNN **Disadvantages**:

- Recurrent computation is slow
- In practice, difficult to access information from many steps back

More on these later in the course

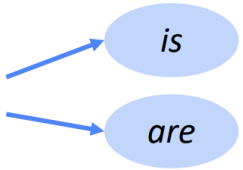


Slide credit: Abigail See and John Hewitt

Effect of Vanishing Gradients

Effect of vanishing gradient on RNN-LM

- **LM task:** *The writer of the books ___*



- **Correct answer**: *The writer of the books is planning a sequel*
- **Syntactic recency:** *The writer of the books is* (correct)
- **Sequential recency:** *The writer of the books are* (incorrect)
- Vanishing gradient problems may bias RNN-LMs towards learning from sequential recency, so they make this type of error more often than we'd like. [Linzen et al 2016]

Slide credit: Christopher Manning , Abigail See and John Hewitt

Long Short-Term Memory (LSTM)

We have a sequence of inputs $x^{(t)}$, and we will compute a sequence of hidden states $h^{(t)}$ and cell states $c^{(t)}$. On timestep t:

Forget gate: controls what is kept vs forgotten, from previous cell state

Input gate: controls what parts of the new cell content are written to cell

Output gate: controls what parts of cell are output to hidden state

New cell content: this is the new content to be written to the cell

Cell state: erase ("forget") some content from last cell state, and write ("input") some new cell content

Hidden state: read ("output") some content from the cell

Sigmoid function: all gate values are between 0 and 1

$$\boldsymbol{f}^{(t)} = \sigma\left(\boldsymbol{W}_f \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_f \boldsymbol{x}^{(t)} + \boldsymbol{b}_f\right)$$

$$\boldsymbol{i}^{(t)} = \sigma\left(\boldsymbol{W}_i \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_i \boldsymbol{x}^{(t)} + \boldsymbol{b}_i\right)$$

$$\boldsymbol{o}^{(t)} = \sigma\left(\boldsymbol{W}_o \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_o \boldsymbol{x}^{(t)} + \boldsymbol{b}_o\right)$$

$$\tilde{\boldsymbol{c}}^{(t)} = \tanh\left(\boldsymbol{W}_c \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_c \boldsymbol{x}^{(t)} + \boldsymbol{b}_c\right)$$

$$\boldsymbol{c}^{(t)} = \boldsymbol{f}^{(t)} \circ \boldsymbol{c}^{(t-1)} + \boldsymbol{i}^{(t)} \circ \tilde{\boldsymbol{c}}^{(t)}$$

$$\boldsymbol{h}^{(t)} = \boldsymbol{o}^{(t)} \circ \tanh \boldsymbol{c}^{(t)}$$

Gates are applied using element-wise product

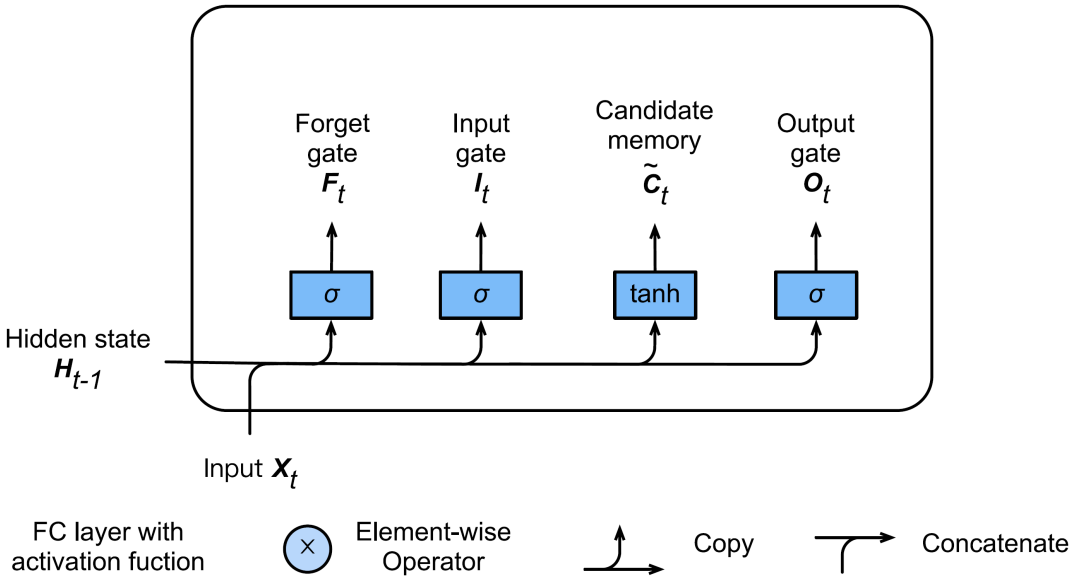
All these are vectors of same length n

Slide credit: Christopher Manning , Abigail See and John Hewitt

LSTM

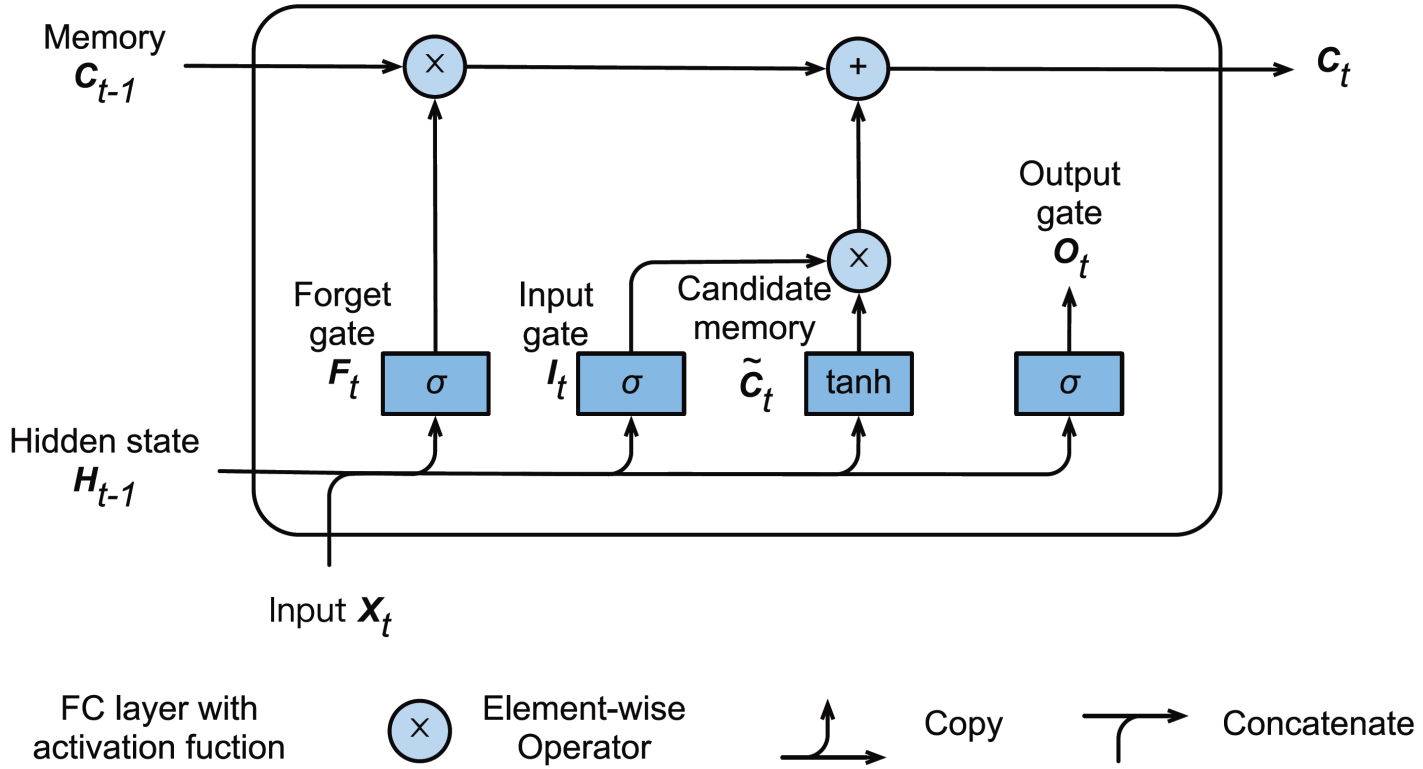
Gates

$$\boldsymbol{I}_t = \sigma(\boldsymbol{X}_t \boldsymbol{W}_{xi} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{hi} + \boldsymbol{b}_i)$$
$$\boldsymbol{F}_t = \sigma(\boldsymbol{X}_t \boldsymbol{W}_{xf} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{hf} + \boldsymbol{b}_f)$$
$$\boldsymbol{O}_t = \sigma(\boldsymbol{X}_t \boldsymbol{W}_{xo} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{ho} + \boldsymbol{b}_o)$$



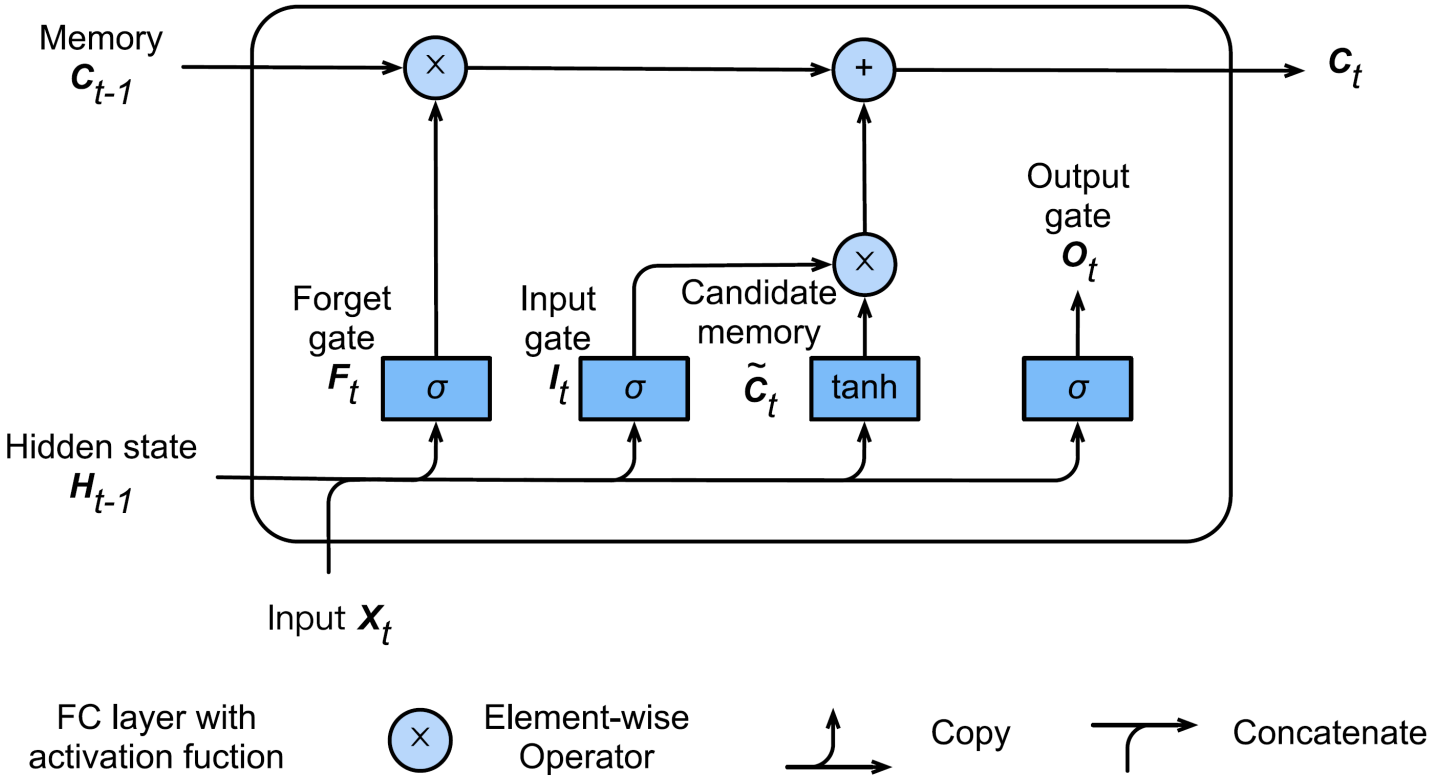
Slide credit: Alex Smola and Mu Li (Berkley)

$$\tilde{C}_t = \tanh(X_t W_{xc} + H_{t-1} W_{hc} + b_c)$$



Slide credit: Alex Smola and Mu Li (Berkley)

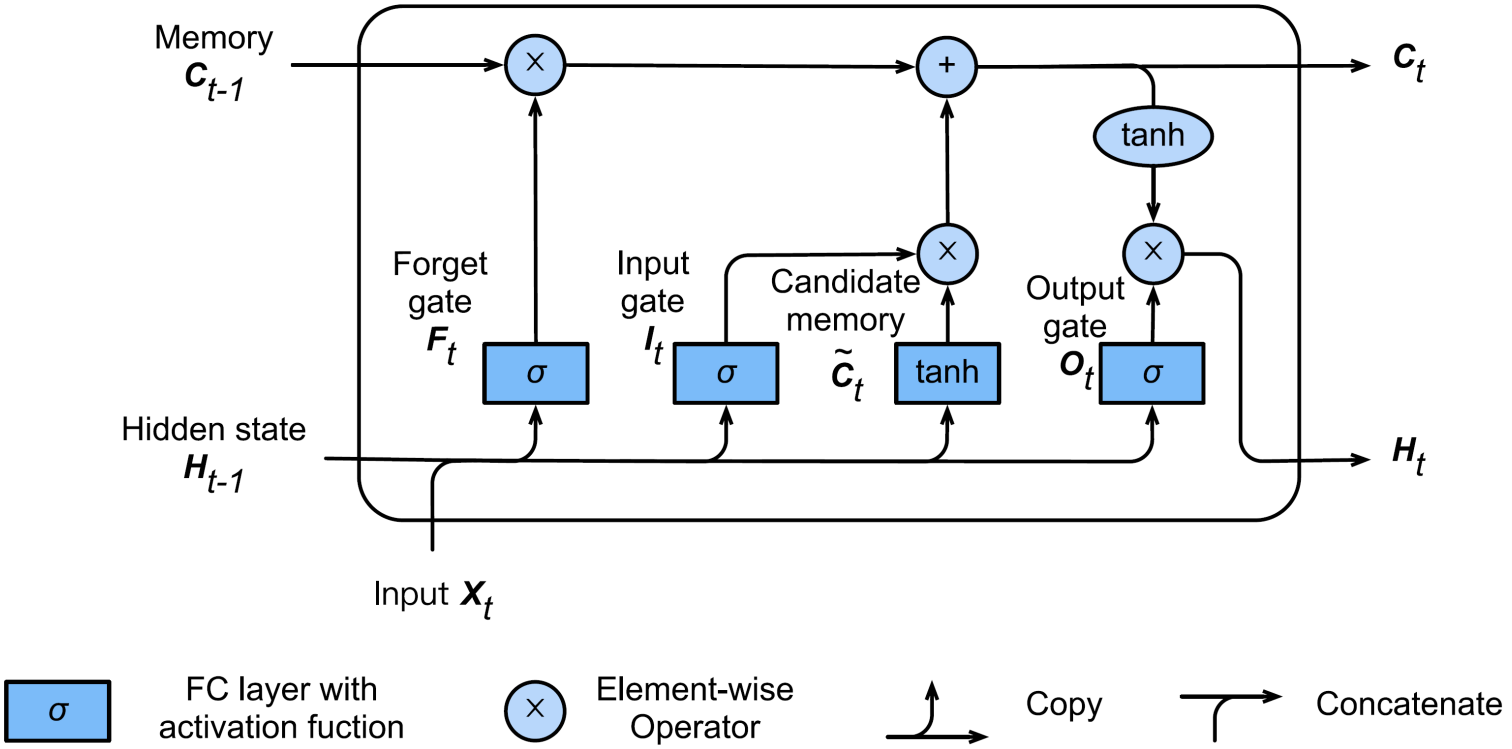
$$C_t = F_t \odot C_{t-1} + I_t \odot \tilde{C}_t$$



Slide credit: Alex Smola and Mu Li (Berkley)

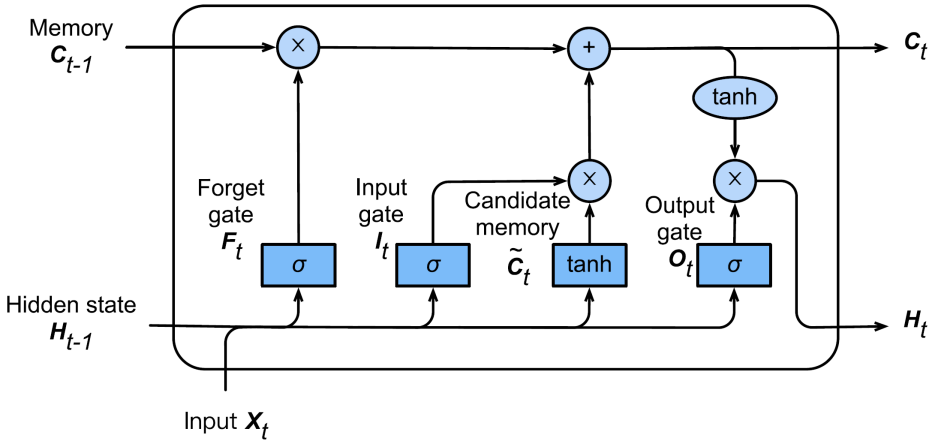
Hidden State / Output

$$H_t = O_t \odot \tanh(C_t)$$



Slide credit: Alex Smola and Mu Li (Berkley)

Hidden State / Output



$$I_t = \sigma(X_t W_{xi} + H_{t-1} W_{hi} + b_i)$$

$$F_t = \sigma(X_t W_{xf} + H_{t-1} W_{hf} + b_f)$$

$$O_t = \sigma(X_t W_{xo} + H_{t-1} W_{ho} + b_o)$$

$$\tilde{C}_t = \tanh(X_t W_{xc} + H_{t-1} W_{hc} + b_c)$$

$$C_t = F_t \odot C_{t-1} + I_t \odot \tilde{C}_t$$

$$H_t = O_t \odot \tanh(C_t)$$

Slide credit: Alex Smola and Mu Li (Berkley)

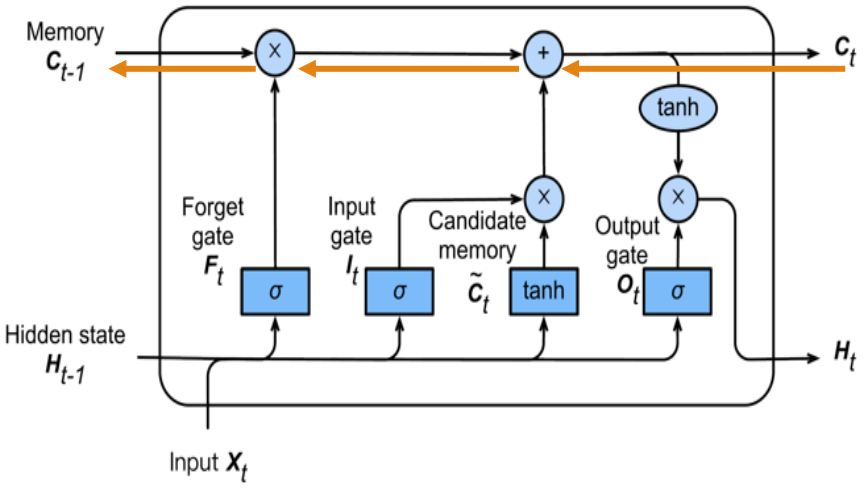
How does LSTM solve vanishing gradients?

- The LSTM architecture makes it easier for the RNN to preserve information over many timesteps
  - e.g. if the forget gate is set to 1 for a cell dimension and the input gate set to 0, then the information of that cell is preserved indefinitely.
  - By contrast, it's harder for vanilla RNN to learn a recurrent weight matrix W_h that preserves info in hidden state

- LSTM doesn't *guarantee* that there is no vanishing/exploding gradient, but it does provide an easier way for the model to learn long-distance dependencies

Slide credit: Christopher Manning , Abigail See and John Hewitt

How does LSTM solve vanishing gradients?

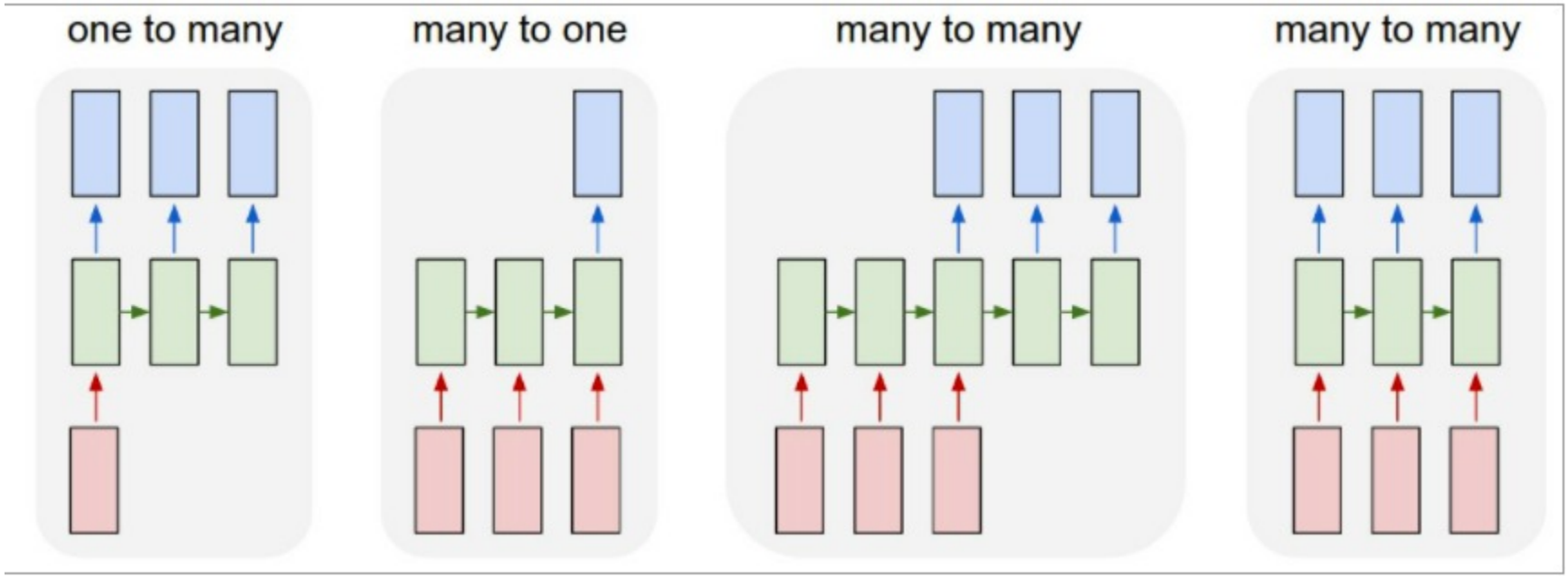


Back propagation from c_t to c_{t-1} : only elementwise multiplication with f, no matrix multiplication by W

$$C_t = F_t \odot C_{t-1} + I_t \odot \tilde{C}_t$$

Slide credit: Alex Smola and Mu Li , Justin Johnson

RNN/LSTM Applications



http://karpathy.github.io/2015/05/21/rnn-effectiveness/

LSTMs: real-world success

- In 2013–2015, LSTMs started achieving state-of-the-art results
  - Successful tasks include handwriting recognition, speech recognition, machine translation, parsing, and image captioning, as well as language models
  - LSTMs became the dominant approach for most NLP tasks

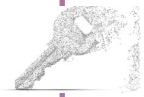
- Now (2021), other approaches (e.g., Transformers) have become dominant for many tasks
  - For example, in **WMT** (a Machine Translation conference + competition):
  - In WMT 2016, the summary report contains "RNN" 44 times
  - In WMT 2019: "RNN" 7 times, "Transformer" 105 times

Slide credit: Christopher Manning , Abigail See and John Hewitt

What we Learnt

Context is key

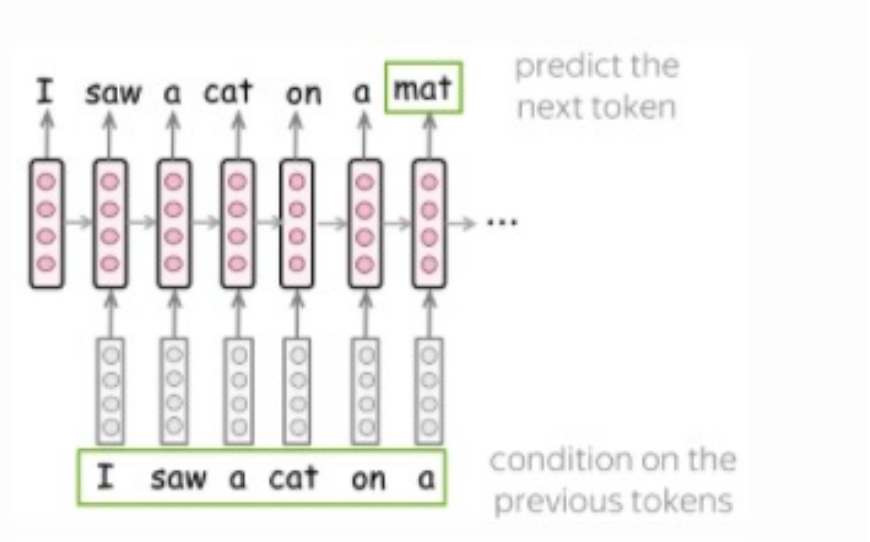
- Previous models (GloVe, Word2Vec, etc.) only have one representation per word
  - They can't capture these ambiguities.
- We just have one representation for a word, but words have different **aspects**, including semantics, syntactic behavior, and register/connotations.



- When you only have one representation, all levels of meaning are combined.
- Solution: have multiple levels of understanding.
  - Embeddings from ***Language Model*** representations

Source: https://lena-voita.github.io/nlp_course/transfer_learning.html

Language Model



- Those RNN layers are trained to predict the next word
- But those language models are producing context-specific word representations at each position

Image source https://lena-voita.github.io/nlp_course/seq2seq_and_attention.htm

Language Model – Context Specific Embeddings

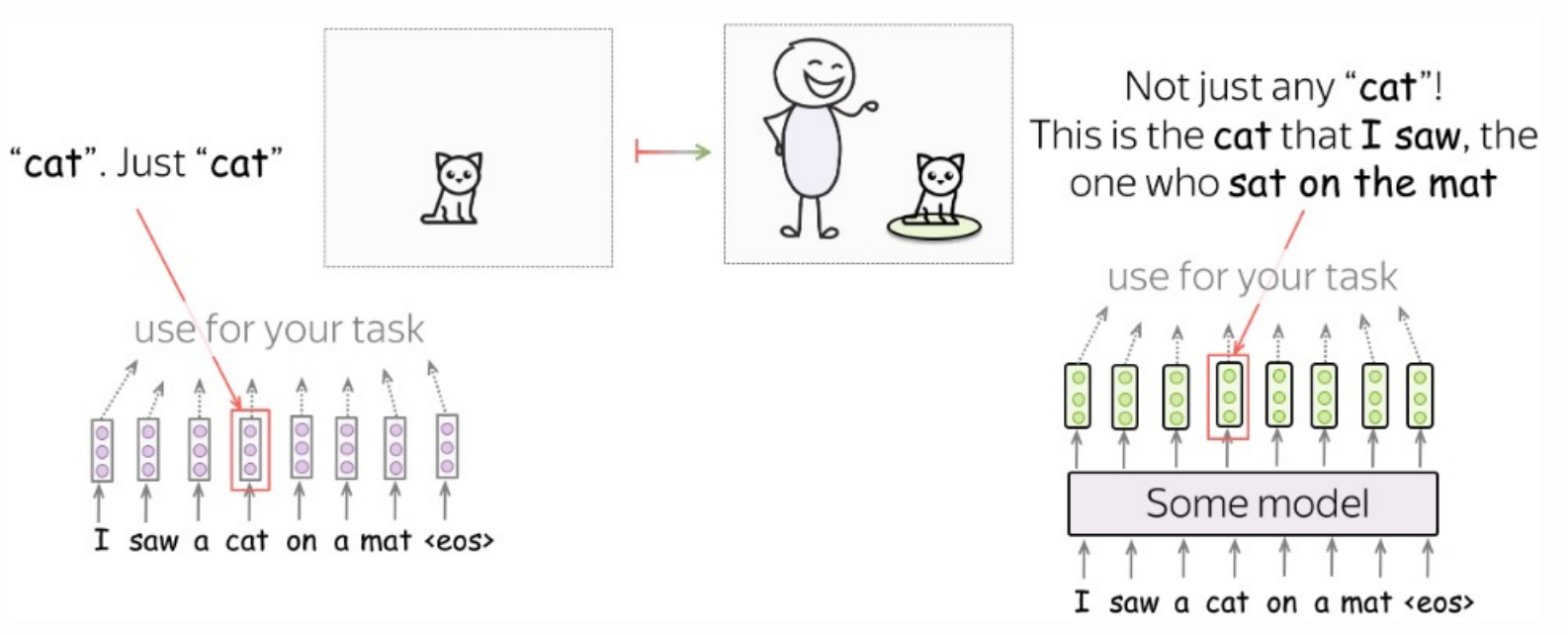
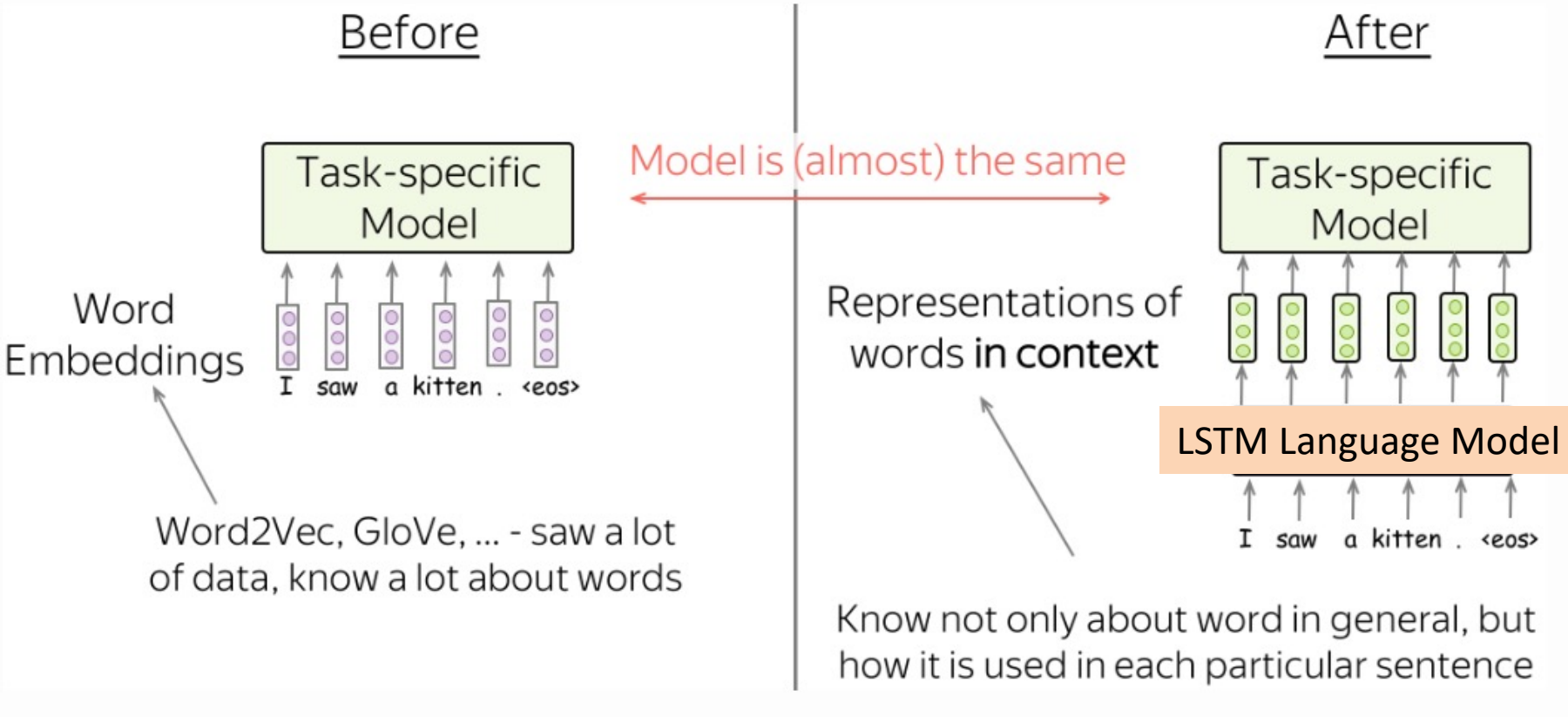


Image source https://lena-voita.github.io/nlp_course/seq2seq_and_attention.htm

Language Model – Context Specific Embeddings



Source: https://lena-voita.github.io/nlp_course/transfer_learning.html

Embeddings from Language/Translation Models

Source: https://lena-voita.github.io/nlp_course/transfer_learning.html

RNN/LSTM solves following issues

Polysemy: Both assign one vector per word, unable to differentiate meanings for words like "bank," which can signify a river's edge or a financial institution.

Static Representations: Word2Vec and GloVe produce fixed embeddings, not adapting to varying word meanings in different contexts.

Word Order Sensitivity: These models miss the word order, so phrases like "free drug" and "drug free" might get similar representations despite contrasting meanings.

Issues With RNN/LSTM

Recurrent computation is slow:

RNNs process data sequentially, which means each step must wait for the previous step to complete. This sequential dependency slows down the computation, particularly compared to models that can process data in parallel.

In practice, difficult to access information from many steps back:

RNNs are theoretically capable of handling long-range dependencies, but in practice, they often struggle to learn and retain information from many steps back in the input sequence. This is due to issues like vanishing gradients, where the contribution of information decays geometrically over time, making it hard for the RNN to maintain long-term dependencies

Still needs Task Specific Models

ULMfit: A New Approach to Transfer Learning

Fine-tune the Language model for downstream tasks instead of training Task specific models

ULMfit

- **Pretraining**: Train LM on big general domain corpus
- **Domain Adaptation**: Tune LM on target task data (Task is same as a pre-training task)
- Fine-tune as classifier on target task

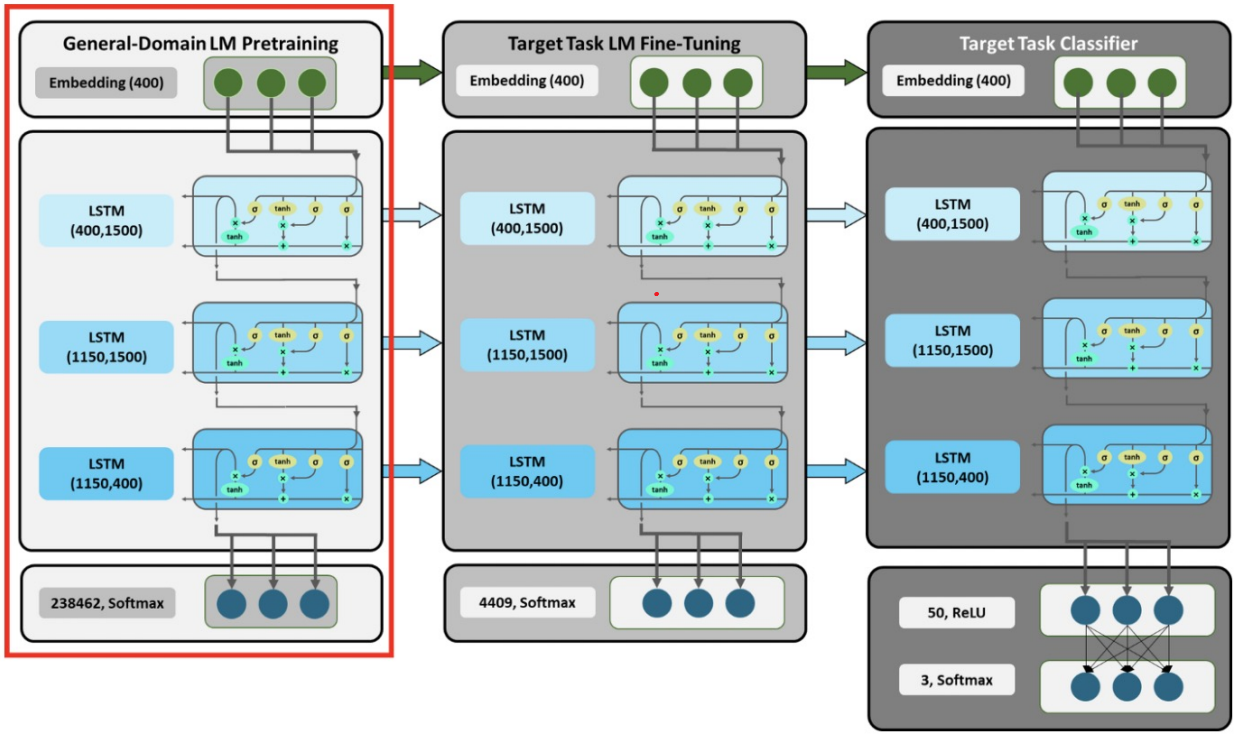


Image source: https://towardsdatascience.com/understanding-ulmfit-and-elmo-the-shift-towards-transfer-learning-in-nlp-b5d8e2e3f664

ULMfit emphases

- A lot of care in LM fine-tuning
  - Different per-layer learning rates
  - Slanted triangular learning rate (STLR) schedule
- Gradual layer unfreezing and STLR when learning classifier

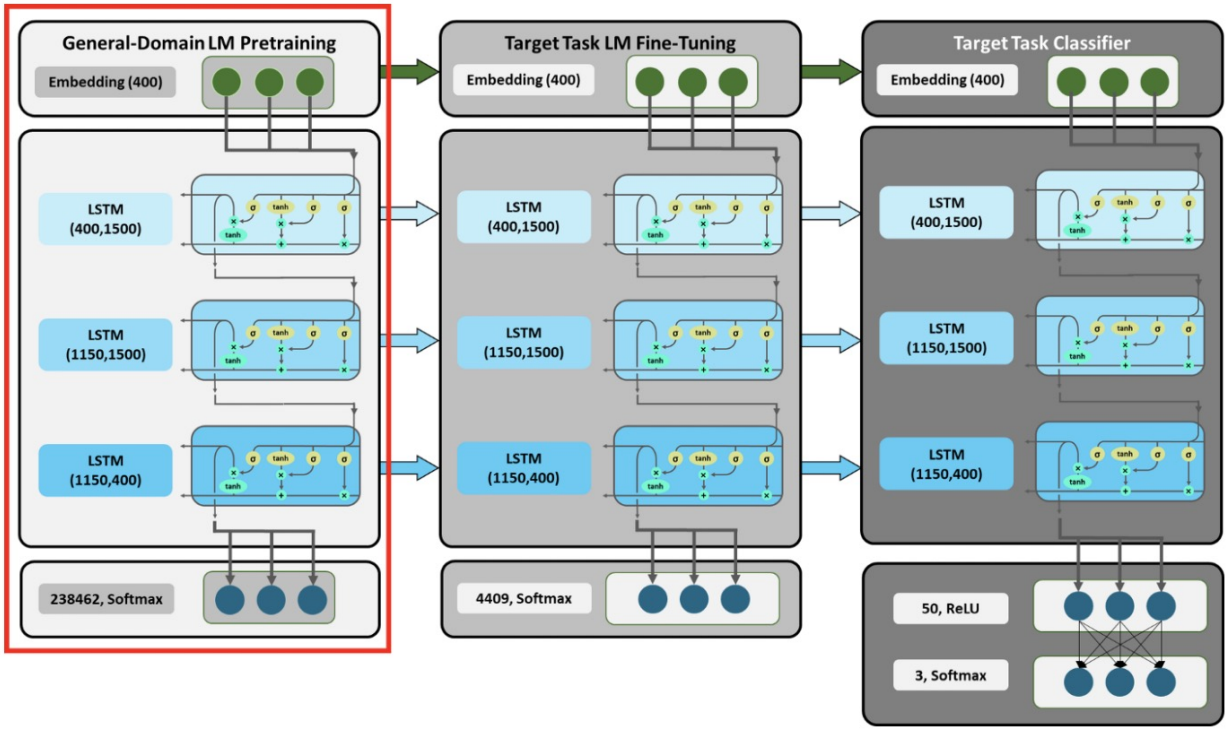


Image source: https://towardsdatascience.com/understanding-ulmfit-and-elmo-the-shift-towards-transfer-learning-in-nlp-b5d8e2e3f664

ULMfit performance

	Model	Test		Model	Test
IMDb	CoVe (McCann et al., 2017)	8.2	TREC-6	CoVe (McCann et al., 2017)	4.2
	oh-LSTM (Johnson and Zhang, 2016)	5.9		TBCNN (Mou et al., 2015)	4.0
	Virtual (Miyato et al., 2016)	5.9		LSTM-CNN (Zhou et al., 2016)	3.9
	ULMFiT (ours)	**4.6**		ULMFiT (ours)	**3.6**

ULMfit transfer learning