CityU
香港城市大學（東莞）
City University of Hong Kong (Dongguan)
Celebrating the Establishment in Year 2024

Lecture 3: Word Embedding

CS6493 Natural Language Processing

Instructor: Linqi Song

Outline

- 1. Word embedding definition and principles
- 2. Embedding methods – word2vec
  - Continuous bag-of-words
  - Skip-gram
- 3. Improve training efficiency
  - Negative sampling
  - Hierarchical softmax
- 4. Other word embedding methods
  - GloVe
- 5. Contextualized word embeddings (ELMo)

Meaning of words in human languages

Definition from Webster dictionary:

- the thing one intends to convey especially by language
- the thing that is conveyed by language

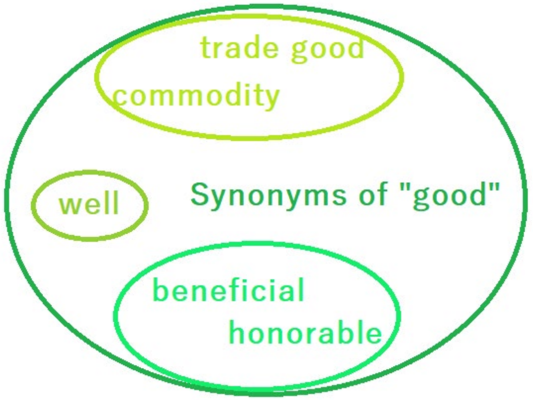
Linguistic way of thinking of meaning:

signifier (symbol) $\Leftrightarrow$ signified (idea or thing) = denotational semantics

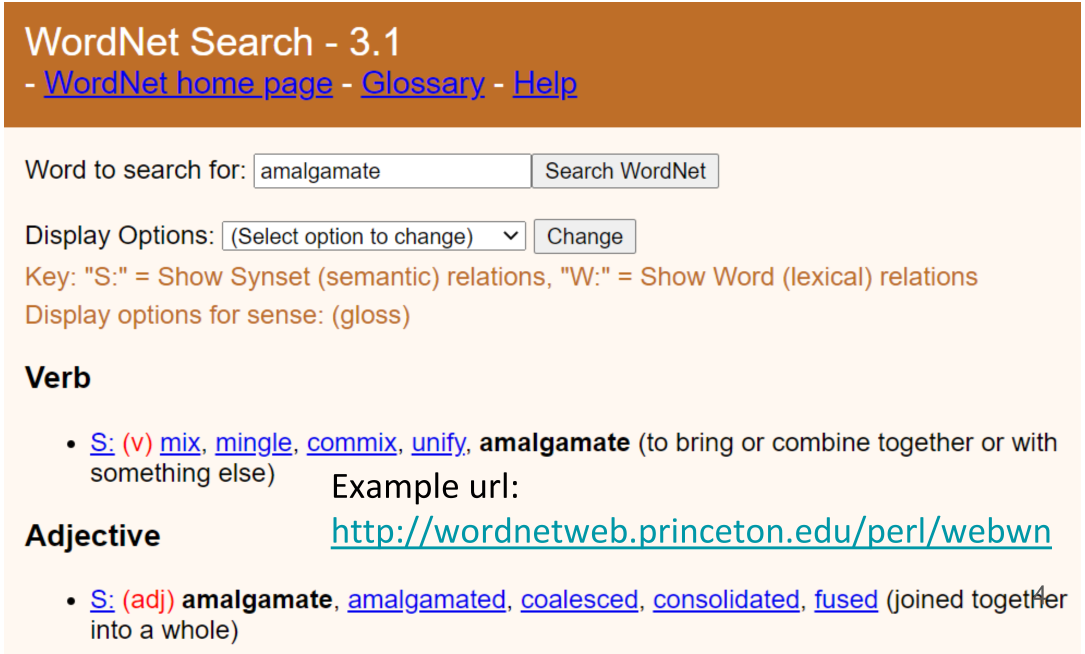
WordNet: synonyms and hypernyms

Common NLP solutions: by lists of synonym sets and hypernyms

e.g., WordNet - a lexical database of semantic relations between words in more than 200 languages



‘red’ is the hypernym of ‘scarlet’, ‘vermilion’, and ‘crimson’



Example url:
http://wordnetweb.princeton.edu/perl/webwn

Problems with resources like WordNet

- Missing nuance
  - Is “proficient” always a synonym of “good”?
- Impossible to keep up-to-date
  - Example: ninja (a person skilled in the Japanese art of ninjutsu), bombest (a word to describe something that is amazing), nifty (particularly good, skillful, or effective)
- Subjective
- Human labor for creation and adaptation
- Cannot compute accurate word similarity

One-hot vector: discrete symbols

- A localist representation
- One-hot vectors
  - one 1, the rest 0s

    hotel = [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
    motel = [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0 ,0 ,0 ,0, 0, 0, 0, 0]

- Vector diemension = number of words in vocabulary

Problems with discrete symbol representation

Consider the consine similarity of the “hotel, motel” examples

There’s no natural notation for one-hot vectors!

Solution

Learn to encode similarity in the vectors themselves.

Distributional hypothesis

Distributional hypothesis (1)

- Words that occur in similar contexts tend to have similar meanings
- Proposed by J. R. Firth in 1957
- "You shall know a word by the company it keeps."
- One of the most successful ideas of modern statistical NLP

John Rupert Firth

Distributional hypothesis (2)

- When a word *w* appears in a text, its context is the set of words that appear nearby (with a fixed-size window)
- Use many contexts of *w* to build up a representation of *w*
- In the example, the context words will represent ***banking***

Example: *context words represent banking*

... government debt problems turning into **banking** crises as happened in 2009...

...saying that Europe needs unified **banking** regulation to replace the hodgepodge...

...India has just given its **banking** system a shot in the arm...

Inspired by distributional hypothesis

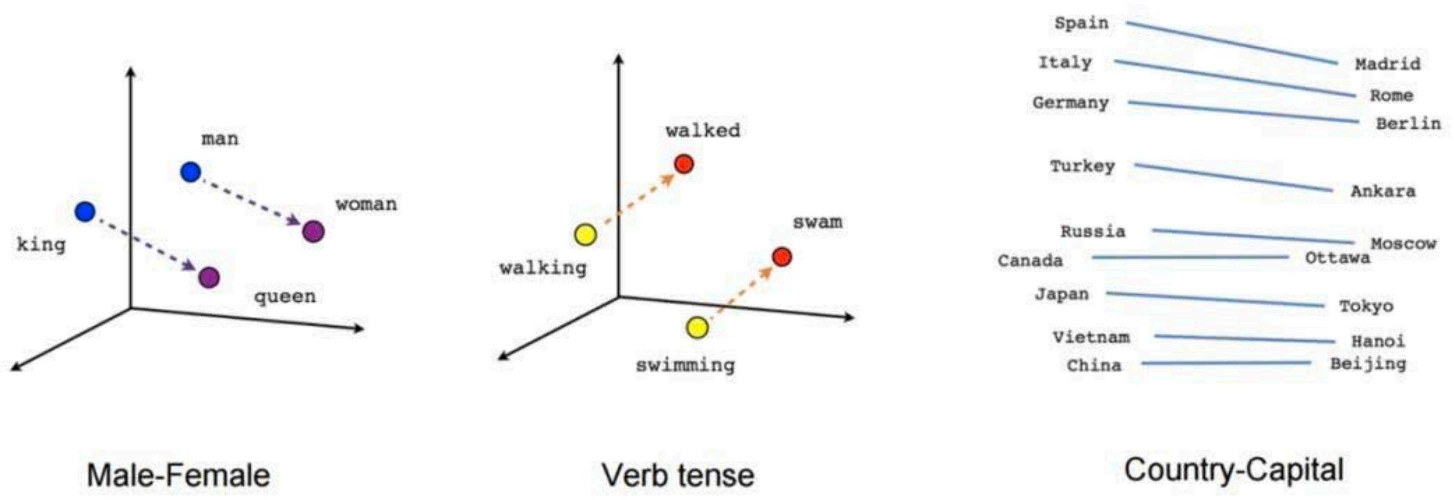
- Latent Semantic Analysis – representation
  - (Deerwester et al, 1990): co-occurrence counting + SVD
- Collobert & Weston vectors - first neural pretrained word embedding
  - (Collobert et al, 2008: A unified architecture for natural language processing) based on joint probability of target and context words, word embedding to support multiple downstream tasks
- Word2vec - learn good vector presentations for words/phrases
  - (Mikolov et al, 2013 two papers: Distributed representations of sentences and documents, Efficient estimation of word representations in vector space) word2vec, negative sampling and hierarchical softmax
- GloVe – global statistics (LSA) + local contexts (word2vec)
  - (Pennington et al., 2014: GloVe: Global Vectors for Word Representation): co-occurrence matrix to capture global statistics and local contexts training

Word embeddings - goal

- Build a dense vector for each word
- A word vector should be similar to vectors of words that appear in similar contexts
- Word embeddings also called word vectors or (neural) word representation
- A distributed representation

$$\textit{banking} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$$

Word2vec examples



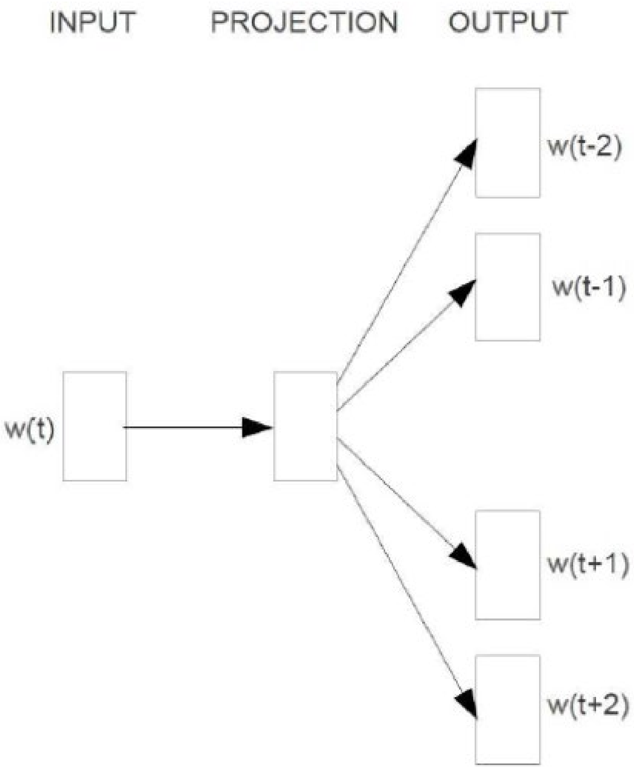
Word2vec

Word2vec (Mikolov et al. 2013) is a framework for learning word vectors.

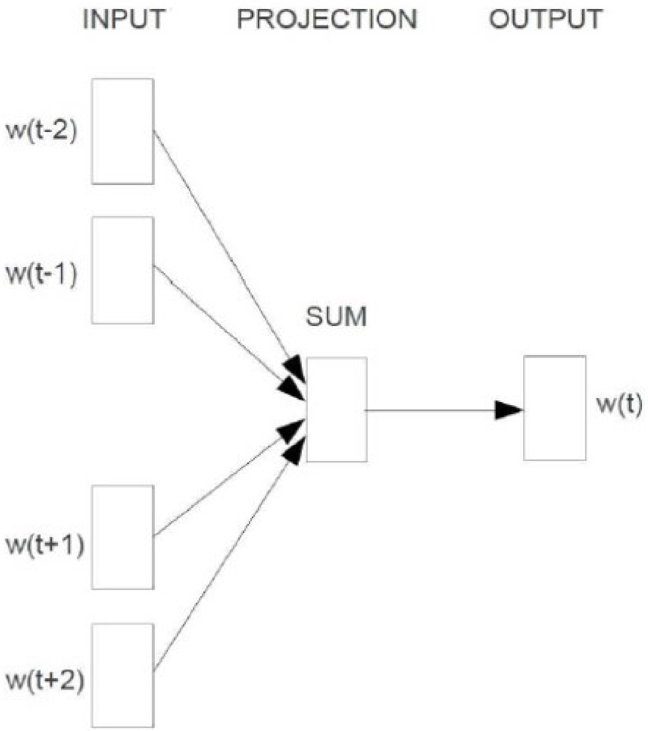
Idea:

- **Input**: Given a large corpus of text (e.g., a bunch of sentences or documents)
- **Output**: Every word in a fixed vocabulary is represented by a vector
- Go through each position *t* in the text, which has a target word (or center word) w_t and several context words w_c
- Use the similarity of the word vectors for w_t and w_c
    - Skip-gram: to calculate the probability of *context words* w_c given *the target word* w_t
    - Continuous bag of words (CBOW): to calculate the probability of target word w_t given context words w_c
- Keep adjusting the word vectors to maximize the probability

Skip-gram vs CBOW (1)

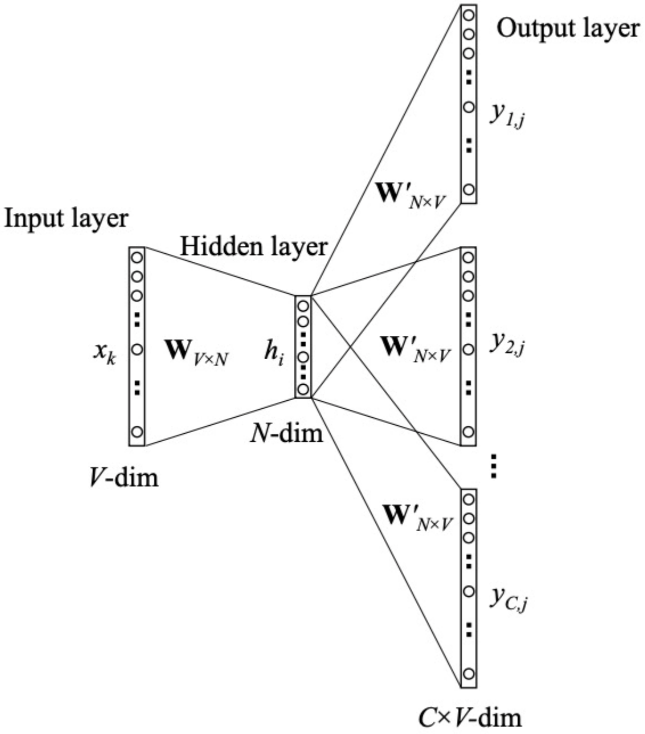


Skip-gram model

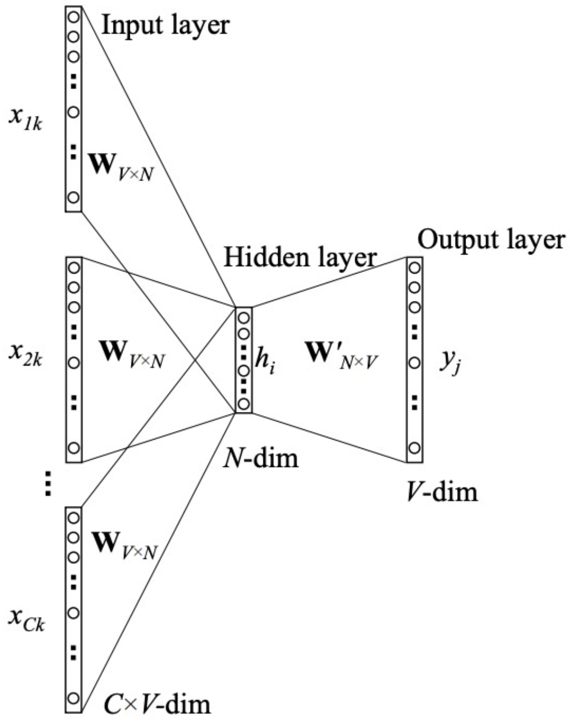


CBOW model

Skip-gram vs CBOW (2)



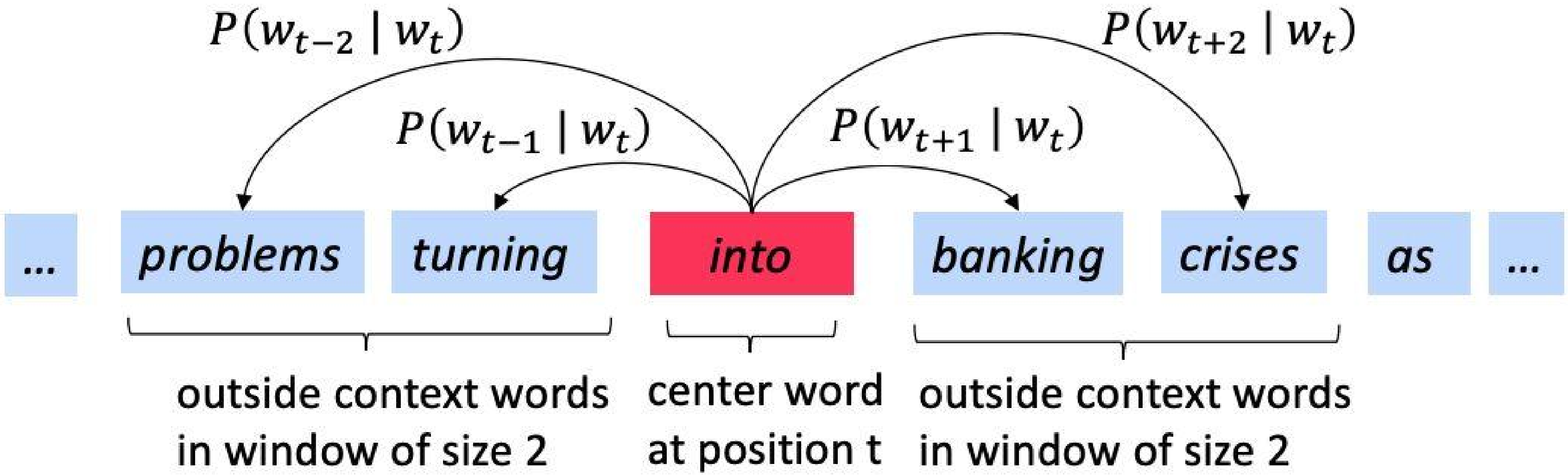
Skip-gram network structure



CBOW network structure

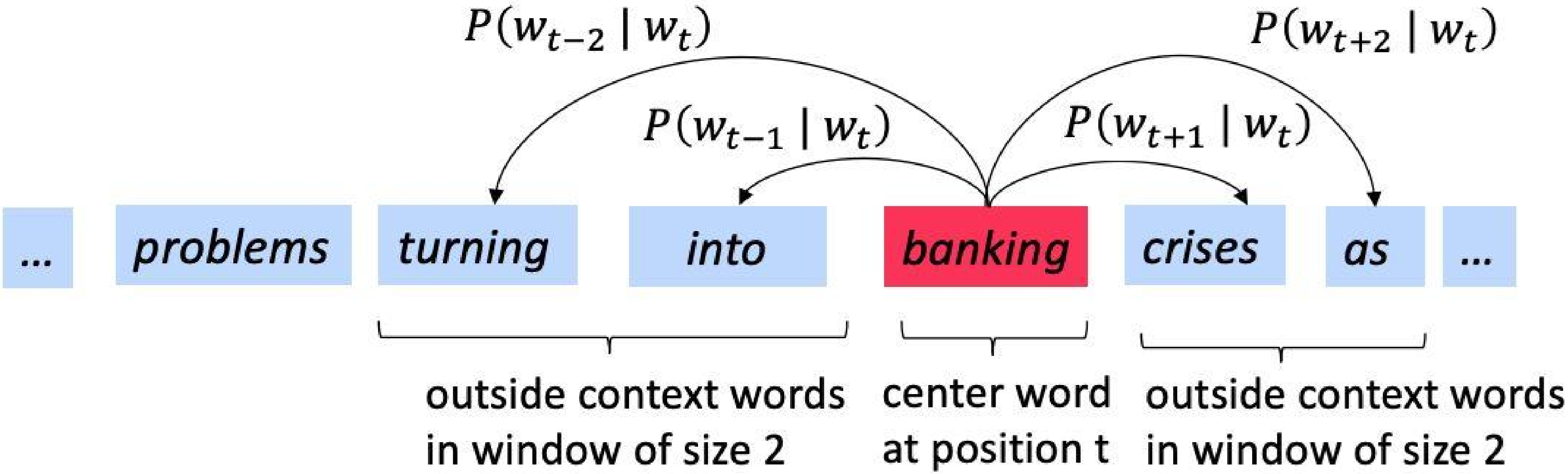
Skip-gram (1)

Computing P(context words | “into”)



Skip-gram (2)

Computing P(context words | “banking”)



Skip-gram's optimization problem

For each position $t = 1, \dots, T$, predict context words within a window of fixed size m, given center word w_j. Data likelihood:

$$\text{Likelihood} = L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \leq j \leq m \\ j \neq 0}} P\left(w_{t+j} \mid w_t; \theta\right)$$

θ is all variables to be optimized

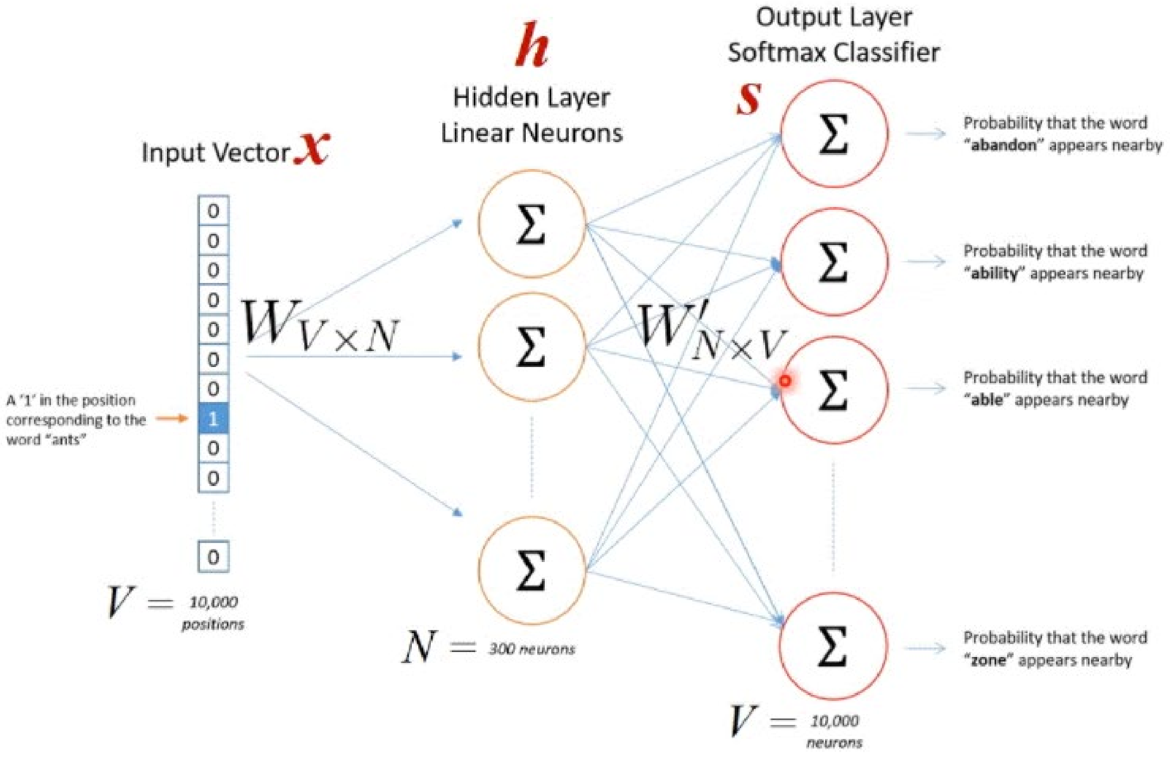
sometimes called a *cost* or *loss* function

The objective function $J(\theta)$ is the (average) negative log likelihood:

$$J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \leq j \leq m \\ j \neq 0}} \log P\left(w_{t+j} \mid w_t; \theta\right)$$

Minimizing objective function $\Leftrightarrow$ Maximizing predictive accuracy

Network structure of skip-gram (1)



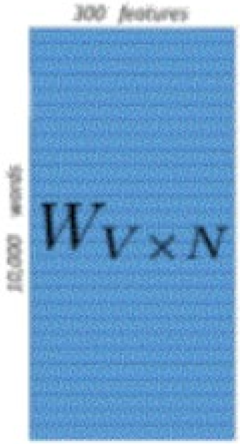
- Input ***X*** as one-hot encoding
- *V*: size of vocabulary
- *N*: number of hidden layers in ***h***
- The number of nodes in each output layer ***S*** = *V*.

Network structure of skip-gram (2)

Hidden layer weight matrix = word vector lookup

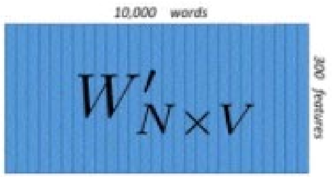
$$h = x^T W = W_{(k,.)} := v_{w_I}$$

$$[0 \quad 0 \quad 0 \quad \boxed{1} \quad 0] \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ 10 & 12 & 19 \\ 11 & 18 & 25 \end{bmatrix} = [10 \quad 12 \quad 19]$$



Network structure of skip-gram (3)

Output layer weight matrix = weighted sum as final score

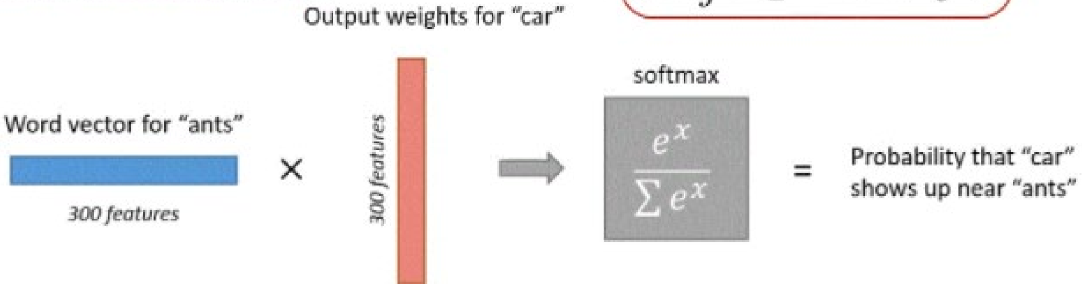


$$s_j = h v'_{w_j}$$

$$p(w_j = w_{O,c} \mid w_I) = y_{j_c} = \boxed{\frac{\exp(s_{j_c})}{\sum_{j'=1}^{V} \exp(s_{j'})}}$$

within the context window

softmax



Softmax output as co-occurence probability

$$P(w_O|w_I) = \frac{\exp v'^T_O v_I}{\sum_{w \in Voc} \exp(v'^T_w v_I)}$$

We have two vectors for each word ***w*** in the vocabulary:

- v_w when *w* is a center word (row vectors of $W_{V \times N}$ matrix)
- v'_w when *w* is a context word (column vectors of $W'_{N \times V}$ matrix)
- $v'^T_i v_j$ measures how likely context word *i* appears with center word *j*. Larger product = larger probability.
- **exp()** makes everything positive
- $\sum_{w \in Voc} \exp(v'^T_w v_I)$ normalize over the entire vocabulary to give probability distribution

Loss function for a given target word

Given a target word (w_I)

$$
\begin{aligned}
C(\theta) &= -\log p(w_{O,1}, w_{O,2}, \cdots, w_{O,C} \mid w_I) \\
&= -\log \prod_{c=1}^{C} \frac{\exp(s_{j_c})}{\sum_{j'=1}^{V} \exp(s_{j'})} \\
&= -\sum_{c=1}^{C} s_{j_c} + C \log \sum_{j'=1}^{V} \exp(s_{j'})
\end{aligned}
$$

$$\boxed{s_j = {v'_{w_j}}^T \cdot h}$$

Backpropagation

Given a target word (w_I)

$$\frac{\partial C(\theta)}{\partial w'_{ij}} = \sum_{c=1}^{C} \frac{\partial C(\theta)}{\partial s_{j_c}} \frac{\partial s_{j_c}}{\partial w'_{ij}} = \sum_{c=1}^{C} (y_{j_c} - t_{j_c}) \cdot h_i$$

$$\boxed{s_j = {v'_{w_j}}^T \cdot h}$$

$$\frac{\partial C(\theta)}{\partial s_{j_c}} = y_{j_c} - t_{j_c} := e_{j_c}$$

error term

=1, when w_{jc} is within the context window
=0, otherwise

$$\boxed{{w'_{ij}}^{(t+1)} = {w'_{ij}}^{(t)} - \eta \cdot \sum_{c=1}^{C} (y_{j_c} - t_{j_c}) \cdot h_i}$$

Backpropagation (cont.)

$$\frac{\partial C(\theta)}{\partial w_{ki}} = \frac{\partial C(\theta)}{\partial h_i}\frac{\partial h_i}{\partial w_{ki}} = \sum_{j=1}^{V}\sum_{c=1}^{C}(y_{j_c} - t_{j_c}) \cdot w'_{ij} \cdot x_k$$

$$\boxed{h = x^T W}$$

$$\frac{\partial C(\theta)}{\partial h_i} = \boxed{\sum_{j=1}^{V}} \frac{\partial C(\theta)}{\partial s_j}\frac{\partial s_j}{\partial h_i} = \sum_{j=1}^{V}\sum_{c=1}^{C}(y_{j_c} - t_{j_c}) \cdot w'_{ij}$$

$$\boxed{s_j = {v'_{w_j}}^T \cdot h}$$

$$\boxed{w_{ij}^{(t+1)} = w_{ij}^{(t)} - \eta \cdot \sum_{j=1}^{V}\sum_{c=1}^{C}(y_{j_c} - t_{j_c}) \cdot w'_{ij} \cdot x_j}$$

SGD update via backpropogation

$$w'^{(t+1)}_{ij} = w'^{(t)}_{ij} - \eta \cdot \sum_{c=1}^{C} (y_{j_c} - t_{j_c}) \cdot h_i$$

$$\boxed{EI_j = \sum_{c=1}^{C} (y_{j_c} - t_{j_c})}$$

$$\boxed{v'_{w_j}{}^{(t+1)} = v'_{w_j}{}^{(t)} - \eta \cdot EI_j \cdot h}$$

$$w^{(t+1)}_{ij} = w^{(t)}_{ij} - \eta \cdot \sum_{j=1}^{V} \sum_{c=1}^{C} (y_{j_c} - t_{j_c}) \cdot w'_{ij} \cdot x_j$$

$$\boxed{v^{(t+1)}_{w_I} = v^{(t)}_{w_I} - \eta \cdot EH^T}$$

$$\boxed{EH_i = \sum_{j=1}^{V} EI_j \cdot w'_{ij} \cdot x_j}$$

large vocabularies or large training corpora → expensive computations

Improve the efficiency in training

Reason:

- The size of vocabulary V is impressively large
- Evaluation of the objective function would take $O(V)$ time

Solution:

- Negative sampling
- Hierarchical softmax

Comparison:

- Hierarchical softmax tends to be better for infrequent words.
- Negative sampling works better for frequent words and lower dimensional vectors.

Negative sampling

- A simplified version of NCE (Noise Contrastive Estimation)
- Sample from a noise distribution $P_n(w)$
- The probabilities in $P_n(w)$ match the ordering of the frequency in the vocabulary
- Pick out k words from $P_n(w)$, training together with the center word
- Convert to (k+1) binary classification problems
- e.g., in tensorflow, the probability distribution to select samples: (decreasing in s(w))

$$P(w_i) = \frac{\log(s(w_i)+2) - \log(s(w_i)+1)}{\log(W+1)}$$

where s(w) is the index of word w in the dictionary according to word frequencies descendingly

Negative sampling example

- Target word w_i and context word w_j
- From $P_n(w)$, based on a certain probability distribution, pick out *k* words $\widetilde{w}_1, \widetilde{w}_2, \dots, \widetilde{w}_k$
- Positive sample: $\{w_i, w_j\}$
- Negative samples: $\{w_i, \widetilde{w}_1\}, \dots, \{w_i, \widetilde{w}_k\}$
- Then given w_i, predict the occurrence of w_j using binary classifications:
  - w_i co-occurs with w_j: truth label 1
  - w_i does not co-occur with any $\widetilde{w}_{k'}$ $(1 \leq k' \leq k)$: truth label 0

Negative sampling example (cont.)

- The prob. of correctly distinguishing a positive sample: $P\left(D=1|\{w_i,w_j\}\right)=\sigma\left(u_j^T v_i\right)=\frac{1}{1+\exp(-u_j^T v_i)}$
- The prob. of correctly distinguishing all negative samples:

$$\Pi_{k'=1}^{k} P(D=0|\{w_i,\widetilde{w}_{k'}\})=\Pi_{k'=1}^{k}\left(1-\sigma\left(u_{k'}^T v_i\right)\right)=\Pi_{k'=1}^{k}\sigma\left(-u_{k'}^T v_i\right)=\Pi_{k'=1}^{k}\frac{1}{1+\exp(u_{k'}^T v_i)}$$

- Maximize the probability of correctly distinguishing all the positive and negative samples:

$$\max P\left(D=1|\{w_i,w_j\}\right)\Pi_{k'=1}^{k} P(D=0|\{w_i,\widetilde{w}_{k'}\})$$

- Or the negative log likelihood (cross-entropy) loss function:

$$\min\ -\log\sigma\left(u_j^T v_i\right)-\log\Pi_{k'=1}^{k}\sigma\left(-u_{k'}^T v_i\right)=-\log\frac{1}{1+\exp(-u_j^T v_i)}-\sum_{k'}\log\frac{1}{1+\exp(u_{k'}^T v_i)}$$

cross-entropy in binary classification

time complexity:
O(k) << O(V)

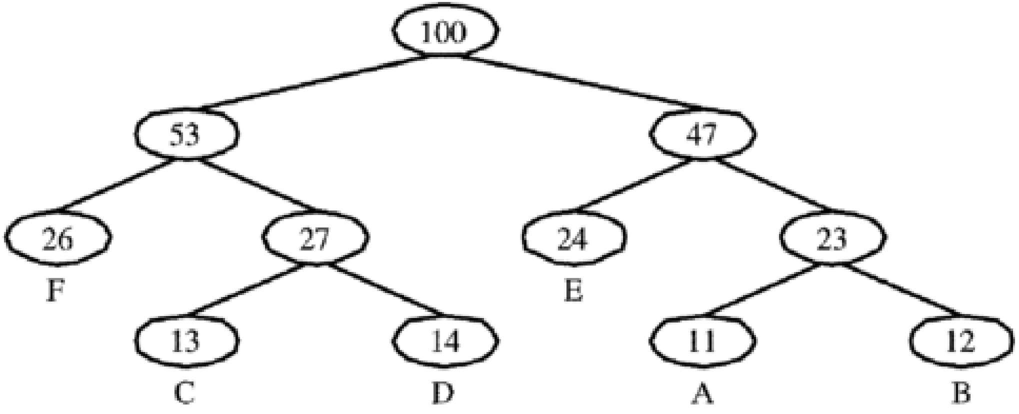
Hierarchical softmax

- Huffman tree: the binary tree with minimal external path weight
- Construct a Huffman tree, with each leaf node representing a word
- Each internal node (a cluster of similar words) of the graph (except the root and the leaves) is associated to a vector that the model is going to learn.
- The probability of a word w given a vector w_i, $P(w|w_i)$, is equal to the probability of a random walk starting at the root and ending at the leaf node corresponding to w.
- Complexity: $O(\log(V))$, corresponding to the length of the path.

Huffman tree

Symbol	Frequency	Encoding type		
		One	Two	Three
A	11	000	111	000
B	12	001	110	001
C	13	100	011	010
D	14	101	010	011
E	24	01	10	10
F	26	11	00	11

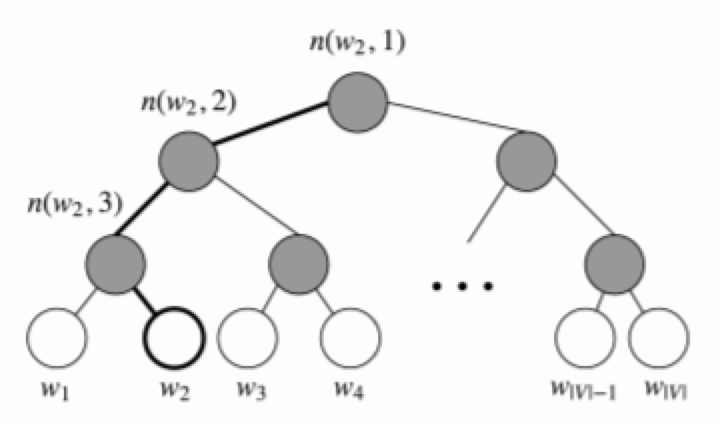
(a)



(b)

Construct a Huffman tree: merge two nodes with the minimum frequencies and consider them together as a single node; repeat until there is only one node.

Hierarchical softmax example (1)



- Let $L(w)$ be the number of nodes (length+1) in the path from the root to the leaf *w*.
- *n(w, i)* as the *i*-th node on this path with associated vector $v_{n(w,i)}$.
- For each inner node *n*, we arbitrarily choose one of its children and call it *ch(n)*.
- The probability can be calculated as follows:

$$P(w|w_i) = \prod_{j=1}^{L(w)-1} \sigma([n(w,j+1) = ch(n(w,j))] \cdot v_{n(w,j)}^T v_{w_i}) \quad [x] = \begin{cases} 1 \text{ if } x \text{ is true} \\ -1 \text{ otherwise} \end{cases}$$

Hierarchical softmax example (2)

$$P(w|w_i) = \prod_{j=1}^{L(w)-1} \sigma(\boxed{[n(w, j+1) = ch(n(w,j))]} \cdot v_{n(w,j)}^T v_{w_i})$$

- compute a product of terms based on the shape of the path.

  e.g., assume ch(n) is always the left node of n, then the term returns 1 when the path goes left, and -1 if right.

- provide normalization.

  e.g., at a node n, if we sum the probabilities for going to the left and right node, you can check that for any value of $v_n^T v_{w_i}$,

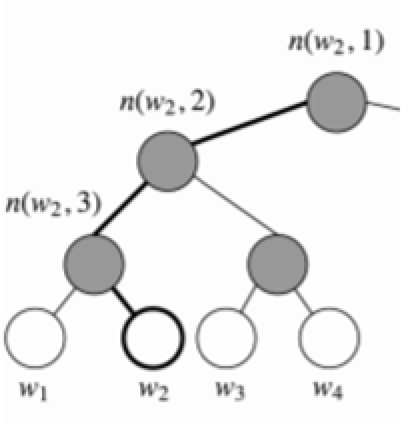
  $$\sigma(v_n^T v_{w_i}) + \sigma(-v_n^T v_{w_i}) = 1$$

  $$\Sigma_{w=1}^{|V|} P(w|w_i) = 1$$

Hierarchical softmax example (3)

$$P(w|w_i) = \prod_{j=1}^{L(w)-1} \boxed{\sigma([n(w,j+1) = ch(n(w,j))] \cdot v_{n(w,j)}^T v_{w_i})}$$

- The dot product compares the similarity of our input vector v_{w_i} to each inner node vector $v_{n(w,j)}^T$.



- Take two left edges and then a right edge to reach w_2 from the root

$$\begin{aligned} P(w_2|w_i) &= p(n(w_2,1),\text{left}) \cdot p(n(w_2,2),\text{left}) \cdot p(n(w_2,3),\text{right}) \\ &= \sigma(v_{n(w_2,1)}^T v_{w_i}) \cdot \sigma(v_{n(w_2,2)}^T v_{w_i}) \cdot \sigma(-v_{n(w_2,3)}^T v_{w_i}) \end{aligned}$$

Hierarchical softmax example (4)

$$P(w|w_i) = \prod_{j=1}^{L(w)-1} \sigma([n(w,j+1) = ch(n(w,j))] \cdot v_{n(w,j)}^T v_{w_i})$$

- Minimize negative log likelihood $-\log P(w|w_i)$
- Update the vectors of the nodes in the binary tree that are in the path from root to leaf node
- Speed of this method is determined by the way in which the binary tree is constructed and words are assigned to leaf nodes
- Huffman tree assigns frequent words shorter paths in the tree

Use word2vec package

- Using this package is extremely simple:
  - Download the code from Mikolov’s git repository: https://github.com/tmikolov/word2vec
  - Compile the package
  - Download the default corpus (wget http://mattmahoney.net/dc/text8.zip) or another corpus of your choice ◦
  - Train the model using the desired parameters
- Jupyter: code for downloading, compiling, and training

Word2vec performance – different window size

Word: **walk**

Window size = 3

Word	Cosine distance
go	0.488083
snipe	0.464912
shoot	0.456677
fly	0.449722
sit	0.449678
pass	0.442459
climbs	0.440931
walked	0.436502
ride	0.434034
stumble	0.426750
bounce	0.425577
travelling	0.419419
walking	0.412107
walks	0.410949
trot	0.410418
leaping	0.406744
sneak	0.401918
climb	0.399793
move	0.396715
wait	0.394463
going	0.391639
shouted	0.388382
roam	0.388073
thrown	0.384087
get	0.383894

Window size = 30

Word	Cosine distance
walking	0.486317
walked	0.430764
walks	0.406772
stairs	0.401518
go	0.399274
sidewalk	0.385786
stand	0.380480
cortege	0.371033
wheelchair	0.362877
strapped	0.360179
hollywood	0.356544
carousel	0.356187
grabs	0.356007
swim	0.355027
breathe	0.354314
tripped	0.352899
cheer	0.352477
moving	0.350943
inductees	0.347791
walkway	0.347164
shout	0.346229
pounding	0.340554
blvd	0.339121
crowd	0.338731
levada	0.334899

Word2vec performance – different No. iterations

Word: **walk**

No. of iterations = 1

Word	Cosine distance
walking	0.851438
walks	0.846485
bat	0.843796
ride	0.830734
crowd	0.821692
quiet	0.812538
spot	0.802777
steal	0.787917
door	0.787571
doors	0.786485
bed	0.773686
dinner	0.772160
shadow	0.769573
luck	0.768221
baby	0.767862
shoot	0.765968
walked	0.765739
sitting	0.765394
shirt	0.759116
rides	0.759047
watching	0.755140
watch	0.750808
gehrig	0.741494
shoots	0.740971
looking	0.740904

No. of iterations = 100

Word	Cosine distance
walked	0.483473
ride	0.470925
walks	0.470889
stand	0.449993
walking	0.449071
go	0.430172
shoot	0.421110
get	0.404258
move	0.403757
live	0.403347
fly	0.400929
climbs	0.396346
throw	0.391768
climb	0.384038
wiggle	0.380892
thrown	0.380426
pull	0.375478
goes	0.375406
moving	0.374447
pass	0.372463
conversing	0.364413
sit	0.362765
crowd	0.361651
kiss	0.359883
stay	0.357015

Word2vec performance – different dimensions

Word: **walk**

No. of dimensions = 5

Word	Cosine distance
catcher	0.998074
shirt	0.996589
lechuck	0.995313
bullseye	0.994644
bowler	0.994381
punter	0.993154
lovell	0.992815
heels	0.992255
whip	0.992085
outfit	0.992047
tore	0.991924
steals	0.991524
guybrush	0.991166
gigs	0.990291
hanging	0.990201
burns	0.990043
backing	0.989966
orser	0.989960
torch	0.989747
beat	0.989435
showdown	0.989381
feat	0.989242
cheers	0.988951
clad	0.988646
lunch	0.988326

No. of dimensions = 1000

Word	Cosine distance
walks	0.304954
walked	0.303322
snipe	0.287221
walking	0.272690
ride	0.266770
canter	0.251025
bandleaders	0.246454
climbs	0.233725
catapulted	0.230075
climb	0.229263
trot	0.228362
shouted	0.227306
stand	0.223288
seagulls	0.221745
fly	0.216602
fences	0.216366
lifts	0.215402
pray	0.214977
paws	0.214865
bounces	0.214449
shoot	0.213457
grabs	0.212018
walkway	0.211136
swim	0.209120
tumble	0.207765

Other word embedding models – GloVe (1)

- GloVe: Global Vectors for Word Representation
  - Global statistics (LSA) + local context window (word2vec)
  - Co-occurrence matrix, decreasing weighting: decay X_{ij}=1/d (distance of word pairs)

- 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

Other word embedding models – GloVe (2)

- Loss function

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

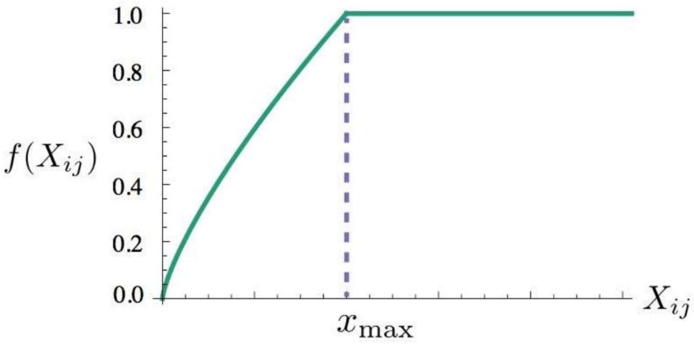
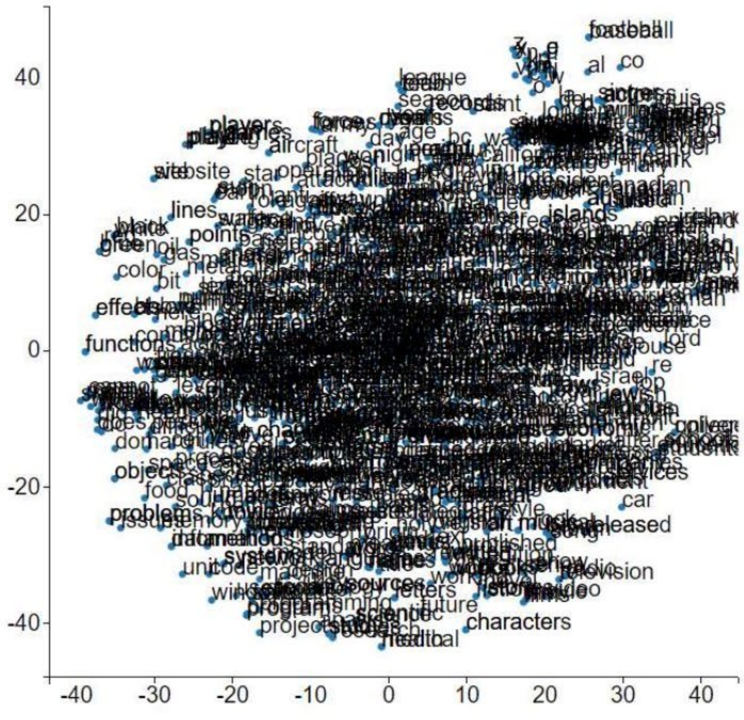
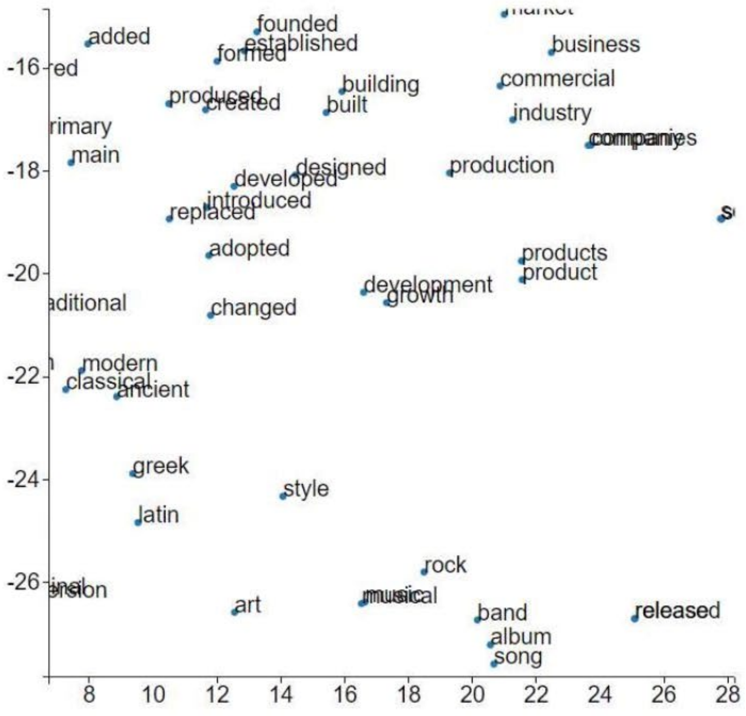


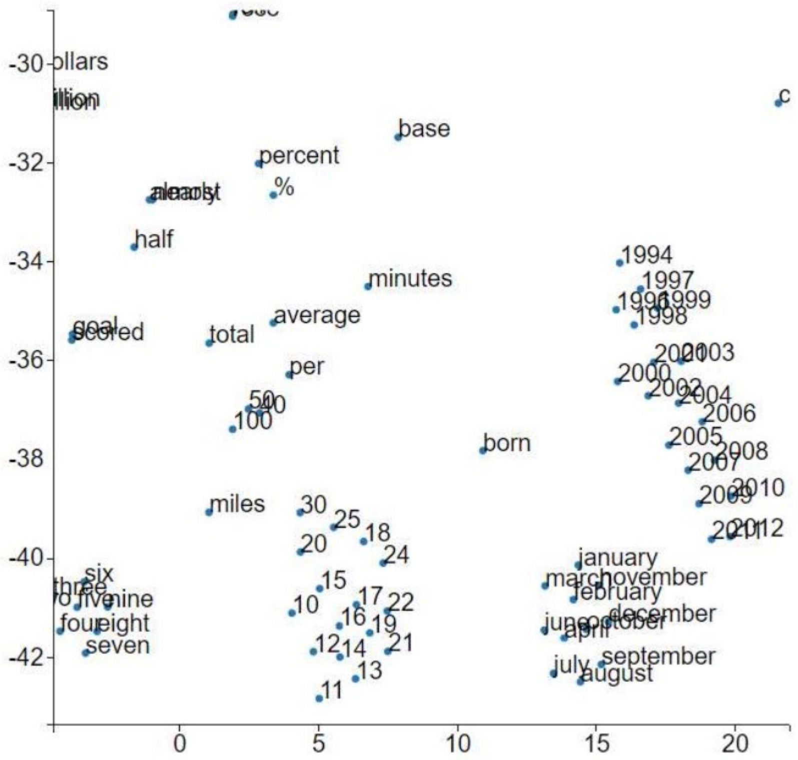
Figure 1: Weighting function f with $\alpha = 3/4$.

Visualizing word embeddings – word2vec





Visualizing word embeddings - GloVe

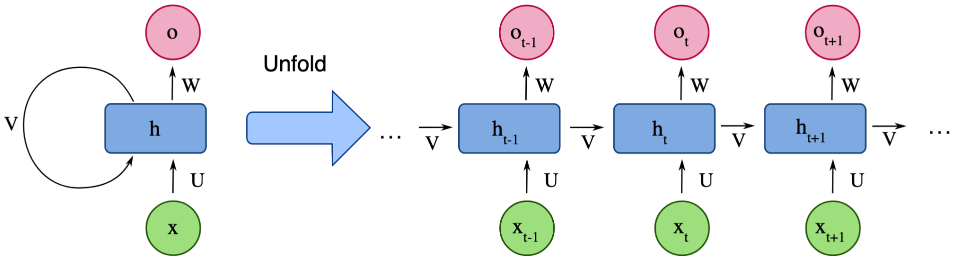


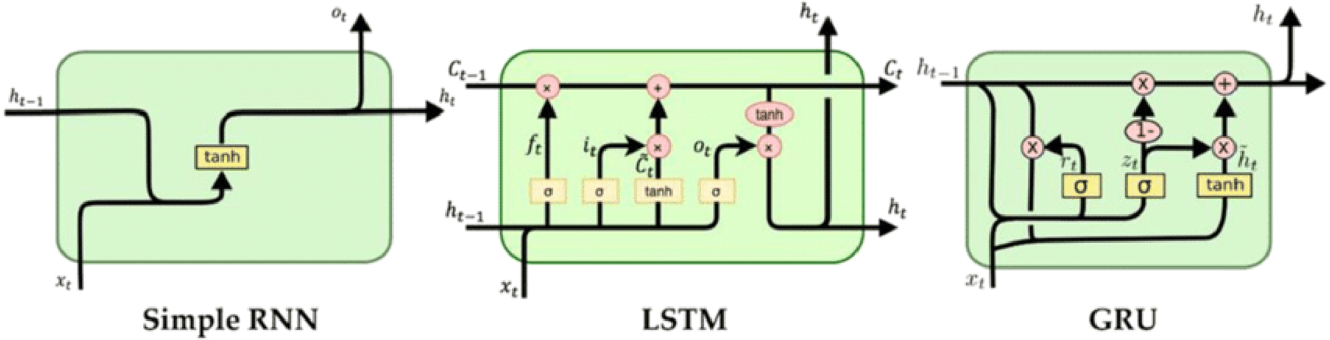
What we have learned so far (1)?

- Language model
  - Predict next words given a sequence of previous words
  - RNN-based language models

RNNs

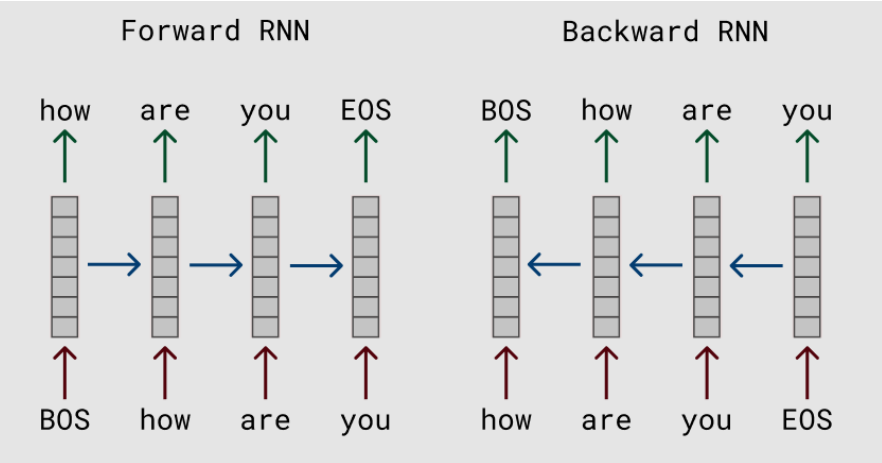
- Basic RNNs, LSTMs, GRUs

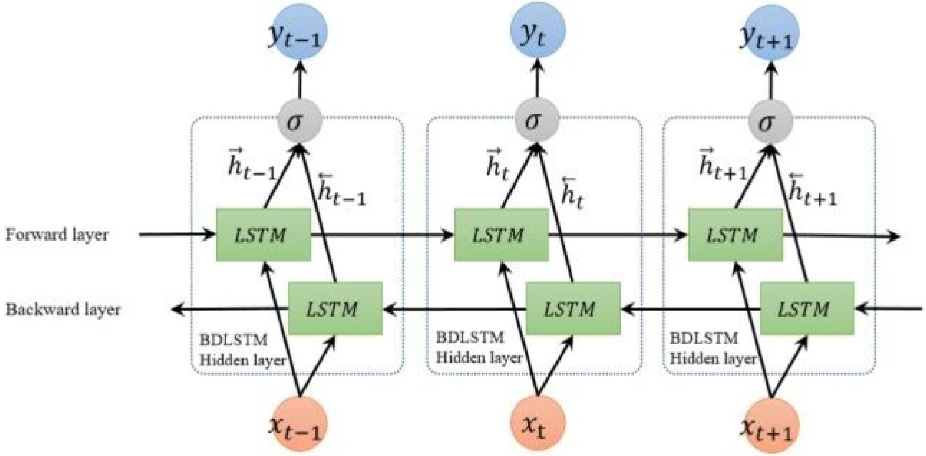




Bi-directional RNNs

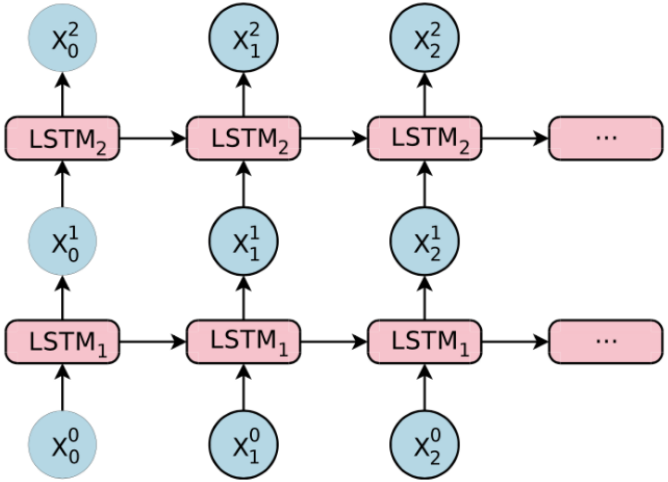
- RNNs can be bi-directional

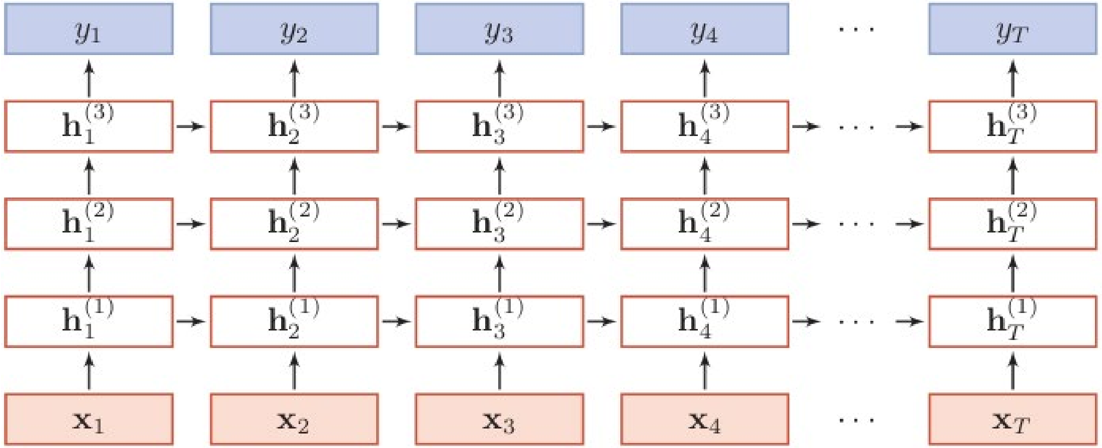




Stacked RNNs

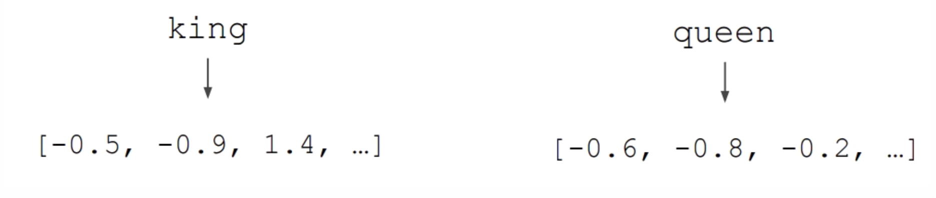
- RNNs can be stacked.
- For each input, multiple representations (hidden states) can be learned.





What we have learned so far (2)?

- Word embeddings
  - Represent each word with a vector
  - Word2vec: CBOW or skip-gram
  - GloVe: global statistics (co-occurrence) + local context window



Problems with (non-contextual) embeddings

The GloVe word embedding of the word ‘stick’ - a vector of 200 floats (rounded to two decimals). It goes on for two hundred values.

-0.34	-0.84	0.20	-0.26	-0.12	0.23	1.04	-0.16	0.31	0.06	0.30	0.33	-1.17	-0.30	0.03	0.09	0.35	-0.28	-(

What if for multi-sense words?

He sticks a stamp on the envelope.

The old lady leant on her stick as she talked.

Contextualized embeddings

ELMo: Deep contextualized word representations, AI2 & Univ. Washington, 2018

From context independent embeddings to context dependent embeddings

open a bank account   on the river bank

[0.3, 0.2, -0.8, …]

[0.9, -0.2, 1.6, …]   [-1.9, -0.4, 0.1, …]

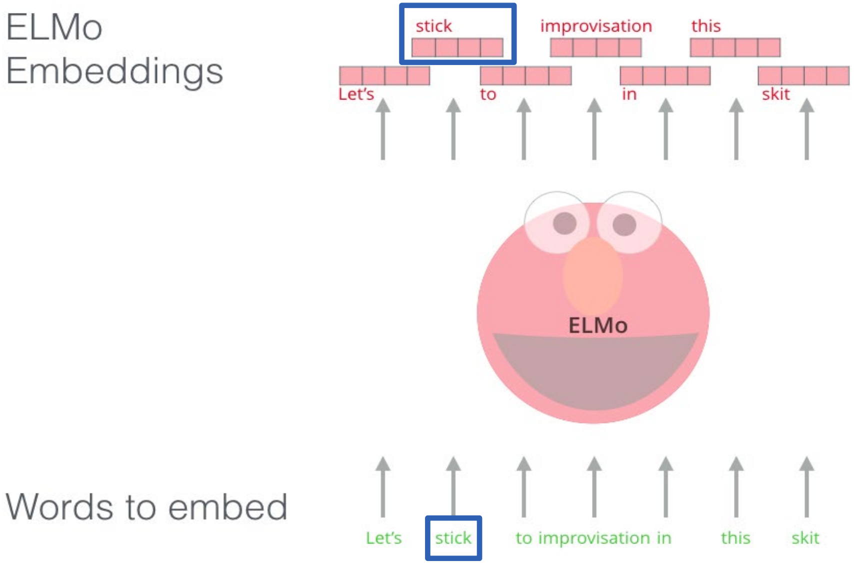
open a bank account   on the river bank

ELMo, what about sentences' context?

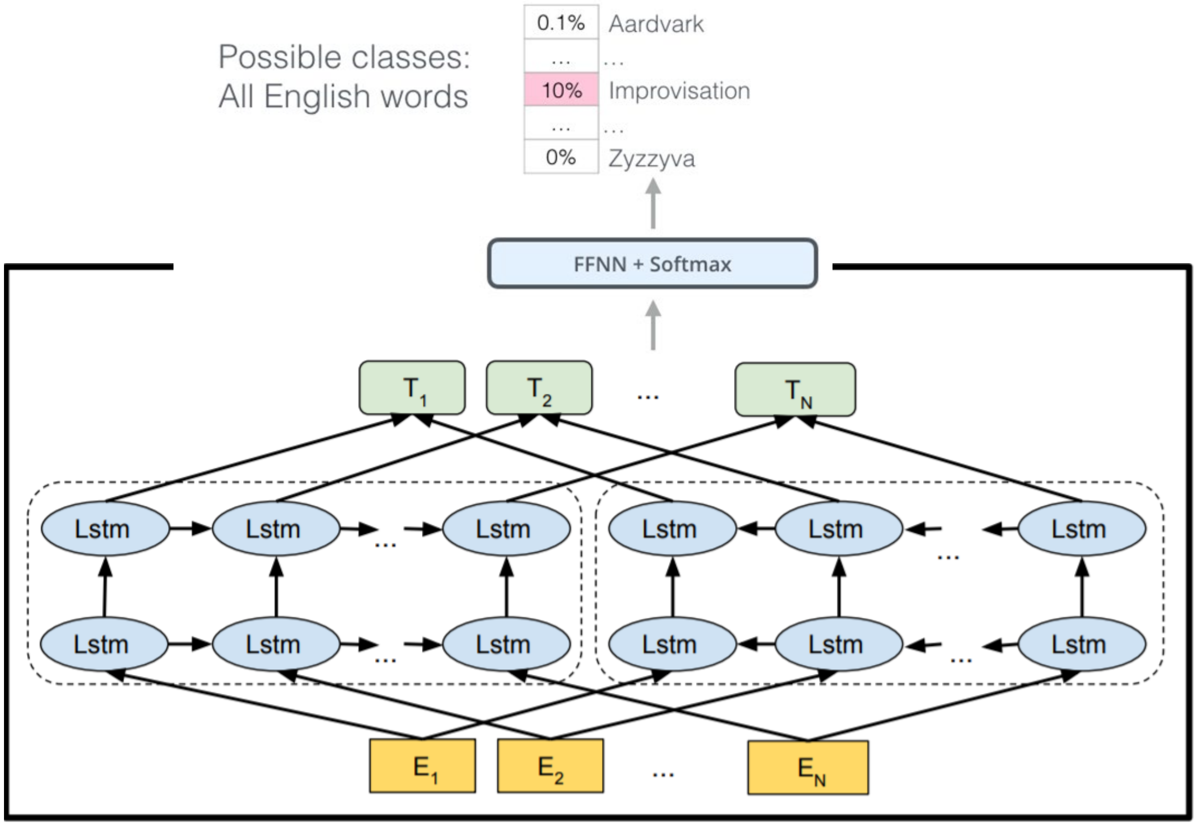
- In ELMo's design, the embedding of one word could have multiple possible answers.
- The model only gives a certain embedding for one word when this word is given in a sentence.
- For example:
  - when 'stick' is given as a word, ELMo may return several possible embeddings
  - when 'stick' is given in a sentence 'let's stick to improvisation in this skit', ELMo will return the embedding '-0.02, -0.16, 0.12, -0.1 …'

High-level view: Embeddings from Language Models

ELMo gives an embedding of a word, e.g., ‘stick’, when inputting together with contexts



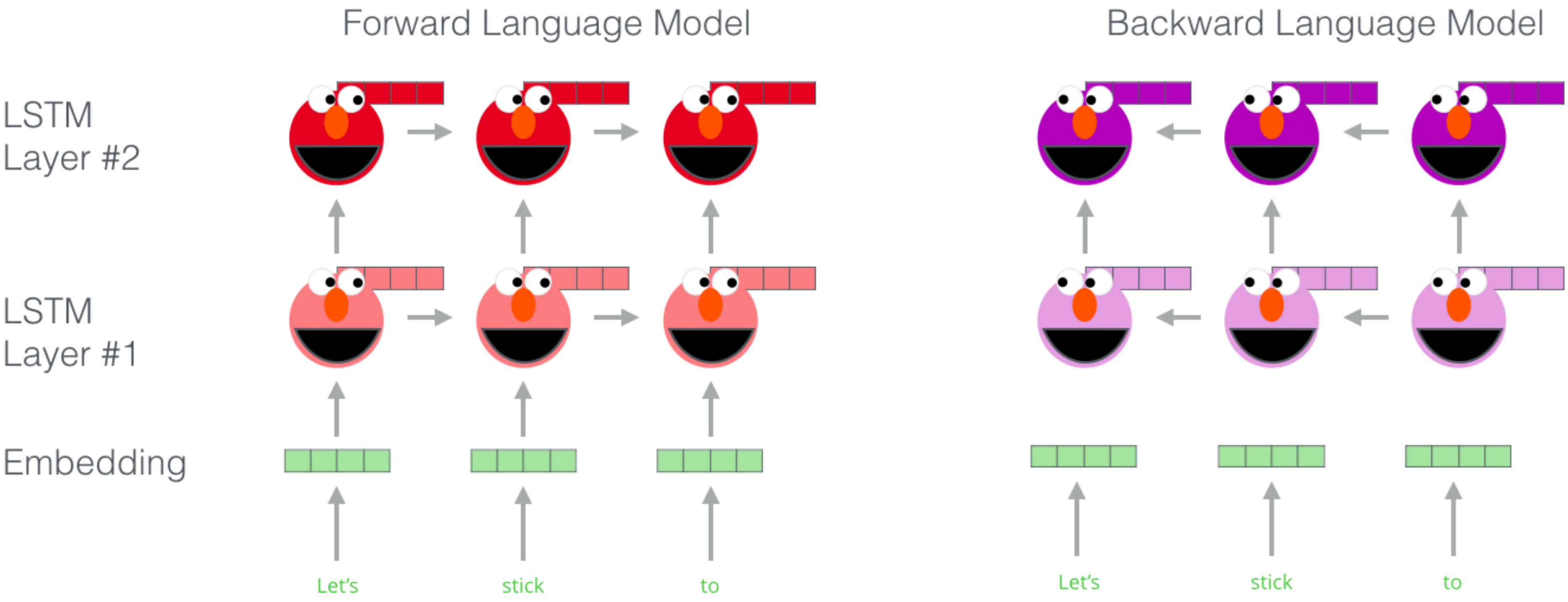
ELMo uses a bi-directional LSTM to pre-train the language model



Key features

- Replace static embeddings (lexicon lookup) with context-dependent embeddings (produced by a deep neural language model), i.e., each token's representation is a function of the entire input sentence.
- Computed by a deep multi-layer, bidirectional language model.
- Return for each token a (task-dependent) linear combination of its representation across layers.
- Different layers capture different information.

Embedding of “stick” in “Let’s stick to” - Step #1



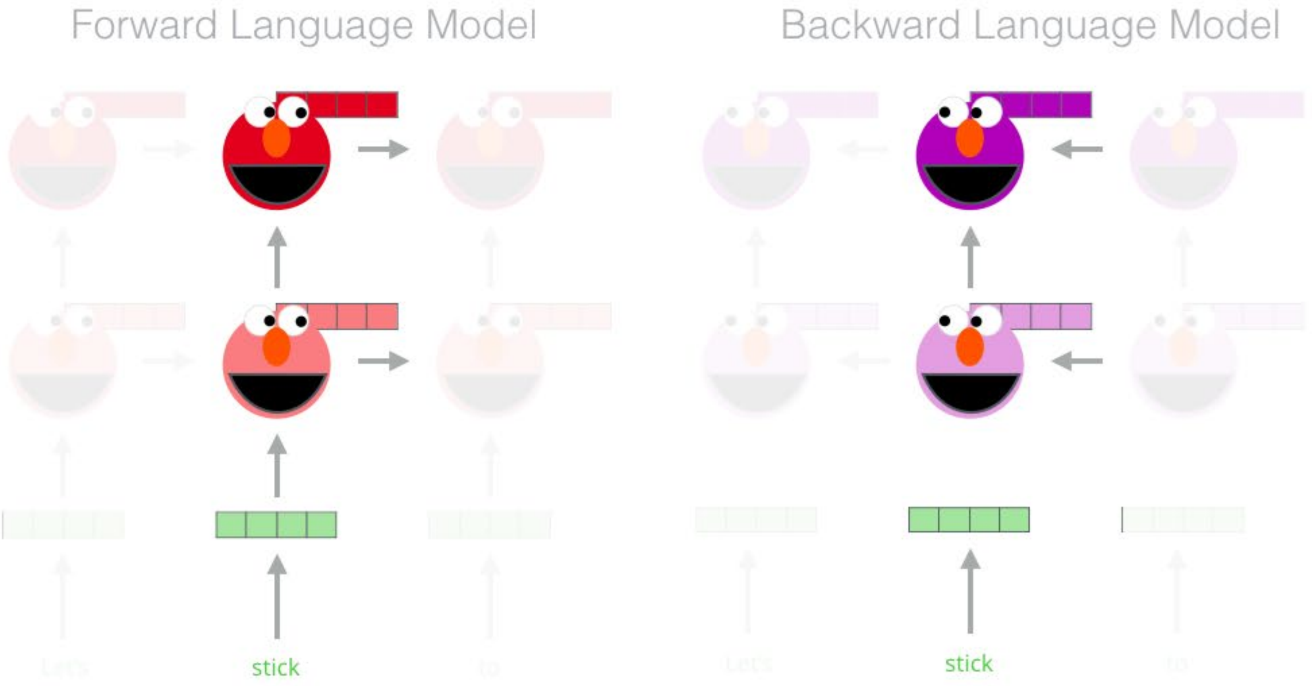
Embedding of “stick” in “Let’s stick to” - Step #2

1- Concatenate hidden layers

2- Multiply each vector by a weight based on the task

3- Sum the (now weighted) vectors

ELMo embedding of “stick” for this task in this context



ELMo architecture

- First layer: character level CNN to get context independent embeddings.
- Each layer of this language model network computes a vector representation for each token.
- Freeze the parameters of the language model.
- For each task: train task-dependent softmax weights to combine the layer-wise representations into a single vector for each token jointly with a task-specific model that uses those vectors.

When ELMo contextual representations was used in task-specific architecture, ELMo advanced the SOTA benchmarks.

— question answering (SQuAD)
— entailment/natural language inference (SNLI)
— semantic role labeling (SRL)
— coreference resolution (Coref)
— named entity recognition (NER)
— sentiment analysis (SST-5)

TASK	PREVIOUS SOTA		OUR BASELINE	ELMO + BASELINE	INCREASE (ABSOLUTE/ RELATIVE)
SQuAD	Liu et al. (2017)	84.4	81.1	85.8	4.7 / 24.9%
SNLI	Chen et al. (2017)	88.6	88.0	88.7 ± 0.17	0.7 / 5.8%
SRL	He et al. (2017)	81.7	81.4	84.6	3.2 / 17.2%
Coref	Lee et al. (2017)	67.2	67.2	70.4	3.2 / 9.8%
NER	Peters et al. (2017)	91.93 ± 0.19	90.15	92.22 ± 0.10	2.06 / 21%
SST-5	McCann et al. (2017)	53.7	51.4	54.7 ± 0.5	3.3 / 6.8%

References

Word2vec Skip gram Explained: https://thinkinfi.com/word2vec-skip-gram-explained/

word2vec Parameter Learning Explained: https://arxiv.org/pdf/1411.2738.pdf

CS224n: Natural Language Processing with Deep Learning Lecture Notes: Part I: http://web.stanford.edu/class/cs224n/readings/cs224n-2019-notes01-wordvecs1.pdf

Natural Language Processing (Almost) from Scratch: https://www.jmlr.org/papers/volume12/collobert11a/collobert11a.pdf?source=post_page---------------------------

Efficient Estimation of Word Representations in Vector Space: https://arxiv.org/pdf/1301.3781.pdf

Backpropagation: https://yuting3656.github.io/yutingblog//aiacademy/week11/nlp-word-embeddings-wrod2vec

Thanks for your attention!