CSE440: NATURAL LANGUAGE PROCESSING II

Farig Sadeque
Assistant Professor
Department of Computer Science and Engineering
BRAC University

Lecture 4: Word Representations

Outline

- Co-occurrence (SLP 6)
- TF-IDF (SLP 6)
- Embeddings (SLP 6 and lecture)

Intro

- Computers do not understand semantics
- Representation of text needs to include some sort of semantic information

Representation

- Sentence-level representation’s problems
- Co-occurrence
- TF-IDF
- Embeddings

Problems with BoW

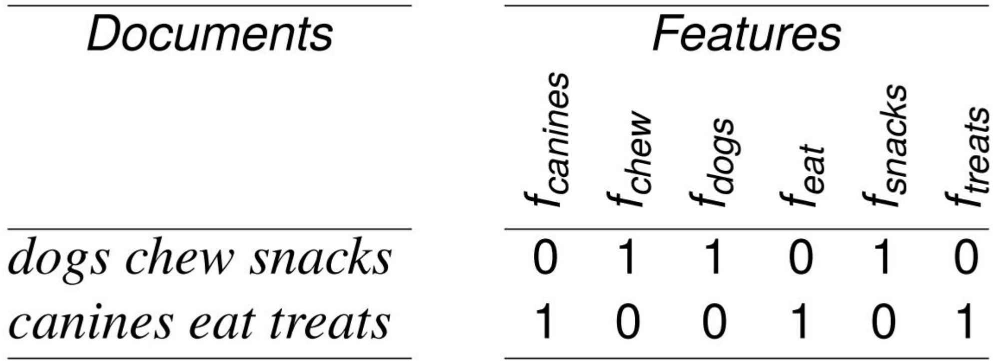
- Too sparse
    - What’s wrong with sparsity?
- Completely ignores word order
- Almost no semantic information preserved
- But, works pretty well!

More problem with sentence-level representation

- Dogs chew snacks
- Canines eat treats

More problem with sentence-level representation

- Dogs chew snacks
- Canines eat treats



Documents	$f_{canines}$	f_{chew}	f_{dogs}	f_{eat}	f_{snacks}	f_{treats}
dogs chew snacks	0	1	1	0	1	0
canines eat treats	1	0	0	1	0	1

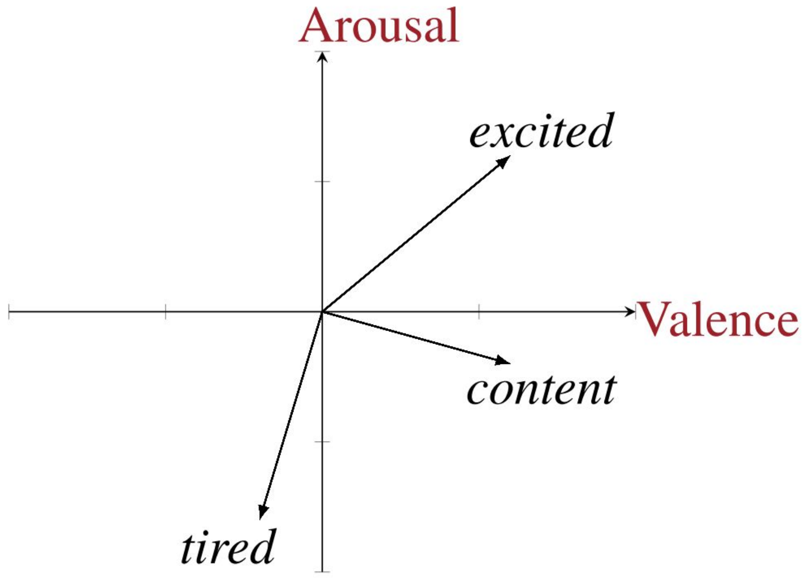
- No feature overlap whatsoever. If we try to calculate similarity, they are 100% dissimilar. But are they?
- Solution: instead of creating sentence level representations, let’s go to smaller units i.e. words

Perfect word representations

- shared lemmas: mouse/mice, dormir/duermes, etc.
- different word senses: computer mouse vs. pet mouse, river bank vs. financial bank, etc.
- synonyms: couch/sofa, car/automobile, etc.
- antonyms: long/short, dark/light, etc.
- word similarity: dog/cat, doctor/nurse, etc.
- word relatedness: cup/coffee, scalpel/surgeon, etc.
- word valence: excited and relaxed are high valence, depressed and angry are low valence
- word arousal: excited and angry are high arousal, relaxed and depressed are low arousal

Words as vectors

Words can be represented as vectors, where each entry in the vector represents a dimension of word meaning.



How to get numbers for each dimension of each word?

Neighboring words hint at semantics

- Imagine you didn’t know what ignite meant:

  . . . fusion fire does not ignite till temperatures . . .

  . . . plumes of flame ignite from the smokestacks . . .

  . . . over low heat. Ignite with a match …

- But you had seen another word in similar contexts:

  . . . the way the fire is lit or the heat source . . .

  ... flame couldn’t have lit a cigarette . . .

  . . . kiln-dried logs that lit with a match ...

Intuition: if two words are semantically similar, they will appear in text with similar surrounding words

Word vectors from neighboring words

Example sentences:

the spinach artichoke dip

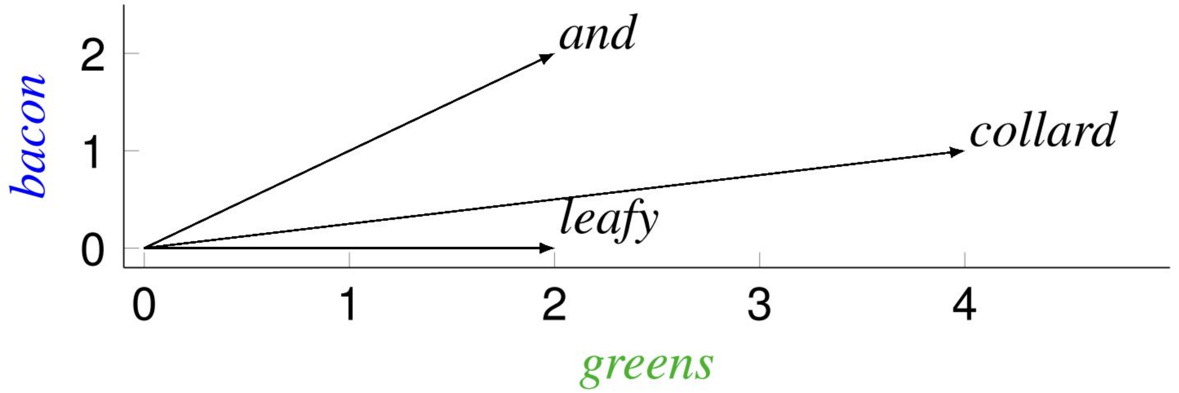
the leafy greens like spinach, kale, or collard greens

the bacon and spinach quiche

the collard greens and bacon

the cornbread and collard greens

Word vectors by co-occurrence:



Term-term matrix

A term-term co-occurrence matrix X is a |V|x|V| matrix where:

- |V| is the number of words in the vocabulary
- each cell $X_{i,j}$ records how often word j occurred in the context of word i
- each row X_i is the vector representation for word i

Context may be defined in different ways:

- The same document
- The same sentence
- Within ±n words of each other

V is typically the 10,000 - 50,000 most frequent words)

- Each word is represented by a large vector

Easy way to build a term-term matrix

- Build a binary BoW for the sentences
- Transpose it
- Multiply it with the original matrix
- Voila
- Try it: docs = ["any big cat", "big cat", "cat dog cat"]

Comparing word vectors

- How do you know the vectors you built make any sense?
- You need to compare these vectors
- What techniques do we have?

Cosine similarity

- Most common similarity measure

$$cosine(\mathbf{v},\mathbf{w}) = \frac{\mathbf{v}^\top \mathbf{w}}{|\mathbf{v}||\mathbf{w}|} = \frac{\overbrace{\sum_i v_i w_i}^{\text{dot product}}}{\underbrace{\sqrt{\sum_i v_i^2}}_{\text{length of } \mathbf{v}} \underbrace{\sqrt{\sum_i w_i^2}}_{\text{length of } \mathbf{w}}}$$

Cosine similarity: Why?

- Range is between 1 and -1
- Why not Euclidean distance? $\sqrt{\sum_{i=1}^{n}(v_i - w_i)^2}$

Let’s try this for this three vectors: u = [0, 1, 0, 1] v = [1, 0, 1, 0] w = [3, 0, 3, 0]

What is the cosine similarity? What is the Euclidean distance? Which one makes more sense?

What’s wrong with term term matrix?

- Sparse.Very sparse.
- Does not carry any contextual information
- Does not represent how important a word is in a sentence

TF-IDF

TF-IDF: Term Frequency - Inverse Document Frequency

Intuition: An informative term should:

- Occurs many times in some specific contexts (TF)
- Does not occur in every context (IDF)

Examples:

- high TF *vector* is informative; it’s frequent in these slides
- high DF *the* is uninformative; it’s frequent everywhere

TF-IDF

$$tf(w,d) = log(1 + f(w,d))$$

$$idf(w,D) = log(\frac{N}{f(w,D)})$$

w is a word, d is a document, D is the corpus, N = |D|

Intuitions:

- frequent in a single context is good
- avoid infinities
- appearing in every document is bad
- score of 100 (vs. 1) is not 100 times more relevant

Using word vectors

For word tasks:

- finding synonyms via cosine
- as classifier features when the input is one word

For sentence/document tasks:

- First, combine all word vectors
- You can combine yourself (using centroid technique): usually needed for classical ML models; or
- You can let an RNN handle things
- These vectors can then be used for classification

Sparse vs. dense vectors

Vectors we studied are very sparse

Advantages of small, dense word vectors:

- fewer feature weights to learn in machine learning
- fewer features can reduce overfitting
- forces sharing; there are not enough dimensions for
- each word to be completely independent

Word embeddings

- Rather than count co-occurrence, let’s try to predict it
- We can do it in two ways
    - Predict the target word given the neighboring words: CBOW
    - Predict the neighboring words given the target word: Skip-gram
- CBOW is easy, but…
- We will focus on Skip-gram

Skip-gram embeddings

- Input: a word, taken from some text
- Output: the 5 preceding and 5 following words
- Try it!
- Input: hippopotamus
- Output: [?, ?, ?, ?, ?, hippopotamus, ?, ?, ?, ?, ?]

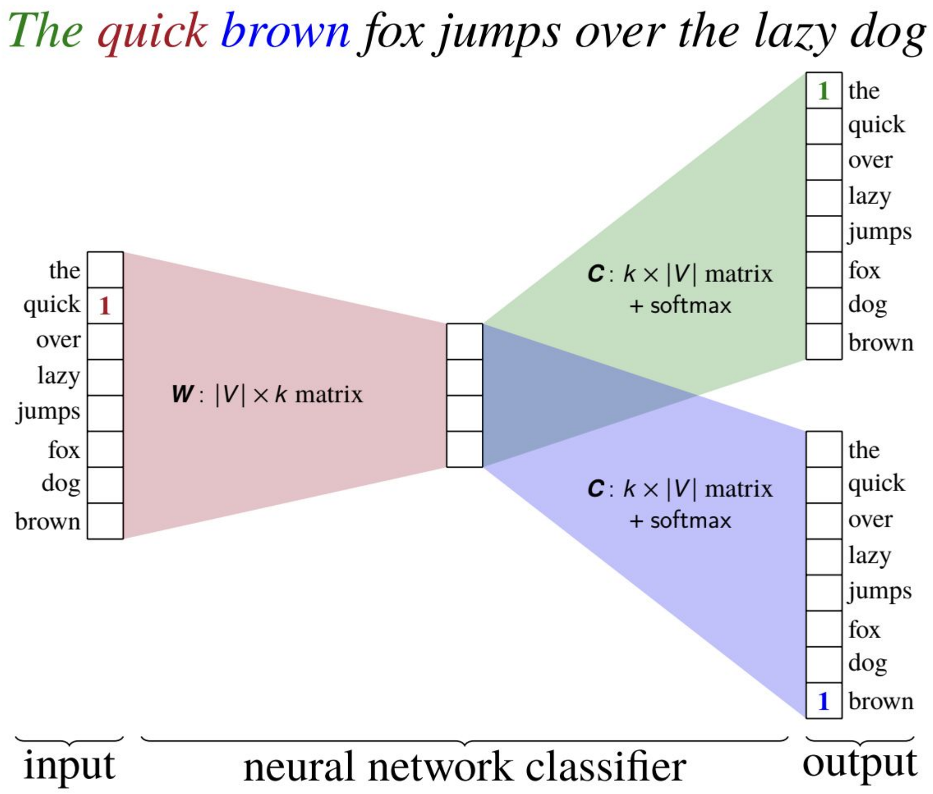
Skip-gram embeddings

- Input: a word, taken from some text
- Output: the 5 preceding and 5 following words
- Try it!
- Input: hippopotamus
- Output: [?, ?, ?, ?, ?, hippopotamus, ?, ?, ?, ?, ?]
- This task is impossible! But that’s okay because
    - We minimize cross-entropy loss, not classification error
    - We won’t ever actually use the model for prediction
    - We’ll only use word vectors learned as part of training

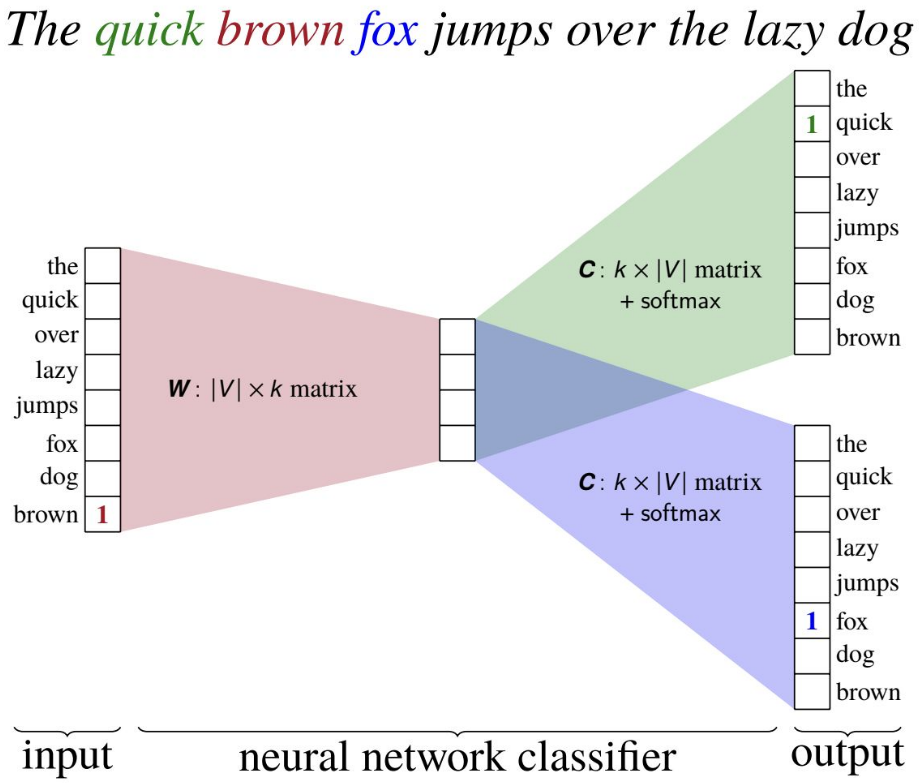
Skip-gram classifier

- We will use a feedforward neural network to predict the preceding and following words
- For simplicity's sake, we will only try to predict one preceding and one following word
- Steps:
    - Initialize all weights to random
    - Create one-hot vector for each of the words
    - Go from left to right
    - Predict the previous and the next word
    - Backpropagate the error and edit the weights
    - Continue

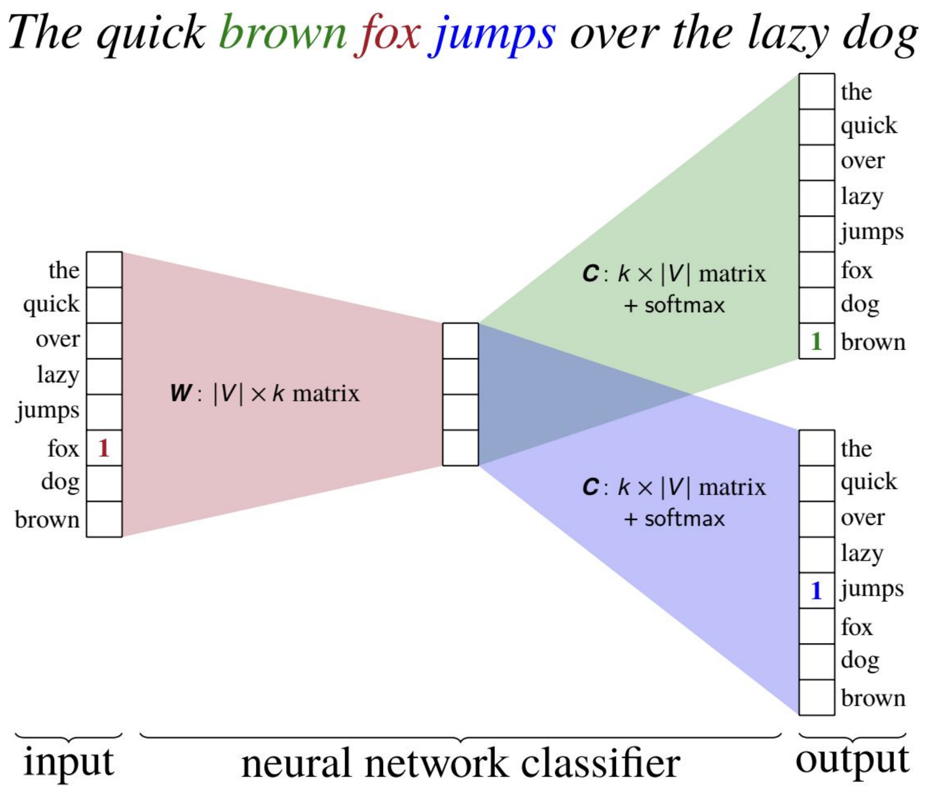
Skip-gram classifier


The quick brown fox jumps over the lazy dog
the
quick
over
lazy
jumps
fox
dog
brown
1
$\boldsymbol{W}: |V| \times k$ matrix
$\boldsymbol{C}: k \times |V|$ matrix + softmax
$\boldsymbol{C}: k \times |V|$ matrix + softmax
1
the
quick
over
lazy
jumps
fox
dog
brown
the
quick
over
lazy
jumps
fox
dog
1
brown
input
neural network classifier
output

Skip-gram classifier


The quick brown fox jumps over the lazy dog
the
quick
over
lazy
jumps
fox
dog
brown
1
$\boldsymbol{W}$: $|V| \times k$ matrix
$\boldsymbol{C}$: $k \times |V|$ matrix + softmax
$\boldsymbol{C}$: $k \times |V|$ matrix + softmax
the
1 quick
over
lazy
jumps
fox
dog
brown
the
quick
over
lazy
jumps
1 fox
dog
brown
input
neural network classifier
output

Skip-gram classifier



GloVe embeddings

- Self study
- Will be a part of your assignment

Using embeddings

It's rarely necessary to train skip-gram or GloVe directly.

Download pre-trained word embeddings:

- Skip-gram https://code.google.com/archive/p/word2vec/
- GloVe https://nlp.stanford.edu/projects/glove/

Many models provide pre-trained embeddings:
https://github.com/Hironsan/awesome-embedding-models

Which one should I choose? Try a few and see what works!

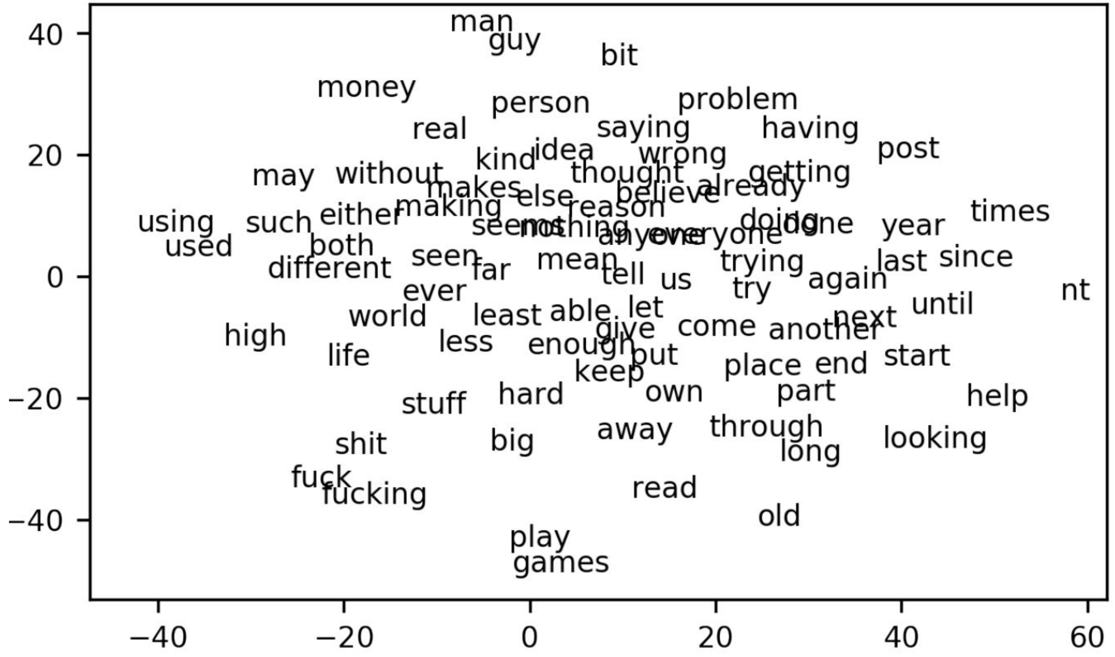
Inspecting embeddings

How do I know if my word embeddings make sense?

- Check by hands
- Project the words to a visible dimension
- Use linear algebra

Visualizing embeddings

We usually have very high-dimensional vectors for each words. t-SNE can project down to 2.



Algebra

```
>>> cosine(vector("queen"), vector("king"))
0.7252606
>>> cosine(vector("queen"),
...  vector("king")-vector("man")+vector("woman"))
0.7880841
>>> cosine(vector("Paris"), vector("Rome"))
0.58241177
>>> cosine(vector("Paris"),
...  vector("Rome")-vector("Italy")+vector("France"))
0.71733016
```

Other standard evaluations

Correlation with human judgments of similarity

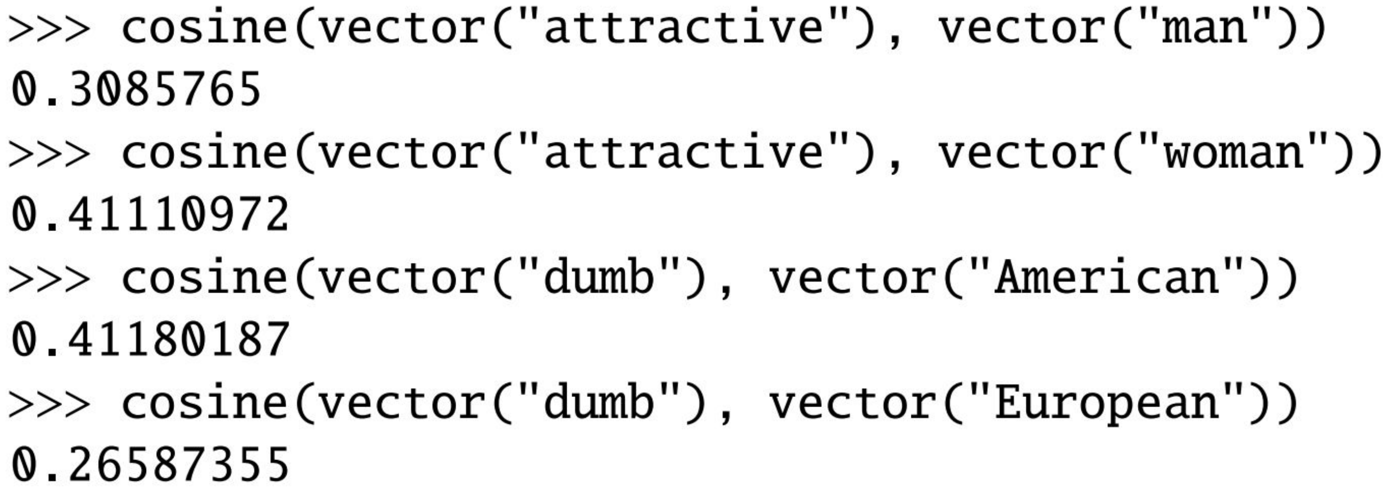
- **WordSim-353** noun similarity, e.g., (plane, car, 5.77)
- **SimLex-999** adjective, noun, and verb similarities
- **SCWS** word similarity given sentential context
- **STS** sentence-level similarity

Accuracy at similarity-based task

- **TOEFL** e.g., Levied is closest in meaning to: imposed, believed, requested, correlated
- **analogies** e.g., Athens is to Greece as Oslo is to ____

Bias in embeddings

Embeddings reflect the language they were trained on



```
>>> cosine(vector("attractive"), vector("man"))
0.3085765
>>> cosine(vector("attractive"), vector("woman"))
0.41110972
>>> cosine(vector("dumb"), vector("American"))
0.41180187
>>> cosine(vector("dumb"), vector("European"))
0.26587355
```

Contextual word embeddings

Traditional word vectors ignore context

The river bank: [0.3, -0.1, -0.2] [0.1, -0.3, -0.2] [-0.6, 0.3, -0.1]

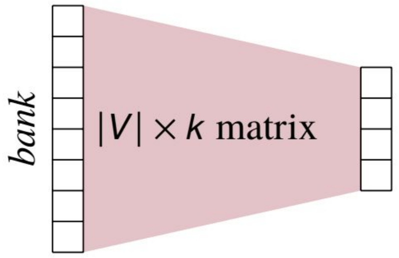
A bank deposit: [0.0 , 0.0 , -0.2] [-0.6, 0.3, -0.1] [-0.3, -0.3, 0.0]

Should these two banks really have the same vectors?

Contextual word embeddings

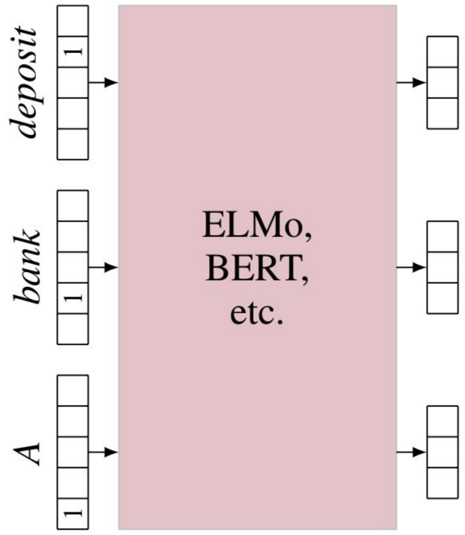
Word embeddings

Input	1 word
Output	1 embedding



Contextual word embeddings

Input	n words
Output	n embeddings



Learning contextual word embeddings

We need to make up a prediction task that

- takes n words as input
- produces n vectors as output
- requires only unlabeled data

ELMo’s task: language modeling

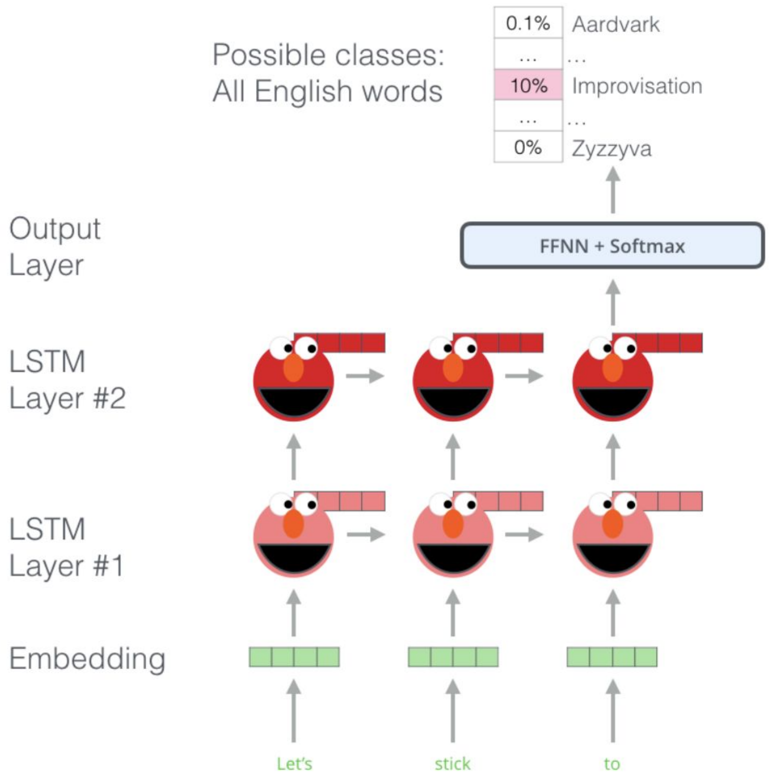
What is a language model?

- Given a sequence of words, what is the next most probable word?
- Unsupervised, great for learning representations

ELMo combines a forward language model and a backward language model.

Transformers use the same idea, but in a much larger canvas.

ELMo’s task: language modeling



How to use contextual word embeddings?

Contextual word embeddings are trained on unlabeled data. How do we use them on the task we care about?

- Extract word vectors, use as features
- Fine-tune contextual embedding model, i.e., continue training the model, but now on our labeled data instead of the unlabeled data