Word-embeddings and **Word2Vec**

Yoav Goldberg

Last Time

- Language modeling.
- 1-hot X matrix
- Embedding layer
- (pre-trained) Word Embeddings.

Neural LM

$$\text{softmax}(\square) \qquad \mathbb{R}^{d_{out}}$$

$$\uparrow$$

$$\square\mathbf{W^3} + \mathbf{b^3} \qquad \mathbb{R}^{d_{out}}$$

what's in **W3**?

$$\uparrow$$

$$g(\square\mathbf{W^2} + \mathbf{b^2}) \qquad \mathbb{R}^{d_2}$$

columns of W3 correspond to vocab items!

$$\uparrow$$

$$g(\square\mathbf{W^1} + \mathbf{b^1}) \qquad \mathbb{R}^{d_1}$$

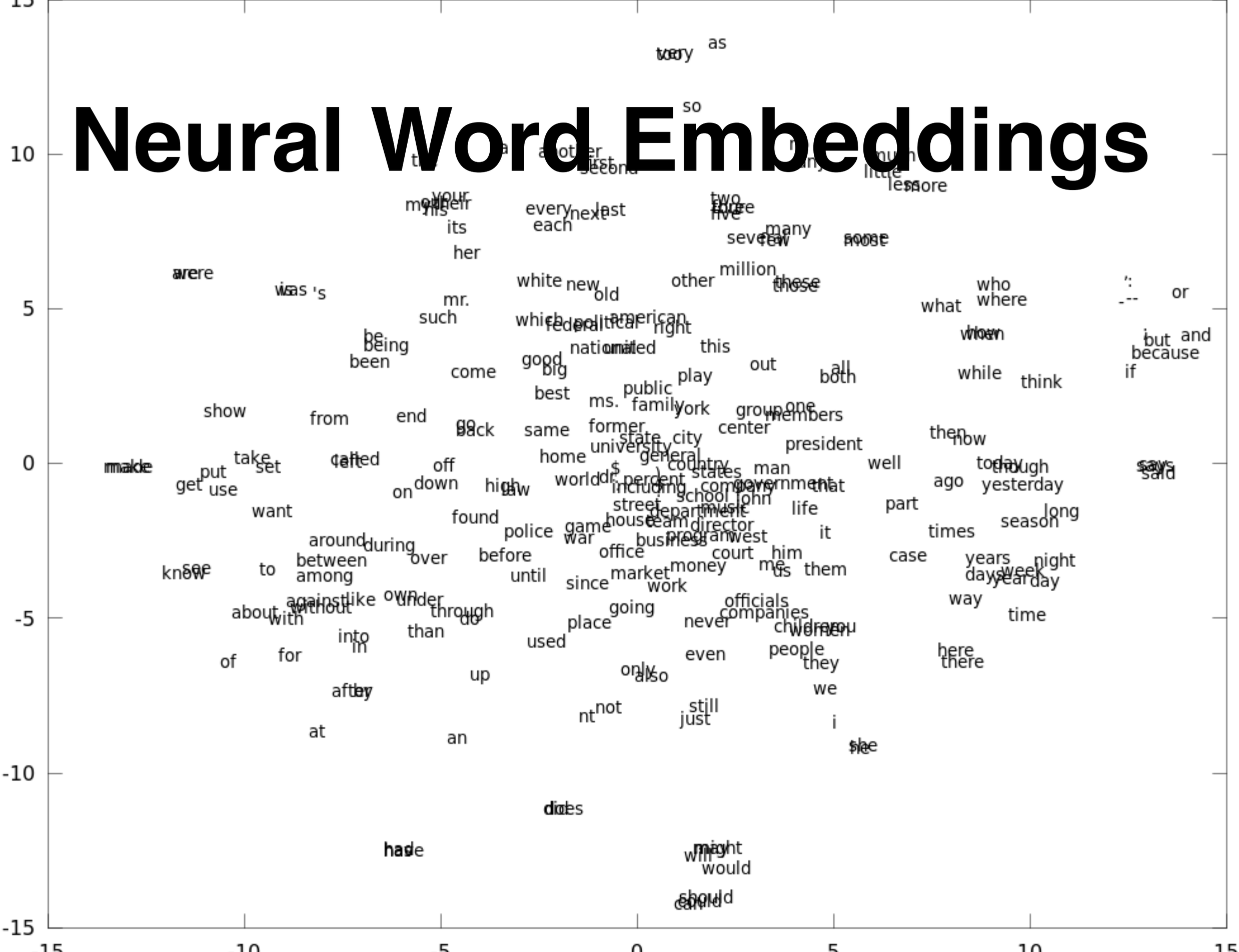
$$\uparrow$$

$$\mathbf{E}_{[x_{k-4}]} \circ \mathbf{E}_{[x_{k-3}]} \circ \mathbf{E}_{[x_{k-2}]} \circ \mathbf{E}_{[x_{k-1}]} \qquad \mathbb{R}^{d_{in}}$$

$$\uparrow$$

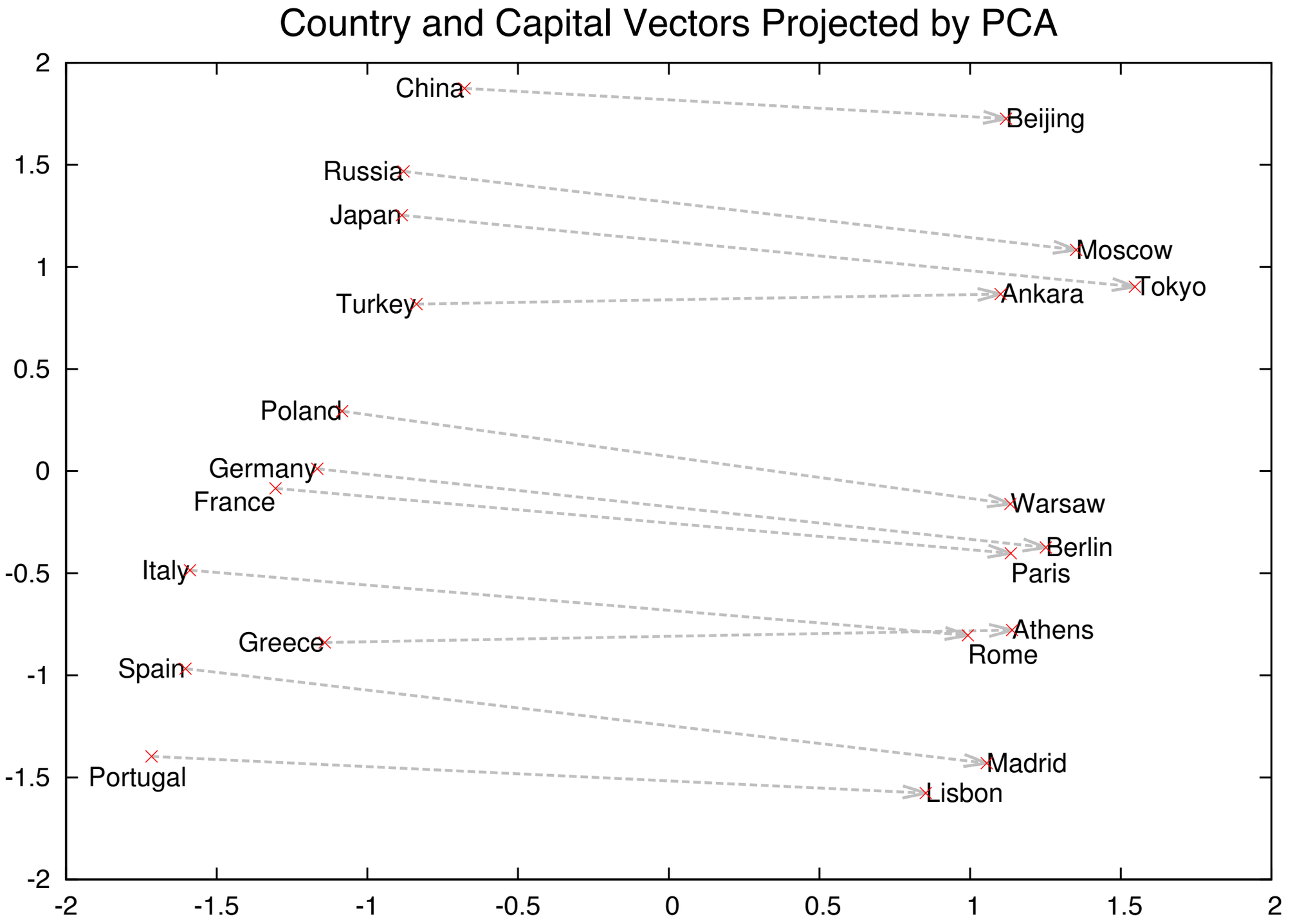
$$encode(x_{k-4}, x_{k-3}, x_{k-2}, x_{k-1})$$

Neural Word Embeddings



Neural Word Embeddings

Neural Word Embeddings



Neural Word Embeddings

Target Word	BoW5
batman	nightwing aquaman catwoman superman manhunter
hogwarts	dumbledore hallows half-blood malfoy snape
turing	nondeterministic non-deterministic computability deterministic finite-state
florida	gainesville fla jacksonville tampa lauderdale
object-oriented	aspect-oriented smalltalk event-driven prolog domain-specific
dancing	singing dance dances dancers tap-dancing

[word similarity example]

Neural Word Embeddings

- We trained a language model.
- We ended up with **vector representations of words**.
- These representations are useful -- they encode various aspects of word similarity.

Neural LM

How are the softmax scores determined?

$$\text{softmax}(\square) \qquad \mathbb{R}^{d_{out}}$$

$$\uparrow$$

$$\square\mathbf{W^3} + \mathbf{b^3} \qquad \mathbb{R}^{d_{out}}$$

$$\uparrow$$

$$g(\square\mathbf{W^2} + \mathbf{b^2}) \qquad \mathbb{R}^{d_2}$$

$$\uparrow$$

$$g(\square\mathbf{W^1} + \mathbf{b^1}) \qquad \mathbb{R}^{d_1}$$

$$\uparrow$$

$$\mathbf{E}_{[x_{k-4}]} \circ \mathbf{E}_{[x_{k-3}]} \circ \mathbf{E}_{[x_{k-2}]} \circ \mathbf{E}_{[x_{k-1}]} \qquad \mathbb{R}^{d_{in}}$$

$$\uparrow$$

$$encode(x_{k-4}, x_{k-3}, x_{k-2}, x_{k-1})$$

Neural LM

How are the softmax scores determined?

$$\text{softmax}(\square)$$

$$\mathbb{R}^{d_{out}}$$

$$\uparrow$$

$$\mathbb{R}^{d_{out}}$$

$$\square\mathbf{W^3} + \mathbf{b^3}$$

$$\uparrow$$

$$\mathbb{R}^{d_2}$$

$$g(\square\mathbf{W^2} + \mathbf{b^2})$$

$$\uparrow$$

$$\mathbb{R}^{d_1}$$

$$g(\square\mathbf{W^1} + \mathbf{b^1})$$

$$\uparrow$$

$$\mathbb{R}^{d_{in}}$$

$$\mathbf{E}_{[x_{k-4}]} \circ \mathbf{E}_{[x_{k-3}]} \circ \mathbf{E}_{[x_{k-2}]} \circ \mathbf{E}_{[x_{k-1}]}$$

$$\uparrow$$

$$encode(x_{k-4}, x_{k-3}, x_{k-2}, x_{k-1})$$

Neural LM

How are the softmax scores determined?

each column of **W3** corresponds to a word

$\uparrow$

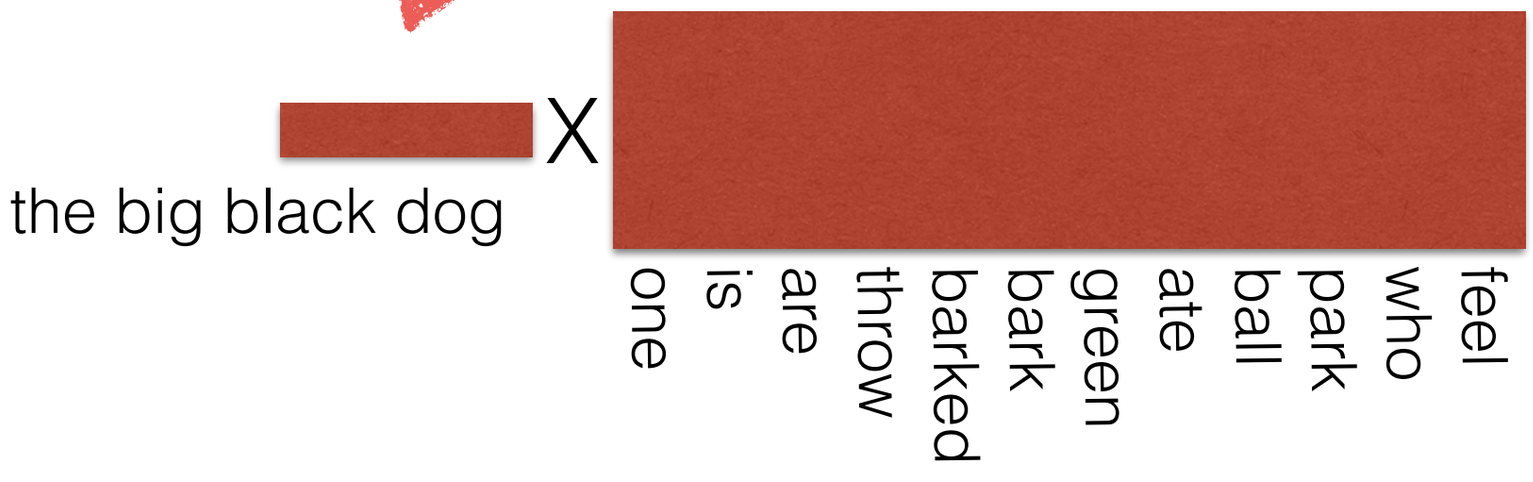
$\square \mathbf{W^3} + \mathbf{b^3}$

$\uparrow$

$g(\square \mathbf{W^2} + \mathbf{b^2})$

$\uparrow$

a vector representation of the context words



towards word2vec

- Training the language model is expensive. (Why?)
- Predicting a word based on previous words is nice, but can we do better?
- If all we care about are the word vectors...

Word2Vec

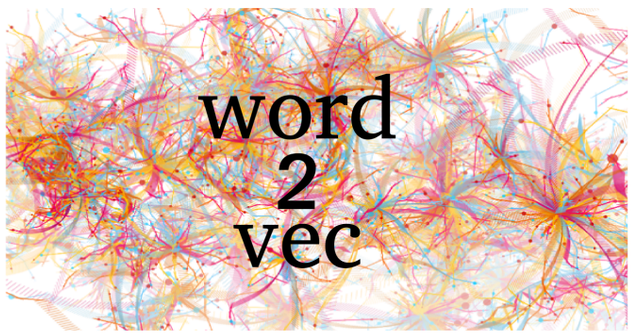
- Radically simplify the neural LM.
- Very fast training.
- Obtain good word representations.

word2vec

feed in text

Text

WIKIPEDIA



wait a few hours

d

|V| words

dog = (0.12,-0.32,0.92,0.43,-0.3 ...)
cat = (0.15,-0.29,0.90,0.39,-0.32 ...)

chair = (0.8,0.9,-0.76,0.29,0.52 ...)

get a |V|xd matrix W where each row is a vector for a word

word2vec

- dog
  - cat, dogs, dachshund, rabbit, puppy, poodle, rottweiler, mixed-breed, doberman, pig
- sheep
  - cattle, goats, cows, chickens, sheeps, hogs, donkeys, herds, shorthorn, livestock
- november
  - october, december, april, june, february, july, september, january, august, march
- jerusalem
  - tiberias, jaffa, haifa, israel, palestine, nablus, damascus katamon, ramla, safed
- teva
  - pfizer, schering-plough, novartis, astrazeneca, glaxosmithkline, sanofi-aventis, mylan, sanofi, genzyme, pharmacia

Collobert and Weston model

- instead of predicting word based on **previous** words...
- ...predict word based on **surrounding** words.

$$P(w_5|w_1 w_2 w_3 w_4 \square) \longrightarrow P(w_3|w_1 w_2 \square w_4 w_5)$$

Collobert and Weston model

- Getting rid of the softmax over the vocabulary:
  - probabilities --> scores
  - score the quality of a sequence, not all the words
  - ranking based loss

Collobert and Weston model

$$\text{score}(D, A, \square, C, F; G) = g(\mathbf{x}\mathbf{U})\mathbf{v}$$

$$\mathbf{x} = \left(\mathbf{E}_{[D]} \circ \mathbf{E}_{[A]} \circ \mathbf{E}_{[G]} \circ \mathbf{E}_{[C]} \circ \mathbf{E}_{[F]}\right)$$

Collobert and Weston model

$$\text{score}(D, A, \square, C, F; G) = g(\mathbf{x}\mathbf{U})\mathbf{v}$$

$$\mathbf{x} = \left(\mathbf{E}_{[D]} \circ \mathbf{E}_{[A]} \circ \mathbf{E}_{[G]} \circ \mathbf{E}_{[C]} \circ \mathbf{E}_{[F]}\right)$$

$$\text{score}(c_1, c_2, \square, c_3, c_4; w) > \text{score}(c_1, c_2, \square, c_3, c_4; w') + 1$$

observed word

random word

Collobert and Weston model

$$\text{score}(D, A, \square, C, F; G) = g(\mathbf{x}\mathbf{U})\mathbf{v}$$

$$\mathbf{x} = (\mathbf{E}_{[D]} \circ \mathbf{E}_{[A]} \circ \mathbf{E}_{[G]} \circ \mathbf{E}_{[C]} \circ \mathbf{E}_{[F]})$$

$$\text{score}(c_1, c_2, \square, c_3, c_4; w) > \text{score}(c_1, c_2, \square, c_3, c_4; w') + 1$$

observed word

random word

$$L(w, c, w') = max(0, 1 - (\text{score}(c_{1:k}; w) - \text{score}(c_{1:k}; w')))$$

Collobert and Weston model

How do you train this?
(discuss)

$$\text{score}(D, A, \square, C, F; G) = g(\mathbf{x}\mathbf{U})\mathbf{v}$$

$$\mathbf{x} = (\mathbf{E}_{[D]} \circ \mathbf{E}_{[A]} \circ \mathbf{E}_{[G]} \circ \mathbf{E}_{[C]} \circ \mathbf{E}_{[F]})$$

$$\text{score}(c_1, c_2, \square, c_3, c_4; w) > \text{score}(c_1, c_2, \square, c_3, c_4; w') + 1$$

observed word

random word

$$L(w, c, w') = max(0, 1 - (\text{score}(c_{1:k}; w) - \text{score}(c_{1:k}; w')))$$

from Collobert and Weston to Word2Vec

- Back to probabilities
- Simplify the scoring function

Word2Vec

- Back to probabilities

$$L(w, c, w') = max(0, 1 - (\text{score}(c_{1:k}; w) - \text{score}(c_{1:k}; w')))$$

$$P(D = 1|w, c) = \sigma(s(w, c)) = \frac{1}{1 + e^{-s(w,c)}}$$

Word2Vec

- Back to probabilities

$$P(D = 1|w, c) = \sigma(s(w, c)) = \frac{1}{1 + e^{-s(w,c)}}$$

$$\mathcal{L}(\Theta; D, \bar{D}) = \sum_{(w,c)\in D} \log P(D = 1|w, c) + \sum_{(w',c)\in \bar{D}} \log P(D = 0|w', c)$$

Word2Vec

- Back to probabilities

good word+context pairs **bad** word+context pairs

$$P(D = 1|w, c) = \sigma(s(w, c)) = \frac{1}{1 + e^{-s(w,c)}}$$

$$\mathcal{L}(\Theta; D, \bar{D}) = \sum_{(w,c)\in D} \log P(D = 1|w, c) + \sum_{(w',c)\in \bar{D}} \log P(D = 0|w', c)$$

Word2Vec

- Back to probabilities

$$P(D = 0|w, c) = 1 - P(D = 1|w, c)$$

$$P(D = 1|w, c) = \sigma(s(w, c)) = \frac{1}{1 + e^{-s(w,c)}}$$

$$\mathcal{L}(\Theta; D, \bar{D}) = \sum_{(w,c)\in D} \log P(D = 1|w, c) + \sum_{(w',c)\in \bar{D}} \log P(D = 0|w', c)$$

Word2Vec

- Simplify the score function:

  CBOW

  Skip-grams

Word2Vec

- Simplify the score function:

  CBOW

  Skip-grams

$$P(D = 1|w, c) = \sigma(s(w, c)) = \frac{1}{1 + e^{-s(w,c)}}$$

what is c?

Word2Vec

- Simplify the score function:

CBOW

$$\text{score}(w; c_1, ..., c_k) = (\sum_{i=1}^{k} \mathbf{E}_{[c_i]}) \cdot \mathbf{E}'_{[w]}$$

$$P(D = 1|w, c_{1:k}) = \frac{1}{1 + e^{-(\mathbf{w \cdot c_1} + \mathbf{w \cdot c_2} + ... + \mathbf{w \cdot c_k})}}$$

Word2Vec

- Simplify the score function:

Skip-grams:

$$P(D=1|w,c_i) = \frac{1}{1+e^{-\mathbf{w}\cdot\mathbf{c_i}}}$$

$$P(D=1|w,c_{1:k}) = \prod_{i=1}^{k} P(D=1|w,c_i) = \prod_{1=i}^{k} \frac{1}{1+e^{-\mathbf{w}\cdot\mathbf{c_i}}}$$

$$\log P(D=1|w,c_{1:k}) = \log \sum_{i=1}^{k} \frac{1}{1+e^{-\mathbf{w}\cdot\mathbf{c_i}}}$$

Word2Vec

- Simplify the score function:

Skip-grams:

$$\mathcal{L}(\Theta; D, \bar{D}) = \sum_{(w,c)\in D} \log P(D = 1|w, c) + \sum_{(w',c)\in \bar{D}} \log P(D = 0|w', c)$$

$$= \arg\max_{\theta} \sum_{(w,c)\in D} \log \frac{1}{1 + e^{-v_c \cdot v_w}} + \sum_{(w,c)\in D'} \log(\frac{1}{1 + e^{v_c \cdot v_w}})$$

Word2Vec

- How to train a word2vec model? (discuss)

"FastText"

Enriching Word Vectors with Subword Information

Piotr Bojanowski* and **Edouard Grave*** and **Armand Joulin** and **Tomas Mikolov**
Facebook AI Research
{bojanowski,egrave,ajoulin,tmikolov}@fb.com

$$= \arg\max_{\theta} \sum_{(w,c)\in D} \log \frac{1}{1+e^{-v_c \cdot v_w}} + \sum_{(w,c)\in D'} \log\left(\frac{1}{1+e^{v_c \cdot v_w}}\right)$$

"FastText"

Represent a word by the sum of its char ngrams.

$$= \arg\max_{\theta} \sum_{(w,c)\in D} \log \frac{1}{1+e^{-v_c \cdot v_w}} + \sum_{(w,c)\in D'} \log\left(\frac{1}{1+e^{v_c \cdot v_w}}\right)$$

"FastText"

Represent a word by the sum of its char ngrams.

where = "<where>" "<where", "where>", "<wher", "where", "here>", "<whe", "wher", "here", "ere>", "<wh", ...

v_{where} = E["<where>"] + E["<where"] + ...

$$= \arg\max_{\theta} \sum_{(w,c) \in D} \log \frac{1}{1 + e^{-v_c \cdot v_w}} + \sum_{(w,c) \in D'} \log\left(\frac{1}{1 + e^{v_c \cdot v_w}}\right)$$

"FastText"

Represent a word by the sum of its char ngrams.

"<where>" "<where", "where>", "<wher", "where",
"here>", "<whe", "wher", "here", "ere>", "<wh", ...

- What are the benefits of FastText? Where do we expect it to be **better**?
- When do we expect it to work **worse**?
- Can you think of a **generalization of it to non-human-language tasks**?
- Can you think of **non-human-language-tasks where it will hurt** and not help?
- Can you think of a **human languages where it may hurt**?

"FastText"

Represent a word by the sum of its char ngrams.

"<where>" "<where", "where>", "<wher", "where",
"here>", "<whe", "wher", "here", "ere>", "<wh", ...

$$v_{where} = \text{E["<where>"] + E["<where"] + ...}$$

Think:

Isn't this very expensive to run at inference time?
Why or why not?

Other word vectors

- Other **contexts** are also possible.
- Other **algorithms** are also possible.
    - GloVe
    - NCE
    - SVD
- Implementations:
    - word2vec, word2vecf, GloVe, gensim, your own...

Using Word Vectors

- Use for initializing the word embeddings in other models. ("pre-training". more soon.)
- Use for finding similar words. (how?)
- Find a word similar to a group of words. (how?)
- Find the word that does not belong to a group. (how?)

Working with Dense Vectors

Word Similarity

- Similarity is calculated using *cosine similarity*:

$$sim(\vec{dog}, \vec{cat}) = \frac{\vec{dog} \cdot \vec{cat}}{||\vec{dog}||\ ||\vec{cat}||}$$

- For normalized vectors ($||x|| = 1$), this is equivalent to a dot product:

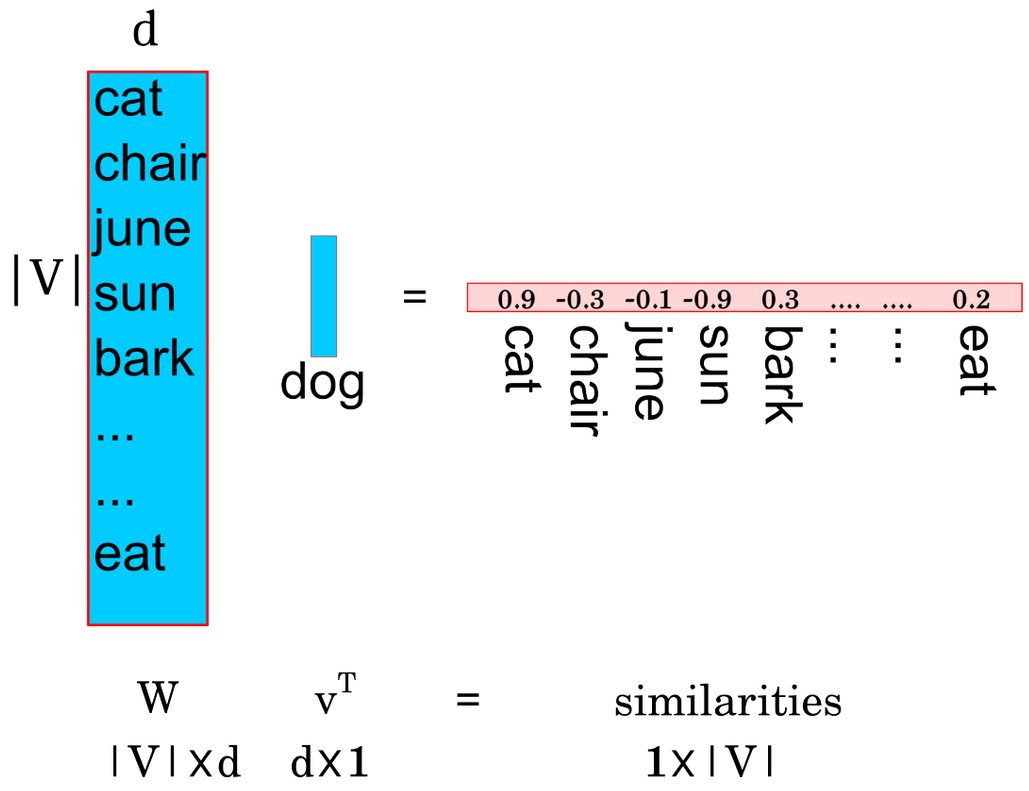
$$sim(\vec{dog}, \vec{cat}) = \vec{dog} \cdot \vec{cat}$$

- **Normalize the vectors when loading them.**

Working with Dense Vectors

Finding the most similar words to $\vec{dog}$

- Compute the similarity from word $\vec{v}$ to all other words.
- This is a **single matrix-vector product**: $W \cdot \vec{v}^{\top}$



- Result is a $|V|$ sized vector of similarities.
- Take the indices of the k-highest values.
- **FAST! for 180k words, d=300: $\sim$30ms**

Working with Dense Vectors

Most Similar Words, in python+numpy code

```
W,words = load_and_normalize_vectors("vecs.txt")
# W and words are numpy arrays.
w2i = {w:i for i,w in enumerate(words)}

dog = W[w2i['dog']] # get the dog vector

sims = W.dot(dog)    # compute similarities

most_similar_ids = sims.argsort()[-1:-10:-1]
sim_words = words[most_similar_ids]
```

Working with Dense Vectors

Similarity to a group of words

- "Find me words most similar to cat, dog and cow".
- Calculate the pairwise similarities and sum them:

$$W \cdot \vec{cat} + W \cdot \vec{dog} + W \cdot \vec{cow}$$

- Now find the indices of the highest values as before.
- Matrix-vector products are wasteful. **Better option:**

$$W \cdot (\vec{cat} + \vec{dog} + \vec{cow})$$

tagging + pre-training

we can use the **E** we got from LM training to initialize **E** for the POS tagging task.

(why is that helpful?)

NOUN

The brown **fox** jumped over

VERB

brown fox **jumped** over the

PREP

fox jumped **over** the lazy

Pre-training

- A large part of the success of feed-forward networks in NLP comes from the use of pre-trained word embeddings.

- Pre-trained embeddings are an easy way to perform semi-supervised learning (or transfer learning).

- But notice: fine-tuning the pre-trained embeddings means that some features change, while most stay the same...

Pre-training

- Define an auxiliary task that you suspect is correlated with your prediction problem.

- Train a model to perform this task.

- Take features representations from this model as inputs to another model.

Pre-training

- In word2vec, auxiliary tasks are "predict word based on a window of size k around it" (CBOW) or "predict neighboring words in a window of k around a focus word" (skipgram).
- More generally, "predict word based on some context of the word".
- This is useful, as we can get tons of training data for free.
- The choice of contexts determines the resulting word representations.

Some possible contexts

- **Window of k around a word.**
  (smaller k: more syntactic. Larger k: more semantic)

- **Positional window around a word.**
  (more syntactic)

- **Aligned words in a different language.** Requires parallel corpus
  (synonyms, paraphrases)

- **Neighbors in a dependency tree.** Requires parser.
  (functional similarity)

Effect of Context

Target Word	Bag of Words (k=5)	Dependencies
Hogwarts (Harry Potter's school)	Dumbledore hallows half-blood Malfoy Snape	Sunnydale Collinwood Calarts Greendale Millfield
	Related to Harry Potter	**Schools**

Levy and Goldberg 2014
Dependency-based word embeddings

Effect of Context

Target Word	Bag of Words (k=5)	Dependencies
Turing (computer scientist)	nondeterministic non-deterministic computability deterministic finite-state	Pauling Hotelling Heting Lessing Hamming
	Related to computability	**Scientists**

Levy and Goldberg 2014
Dependency-based word embeddings

Effect of Context

Target Word	Bag of Words (k=5)	Dependencies
dancing (dance gerund)	singing dance dances dancers tap-dancing	singing rapping breakdancing miming busking
	Related to dance	**Gerunds**

Levy and Goldberg 2014
Dependency-based word embeddings

Word2Vec

- no layers.
- no softmax. (so negative sampling.)
- cbow or skipgrams objectives.
- many implementations available.
- more in the NLP course.