BUS 243

Lecture 6: Distributional Semantics

We'll represent $\hat{y}$ as $f(x; \theta)$ to make the dependence on θ more obvious:

$$\nabla_{\theta} L(f(x;\theta),y)) = \begin{bmatrix} \frac{\partial}{\partial w_1} L(f(x;\theta),y) \\ \frac{\partial}{\partial w_2} L(f(x;\theta),y) \\ \vdots \\ \frac{\partial}{\partial w_n} L(f(x;\theta),y) \end{bmatrix}$$

The final equation for updating θ based on the gradient is thus

$$\theta_{t+1} = \theta_t - \eta \nabla L(f(x;\theta),y)$$

WORD VECTORS

- Arguably the most exciting discovery in NLP is **word vectors**
- Previously, we understood the semantics in a statistical way
  - Consider the meaning of frequency
- Now, introduce the effect of the neighbors of a word
  - have on its meaning
  - how those relationships affect the overall meaning of a statement

PREVIOUS APPROACH

- Suppose want to search a hotel in Boston on the web
  - Hostel, motel
  - How about Airbnb?
- Previous approach to represent a word: one-hot vector
  - Hotel in Boston: [0 0 1 0 0 0 0 … 0 0 0]
  - Motel in Boston: [0 0 0 0 0 0 … 0 1 0]
  - Two vectors are orthogonal
- Hard to measure a similarity among similar words

REPRESENTING WORDS BY THEIR CONTEXT

- J. R. Firth 1957: *You shall know a word by the company it keeps*
  - **Distributional semantics**
  - One of the most successful ideas of modern statistical NLP
- When a word *w* appears in a text, its **context** is the set of words that appear nearby

CONTEXT, CONTEXT, CONTEXT

- Consider the meanings of the word “star” in various contexts
  - The night sky was clear, and the **stars** twinkled above.
  - Kim made a wish upon a **star**, hoping for a positive change in her life.
  - She would be the next **star** of the classical music world.
  - The movie features a cast of **stars** from various parts of the world.

CONTEXT, CONTEXT, CONTEXT

- What does the term "*jejune*" indicate?
  - *The poem seems to be rather jejune.*
  - Two choices: uninteresting vs. exciting
  - How did you know?

WORD VECTORS

- Want to build a vector for each word
  - Dense vector with a limited length (300 - 500)
    - Previous one-hot vectors or BOW are spare vector
  - Capture similarity in similar contexts
  - **Word vectors** are also called (**word or neural**) **embeddings**
  - Called **distributed** representation
    - Previous representation is called **denotational** representation

WORD VECTORS

- Why dense vectors for similarity?
- More important question is: How
- Let's consider three words
  - dog, cat, pizza: represent them in vectors

DISCRETENESS OF LANGUAGE

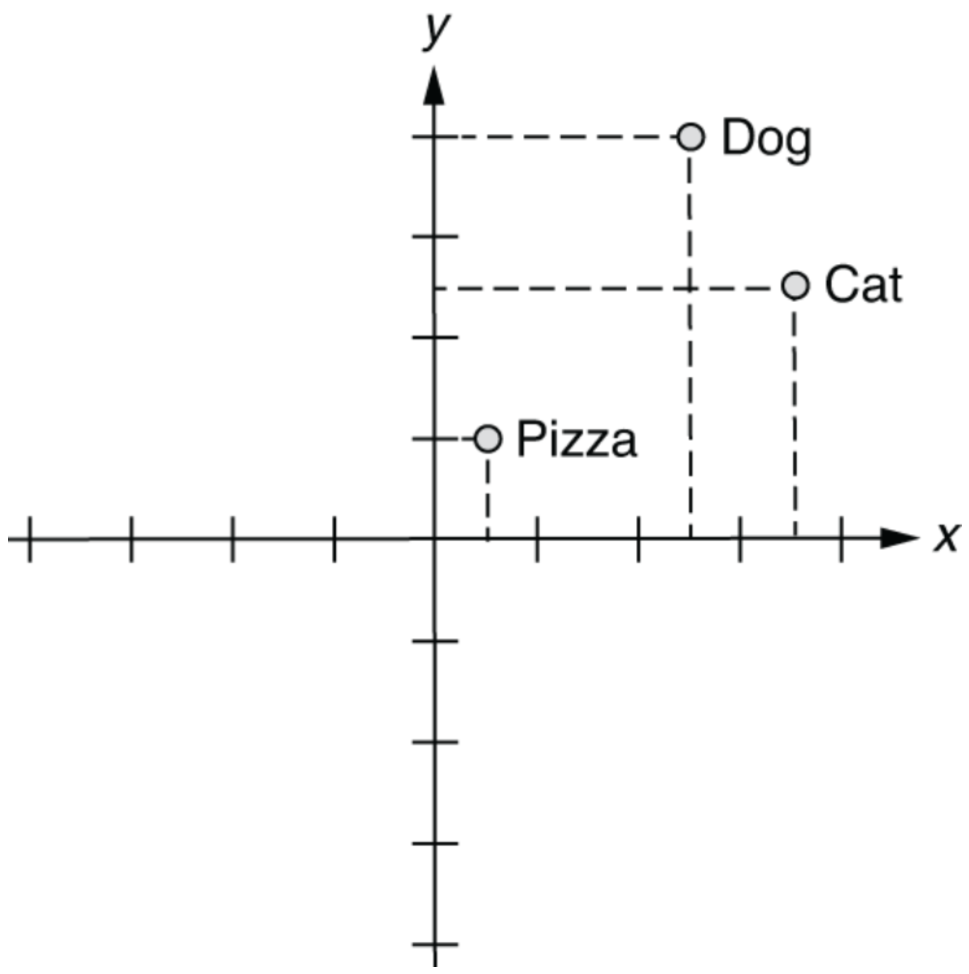
- Consider one dim vector: scalar
- Let’s assign indices:
  - Index(“cat”) = 1
  - Index(“dog”) = 2
  - Index(“pizza”) = 3
- But this method isn’t any better than dealing with raw words

WORD EMBEDDINGS IN A 1-D SPACE

- What if we can represent them on a numerical scale?

- This is a step forward.
- What if you wanted to place it somewhere that is equally far from "cat" and "dog?"

WORD EMBEDDINGS IN A 2-D SPACE



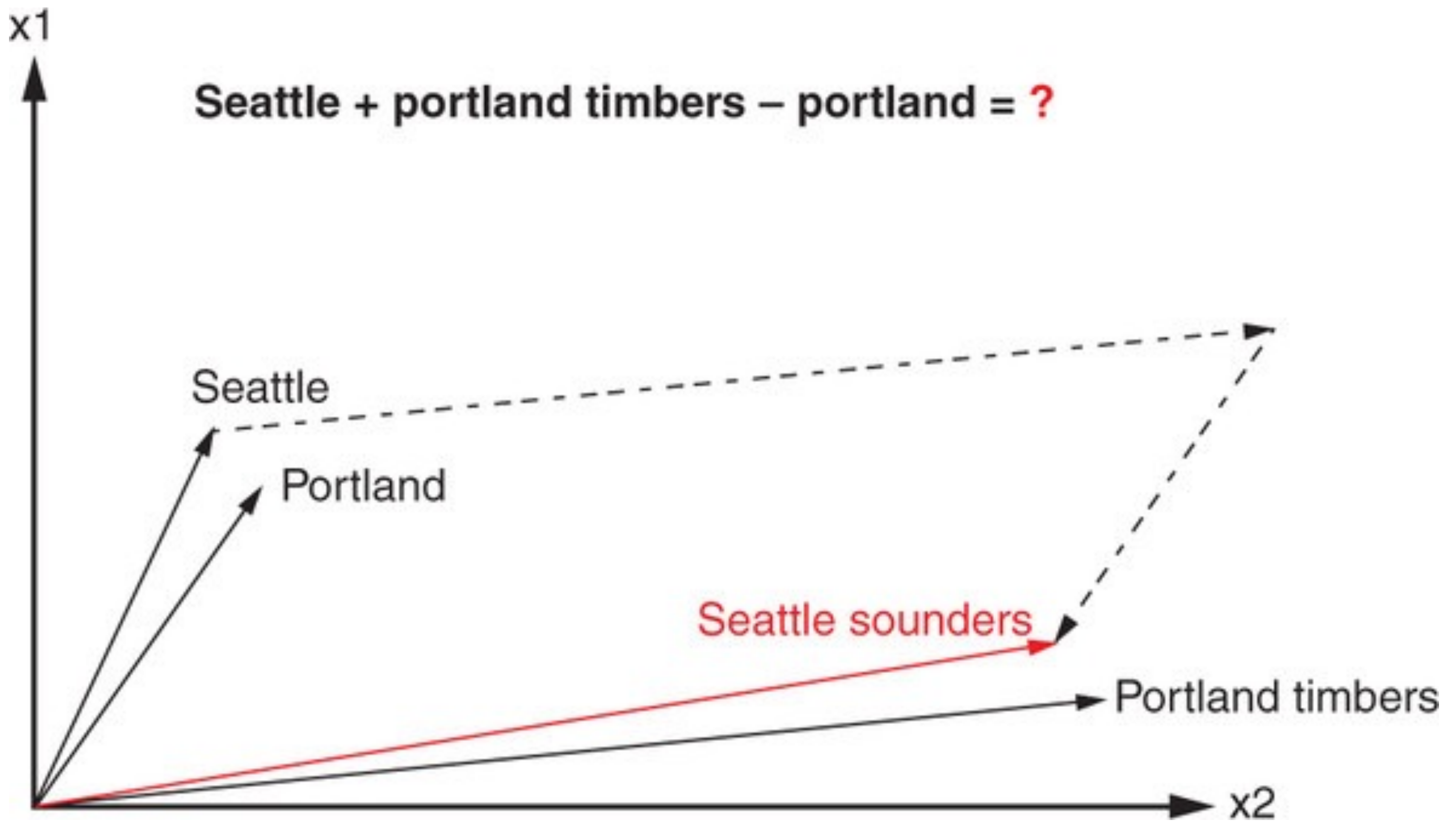
HOW ABOUT 3-D?

- index("cat") = [0.7, 0.5, 0.1]
- index("dog") = [0.8, 0.3, 0.1]
- index("pizza") = [0.1, 0.2, 0.8]
- Possibly attach meanings here?
  - Look at the number on X and Z axes
  - This is essentially what word embeddings are
- Each axis may preserve some special meanings

WORK WELL ON ANALOGY QUESTION

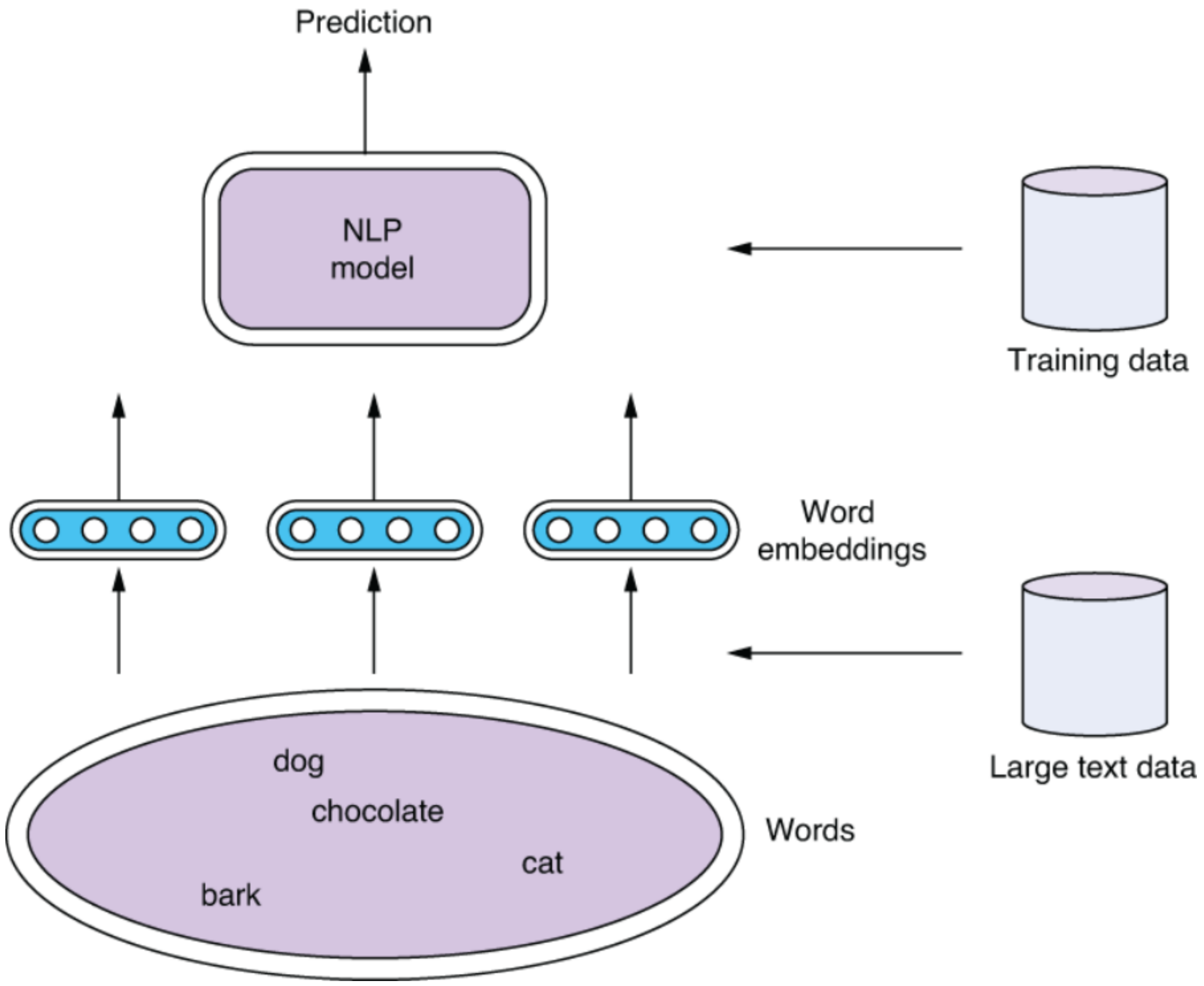
- Word vectors are vectors, so we can do vector reasoning
  - Portland Timbers: Men’s soccer club in Portland
  - What is the name of men’s soccer club in Seattle?

WORK WELL ON ANALOGY QUESTION



AGAIN, PREVIOUSLY

- Much simpler method to “embed” words into a vector space
  - index(“cat”) = [1, 0, 0]
  - index(“dog”) = [0, 1, 0]
  - index(“pizza”) = [0, 0, 1]
- Not very useful in representing semantic relationship between them
  - All at equal distance from each other


Prediction
NLP model
Training data
Word embeddings
Large text data
dog
chocolate
cat
bark
Words

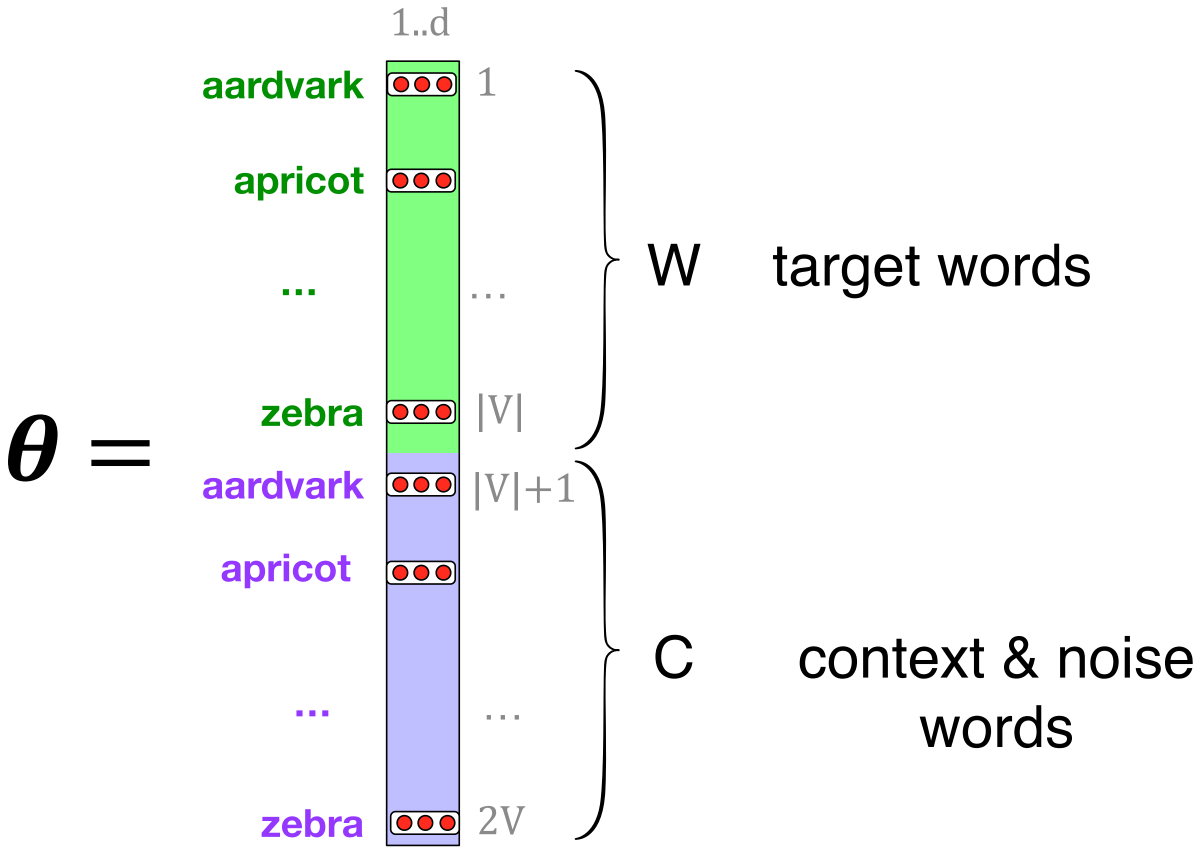
WORD2VEC

- In 2012, Thomas Mikolov, an intern at Microsoft, found a way to encode the meaning of words
- "Word2Vec" was indeed one of the most cited papers in NLP
- Static embeddings, but remarkably well on vector reasoning

WHY WORD2VEC?

- Fast to train
- Easy access to the legacy code
  - Archive: https://code.google.com/archive/p/word2vec/
  - Easy way: Gensim
  - Hard way: https://www.tensorflow.org/tutorials/text/word2vec

LET'S SEE WHAT WE WANT, FIRST



SKIP-GRAM ALGORITHM

- Conceptually, use logistic regression
- However, no interest on classification task
- The learned weights are of interest, and they are the embeddings
- Inputs are the word pairs, and the target is whether they are neighbors (loosely)

- Consider the word '*apricot*'
  - *…lemon, a tablespoon of* ***apricot*** *jam, a pinch…*
- Classify whether a word around '*apricot*' is its context
  - Using a window around *apricot*, we can define the **context**
  - …lemon, a [tablespoon of apricot jam, a] pinch…
- Train a classifier for a candidate (word, context) pair
  - given a specific word w and c, want to predict $P(C{=}c\,|\,W{=}w)$
- Two questions
  - How to construct negative (-) word pair?
  - How to estimate $P(C{=}c\,|\,W{=}w)$?

- How to estimate $P(C{=}c \mid W{=}w)$?
- Word embedding takes accounts word's context
  - A word may occur near the target if their embeddings are similar
- Word embedding is a vector
- What does it mean when a dot product of two vector is high?
  - $\text{sim}(d_1, d_2) = \frac{\vec{V}(d_1)\cdot\vec{V}(d_2)}{|\vec{V}(d_1)||\vec{V}(d_2)|} = \vec{v}(d_1) \cdot \vec{v}(d_2)$
  - Similarity$(c,w) \cong \boldsymbol{c} \cdot \boldsymbol{w}$

- Note that $\boldsymbol{c} \cdot \boldsymbol{w}$ is not a probability
  - Need to transform using logistic function!
  - $P(C = c|W = w) = \sigma(c \cdot w) = \frac{1}{1+\exp(-c \cdot w)}$
- It is the probability for one word, but there could be many context words
- Skip-gram makes the simplifying assumption that all context words are independent, so we can use
  - $P(C = c_1, \ldots c_L|W = w) = \Pi_i \sigma(c_i \cdot w)$

- Now consider how to construct the negative word pairs
  - …lemon, a [tablespoon of apricot jam, a] pinch…
  - Not feasible to use entire word outside the window
- Word2vec uses the Negative Sampling
- Treat the target word *w* and a context word *c* as positive examples
- Randomly sample other words in the lexicon to get negative examples
  - Sample words more than context words
  - Those words should be rare

SKIP-GRAM TRAINING DATA

…lemon, a [tablespoon of apricot jam, a] pinch…

positive examples +

t	c
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

negative examples -

t	c	t	c
apricot	aardvark	apricot	seven
apricot	my	apricot	forever
apricot	where	apricot	dear
apricot	coaxial	apricot	if

- Same intuition from logistic regression
- Training data: the set of positive and negative instances
- Weights: embeddings
- The goal of the learning algorithm is to adjust those embeddings to
  - Maximize the similarity of the word pairs (target word and context word)
    - Dot product of the word with the actual context words
  - Minimize the similarity of the word pairs (target word and non context word)
    - Dot product of the word with non context words

REMEMBER?

We'll represent $\hat{y}$ as $f(x; \theta)$ to make the dependence on θ more obvious:

$$\nabla_\theta L(f(x;\theta),y)) = \begin{bmatrix} \frac{\partial}{\partial w_1} L(f(x;\theta),y) \\ \frac{\partial}{\partial w_2} L(f(x;\theta),y) \\ \vdots \\ \frac{\partial}{\partial w_n} L(f(x;\theta),y) \end{bmatrix}$$

The final equation for updating θ based on the gradient is thus

$$\theta_{t+1} = \theta_t - \eta \nabla L(f(x;\theta),y)$$

- $L_{ce} = -[\log \sigma(c_{pos} \cdot w) + \sum \log \sigma(-c_{neg_i} \cdot w)]$

$$
\begin{aligned}
\mathbf{c}_{pos}^{t+1} &= \mathbf{c}_{pos}^{t} - \eta[\sigma(\mathbf{c}_{pos}^{t} \cdot \mathbf{w}^{t}) - 1]\mathbf{w}^{t} \\
\mathbf{c}_{neg}^{t+1} &= \mathbf{c}_{neg}^{t} - \eta[\sigma(\mathbf{c}_{neg}^{t} \cdot \mathbf{w}^{t})]\mathbf{w}^{t} \\
\mathbf{w}^{t+1} &= \mathbf{w}^{t} - \eta\left[[\sigma(\mathbf{c}_{pos} \cdot \mathbf{w}^{t}) - 1]\mathbf{c}_{pos} + \sum_{i=1}^{k}[\sigma(\mathbf{c}_{\mathrm{neg}_i} \cdot \mathbf{w}^{t})]\mathbf{c}_{neg_i}\right]
\end{aligned}
$$

APPLICATION: GENSIM

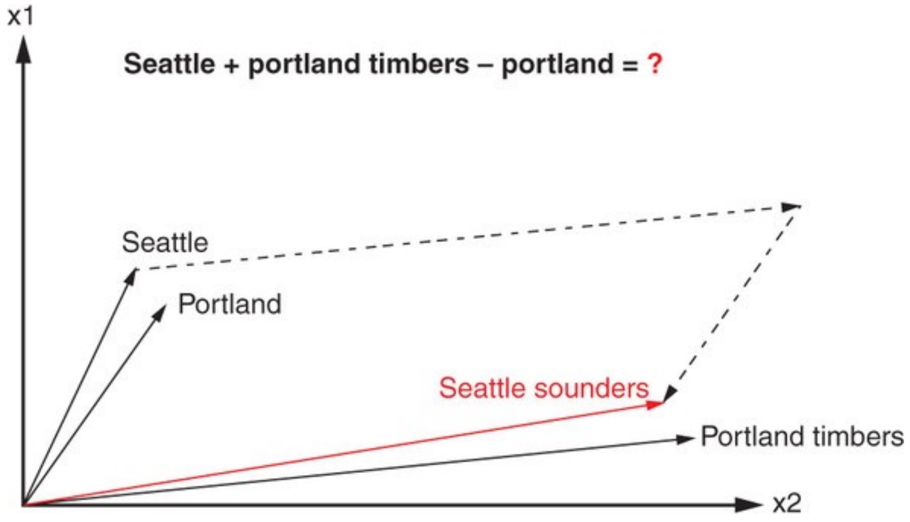
- We will train Word2vec later
  - Use Keras
- Gensim is handy way to play with distributional semantics
- Gensim isn’t really a deep learning package.
- But its efficient and scalable, and quite widely used

DISCUSSIONS

- What's the size of window in the previous example?
- Small windows (C= +/- 2)
  - nearest words are syntactically similar words in same taxonomy
    - Hogwarts nearest neighbors are other fictional schools: Sunnydale, Evernight,
- Large windows (C= +/- 5)
  - nearest words are related words in same semantic field
    - Hogwarts nearest neighbors are Harry Potter world: Dumbledore, half-blood, Malfoy

WORD ANALOGY, AGAIN?

- Portland Timbers + Seattle - Portland = ?



VECTOR-ORIENTED REASONING

- The Word2vec model contains information about the relationships between words

$$\begin{bmatrix} 0.0168 \\ 0.007 \\ 0.247 \\ \cdots \end{bmatrix} + \begin{bmatrix} 0.093 \\ -0.028 \\ -0.214 \\ \cdots \end{bmatrix} - \begin{bmatrix} 0.104 \\ 0.0883 \\ -0.318 \\ \cdots \end{bmatrix} = \begin{bmatrix} 0.006 \\ -0.109 \\ 0.352 \\ \cdots \end{bmatrix}$$

- After adding and subtracting word vectors, your resultant vector will almost never exactly equal one of the vectors in your word vector vocabulary

PROSPERITIES OF EMBEDDINGS

- Word2Vec requires a substantial amount of text data to learn meaningful word representations
- Any concerns?

LET'S FIND OUT SOME ERRORS

- One day, a father is driving with his son, but suddenly a massive crash happened.
- The Dad died, but fortunately the son survived.
- His condition was critical, so he was rushed to the hospital and awaited a major surgery.
- However, when the son reached the hospital, the doctor refused to perform the surgery.
- Saying: "I cannot do this, since he is my son."