CSC321 Lecture 10: Distributed Representations

Roger Grosse

Overview

- So far, the course has focused on the math of training neural nets.
- Now let's switch gears and talk look at particular neural net architectures and how they can be used to understand various kinds of data. This will take us the next 4 weeks.
- Today's lecture: learning distributed representations of words

Language Modeling

Motivation: suppose we want to build a speech recognition system.

We'd like to be able to infer a likely sentence $\mathbf{s}$ given the observed speech signal $\mathbf{a}$. The generative approach is to build two components:

- An observation model, represented as $p(\mathbf{a} \mid \mathbf{s})$, which tells us how likely the sentence $\mathbf{s}$ is to lead to the acoustic signal $\mathbf{a}$.
- A prior, represented as $p(\mathbf{s})$, which tells us how likely a given sentence $\mathbf{s}$ is. E.g., it should know that "recognize speech" is more likely that "wreck a nice beach."

Language Modeling

Motivation: suppose we want to build a speech recognition system.

We'd like to be able to infer a likely sentence $\mathbf{s}$ given the observed speech signal $\mathbf{a}$. The generative approach is to build two components:

- An observation model, represented as $p(\mathbf{a} \mid \mathbf{s})$, which tells us how likely the sentence $\mathbf{s}$ is to lead to the acoustic signal $\mathbf{a}$.
- A prior, represented as $p(\mathbf{s})$, which tells us how likely a given sentence $\mathbf{s}$ is. E.g., it should know that "recognize speech" is more likely that "wreck a nice beach."

Given these components, we can use Bayes' Rule to infer a posterior distribution over sentences given the speech signal:

$$p(\mathbf{s} \mid \mathbf{a}) = \frac{p(\mathbf{s})p(\mathbf{a} \mid \mathbf{s})}{\sum_{\mathbf{s}'} p(\mathbf{s}')p(\mathbf{a} \mid \mathbf{s}')}.$$

Language Modeling

In this lecture, we focus on learning a good distribution $p(\mathbf{s})$ of sentences. This problem is known as language modeling.

Assume we have a corpus of sentences $\mathbf{s}^{(1)}, \ldots, \mathbf{s}^{(N)}$. The maximum likelihood criterion says we want our model to maximize the probability our model assigns to the observed sentences. We assume the sentences are independent, so that their probabilities multiply.

$$\max \prod_{i=1}^{N} p(\mathbf{s}^{(i)}).$$

Language Modeling

In maximum likelihood training, we want to maximize $\prod_{i=1}^{N} p(\mathbf{s}^{(i)})$.

The probability of generating the whole training corpus is vanishingly small — like monkeys typing all of Shakespeare.

- The log probability is something we can work with more easily. It also conveniently decomposes as a sum:

$$\log \prod_{i=1}^{N} p(\mathbf{s}^{(i)}) = \sum_{i=1}^{N} \log p(\mathbf{s}^{(i)}).$$

- Let's use *negative* log probabilities, so that we're working with positive numbers.

Language Modeling

In maximum likelihood training, we want to maximize $\prod_{i=1}^{N} p(\mathbf{s}^{(i)})$.

The probability of generating the whole training corpus is vanishingly small — like monkeys typing all of Shakespeare.

- The log probability is something we can work with more easily. It also conveniently decomposes as a sum:

$$\log \prod_{i=1}^{N} p(\mathbf{s}^{(i)}) = \sum_{i=1}^{N} \log p(\mathbf{s}^{(i)}).$$

- Let's use *negative* log probabilities, so that we're working with positive numbers.

Better trained monkeys are slightly more likely to type *Hamlet*!

Language Modeling

Probability of a sentence? What does that even mean?

Language Modeling

Probability of a sentence? What does that even mean?
A sentence is a sequence of words $w_1, w_2, \ldots, w_T$. Using the chain rule of conditional probability, we can decompose the probability as

$$p(\mathbf{s}) = p(w_1, \ldots, w_T) = p(w_1)p(w_2 \mid w_1) \cdots p(w_T \mid w_1, \ldots, w_{T-1}).$$

Therefore, the language modeling problem is equivalent to being able to predict the next word!

Language Modeling

Probability of a sentence? What does that even mean?
A sentence is a sequence of words $w_1, w_2, \ldots, w_T$. Using the chain rule of conditional probability, we can decompose the probability as

$$p(\mathbf{s}) = p(w_1, \ldots, w_T) = p(w_1)p(w_2 \mid w_1) \cdots p(w_T \mid w_1, \ldots, w_{T-1}).$$

Therefore, the language modeling problem is equivalent to being able to predict the next word!

We typically make a Markov assumption, i.e. that the distribution over the next word only depends on the preceding few words. I.e., if we use a context of length 3,

$$p(w_t \mid w_1, \ldots, w_{t-1}) = p(w_t \mid w_{t-3}, w_{t-2}, w_{t-1}).$$

Such a model is called memoryless.
Now it's basically a supervised prediction problem. We need to predict the conditional distribution of each word given the previous K.

N-Gram Language Models

- One sort of Markov model we can learn uses a conditional probability table, i.e.

	cat	and	city	⋯
the fat	0.21	0.003	0.01	
four score	0.0001	0.55	0.0001	⋯
New York	0.002	0.0001	0.48	
⋮		⋮		

- Maybe the simplest way to estimate the probabilities is from the empirical distribution:

$$p(w_3 = \text{cat} \mid w_1 = \text{the}, w_2 = \text{fat}) = \frac{\text{count}(\text{the fat cat})}{\text{count}(\text{the fat})}$$

- This is the maximum likelihood solution; we'll see why later in the course.
- The phrases we're counting are called n-grams (where n is the length), so this is an n-gram language model.

N-Gram Language Models

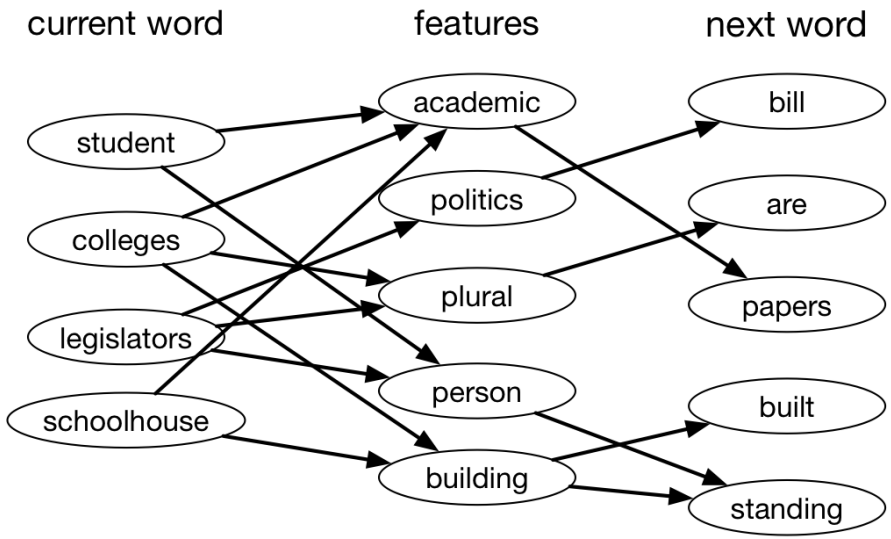
Samples from language models:
https://lagunita.stanford.edu/c4x/Engineering/CS-224N/asset/slp4.pdf

N-Gram Language Models

- Problems with n-gram language models
  - The number of entries in the conditional probability table is exponential in the context length.
  - Data sparsity: most n-grams never appear in the corpus, even if they are possible.
- Ways to deal with data sparsity
  - Use a short context (but this means the model is less powerful)
  - Smooth the probabilities, e.g. by adding imaginary counts
  - Make predictions using an ensemble of n-gram models with different n

Distributed Representations

- Conditional probability tables are a kind of localist representation: all the information about a particular word is stored in one place, i.e. a column of the table.
- But different words are related, so we ought to be able to share information between them. For instance,



- Here, the information about a given word is distributed throughout the representation. We call this a distributed representation.
- In general, unlike in this cartoon, we won't be able to attach labels to the features in our distributed representation.

Distributed Representations

- We would like to be able to share information between related words. E.g., suppose we've seen the sentence

  The cat got squashed in the garden on Friday.

- This should help us predict the words in the sentence

  The dog got flattened in the yard on Monday.

- An n-gram model can't generalize this way, but a distributed representation might let us do so.

Neural Language Model

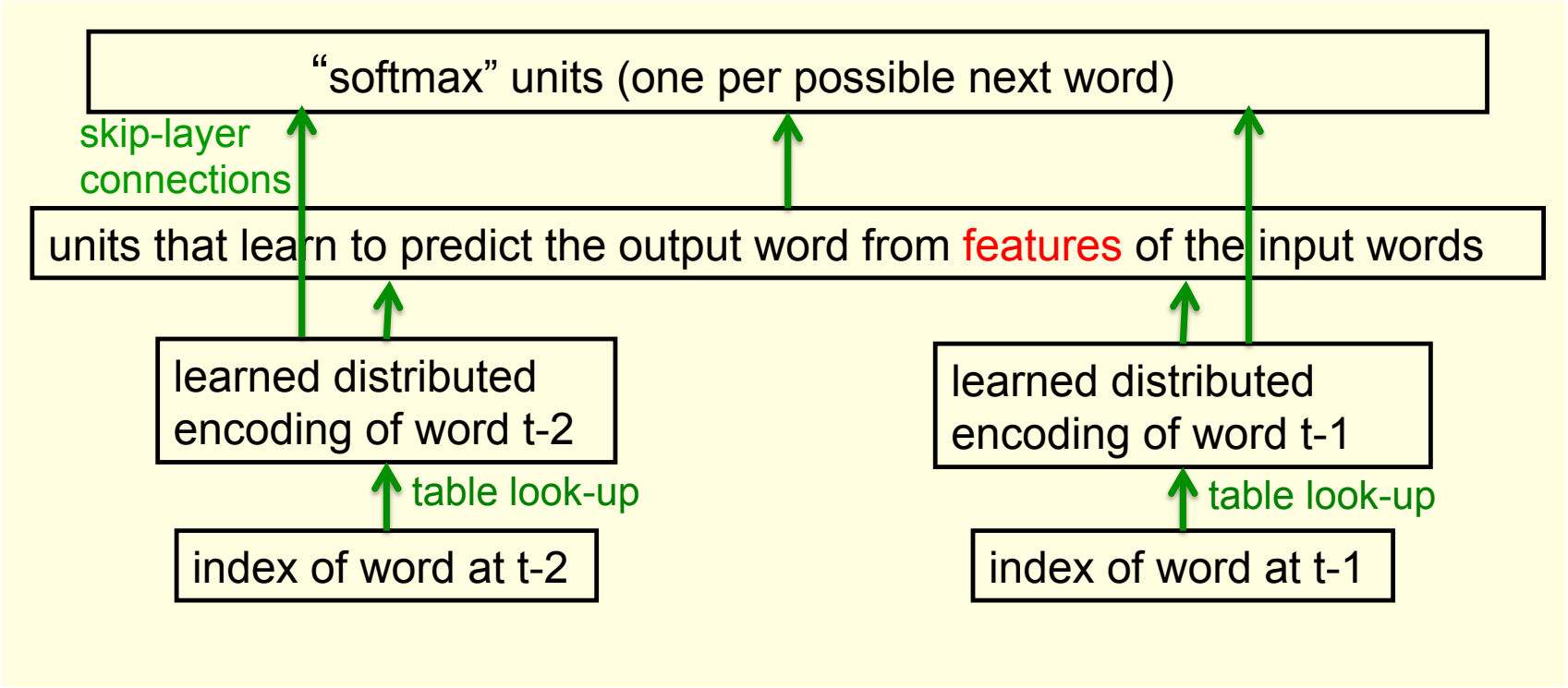
- Predicting the distribution of the next word given the previous K is just a multiway classification problem.
- **Inputs:** previous K words
- **Target:** next word
- **Loss:** cross-entropy. Recall that this is equivalent to maximum likelihood:

$$\begin{aligned} -\log p(\mathbf{s}) &= -\log \prod_{t=1}^{T} p(w_t \mid w_1, \ldots, w_{t-1}) \\ &= -\sum_{t=1}^{T} \log p(w_t \mid w_1, \ldots, w_{t-1}) \\ &= -\sum_{t=1}^{T} \sum_{v=1}^{V} t_{tv} \log y_{tv}, \end{aligned}$$

where t_{iv} is the one-of-K encoding for the ith word and y_{iv} is the predicted probability for the ith word being index v.

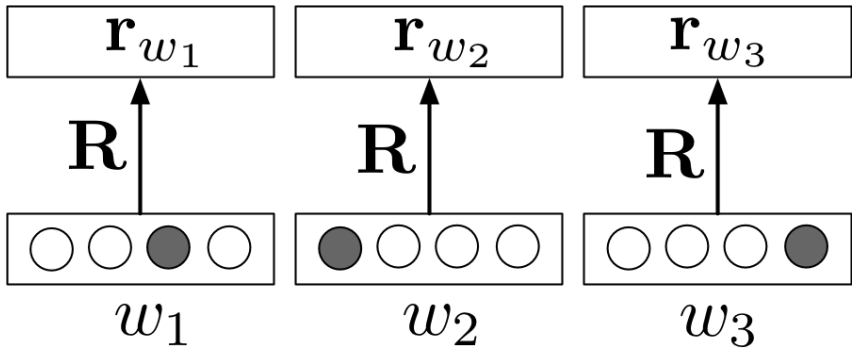
Neural Language Model

- Here is a classic neural probabilistic language model, or just neural language model:



Neural Language Model

- If we use a 1-of-K encoding for the words, the first layer can be thought of as a linear layer with tied weights.



- The weight matrix basically acts like a lookup table. Each column is the representation of a word, also called an embedding, feature vector, or encoding.
  - "Embedding" emphasizes that it's a location in a high-dimensonal space; words that are closer together are more semantically similar
  - "Feature vector" emphasizes that it's a vector that can be used for making predictions, just like other feature mappigns we've looked at (e.g. polynomials)

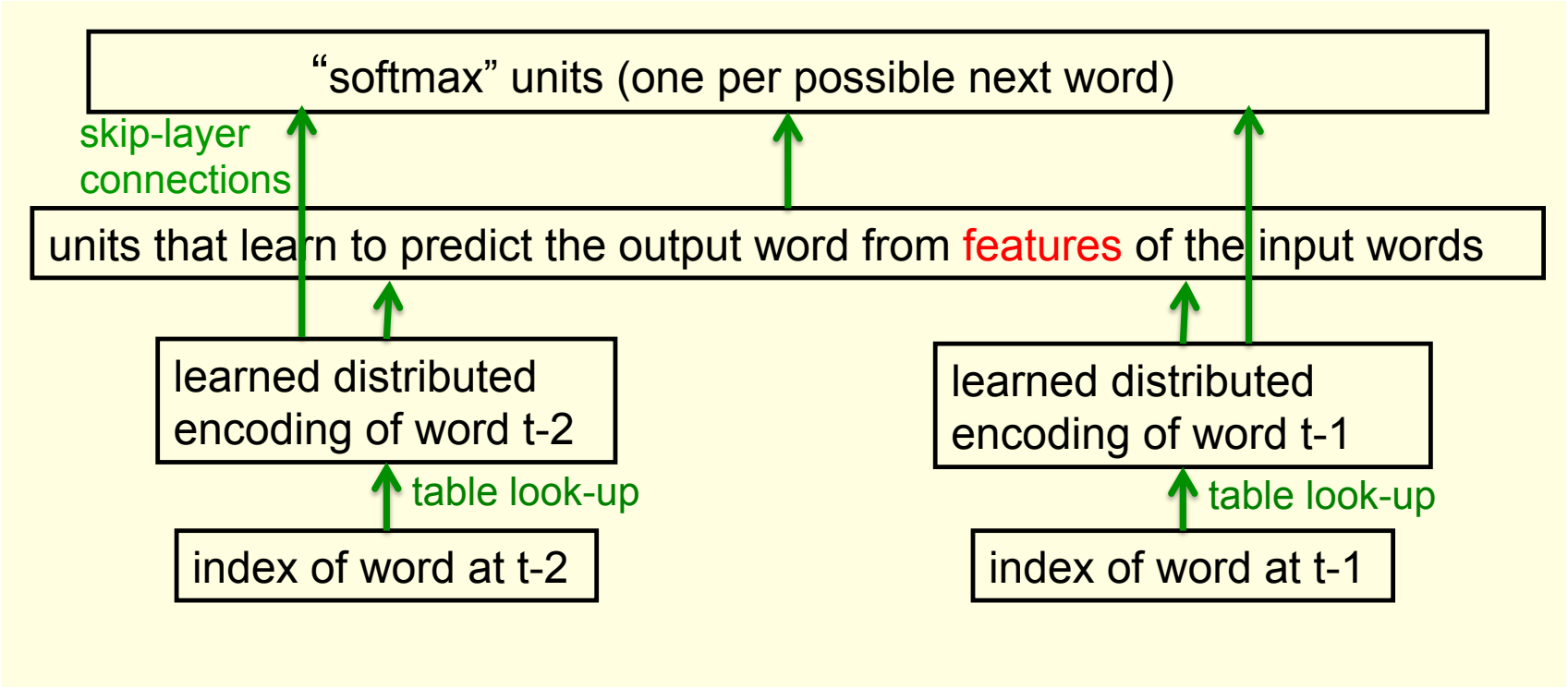
Neural Language Model

- We can measure the similarity or dissimilarity of two words using
  - the dot product $\mathbf{r}_1^\top \mathbf{r}_2$
  - Euclidean distance $\|\mathbf{r}_1 - \mathbf{r}_2\|$
- If the vectors have unit norm, the two are equivalent:

$$\begin{aligned}\|\mathbf{r}_1 - \mathbf{r}_2\|^2 &= (\mathbf{r}_1 - \mathbf{r}_2)^\top (\mathbf{r}_1 - \mathbf{r}_2) \\ &= \mathbf{r}_1^\top \mathbf{r}_1 - 2\mathbf{r}_1^\top \mathbf{r}_2 + \mathbf{r}_2^\top \mathbf{r}_2 \\ &= 2 - 2\mathbf{r}_1^\top \mathbf{r}_2\end{aligned}$$

Neural Language Model

- This model is very compact: the number of parameters is *linear* in the context size, compared with exponential for n-gram models.



Neural Language Model

- What do these word embeddings look like?
- It's hard to visualize an n-dimensional space, but there are algorithms for mapping the embeddings to two dimensions.
- The following 2-D embeddings are done with an algorithm called tSNE which tries to make distnaces in the 2-D embedding match the original 30-D distances as closely as possible.
- Note: the visualizations are from a slightly different model.

Neural Language Model

Neural Language Model

Neural Language Model

Neural Language Model

- Thinking about high-dimensional embeddings
  - Most vectors are nearly orthogonal (i.e. dot product is close to 0)
  - Most points are far away from each other
  - "In a 30-dimensional grocery store, anchovies can be next to fish and next to pizza toppings." – Geoff Hinton
- The 2-D embeddings might be fairly misleading, since they can't preserve the distance relationships from a higher-dimensional embedding. (I.e., unrelated words might be close together in 2-D, but far apart in 30-D.)

Neural language model

When we train a neural language model, is that supervised or unsupervised learning? Does it have elements of both?

Skip-Grams

- Fitting language models is really hard:
  - It's really important to make good predictions about relative probabilities of rare words.
  - Computing the predictive distribution requires a large softmax.
- Maybe this is overkill if all you want is word representations.

Skip-Grams

- Skip-gram model (Mikolov et al., 2013)
  - **Task:** given one word as input, predict (the distribution of) a word in its surrounding context.

    _____ _____ *fish* _____ __*?*__

  - Learn separate embeddings for the input and target word.
  - **Model:** softmax where the log odds are computed as the dot product:

$$p(w_{t+\tau} = a \mid w_t = b) = \frac{\exp(\tilde{\mathbf{r}}_a^\top \mathbf{r}_b)}{\sum_c \exp(\tilde{\mathbf{r}}_c^\top \mathbf{r}_b)}$$

  - **Loss:** cross-entropy, as usual
- Predictions are efficient because it's just a linear model, i.e. no hidden units.
- Problem: this still requires computing a softmax over the entire vocabulary!
  - The original paper used a model called "hierarchical softmax" to get around this, but there's an easier way.

Skip-Grams

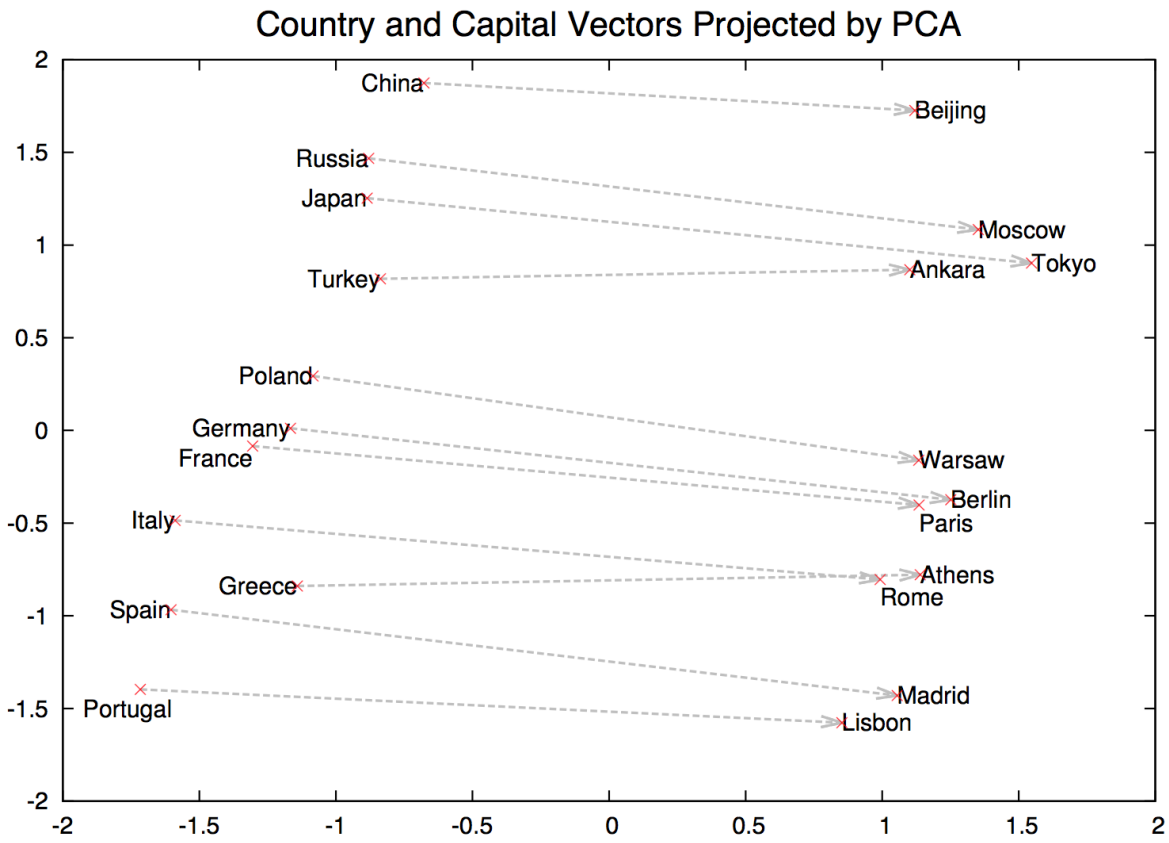
- Instead of predicting a distribution over words, switch to a binary prediction problem.
- Negative sampling: the model is given pairs of words, and needs to distinguish between:
  - real: the two words actually occur near each other in the training corpus
  - fake: the two words are sampled randomly from the training corpus
- Cross-entropy loss, with logistic activation function:

$$p(\mathit{real} \mid w_1 = a, w_2 = b) = \sigma(\tilde{\mathbf{r}}_a^\top \mathbf{r}_b) = \frac{1}{1 + \exp(-\tilde{\mathbf{r}}_a^\top \mathbf{r}_b)}$$

- This forces the dot product to be large for words which co-occur and small (or negative) for words which don't co-occur.
- Skip-grams with negative sampling can be trained very efficiently, so we can use tons of data.

Skip-Grams

- Here's a linear projection of word representations for cities and capitals into 2 dimensions.
- The mapping $\text{city} \rightarrow \text{capital}$ corresponds roughly to a single direction in the vector space:



Skip-Grams

- In other words,
  $\text{vector}(\text{Paris}) - \text{vector}(\text{France}) \approx \text{vector}(\text{London}) - \text{vector}(\text{England})$
- This means we can analogies by doing arithmetic on word vectors:
  - e.g. "Paris is to France as London is to _____"
  - Find the word whose vector is closest to
    $\text{vector}(\text{France}) - \text{vector}(\text{Paris}) + \text{vector}(\text{London})$
- Example analogies:

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza