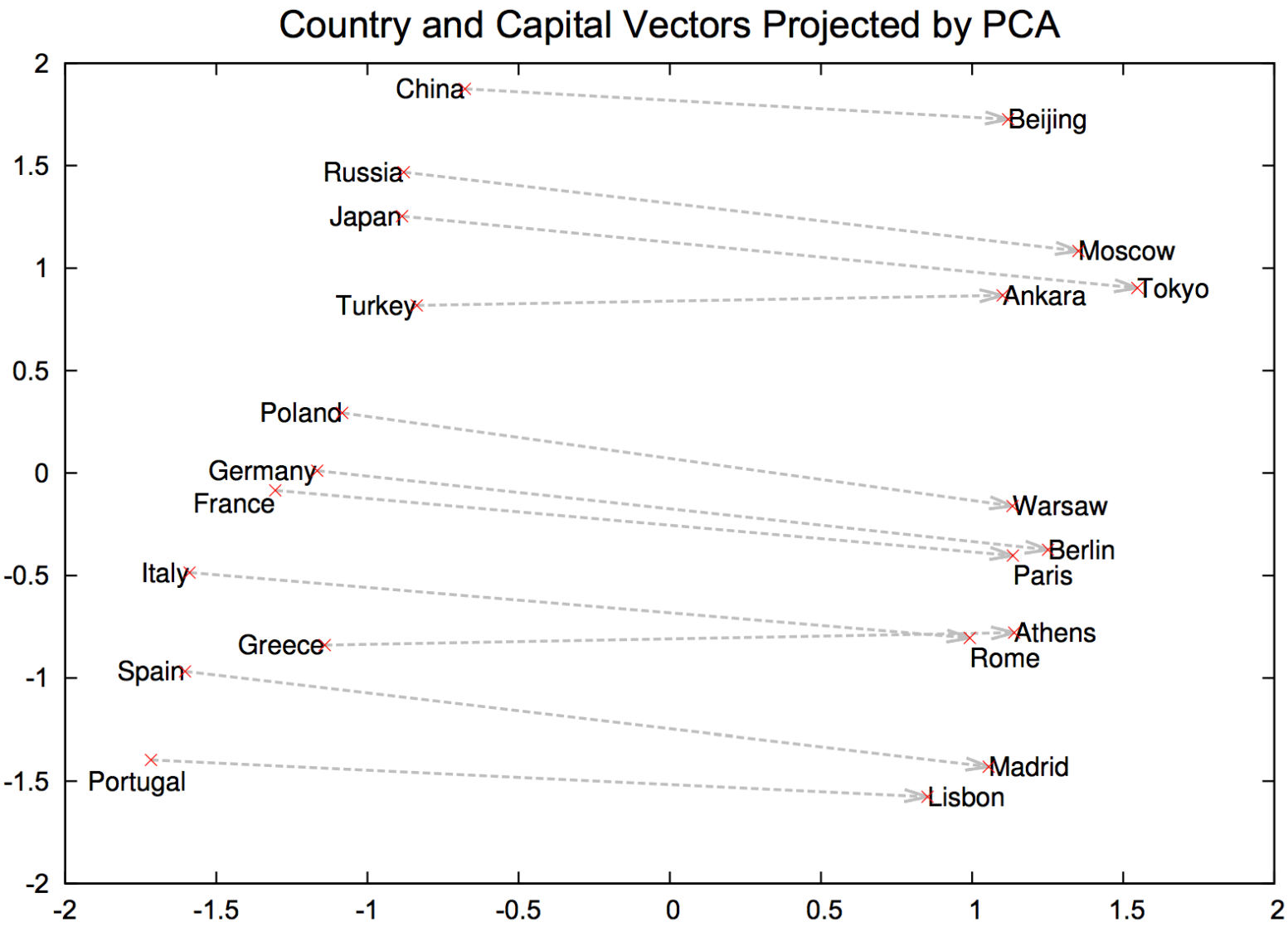
Search Engine Architecture

9. Distributed Word Representations

Agenda

- Distributed representations / distributional hypothesis
- Dimensionality reduction
- Artificial neural networks
- Representation learning
- Word2vec
  - Skip-gram
  - CBOW
- Doc2vec
- SVD reduction

Word2vec Linear Relationships



Source: Mikolov et al. (2013) "Distributed Representations of Words and Phrases and their Compositionality." *NIPS.*

Distributional Hypothesis

“You shall know a word by the company it keeps”

(J. R. Firth 1957: 11)

How to represent meaning?

- One answer: use a taxonomy like WordNet

Hypernyms (is-a relationships):

```
from nltk.corpus import wordnet as wn
panda = wn.synset('panda.n.01')
hyper = lambda s: s.hypernyms()
list(panda.closure(hyper))
```

```
[Synset('procyonid.n.01'),
Synset('carnivore.n.01'),
Synset('placental.n.01'),
Synset('mammal.n.01'),
Synset('vertebrate.n.01'),
Synset('chordate.n.01'),
Synset('animal.n.01'),
Synset('organism.n.01'),
Synset('living_thing.n.01'),
Synset('whole.n.02'),
Synset('object.n.01'),
Synset('physical_entity.n.01'),
Synset('entity.n.01')]
```

Synonym sets (good):

```
S: (adj) full, good
S: (adj) estimable, good, honorable, respectable
S: (adj) beneficial, good
S: (adj) good, just, upright
S: (adj) adept, expert, good, practiced,
proficient, skillful
S: (adj) dear, good, near
S: (adj) good, right, ripe
...
S: (adv) well, good
S: (adv) thoroughly, soundly, good
S: (n) good, goodness
S: (n) commodity, trade good, good
```

Source: Socher et al. CS224d – Deep Learning for Natural Language Processing, Stanford Spring 2016.

Problems with discrete representation

- Missing nuances
  - adept, expert, good, practiced, proficient, skillful
- Impossible to keep up to date
  - wicked, badass, nifty, legend, ninja
- Subjective
- Requires human labor
- *Hard to compute word similarity*

Source: Manning et al. CS276 - Information Retrieval and Web Search, Stanford Spring 2016

How can we represent term relations?

- With the standard symbolic encoding of terms, each term is a dimension
- Different terms have no inherent similarity
- motel $[0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0]^T$
  hotel $[0\ 0\ 0\ 0\ 0\ 0\ 0\ 3\ 0\ 0\ 0\ 0\ 0\ 0\ 0] = 0$
- If query on *hotel* and document has *motel*, then our query and document vectors are **orthogonal**

Distributional similarity-based representations

- You can get a lot of value by representing a word by means of its neighbors
- "You shall know a word by the company it keeps"
  - (J. R. Firth 1957: 11)
- One of the most successful ideas of modern statistical NLP

government debt problems turning into	banking	crises as has happened in
saying that Europe needs unified	banking	regulation to replace the hodgepodge

↖ These words will represent *banking* ↗

Source: Manning et al. CS276 - Information Retrieval and Web Search, Stanford Spring 2016

Representing words with neighbors

- Use co-occurrence matrix
- Two options for context – document or window
- Word-document co-occurrence matrix leads to general topics (sports terms will have similar entries)
  - "Latent Semantic Analysis"
- Instead – window around each word captures syntactic and semantic information

Source: Manning et al. CS276 - Information Retrieval and Web Search, Stanford Spring 2016

Window-based co-occurrence matrix

- Corpus
  - I like deep learning.
  - I like NLP.
  - I enjoy flying.
- (Window size 1)

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

Source: Socher et al. CS224d – Deep Learning for Natural Language Processing, Stanford Spring 2016.

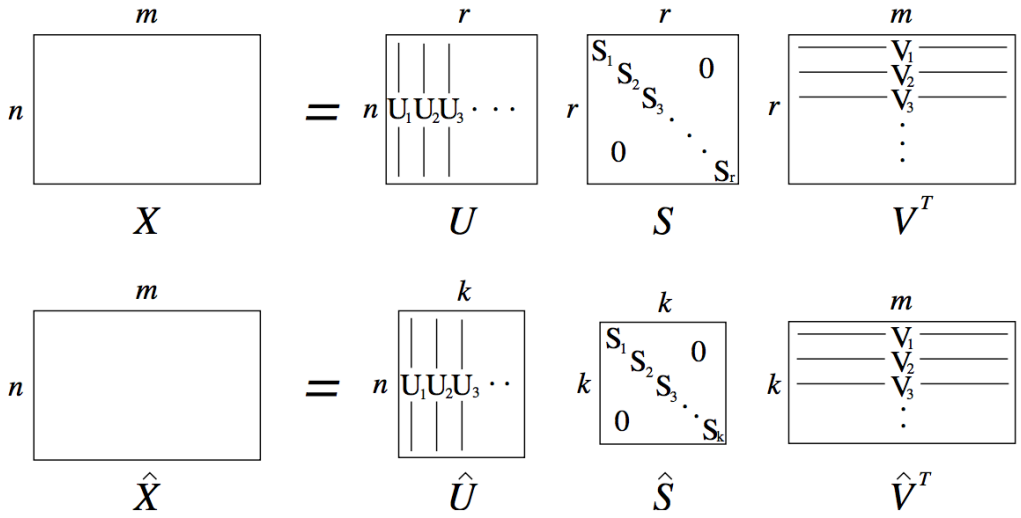
Problems with co-occurrence vectors

- Increase in size with vocabulary
- High dimensional
- Extreme sparsity leads to less robust downstream models
- *How to reduce dimensionality?*

Source: Socher et al. CS224d – Deep Learning for Natural Language Processing, Stanford Spring 2016.

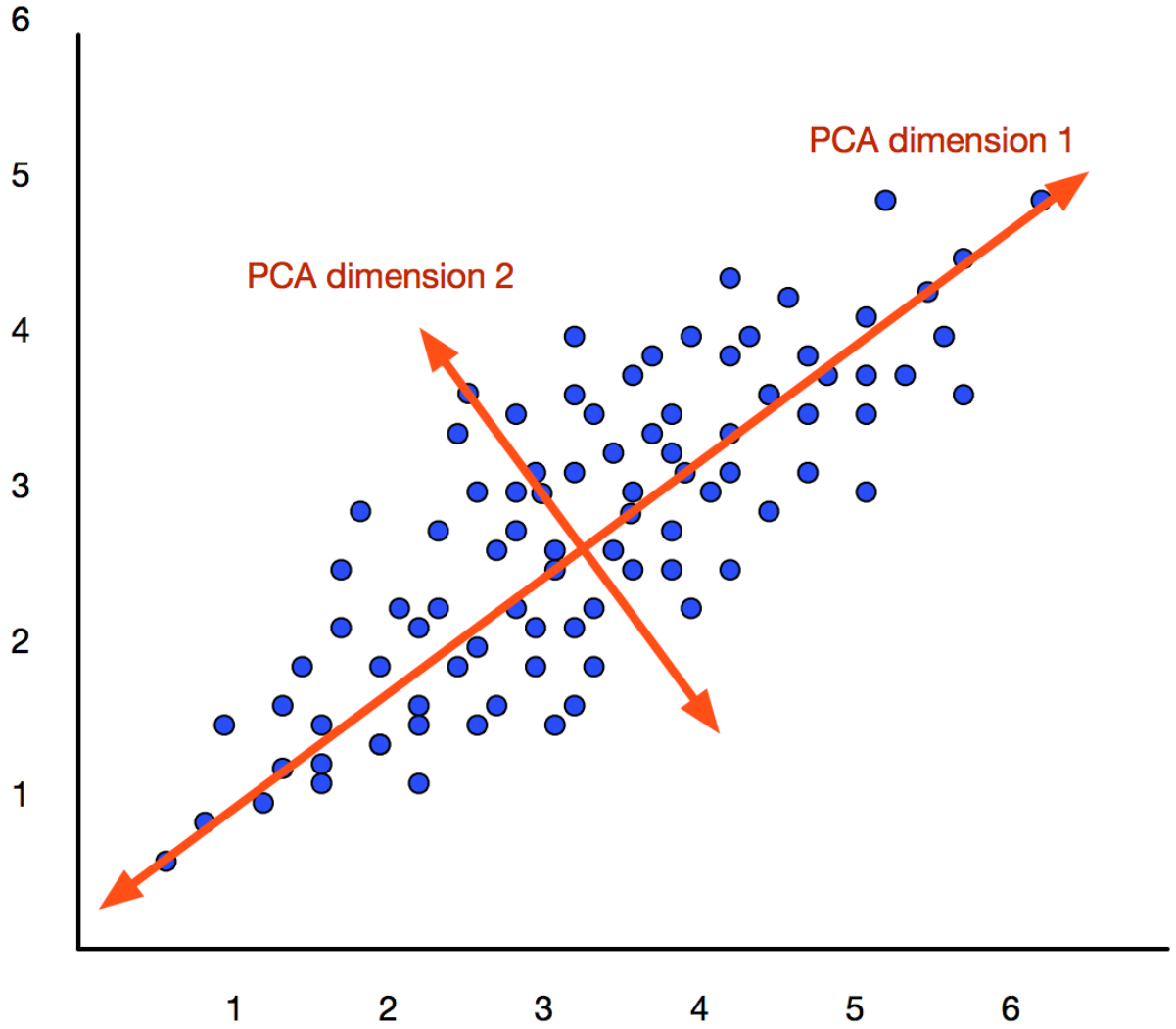
Traditional Way: Latent Semantic Indexing/Analysis

- Use Singular Value Decomposition (SVD) – kind of like Principal Components Analysis (PCA) for an arbitrary rectangular matrix
- Theory is that similarity is preserved as much as possible
- Weakly, you can actually gain in IR by doing LSA as "noise" of term variation gets replaced by semantic "concepts"



Source: Socher et al. CS224d – Deep Learning for Natural Language Processing, Stanford Spring 2016.

Dimensionality Reduction



Source: Socher et al. CS224d – Deep Learning for Natural Language Processing, Stanford Spring 2016.

Word meaning in terms of vectors

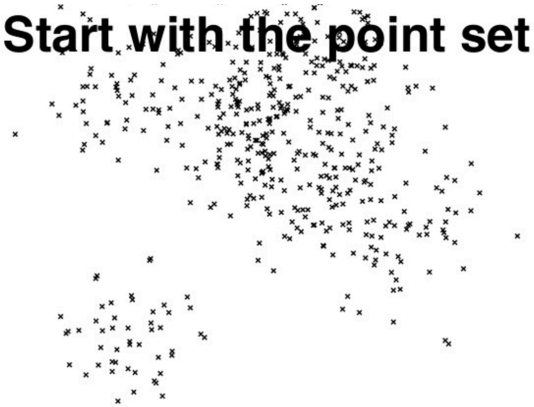
- In all subsequent models, including deep learning models, a word is represented as a dense vector

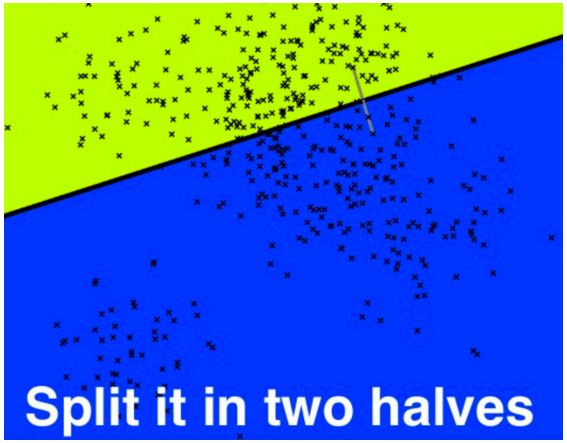
$$\textit{linguistics} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$$

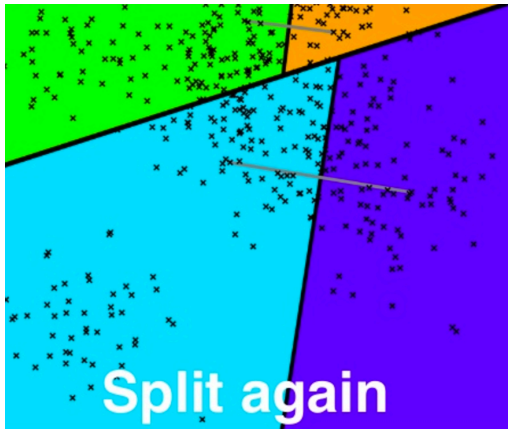
Source: Manning et al. CS276 - Information Retrieval and Web Search, Stanford Spring 2016

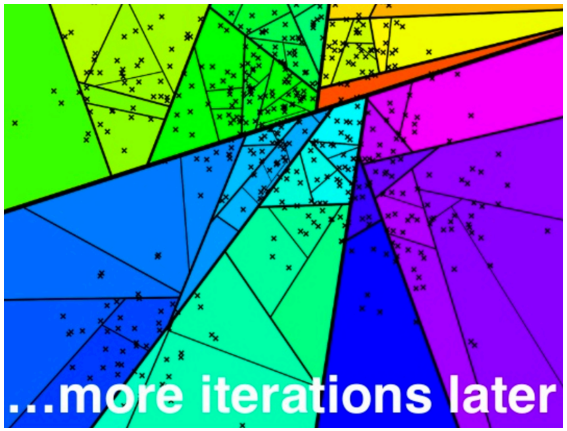
What's the point?

- Word models have many applications
- E.g. find synonyms with Spotify's Annoy:





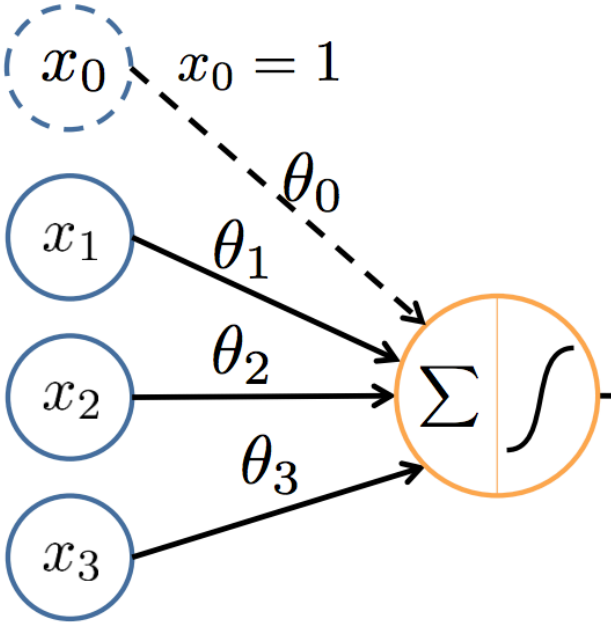




Source: Bernhardsson. (2015) "Approximate nearest neighbor methods and vector models."

Detour:
Artificial Neural Networks

Neuron



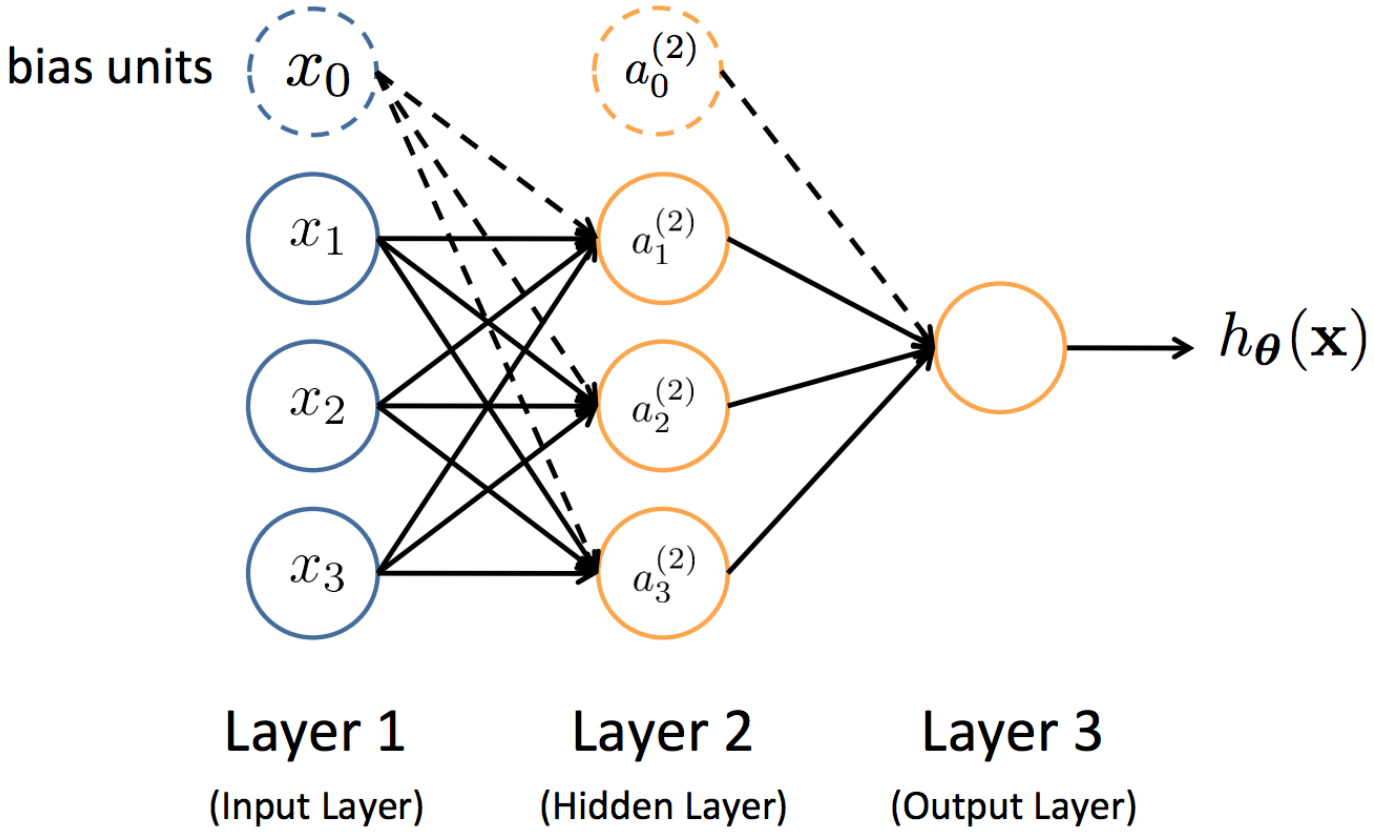
$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = g\left(\boldsymbol{\theta}^{\mathsf{T}}\mathbf{x}\right) = \frac{1}{1 + e^{-\boldsymbol{\theta}^{\mathsf{T}}\mathbf{x}}}$$

Sigmoid (logistic) activation function: $g(z) = \dfrac{1}{1 + e^{-z}}$

Based on slide by Andrew Ng

Neural Network



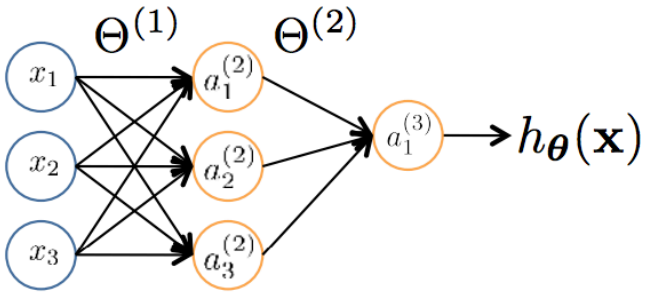
Based on slide by Andrew Ng

Feed-Forward Process

- Input layer units are set by some external function (think of these as **sensors**), which causes their output links to be **activated** at the given level
- Working forward through the network, the **input function** of each unit is applied to compute the input value
  - Usually this is just the weighted sum of the activation on the links feeding into the node
- The **activation function** transforms this input into a final value
  - Typically this is a **nonlinear** function, often a **sigmoid** function corresponding to the “threshold” of the node

Based on slide by Andrew Ng

Feed-Forward Process



$a_i^{(j)}$ = "activation" of unit i in layer j

$\Theta^{(j)}$ = weight matrix controlling function mapping from layer j to layer j + 1

$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$$
$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3)$$
$$a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3)$$
$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})$$

If network has s_j units in layer j *and* s_{j+1} units in layer $j+1$, then $\Theta^{(j)}$ has dimension $s_{j+1} \times (s_j+1)$.

$$\Theta^{(1)} \in \mathbb{R}^{3\times 4} \qquad \Theta^{(2)} \in \mathbb{R}^{1\times 4}$$

Based on slide by Andrew Ng

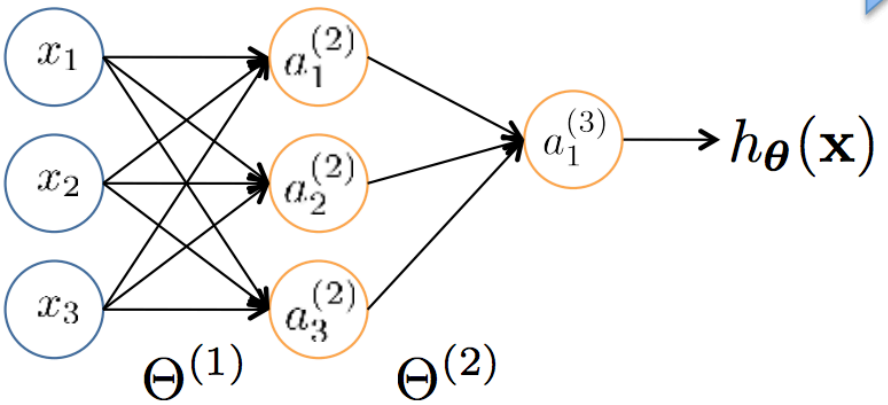
Vectorization

$$a_1^{(2)} = g\left(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3\right) = g\left(z_1^{(2)}\right)$$

$$a_2^{(2)} = g\left(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3\right) = g\left(z_2^{(2)}\right)$$

$$a_3^{(2)} = g\left(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3\right) = g\left(z_3^{(2)}\right)$$

$$h_\Theta(\mathbf{x}) = g\left(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)}\right) = g\left(z_1^{(3)}\right)$$



Based on slide by Andrew Ng

Feed-Forward Steps:

$$\mathbf{z}^{(2)} = \Theta^{(1)}\mathbf{x}$$

$$\mathbf{a}^{(2)} = g(\mathbf{z}^{(2)})$$

$$\text{Add } a_0^{(2)} = 1$$

$$\mathbf{z}^{(3)} = \Theta^{(2)}\mathbf{a}^{(2)}$$

$$h_\Theta(\mathbf{x}) = \mathbf{a}^{(3)} = g(\mathbf{z}^{(3)})$$

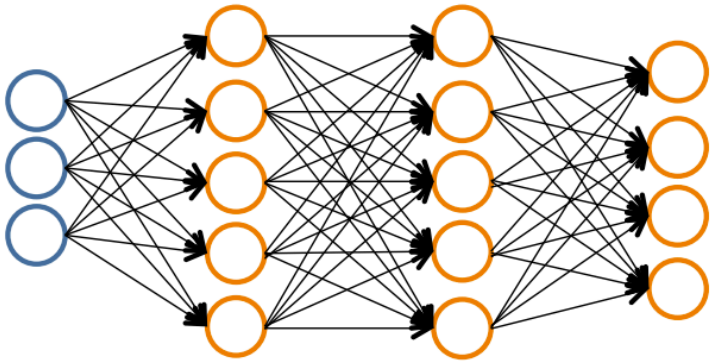
Multiple Output Units: One Vs. Rest

Pedestrian

Car

Motorcycle

Truck



$$h_\Theta(\mathbf{x}) \in \mathbb{R}^K$$

We want:

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

when pedestrian

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

when car

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

when motorcycle

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

when truck

Based on slide by Andrew Ng

Learning Via Backpropagation

- Cycle through examples
  - If the output of the network is correct, no changes are made
  - If there is an error, weights are adjusted to reduce the error
- The trick is to assess the blame for the error and divide it among the contributing weights
- Iteratively update weights by gradient descent

Based on slide by Andrew Ng

Cost Function

Logistic Regression:

$$J(\theta) = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log h_{\boldsymbol{\theta}}(\mathbf{x}_i) + (1-y_i)\log(1-h_{\boldsymbol{\theta}}(\mathbf{x}_i))] + \frac{\lambda}{2n}\sum_{j=1}^{d}\theta_j^2$$

Neural Network:

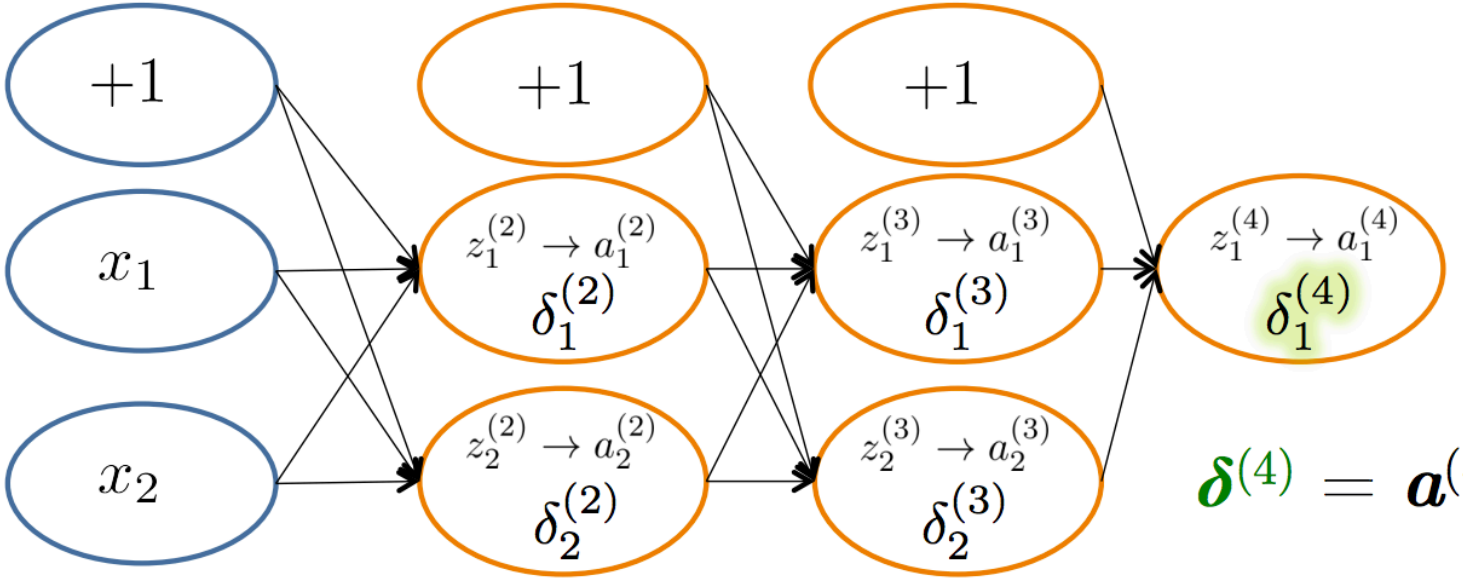
$$h_\Theta \in \mathbb{R}^K \qquad (h_\Theta(\mathbf{x}))_i = i^{th}\text{output}$$

$$J(\Theta) = -\frac{1}{n}\left[\sum_{i=1}^{n}\sum_{k=1}^{K} y_{ik}\log(h_\Theta(\mathbf{x}_i))_k + (1-y_{ik})\log\left(1-(h_\Theta(\mathbf{x}_i))_k\right)\right]$$
$$+ \frac{\lambda}{2n}\sum_{l=1}^{L-1}\sum_{i=1}^{s_{l-1}}\sum_{j=1}^{s_l}\left(\Theta_{ji}^{(l)}\right)^2$$

k^{th} class: true, predicted
not k^{th} class: true, predicted

Based on slide by Andrew Ng

Backpropagation Intuition



$$\boldsymbol{\delta}^{(4)} = \boldsymbol{a}^{(4)} - \mathbf{y}$$

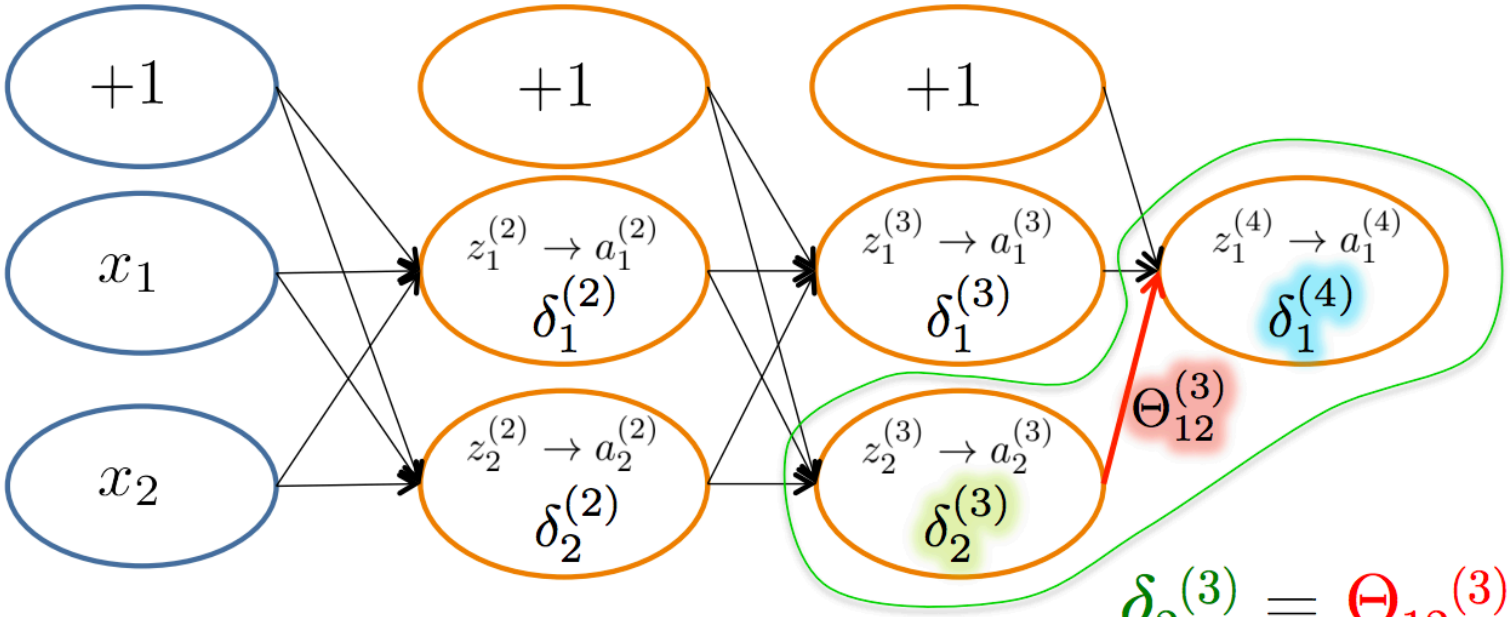
$\delta_j^{(l)}$ = "error" of node j in layer l

Formally, $\delta_j^{(l)} = \dfrac{\partial}{\partial z_j^{(l)}} \text{cost}(\mathbf{x}_i)$

$$\text{where } \text{cost}(\mathbf{x}_i) = y_i \log h_\Theta(\mathbf{x}_i) + (1 - y_i) \log(1 - h_\Theta(\mathbf{x}_i))$$

Based on slide by Andrew Ng

Backpropagation Intuition



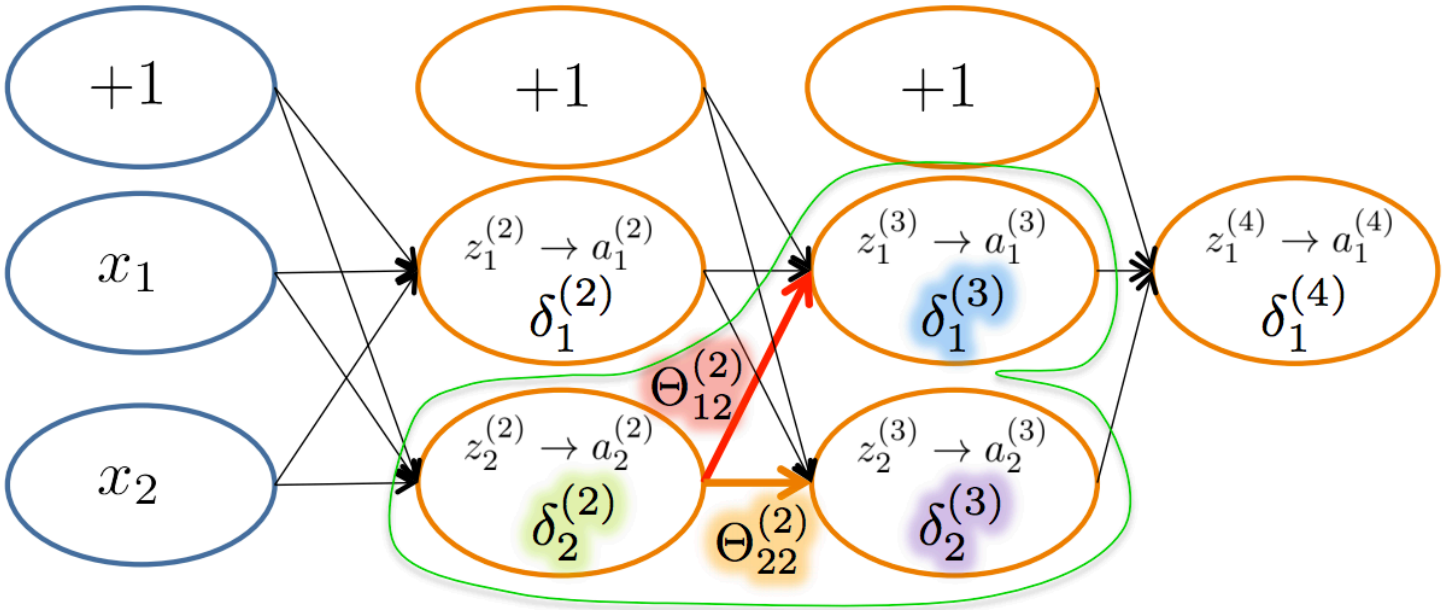
$\delta_j^{(l)}$ = "error" of node j in layer l

Formally, $\delta_j^{(l)} = \dfrac{\partial}{\partial z_j^{(l)}} \text{cost}(\mathbf{x}_i)$

$$\text{where } \text{cost}(\mathbf{x}_i) = y_i \log h_\Theta(\mathbf{x}_i) + (1 - y_i) \log(1 - h_\Theta(\mathbf{x}_i))$$

Based on slide by Andrew Ng

Backpropagation Intuition



$$\delta_2^{(2)} = \Theta_{12}^{(2)} \times \delta_1^{(3)} + \Theta_{22}^{(2)} \times \delta_2^{(3)}$$

$\delta_j^{(l)}$ = "error" of node j in layer l

Formally, $\delta_j^{(l)} = \dfrac{\partial}{\partial z_j^{(l)}} \text{cost}(\mathbf{x}_i)$

where $\text{cost}(\mathbf{x}_i) = y_i \log h_\Theta(\mathbf{x}_i) + (1 - y_i) \log(1 - h_\Theta(\mathbf{x}_i))$

Based on slide by Andrew Ng

Gradient Descent with Backprop

Given: training set $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$
Initialize all $\Theta^{(l)}$ randomly (NOT to 0!)
Loop // each iteration is called an epoch

Set $\Delta_{ij}^{(l)} = 0 \quad \forall l, i, j$ (Used to accumulate gradient)

For each training instance $(\mathbf{x}_i, y_i)$:

Set $\mathbf{a}^{(1)} = \mathbf{x}_i$

Compute $\{\mathbf{a}^{(2)}, \ldots, \mathbf{a}^{(L)}\}$ via forward propagation

Compute $\boldsymbol{\delta}^{(L)} = \mathbf{a}^{(L)} - y_i$

Compute errors $\{\boldsymbol{\delta}^{(L-1)}, \ldots, \boldsymbol{\delta}^{(2)}\}$

Compute gradients $\Delta_{ij}^{(l)} = \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

Compute avg regularized gradient $D_{ij}^{(l)} = \begin{cases} \frac{1}{n}\Delta_{ij}^{(l)} + \lambda\Theta_{ij}^{(l)} & \text{if } j \neq 0 \\ \frac{1}{n}\Delta_{ij}^{(l)} & \text{otherwise} \end{cases}$

Update weights via gradient step $\Theta_{ij}^{(l)} = \Theta_{ij}^{(l)} - \alpha D_{ij}^{(l)}$

Until weights converge or max #epochs is reached

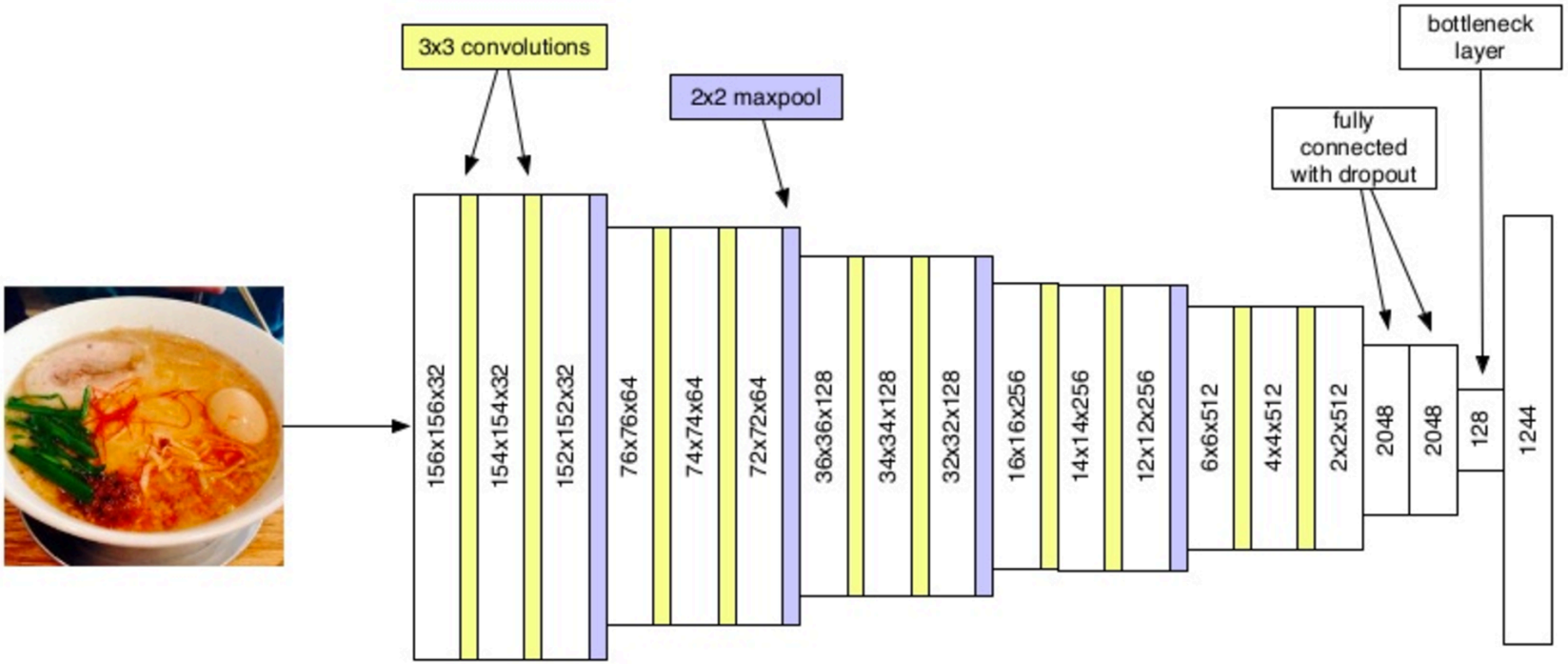
Backpropagation

Based on slide by Andrew Ng

Neural Embeddings

Representation Learning

- Deep model trained on a GPU on 6M random pics downloaded from Yelp



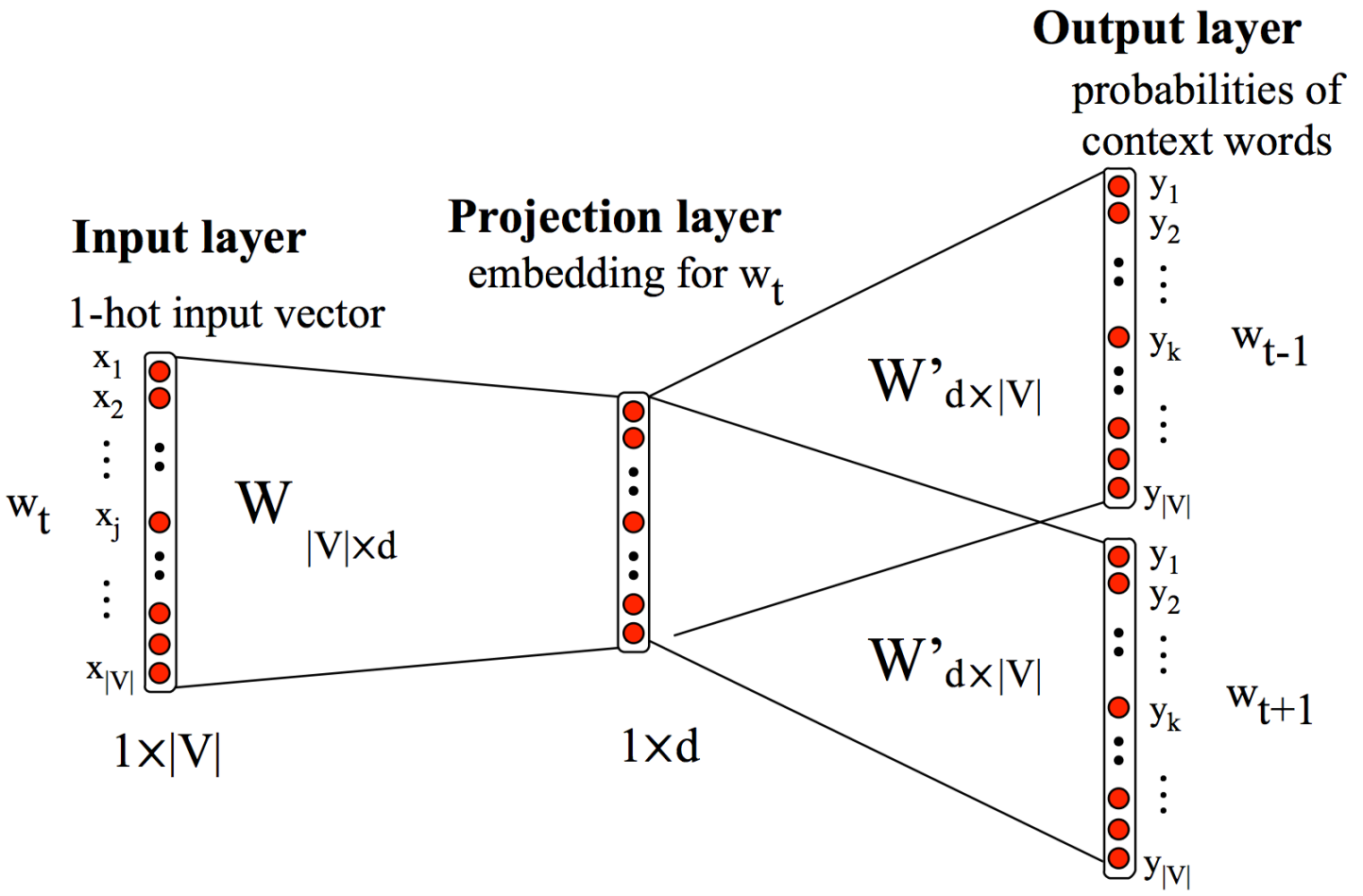
Source: Bernhardsson. (2015) “Approximate nearest neighbor methods and vector models.”

What is word2vec?

- word2vec is **not** a single algorithm
- It is a **software package** for representing words as vectors, containing:
  - Two distinct models
    - Skip-Gram
    - CBOW
  - Various training methods
    - Negative Sampling
    - Hierarchical Softmax
  - A rich preprocessing pipeline
    - Dynamic Context Windows
    - Subsampling
    - Deleting Rare Words
- C.f. GloVe

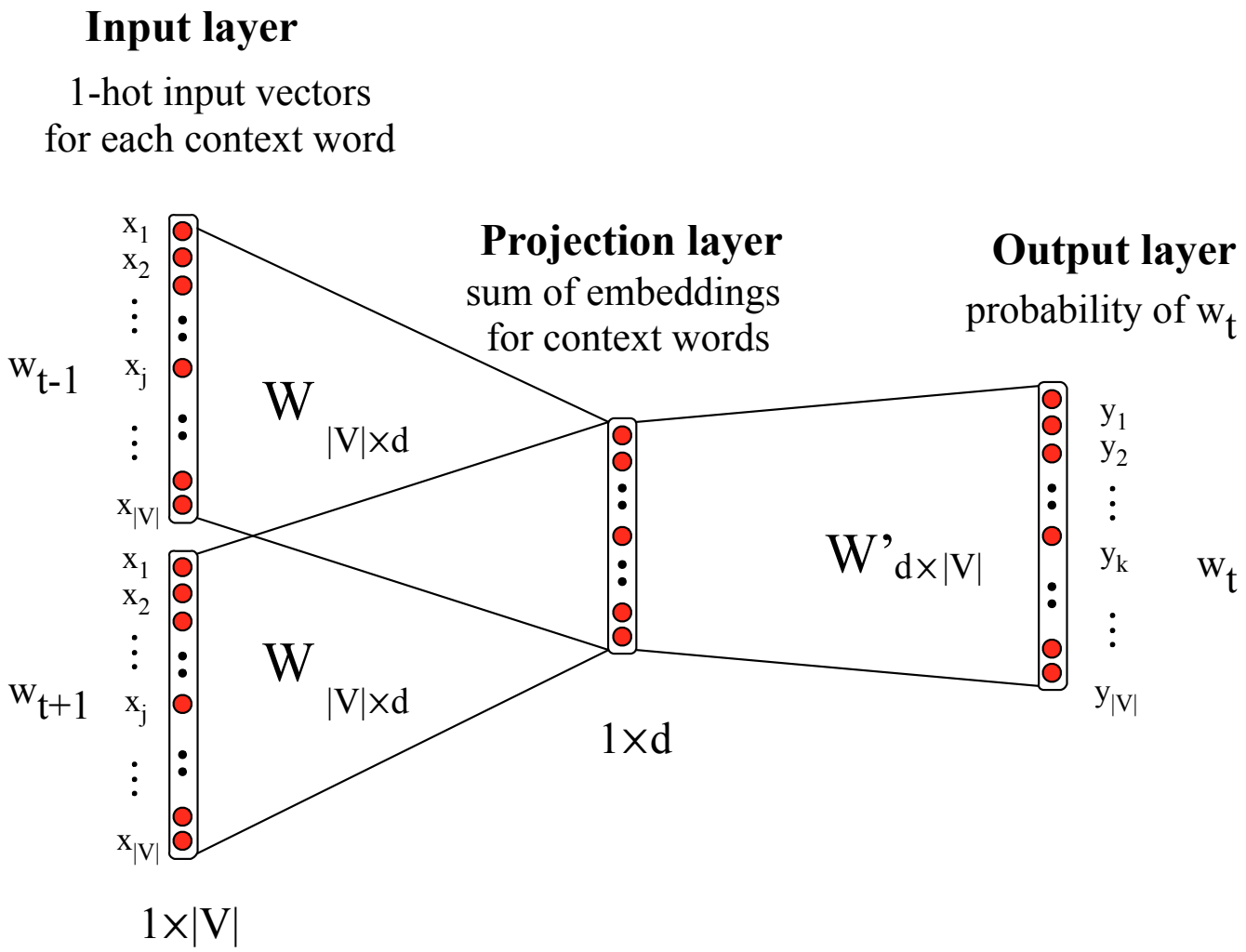
Source: Levy et al. (2015) “Improving Distributional Similarity with Lessons Learned from Word Embeddings.”

Skip-gram



Source: Manning et al. CS276 - Information Retrieval and Web Search, Stanford Spring 2016

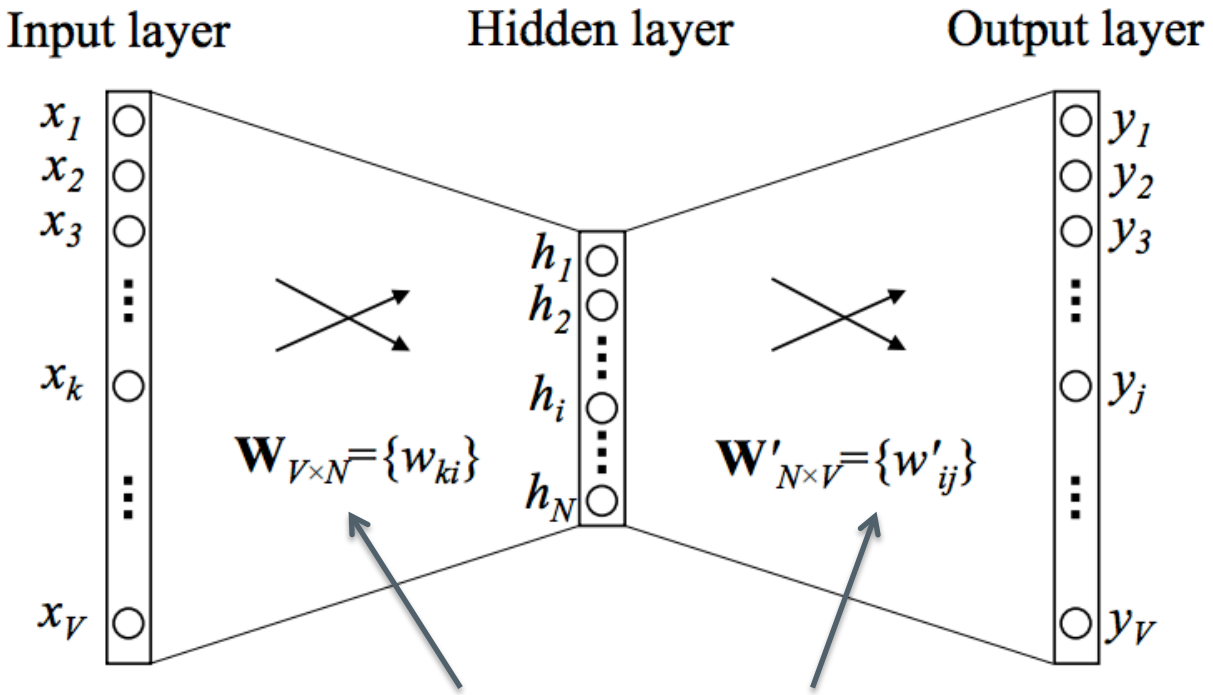
Continuous Bag of Words (CBOW)



Source: Manning et al. CS276 - Information Retrieval and Web Search, Stanford Spring 2016

CBOW with 1-Word Context

One-hot vector



Attempts to predict one-hot vector

These matrices have word vectors!

Source: Manning et al. CS276 - Information Retrieval and Web Search, Stanford Spring 2016

Word2vec Training

- Start with small, random vectors for words
- Iteratively go through millions of words in contexts
  - Work out prediction, work out error
  - Backpropagate error to update word vectors
  - Repeat
- Result is dense vectors for all words

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

Source: Manning et al. CS276 - Information Retrieval and Web Search, Stanford Spring 2016

Word2vec Linear Relationships

- These representations are *very good* at encoding **similarity** and **dimensions of similarity**!
- Analogies testing dimensions of similarity can be solved quite well just by doing vector subtraction in the embedding space

  Syntactically

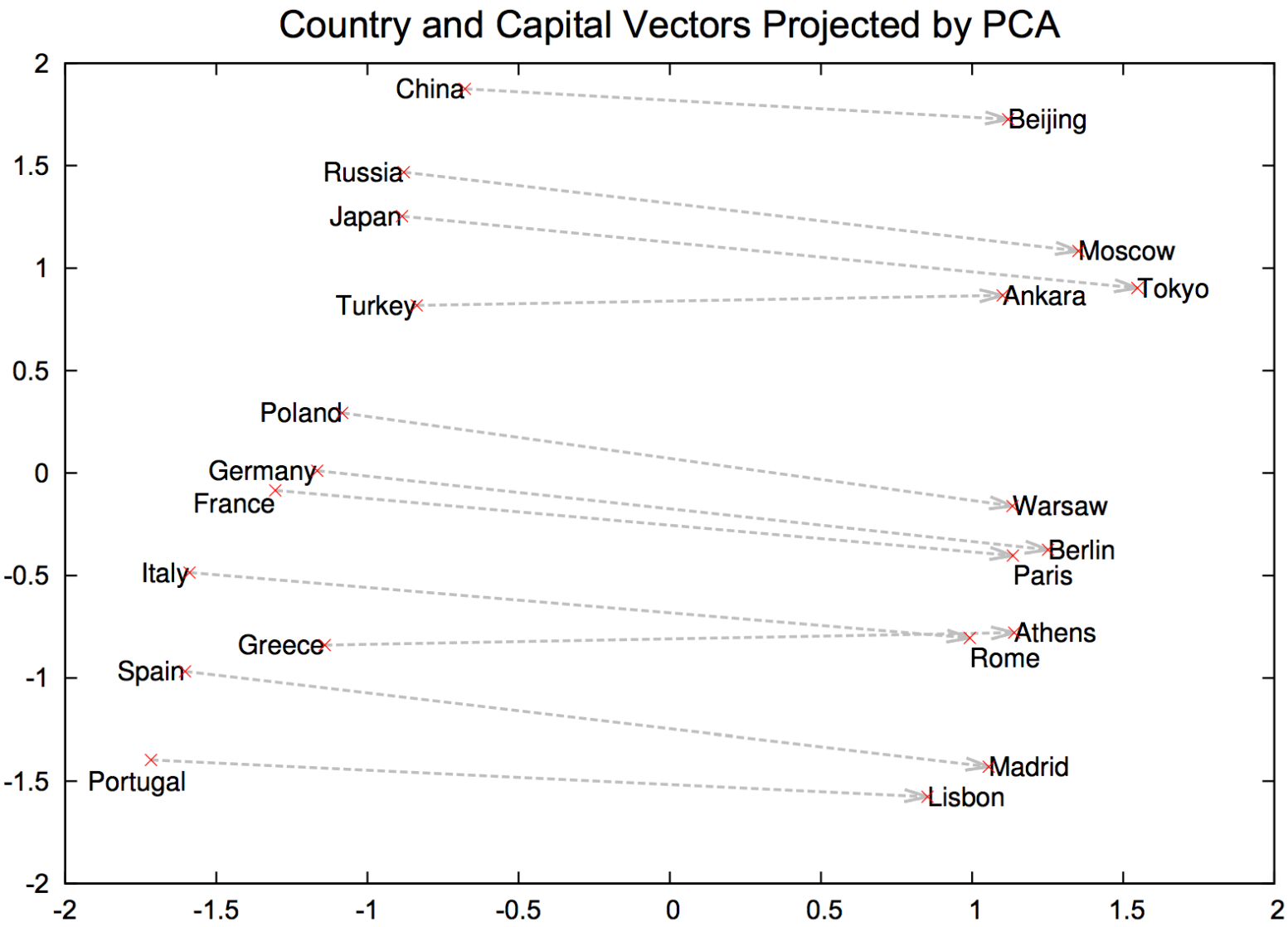
  - $x_{apple} - x_{apples} \approx x_{car} - x_{cars} \approx x_{family} - x_{families}$
  - Similarly for verb and adjective morphological forms

  Semantically (Semeval 2012 task 2)

  - $x_{shirt} - x_{clothing} \approx x_{chair} - x_{furniture}$
  - $x_{king} - x_{man} \approx x_{queen} - x_{woman}$

Source: Manning et al. CS276 - Information Retrieval and Web Search, Stanford Spring 2016

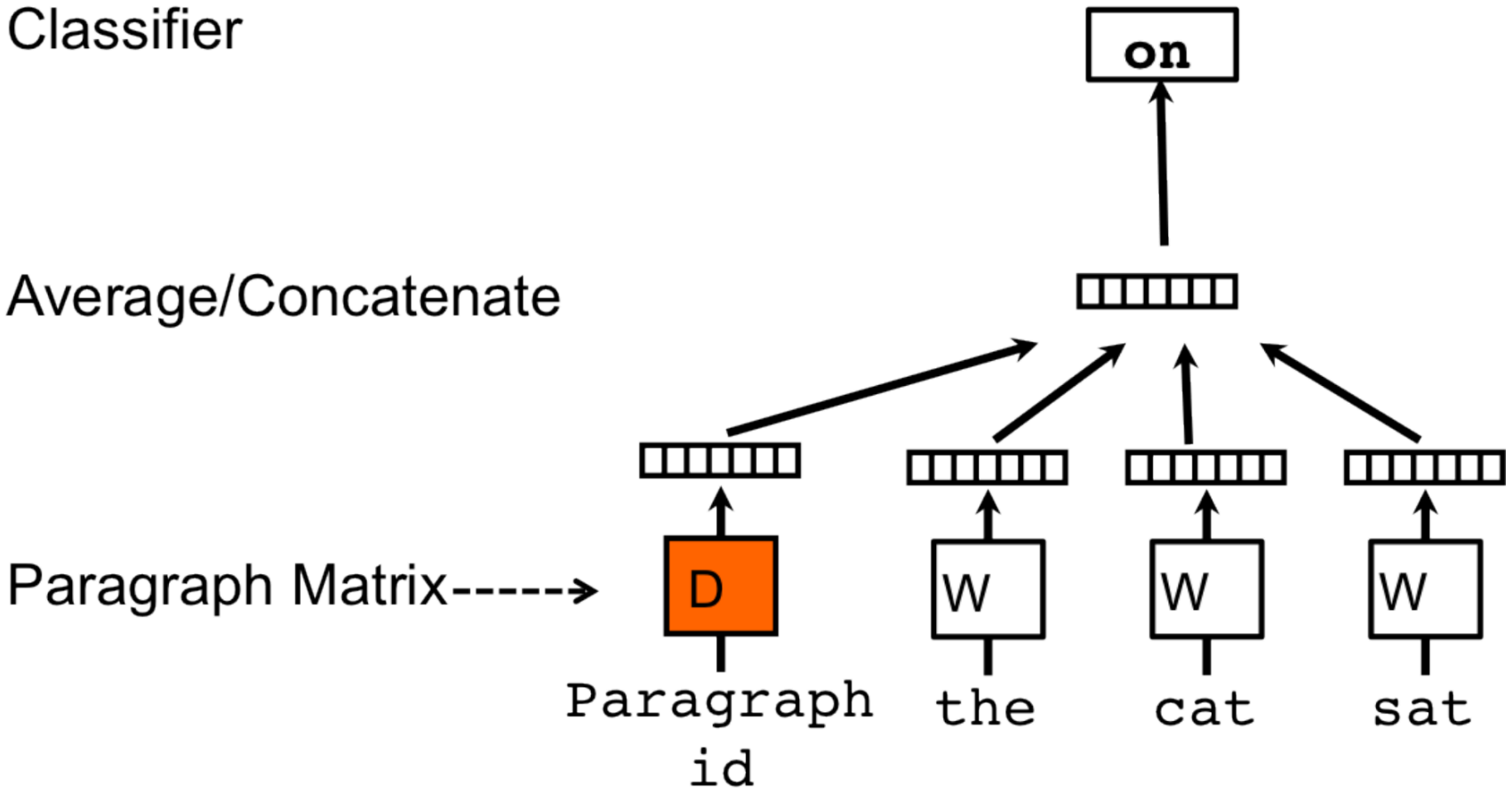
Word2vec Linear Relationships



Source: Mikolov et al. (2013) "Distributed Representations of Words and Phrases and their Compositionality." *NIPS.*

Doc2vec

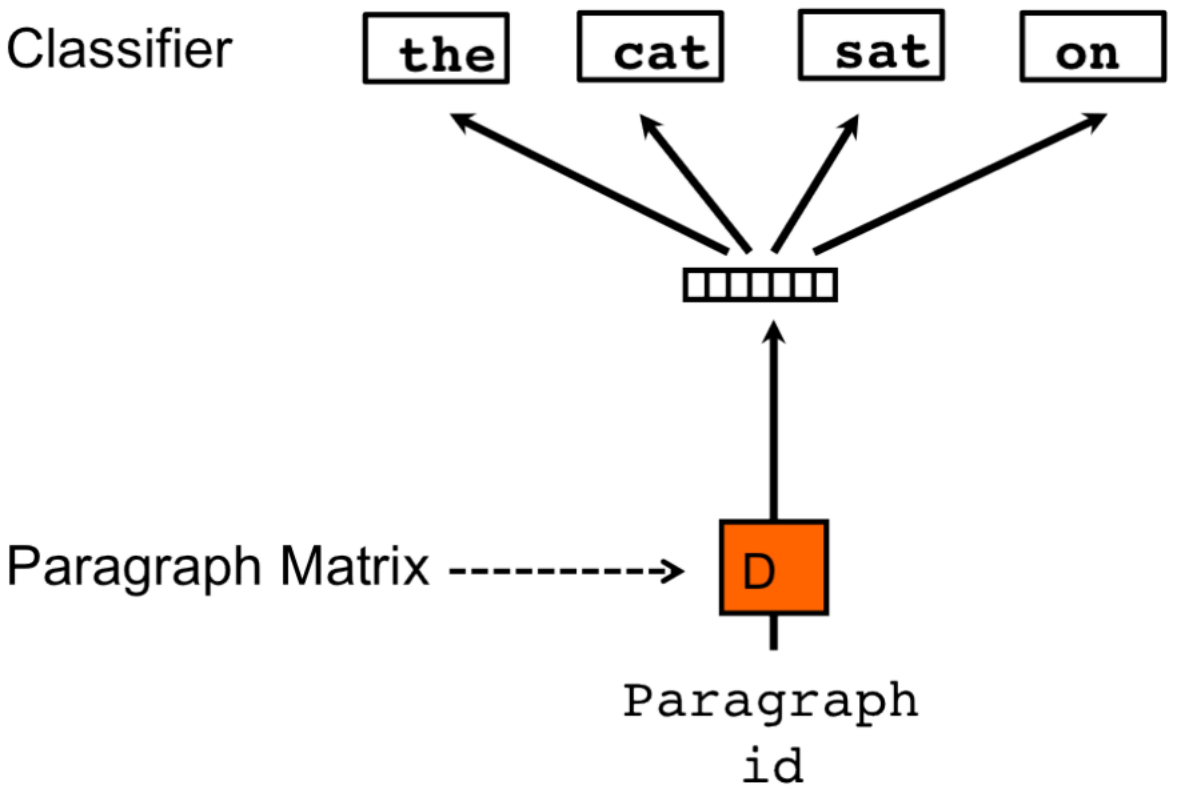
Distributed Memory



Source: Le, Mikolov (2014) “Distributed Representations of Sentences and Documents.” *ICML.*

Doc2vec

Distributed Bag of Words



Source: Le, Mikolov (2014) “Distributed Representations of Sentences and Documents.” *ICML.*

Doc2vec > Bag of Words

- Semantic similarity
- Takes into account word order within context
  - (Distributed Memory only)
  - Similar to bag-of-n-grams, but dense

Source: Le, Mikolov (2014) “Distributed Representations of Sentences and Documents.” *ICML.*

SVD Reduction

- Levy and Goldberg demonstrated skip-gram with negative sampling reduces to SVD on "SPPMI" matrix
  - SPMI is PMI shifted by constant log(k)
    - k := number of negative samples
  - SPPMI is positive SPMI
    - max(0, SPMI)
- Demonstrated near equivalence among many embedding methods
  - (Given the right hyperparameters)
- Also refuted a bunch of the GloVe claims…

Source: Levy et al. (2015) "Improving Distributional Similarity with Lessons Learned from Word Embeddings."

Questions?