Word Embeddings & Intro to NNs for NLP

UMass CS 585 11/15/16 Brendan O'Connor

Last Thursday: Distributional Similarity

Word Embeddings extend the dist. sim. idea and give a foundation for neural network approaches to NLP

Word Embeddings

Sparse Context Vector (10 million+ dimensional):

$$V_i = [0, 1, 0, 0, 0, 4, 0, 0, 0, 2, 0, 0, 1, \ldots]$$

[This can be directly used, but maybe too slow, sparse]

Instead represent every word type as a low-dimensional dense vector (about 100 dimensional).

$$E_i = [.253, 458, 4.56, 78.5, 120, \ldots]$$

These don't come directly from the data. They need to be learned.

Slide: David Belanger

Matrix factorization

SVD ... PCA

Factor Analysis

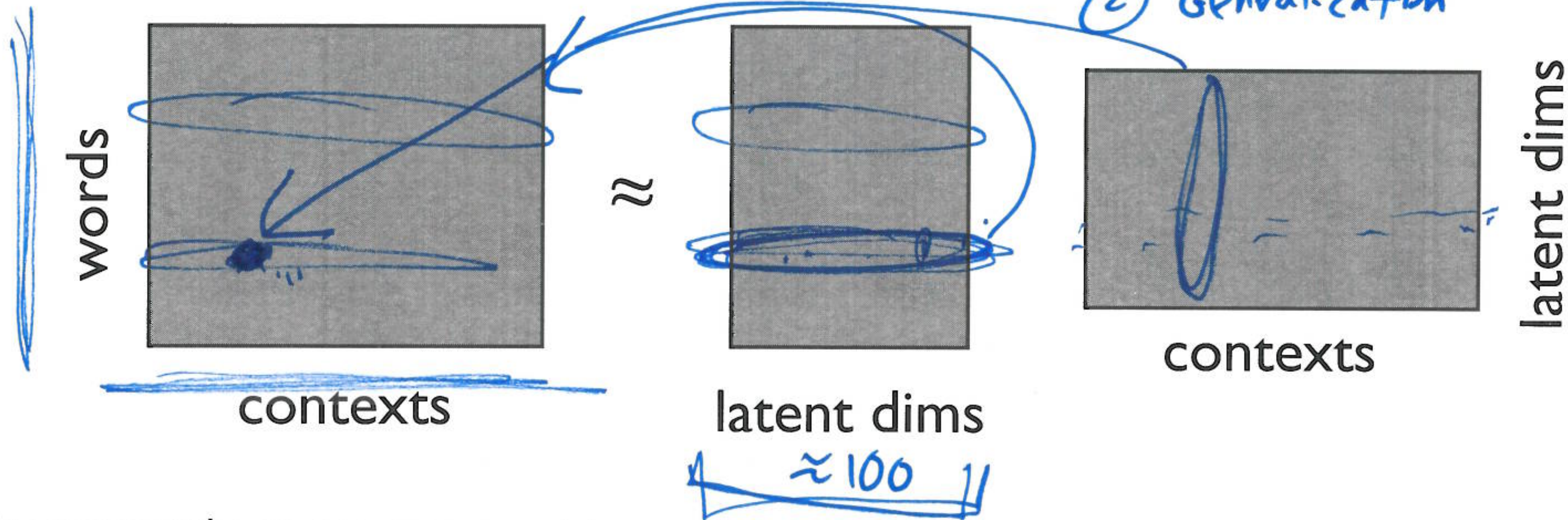
$$V \approx E \quad B$$

(counts) (embeddings)

why?

(1) Make smaller

(2) Generalization



Reconstruct the co-occurrence matrix

$$V_{i,c} \approx \sum_{k \in 1..100} E_{i,k} B_{k,c}$$

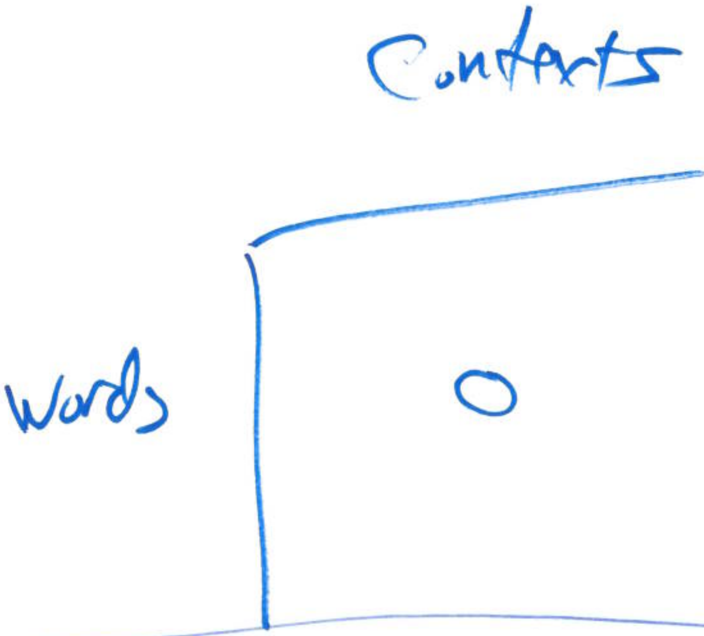
Preserve pairwise distances between words i, j

$$V_i^\top V_j \approx E_i^\top E_j$$

Singular Value Decomposition learns E,B
(or other matrix factorization techniques)

Eigen Decomposition learns E

Slide: David Belanger



$V_{i,c}$?

$V_{i,c} = \text{count}(i, c)$

TF-IDF

Alternate method to upweight globally-rare contexts/words

$\boxed{\text{PMI}}$ $V_{i,c} = \log \frac{P(i,c)}{P(i)\,P(c)}$

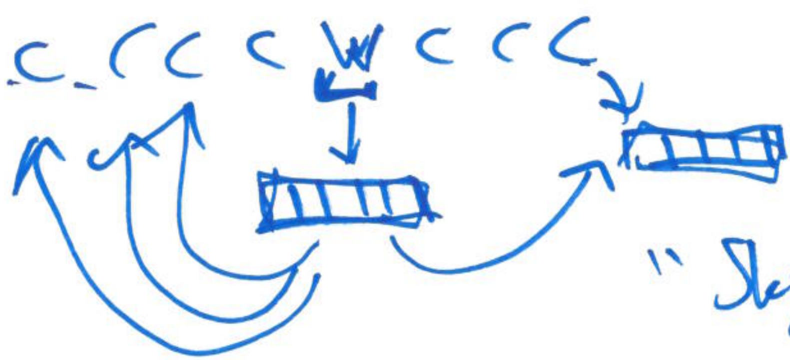
$= \log \frac{P(c \mid i)}{P(c)}$

$\boxed{\text{PPMI}}$ $V_{i,c} = \max(\text{PMI}, 0)$

Trained word embeddings

- Most commonly used methods jointly train word embeddings and context embeddings, to predict contexts from words
  - Captures information about word's distribution
  - *[Levy and Goldberg, 2014]*
- Commonly used software/models/pretrained embeddings
  - *word2vec*
  - *GloVe*
  - Practically speaking: extremely good training speed

=> Lexical Resource



c c c W c c c c

"Skip-gram Model"

Using Word Embeddings

man : woman :: king : ____

- "Intrinsic Eval"
- Extrinsic Eval: use embeddings in a system

- Use as features

"cat" → [0 0000 | 1 | 0000 | 1 | 0000] ← Feat. Engr.

"cat"   "_t"

Concat!

"cat" → [4.5 | -2.1 | ....] E_{cat}

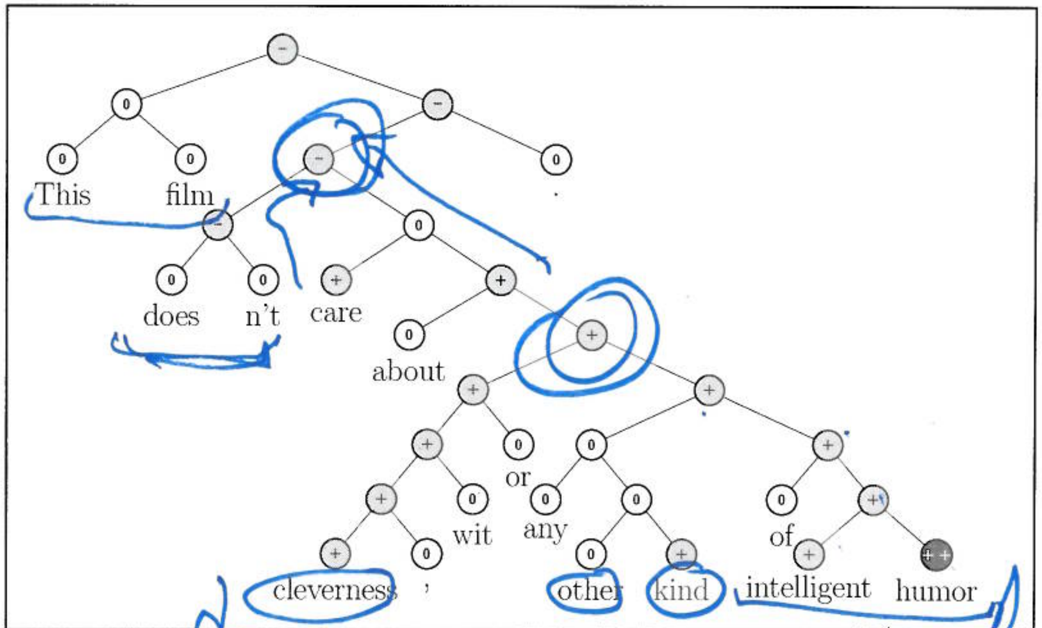
Pro: Generalize!

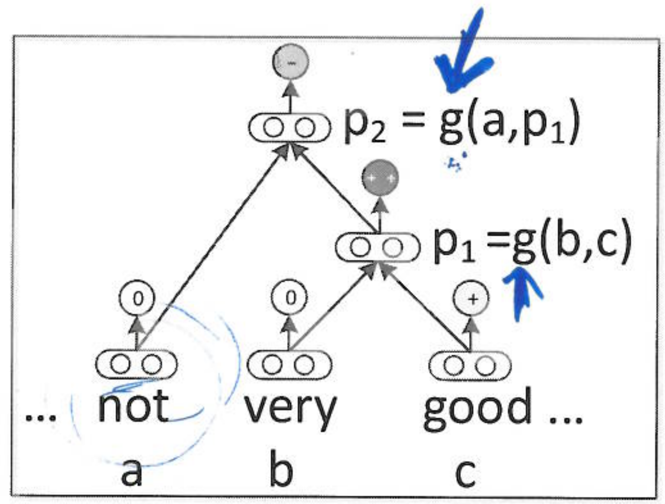
Con: Not specific

Why neural networks?

- Easily incorporate continuous-valued features
  - (If you like word embeddings...)
  - or, new way to learn word embeddings
- Automatically learn features
- Easier to train/deploy than alternative non-linear models
  - Keras / Tensorflow / Torch / etc.
  - DyNet ... etc. etc. etc.
- Current research: capable of modeling complicated language phenomena?

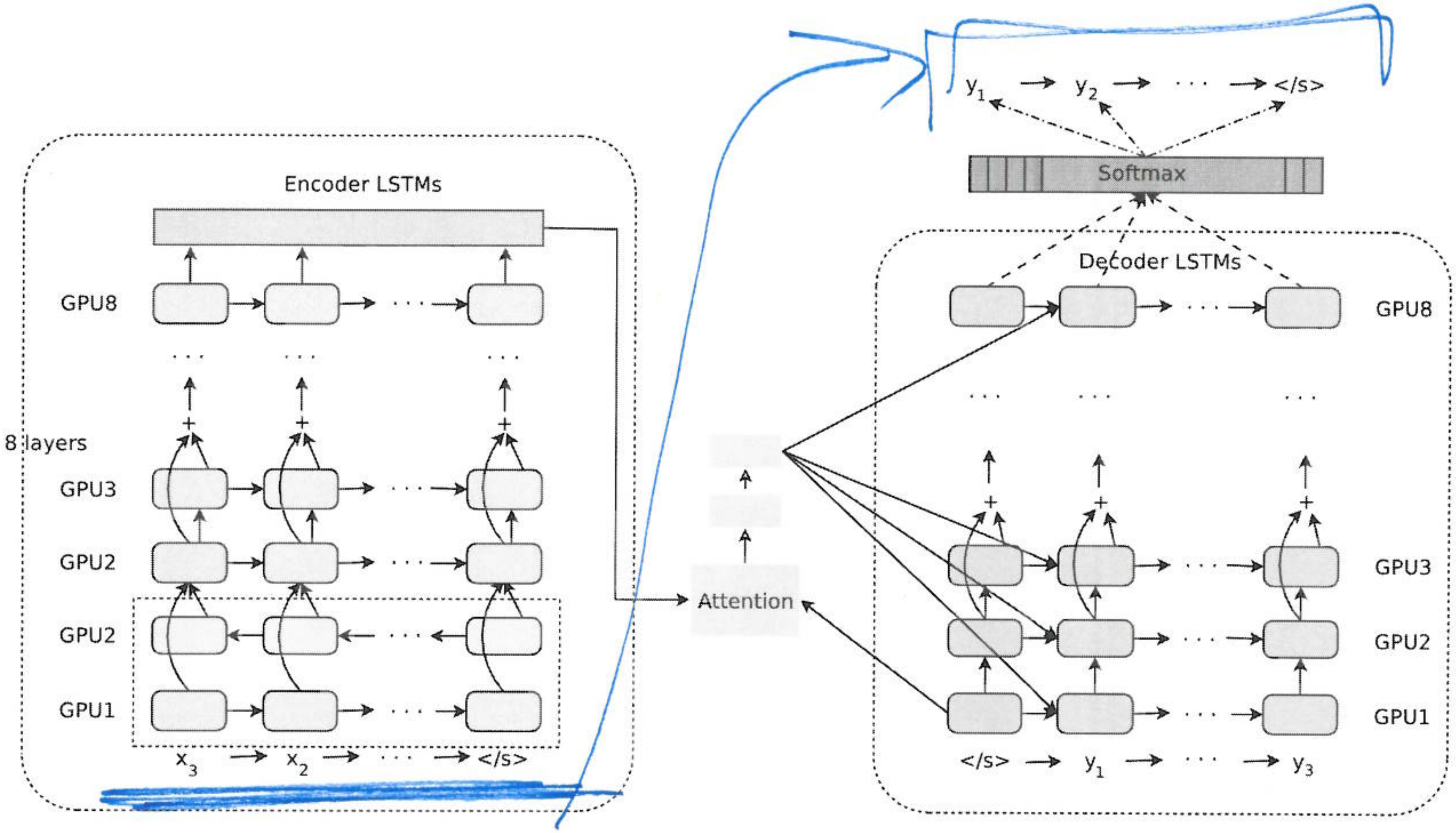
Example: sentiment composition





[Socher et al., EMNLP 2013]

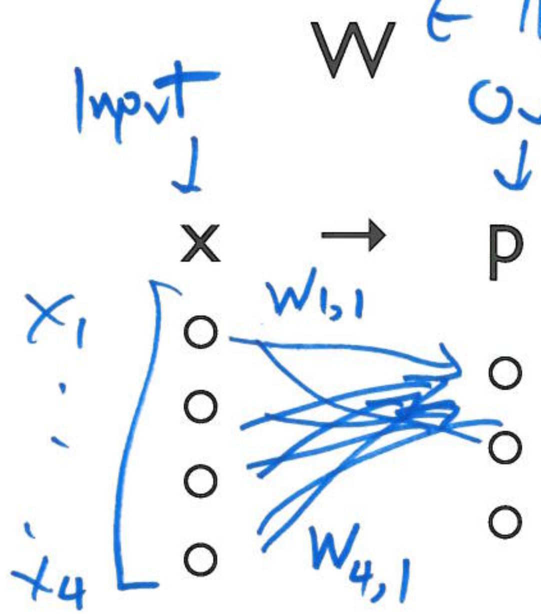
Example: machine translation



[Google's Neural MT system: Wu et al., 2016]

Multi. Log.Reg. as NN

$W \in \mathbb{R}^{I \times K}$

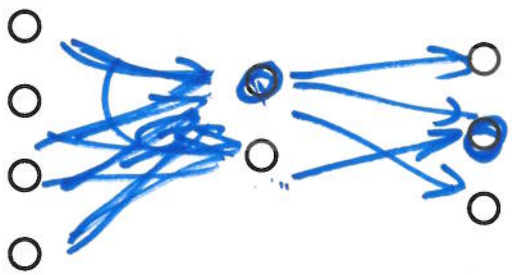


$$P(y=k|x) \equiv p_k = \frac{e^{\sum_i x_i w_{ik}}}{\sum_{k'} p_{k'}}$$

Feedforward NN

$W^{(1)} \quad W^{(2)}$

x → h → p



"tanh"

$$h_j = \tanh(x' W^{(1)}_{\cdot,j})$$

$$p_k = \frac{e^{\sum_j h_j W^{(2)}_{j,k}}}{Z}$$

$$x \to h_1 \to h_2 \to h_3 \to p$$

Fun!

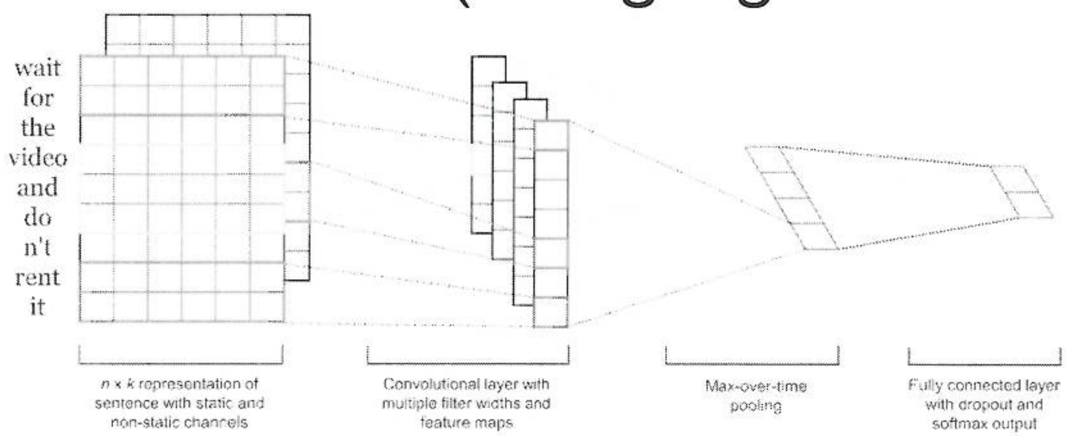
Demo: http://playground.tensorflow.org

Common NN architectures in NLP

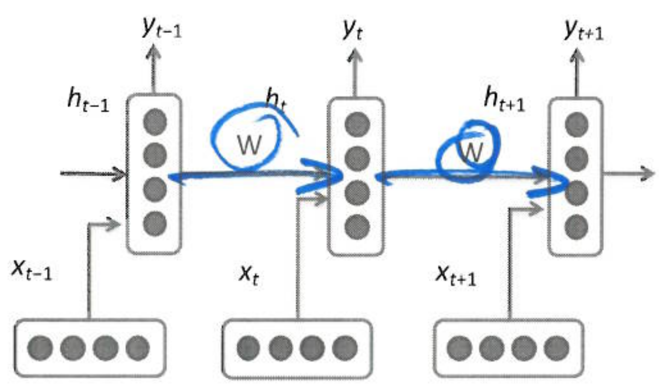
Tie weights at each step or window

[Fairly easy to experiment with architectures]

- Convolutional NN (sliding n-gram window)



- Recurrent NN (hidden state sequence)



The secret sauce (sometimes) is the unsupervised word vector pre-training on a large text collection

	POS WSJ (acc.)	NER CoNLL (F1)
State-of-the-art*	97.24	89.31
Supervised NN	96.37	81.47
Word vector pre-training followed by supervised NN**	97.20	88.87
+ hand-crafted features***	97.29	89.59

* Representative systems: POS: (Toutanova et al. 2003), NER: (Ando & Zhang 2005)

** 130,000-word embedding trained on Wikipedia and Reuters with 11 word window, 100 unit hidden layer – then supervised task training

***Features are character suffixes for POS and a gazetteer for NER

[Slide: Richard Socher "cs224d" website]