ENGR-E 533

Deep Learning Systems

Module 07

Deep Embedding

Minje Kim

Department of Intelligent Systems Engineering

Email: minje@indiana.edu

Website: http://minjekim.com

Research Group: http://saige.sice.indiana.edu

Meeting Request: http://doodle.com/minje

INDIANA UNIVERSITY

SCHOOL OF INFORMATICS, COMPUTING, AND ENGINEERING

Vector Representation of Words

- Once again we seek latent representations

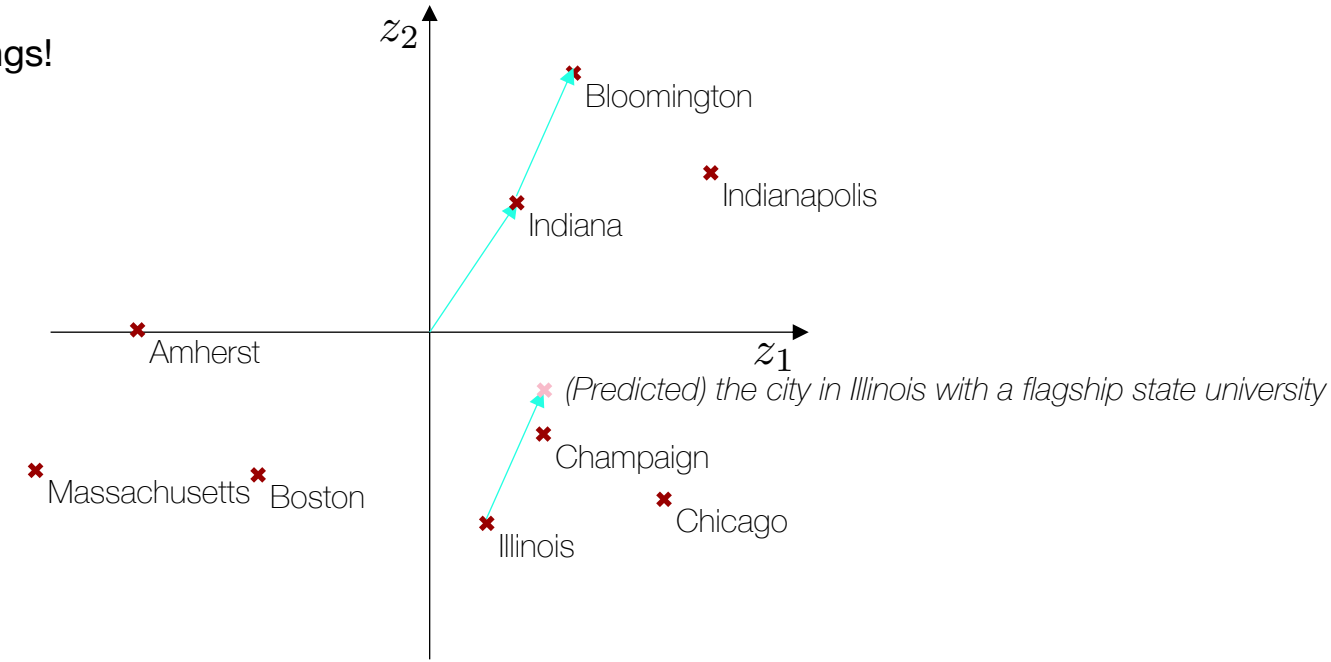
- Suppose we're interested in these words:
  - Illinois, Chicago, Champaign, Indiana, Indianapolis, Bloomington
- How do we represent them for computer algorithms to process?
  - One-hot vectors have been common
- Any problems with this?
  - Too high dimensional
    - There are about 700k words in English
  - Not really a feature vector
    - Very discriminant but that's it
    - Difficult to compare them

Illinois	1	0	0	0	0	0
	…	…	…	…	…	…
	…	…	…	…	…	…
Chicago	0	1	0	0	0	0
	…	…	…	…	…	…
	…	…	…	…	…	…
	…	…	…	…	…	…
Champaign	0	0	1	0	0	0
Indiana	0	0	0	1	0	0
	…	…	…	…	…	…
	…	…	…	…	…	…
Indianapolis	0	0	0	0	1	0
	…	…	…	…	…	…
Bloomington	0	0	0	0	0	1

Vector Representation of Words

- Once again we seek latent representations

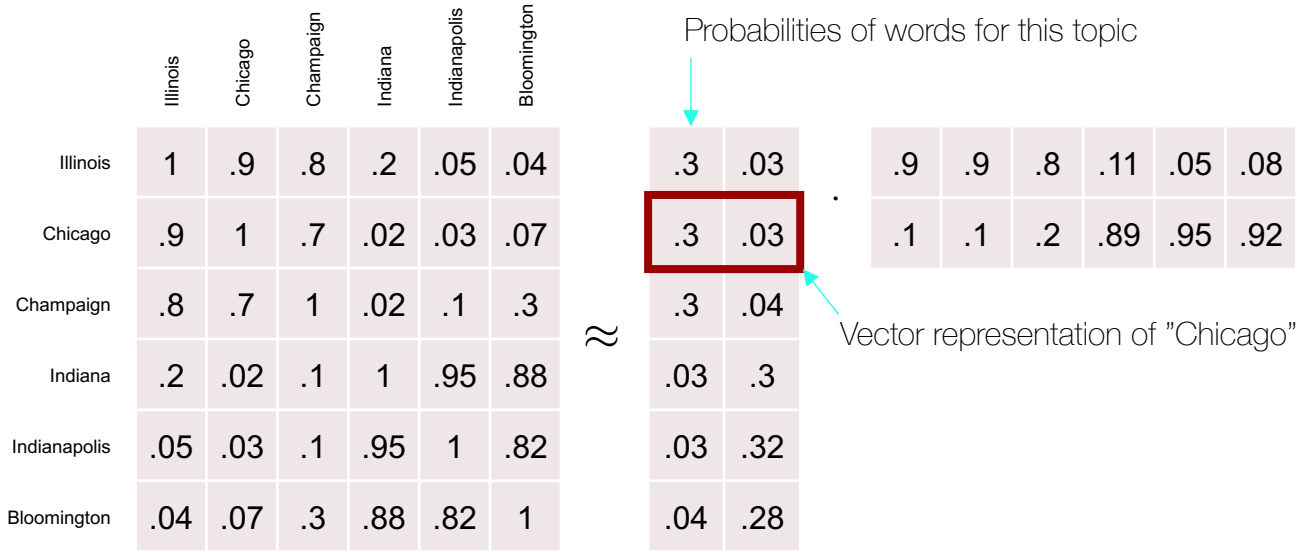
- A better latent representation
- Difference vectors have semantic meanings!



Vector Representation of Words

- A matrix factorization approach

- Similarity matrix
  - e.g. co-occurrence
- Matrix factorization
  - Eigendecomposition
  - SVD
  - NMF
  - Topic modeling
  - You name it
- Implication
  - Each word has a K-d vector representation
    - K is the low rank of your choice
  - Similar words often appear in the same topic together
  - Each word has a unique activation pattern over the latent variables



Vector Representation of Words

- A matrix factorization approach

- Same story for the Term-Frequency (TF) matrix

	d1	d2	d3	d4	…	dN
Illinois	.03	…	…	…	…	…
Chicago	.02	…	…	…	…	…
Champaign	.04	…	…	…	…	…
Indiana	.001	…	…	…	…	…
Indianapolis	.000	…	…	…	…	…
Bloomington	.000	…	…	…	…	…

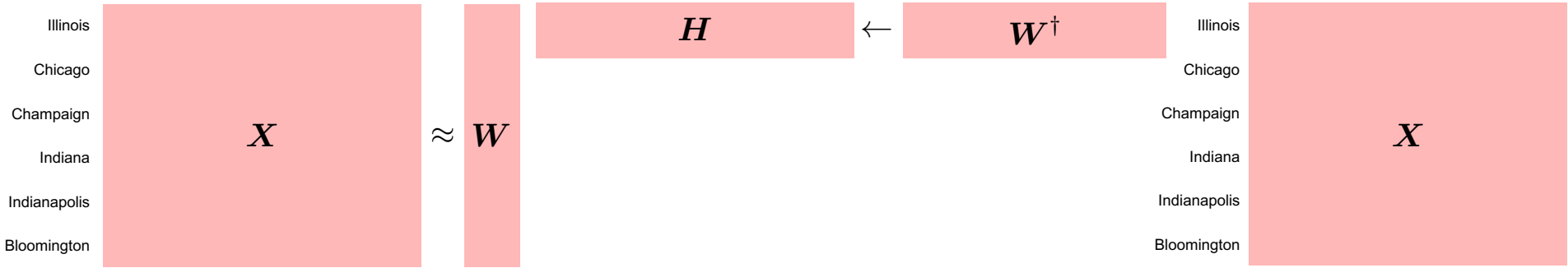
$\approx$

- An equation for this procedure $\boldsymbol{X} \approx \boldsymbol{W}\boldsymbol{H}$
- Where am I going?
  - Autoencoder

Word2vec

- The basic autoencoding structure

- The autoencoding process



$$X \approx WW^\dagger X$$

- You'll see this is just a shallow neural network
- In order to make it practical, we need to define
  - The error function
  - Input data
  - Targets
  - Nonlinearity

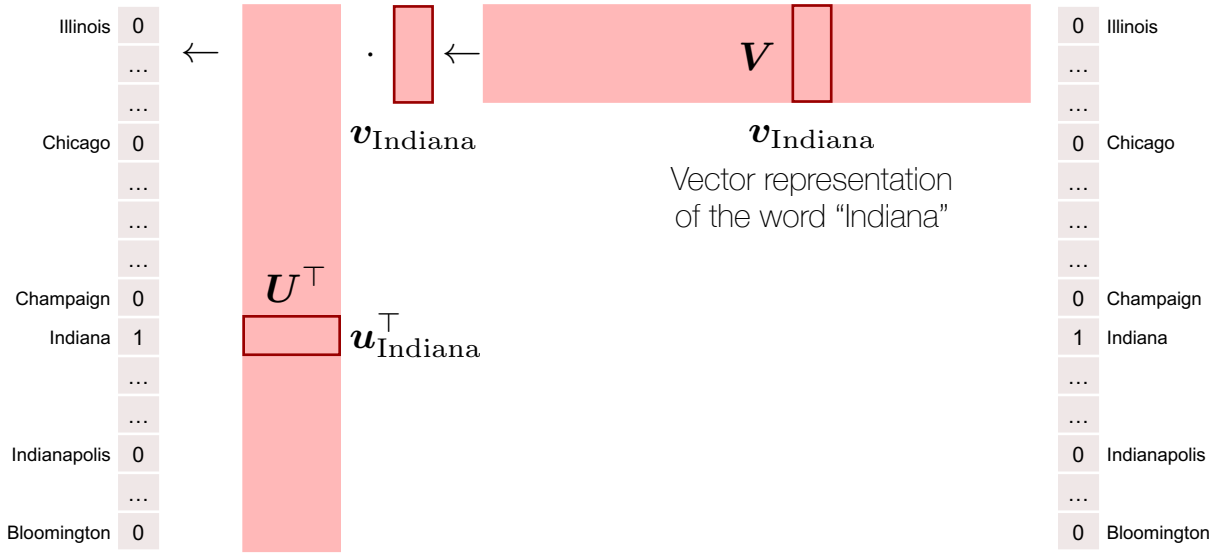
Word2vec

- Skip-gram

- Basic autoencoding network
- Activation function for the output
  - Softmax

$$p(w_t|w_t;\boldsymbol{V},\boldsymbol{U}) = \frac{\exp(\boldsymbol{u}_t^\top \boldsymbol{v}_t)}{\sum_{t'}\exp(\boldsymbol{u}_{t'}^\top \boldsymbol{v}_t)}$$

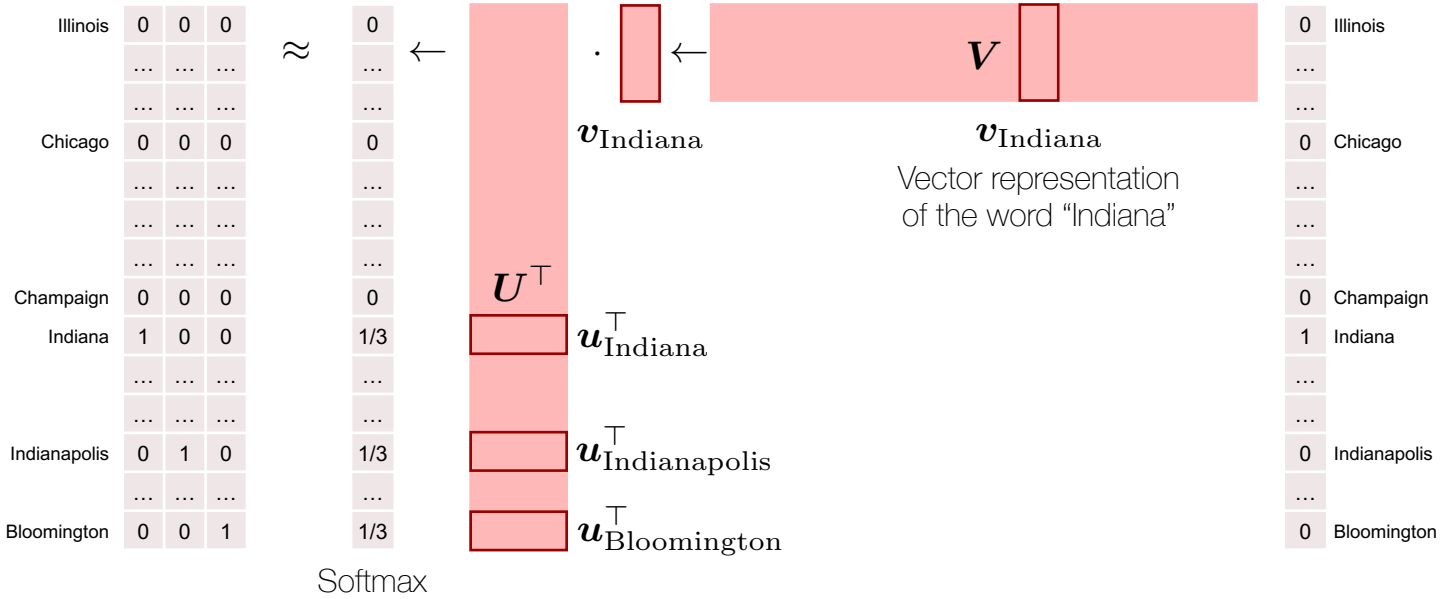
- If this autoencoding job is successful
  - $\boldsymbol{v}_{\text{Indiana}}$ is totally different from any other vectors in $\boldsymbol{U}$, but $\boldsymbol{u}_{\text{Indiana}}$
- Is this good enough? What's the problem with this?
  - Is "Indiana" closer to "Indianapolis" than to "Illinois"?



Word2vec

- Skip-gram

- Another prediction mechanism that encodes co-occurunce



- If we revert this order, it's called continuous bag-of-words (CBOW)

Word2vec

- Negative sampling

- Average negative log-likelihood

$$\mathcal{J}(\boldsymbol{U}, \boldsymbol{V}) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{|\tau|\leq m, \tau\neq 0} \log p(w_{t+\tau}|w_t)$$

$$= -\frac{1}{T}\sum_{t=1}^{T}\sum_{|\tau|\leq m, \tau\neq 0} \log \frac{\exp(\boldsymbol{u}_{t+\tau}^{\top}\boldsymbol{v}_t)}{\sum_{t'}\exp(\boldsymbol{u}_{t'}^{\top}\boldsymbol{v}_t)}$$

context window

except for the input word itself

  - For a given input word word2vec predicts its surrounding words
  - Note that word2vec doesn't predict the input word

- Any problem?
  - Denominator involves all words

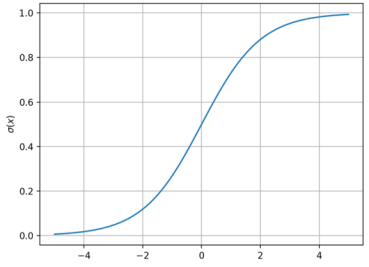
- Negative sampling
  - Word-wise binary classification

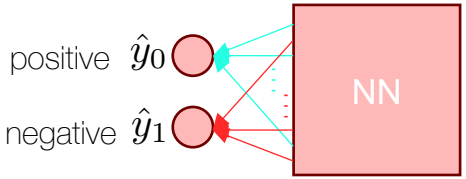
[The word of interest] versus [A randomly sampled word outside of the context]

Word2vec

- Negative sampling

- Suppose softmax with two classes
  - Or logistic regression $\sigma(\boldsymbol{w}^\top \boldsymbol{x}) = \dfrac{1}{1+\exp(-\boldsymbol{w}^\top \boldsymbol{x})}$
  - Loss function?
    - Cross entropy $\mathcal{E}\big(y||\sigma(\boldsymbol{w}^\top \boldsymbol{x})\big) = -y \log \sigma(\boldsymbol{w}^\top \boldsymbol{x})$





- A redundant representation $\hat{y}_i = \dfrac{\exp(\boldsymbol{w}_i^\top \boldsymbol{x})}{\sum_{j=0}^{1} \exp(\boldsymbol{w}_j^\top \boldsymbol{x})}$
  - Cross entropy? $\mathcal{E}(\boldsymbol{y}||\hat{\boldsymbol{y}}) = -\sum_i y_i \log \hat{y}_i \;\; = -y_0 \log \hat{y}_0 - y_1 \log \hat{y}_1$
- Imagine a training set with only positive examples $\mathcal{E}(\boldsymbol{y}||\hat{\boldsymbol{y}}) = -\log \hat{y}_0 \;\; \because y_1 = 0$ for all examples
  - Trivial solution: $\boldsymbol{w}_0 = \infty$
  - But we want $\hat{y}_1$ to be small (e.g. close to zero). How? $\mathcal{E}(\boldsymbol{y}||\hat{\boldsymbol{y}}) = -\log \hat{y}_0 - \log(1-\hat{y}_1)$
- Then, you can maximize this instead $\log \sigma(\boldsymbol{u}_{t+\tau}^\top \boldsymbol{v}_t) \;\; + \log\big(1-\sigma(\boldsymbol{u}_j^\top \boldsymbol{v}_t)\big)$

  Prob. of predicting one of the context words — Prob. of predicting a negative word
- Eventually $\log p(w_{t+\tau}|w_t) \approx \log \sigma(\boldsymbol{u}_{t+\tau}^\top \boldsymbol{v}_t) + \sum_{j \in P(w)} \big(\log \sigma(-\boldsymbol{u}_j^\top \boldsymbol{v}_t)\big)$

Word2vec and Beyond

- GloVe

- Word2vec models we've learned so far define context locally

$$\mathcal{J}(\boldsymbol{U}, \boldsymbol{V}) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{|\tau|\leq m, \tau\neq 0} \log p(w_{t+\tau}|w_t)$$

Small context window

  - How do we take a global context into account?
    - Using co-occurrence matrix

$$\mathcal{J}(\boldsymbol{U}, \boldsymbol{V}) = \frac{1}{2}\sum_{i,j}^{\mathcal{W}} f(P_{i,j})(\boldsymbol{u}_i^\top \boldsymbol{v}_j - \log P_{i,j})^2$$

Weighting function — Co-occurrence

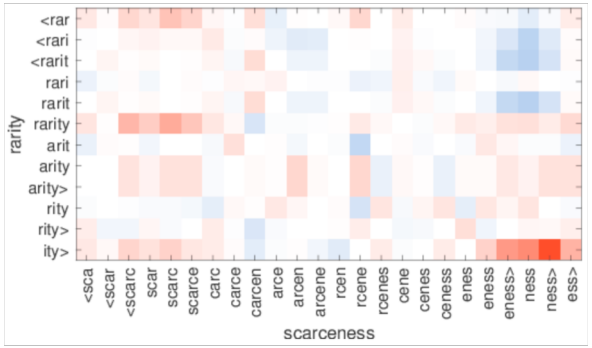
  - Doesn't always necessarily mean GloVe is better than word2vec

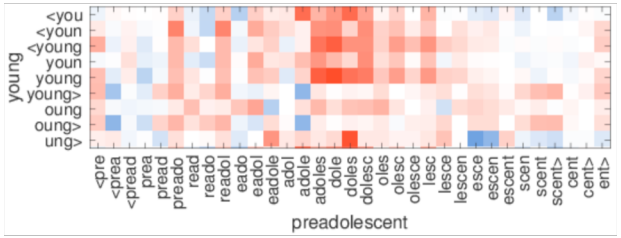
Jeffrey, Pennington, Richard Socher, and Christopher Manning. "Glove: Global vectors for word representation." *EMNLP 2014*

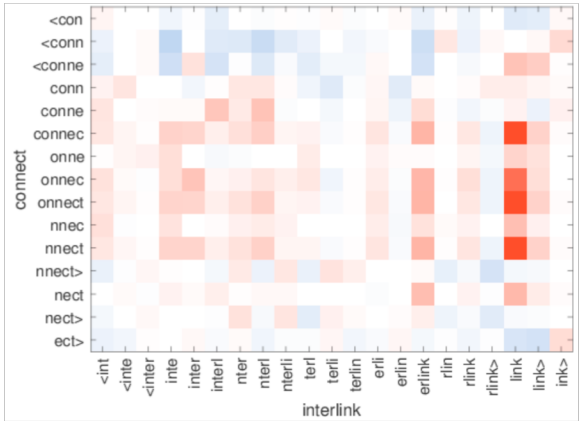
Word2vec and Beyond

- FastText

- Based on character n-grams instead of word-by-word
- N-grams?
  - **Original sentence:** I like the new chicken place on third street
  - **Bigrams:** (I like), (like the), (the new), (new chicken), (chicken place), (place on), (on third), (third street)
- Character n-gram (when n=3)
  - **<chicken>:** <ch, chi, hic, ick, cke, ken, en>
- Works better with rare words and out-of-vocabulary words







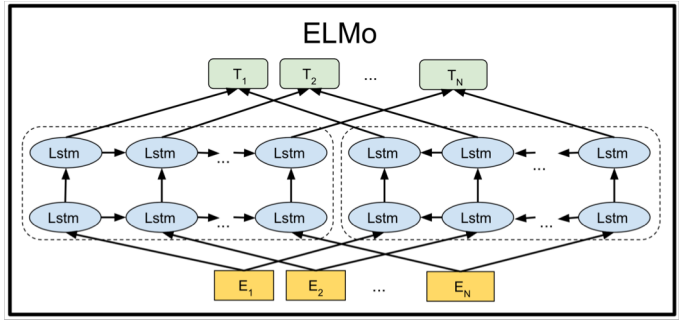
P. Bojanowski, E. Grave, A. Joulin, T. Mikolov, "Enriching Word Vectors with Subword Information," arXiv:1607.04606, 2016

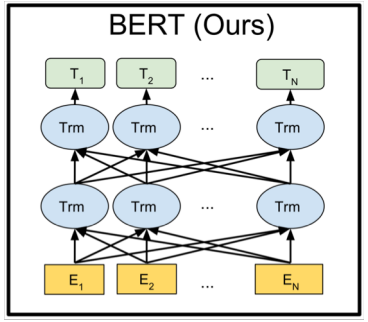
Word2vec and Beyond

- ELMo and BERT

- "The complete meaning of a word is always contextual, and no study of meaning apart from context can be taken seriously." J. R. Firth, 1935
- Chicken can mean something different
  - I like the new chicken place on third street
  - I played chicken at dark night

  In word2vec and GloVe these share same vector representation
- ELMo: an RNN-based approach
- BERT: Transformer-based approach (self attention)





Masking trick for fairer training:

"I like the new *[mask]* place on third street"

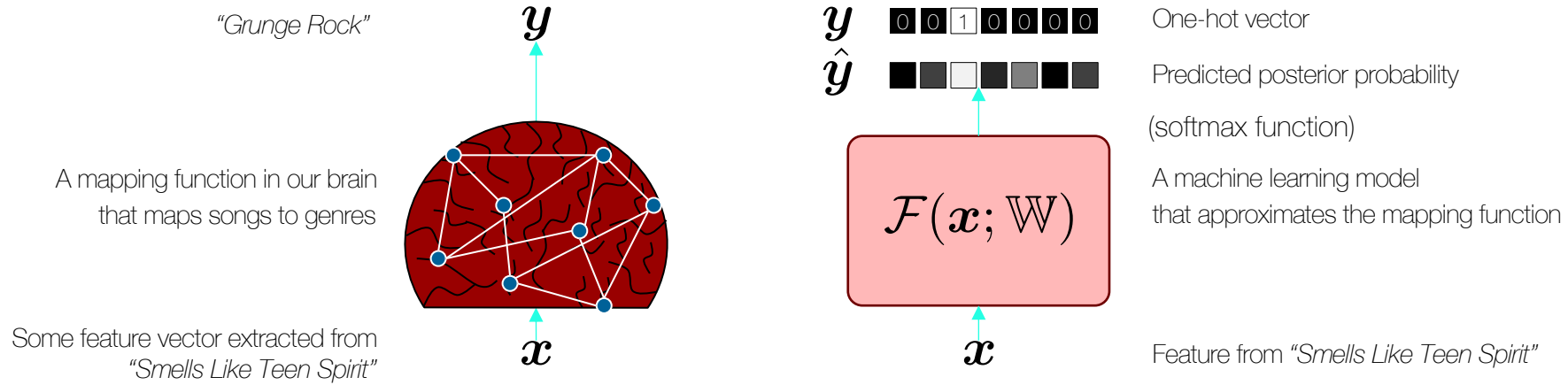
J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," arXiv:1810.04805, 2018

Reading

- Word2vec
  - Mikolov, Tomas, et al. "Efficient estimation of word representations in vector space." *arXiv preprint arXiv:1301.3781* (2013).
  - Mikolov, Tomas, et al. "Distributed representations of words and phrases and their compositionality." *Advances in neural information processing systems*. 2013.
- https://www.tensorflow.org/tutorials/word2vec
- https://fasttext.cc

Not Everything Is about Classification

- Discriminative classification models using softmax



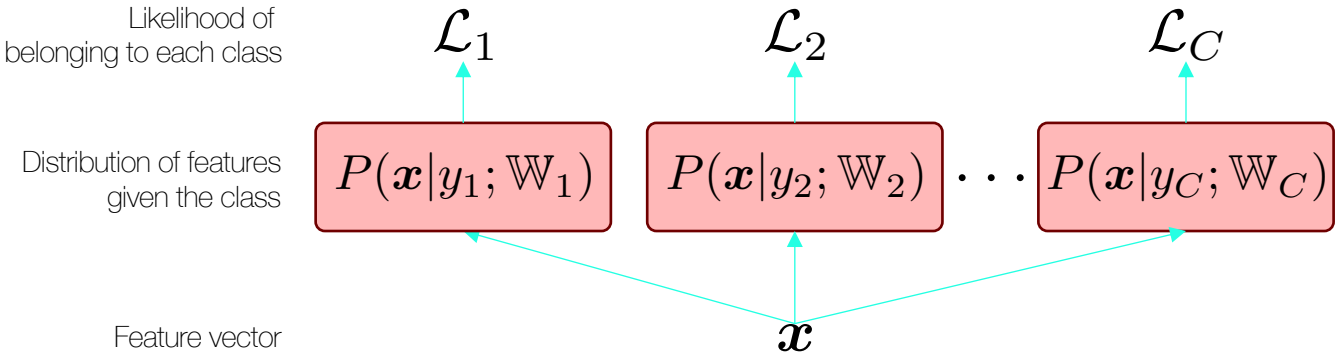
- The training algorithm updates $\mathbb{W}$ to minimize the error between the one-hot vector and prediction

$$\mathcal{E}(\boldsymbol{y}||\hat{\boldsymbol{y}}) = \text{CrossEntropy}(\boldsymbol{y}||\hat{\boldsymbol{y}}) = -\boldsymbol{y}\log\hat{\boldsymbol{y}}$$

- Downsides
  - Classification works with set number of classes (not scalable)
  - Doesn't work well for too many classes
  - Needs a lot of data per class, which should be balanced, too

Not Everything Is about Classification

- Using generative models for classification



- Training algorithm learns class-specific generative models, parameterized by $\mathbb{W}_c$
- For each test example, we calculate all the class-specific likelihoods $\mathcal{L}_c$
- Posterior probability: $P(y_c|\boldsymbol{x}) = \dfrac{\mathcal{L}_c}{\sum_{k=1}^{C} \mathcal{L}_k}$
- Downsides
  - Each class-specific model doesn't know the other models (worse classification performance)
  - Still need many data samples per class for training

Deep Learning for Discriminant Embedding

- Information retrieval and detection detection problems

- Discriminative classifier doesn't work
  - *"Smells Like Teen Spirit"* belongs to one and the only one class, *"Smells Like Teen Spirit"*
    - Millions of classes
  - Binary classification (i.e. one-versus-all) doesn't work either
    - Dataset is way too unbalanced
- Generative models doesn't work, either
  - Each category (a song) needs many examples to learn a model
    - Only one song per category (ordinary matching)
    - A few songs are available for cover song identification, although the problem is still too difficult
- A detection problem
  - Compare the query item with a bunch of examples in the database
  - Computationally challenging
    - Needs more efficient representations (in advance for the database)
  - Needs to be robust to noisy queries
    - Feature engineering

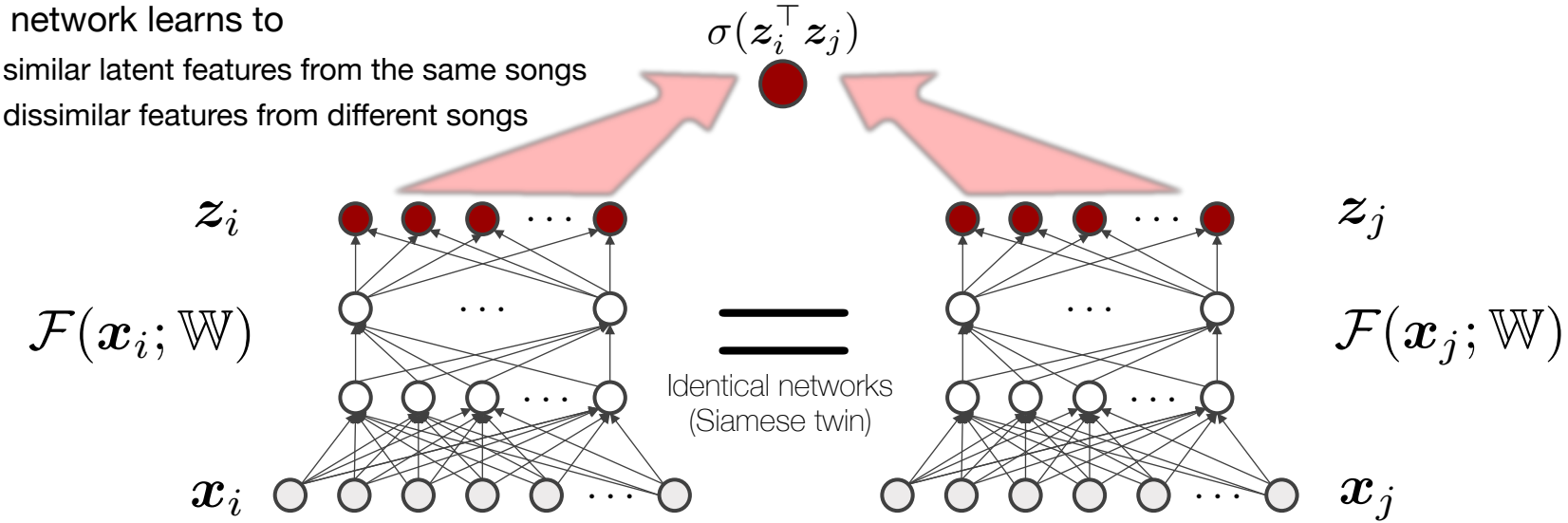
Deep Learning for Discriminant Embedding

- Siamese networks

- Target: $y(\boldsymbol{x}_i, \boldsymbol{x}_j) = \begin{cases} 1 & \text{if they're from the same song} \\ 0 & \text{otherwise} \end{cases}$

Turns on for matching pairs | Turns on for mismatching pairs

- Loss function: $-y(\boldsymbol{x}_i, \boldsymbol{x}_j) \log \sigma(\boldsymbol{z}_i^\top \boldsymbol{z}_j) - \big(1 - y(\boldsymbol{x}_i, \boldsymbol{x}_j)\big) \log \big(1 - \sigma(\boldsymbol{z}_i^\top \boldsymbol{z}_j)\big)$
  - **Similar to the word2vec loss function with negative sampling!**
- Siamese network learns to
  - Predict similar latent features from the same songs
  - Predict dissimilar features from different songs

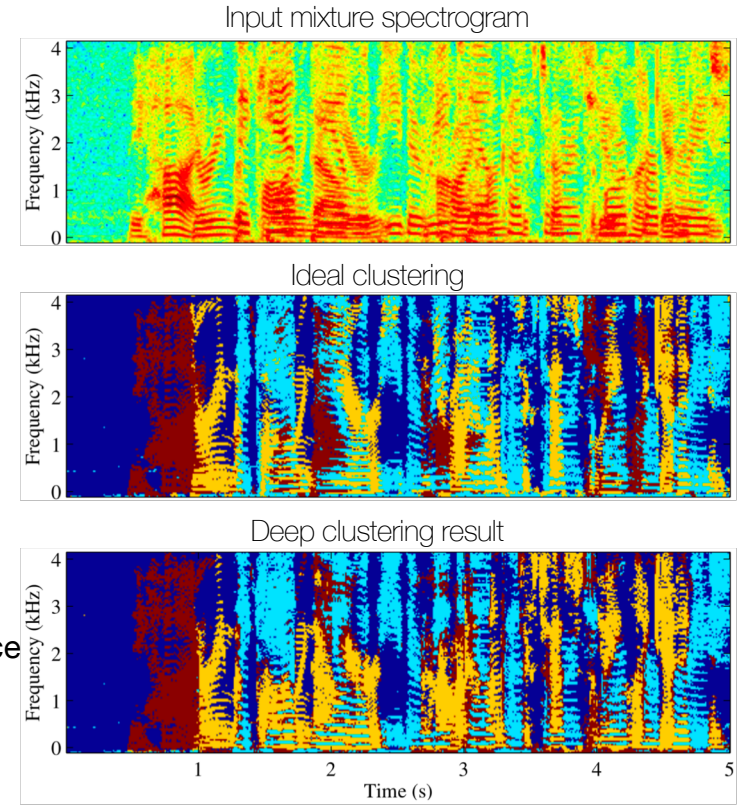


Bromley, Jane, et al. "Signature verification using a" siamese" time delay neural network." *NIPS* 1994
Koch, G. et al. "Siamese neural networks for one-shot image recognition." *ICML Deep Learning Workshop*. Vol. 2. 2015

Deep Clustering

- Discriminant features for source separation

- Deep clustering?
  - In the time-frequency domain audio signals are an image
  - Sources overlap in that space
    - Each pixel can belong to different sources at the same time
  - Clustering the pixels into sources
    - Assumes a pixel can belong to only one source
    - W-disjoint orthogonality
- How would k-means clustering work?
  - Will it work?
  - What's the input?
    - Some scalar, or a 2d vector at best (real and imaginary)
    - Would it be discriminant enough?
  - We want to transform this TF representation into a better feature space
    - So that k-means clustering works



J. R. Hershey et al. "Deep clustering: Discriminative embeddings for segmentation and separation," arXiv:1508.04306

Deep Clustering

- Discriminant features for source separation

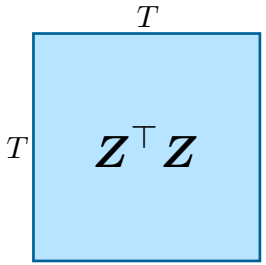
- We know how to learn features by now
  - But how to make them discriminant?
    - For source separation?
- In word2vec models the word vectors were to preserve the co-occurrence relationship
  - Or the pairwise similarity between items
  - Siamese networks are similar
- If in the feature space the samples preserve the some kind of similarity we think the features are nice

$$\boldsymbol{X} \in \mathbb{R}^{D \times T}$$
$$\boldsymbol{Z} \in \mathbb{R}^{K \times T} = \phi(\boldsymbol{X})$$

The (potentially very complicated) transform function

- Kernel methods
  - When people didn't want to deal with the transform function
  - The kernel matrix (pairwise similarity matrix) should preserve "semantic" similarity
  - e.g. the RBF kernel function

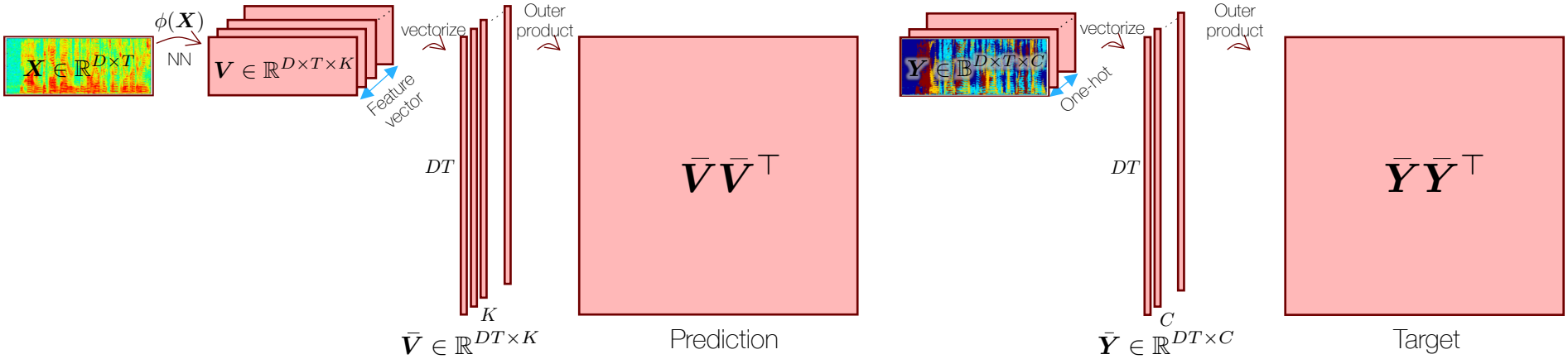
$$\boldsymbol{Z}_{:,i}^{\top} \boldsymbol{Z}_{:,j} = \mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left(-\frac{||\boldsymbol{x}_i - \boldsymbol{x}_j||^2}{\sigma^2}\right)$$



Deep Clustering

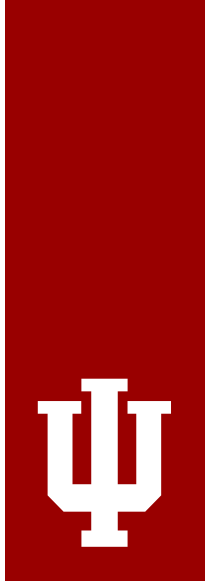
- Discriminant features for source separation

- But, a DNN can directly learn the transform function



- In English
  - The features learned in this way would contain some information about clustering the pixels

J. R. Hershey et al. “Deep clustering: Discriminative embeddings for segmentation and separation,” arXiv:1508.04306



Thank You!