USC Viterbi
School of Engineering
Department of Computer Science

CSCI 544
Applied Natural Language Processing

Mohammad Rostami
USC Computer Science Department

Logistical Notes

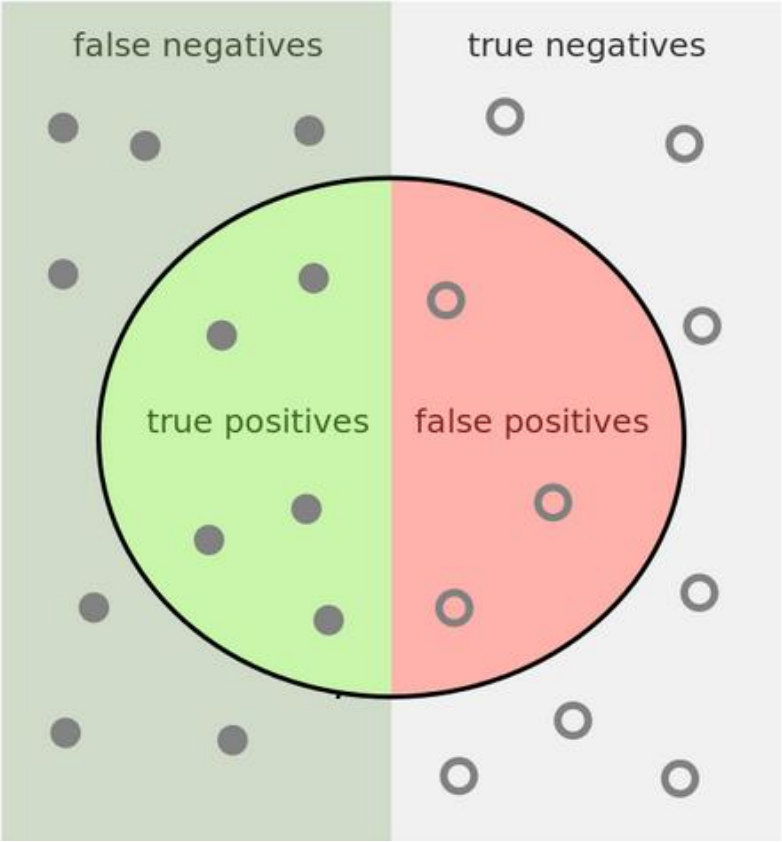
- Quizzes:
  - Quiz Period: 4:05-4:20 and still 10 minutes
  - Quiz 1 average: 84 (108 people scored 100)
  - We will have Quiz 2 next Thursday
- HW1:
  - Please check the homework description for edits done based on feedback from the Slack channel and prepare your report accordingly

Model Evaluation Process

- We use a training dataset for model selection
- A **good** parametric model along with a **suitable** training algorithm guarantees training a model that works well on the training data
- We need to validate that trained models **generalize** well on unseen data instances
- We need a second testing dataset which is fully independent of the training dataset
- We randomly split the annotated dataset into testing and training splits (sometimes, a validation set is generated as well)

Evaluation Metrics

- **Accuracy**: proportion of correctly classified items
- Accuracy can be dominated by **true negatives** (items correctly classified as not in a class).
- Sensitive with respect to imbalance



- Precision: $\frac{\text{True Positives}}{\text{True Positives+False Positive}}$

- Also called positive predictive value

- Recall: $\frac{\text{True Positives}}{\text{True Positives+False negative}}$

- Also called sensitivity

- Precision and recall are not useful metrics when used in isolation?
- We want our model to have good performance with respect to both metrics
- Implemented in sklearn

Evaluation Metrics

- Why having one measure is helpful?
- F1 $= \dfrac{2 \text{ Precision Recall}}{\text{Precision} + \text{Recall}}$
- F1 is biased towards the lower of precision and recall:

- harmonic mean < geometric mean < arithmetic mean
- F1=0 when Precision=0 or Recall=0

- Generalized F score:

$$F_\beta = (1+\beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}}.$$

Natural Language Representation

- Language processing hierarchy levels:
- **Documents**
- Sentences
- Phrase
- Words
- Sparsity in the NLP training datasets: natural language has a very huge space:
- Ex: Average Wikipedia page size is 580 words and English has ~1M words, yet the actual possibilities is far more.
- We need **interpretable** representations or **embeddings** to represent natural language data for model training
- One-hot representation: two large (15M words) and meaningless

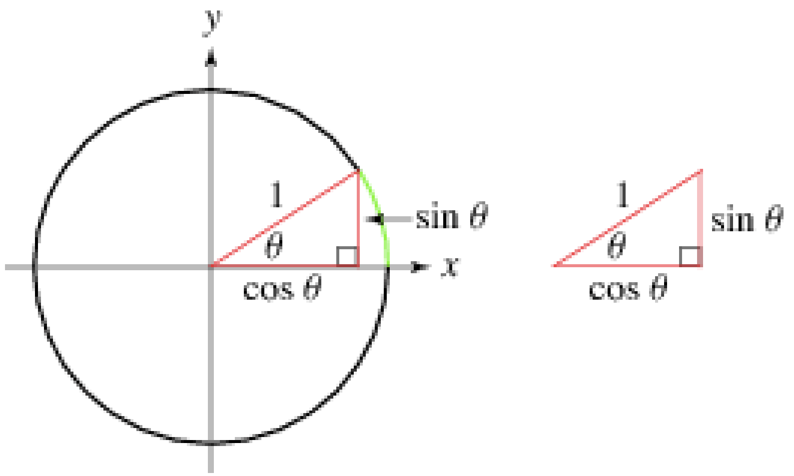
    Hotel: [0,0,0,0,1,0,0,0,0,0,0,…,0,0,0]

    Motel:[0,0,0,0,0,0,0,0,0,1,0,…,0,0,0]

Similarity of Vectors

- Euclidean distance, i.e., geometric closeness :

- Curse of dimensionality



- Dot product:

$a \bullet b = ||a||\,||b||\cos(\theta_{ab})$

$= a_1b_1+a_2b_2+ \ldots +a_nb_n$

- Cosine similarity (scale invariant)

$\cos\theta_{ab} = a \bullet b \ / \ ||a||\,||b||$ -> $1- \cos\theta_{ab}$ is a metric

- Invariant with respect to the vector starting point

- EX: Hotel: [0,0,0,0,1,0,0,0,0,0,0,…,0,0,0], Motel:[0,0,0,0,0,0,0,0,0,0,1,0,…,0,0,0]

Hotel'*Motel = 0

The Distributional Hypothesis

- Words that occur in the **same contexts** tend to have similar meanings (Zellig Harris, 1954)
- Example: nice, good
- Word relatedness association (Budanitsky and Hirst, 2006): related words **co-occur** in different contexts
- Example: cup, coffee
- If semantic similarity and association of words can be encoded into their representations, we may be able to address the challenge of sparsity
- In the absence of a particular word during training, we can rely on its synonyms that exist in the training dataset: Motel vs Hotel
- We can draw conclusions:

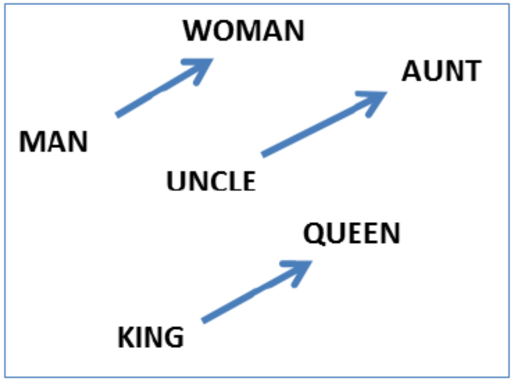
  Lecturers teach in the university-> Professor ___ in the university.

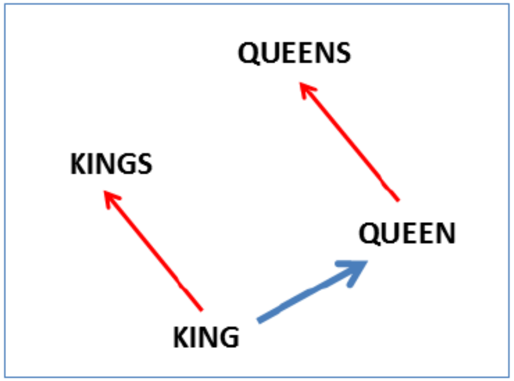
Vector Embedding of Words

- Represent words using dense vectors:
  - Latent Semantic Analysis/Indexing (SC Deerwester et al, 1988)
    - Term weighting-based model
    - Consider occurrences of terms at **document level**.
  - Word2Vec (Mikolov et al, 2013)
    - Prediction-based model.
    - Consider occurrences of terms at **context level**.
  - GloVe (Pennington et al, 2014)
    - Count-based model.
    - Consider occurrences of terms at **context level.**

Word Embedding

- Each word is represented by a vector:
  - The same size is used for all words
  - Relatively low dimensional (~300)
  - Vectors for similar words are similar (measured in dot product)
  - Vector operations can be used for semantic and syntactic deductions, e.g., Queen – Woman + Man = King





- The key idea is to derive the embeddings from the distributions of word context as they appear in a large corpus.

Matrix Factorization

- We can form a matrix of M using the idea of Bag of Words or TF-IDF: the word representations are highly sparce

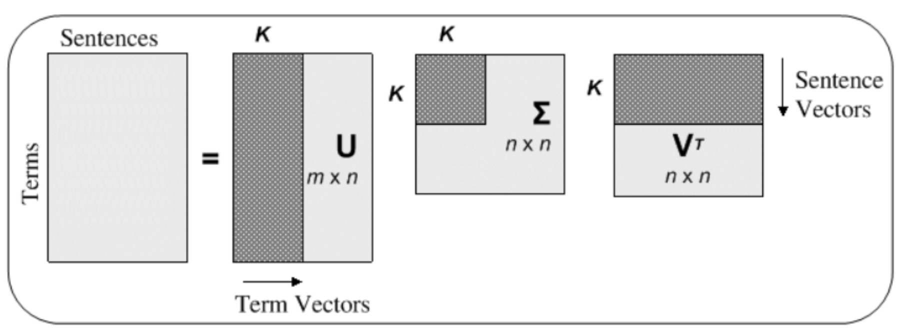
Words

Contexts

	1 This	2 movie	3 is	4 very	5 scary	6 and	7 long	8 not	9 slow	10 spooky	11 good	Length of the review(in words)
Review 1	1	1	1	1	1	1	1	0	0	0	0	7
Review 2	1	1	2	0	0	1	1	0	1	0	0	8
Review 3	1	1	1	0	0	0	1	0	0	1	1	6

- Singular value decomposition (U, V are orthonormal)

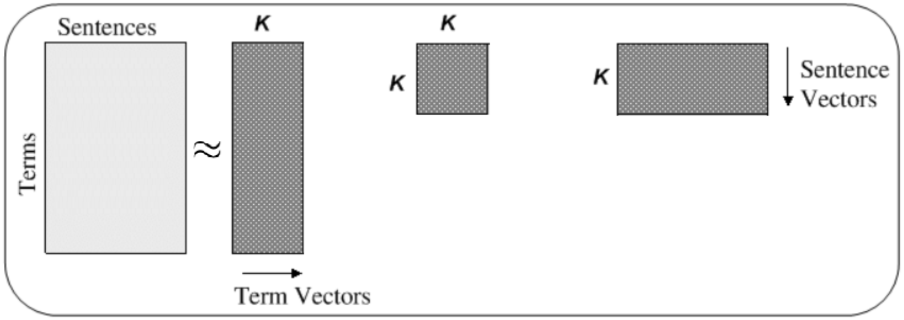
$$M_{m\times n} = U_{m\times n}\Sigma_{n\times n}V^T_{n\times n}$$



Matrix Factorization

- Many singular values are going to be zero or negligible

$$M_{m \times n} \approx U'_{m \times k} \Sigma'_{k \times k} V'^{T}_{k \times n}$$



$$m_l = u_l \Sigma' V'^{T}$$
$$u_l = m_l V' \Sigma'^{-1}$$

- We can use rows of U as word embeddings

- An old idea for dimensionality reduction (it is possible to use other matrix factorization methods, e.g., non-negative matrix factorization)
- Determining context is heuristic
- Computationally expensive
- Hyperparamters: contexts, cell values, optimization
- computational expensive with O(mn^2) cost for an n*m matrix
- Hard to incorporate new words

Word2Vec

- Core idea: find embeddings using a prediction task involving **neighboring words** in a huge real-world corpus.

- Input data can be sets of **successive word-patterns** from meaningful sentences in the corpus, e.g., "one of the most important".
- Try to build **synthetic** prediction tasks using these patterns, e.g., "(one of the __ important, most)"
- Train a model to solve the prediction task
- Embeddings are found as a **byproduct** of this process

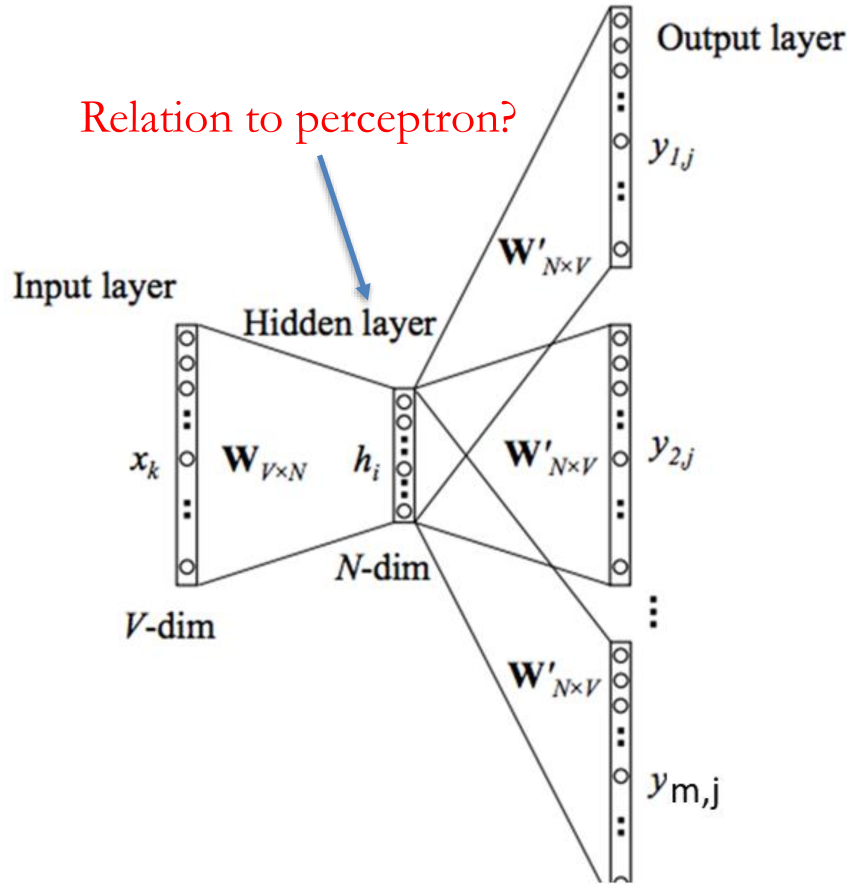
- More specifically:

- **We consider a window with the center word w_t and "context words" $w_{t'}$ with a window fixed size, e.g., (t'=t-5, … t-1, t+1, … , t+5).**
- The model is assumed to be a two-layer neural network
- We train the network to predict all w_t given $w_{t'}$ such that $p(w_{t'}|w_t)$ is maximized
- We learn embeddings such that the prediction loss is minimized, i.e., if two words occur in close proximity, their representations become similar.

Skip-Gram

- Given a center word, we predict the context words:

- Vocabulary size: V
- Input layer: center word in 1-hot form.
- The row k in W_{VxN} is the vector embedding of k-th center word.
- The column k of W'_{NXV} is context vector of the k-th word.
- At output layer y_{ij}, i=1..M is computed:
    1. We use the context word 1-hot vector to choose its column in W'_{NxV}
    2. dot product with h_i for the center word
    3. compute the softmax
    4. **Match** the output one-hot vector
- After optimization, we will have two vectors for each word. We can set the eventual embedding to e the average of these two vectors



$$h_{N\times 1} = W_{V\times N}^{T} x_{V\times 1} = v$$
$$\hat{z}_j = h_{N\times 1}^{T} W'[:, j]_{N\times 1} = v^{T} u$$
$$\hat{y} = \sigma(W'^{T}_{N\times V} h_{N\times 1})$$

Continuous Bag of Words

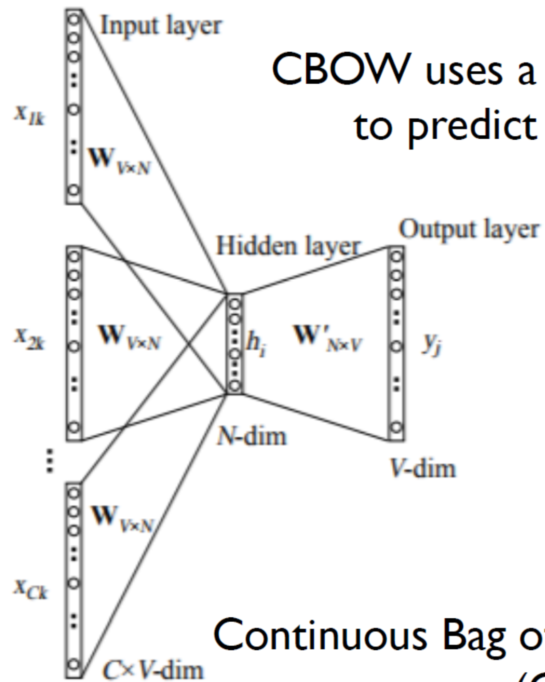
- Given a context word, we predict the center word:
  - Vocabulary size: V
  - Input layer: context words in 1-hot form.
  - The row k in W_{VxN} is the vector embedding of k-th context word.
  - The column k of W'_{NXV} is vector of the k-th center word.
  - The output column y_{ij}, i=1..M is computed:

    1. We use the center word 1-hot vector to choose its column in W'_{NxV}

    2. dot product with h_i for the contex word

    3. compute the softmax
  - We can set the eventual embedding to e the average of these two vectors
- Skip-gram incorporates non-frequent words better than CBOW

CBOW uses a window of word to predict the middle word



Continuous Bag of words (CBOW)

- Ex: Today is really __ day

Skip-gram: delightful+context and nice+context

Word2Vec Optimization Problem

- Consider an arbitrary order on vocabulary and let v_t be the word vector for center word t and u_t be the word vector for the context word t:

- Is the word vector unique?

Ex: [...like to **eat** lunch and...]

- We solve for the word vector by maximizing the following likelihood function

$$v_t, u_t = arg\max \log(\Pi_{t=1}^{T} P(c|w_t)) =$$

$$arg\max \log(\Pi_{t=1}^{T} P(c|w_t)) =$$

$$arg\max \log(\Pi_{t=1}^{T} \Pi_{j=-M,\, j\neq 0}^{M} P(w_{t-j}|w_t)) =$$

Important Step

$$arg\max \frac{1}{T} \sum_{t=1}^{T} \sum_{j=-M,\, j\neq 0}^{M} \log(P(w_{t-j}|w_t)) =$$

Word2Vec Optimization Problem

- Conditional Probability modeling

$$P(w_{t-j}|w_t) = \frac{e^{u_{t-j}^T v_t}}{\sum_{t=1}^{T} e^{u_{t-j}^T v_t}}$$

Is it an extension of the logistic function?

Computationally expensive!

- The log-likelihood optimization problem

$$u_o, v_c = \arg\min \frac{1}{V} \sum_{w=1}^{V} -u_o^T v_c + \log(\sum_{w'=1}^{V} e^{u_o^T v_c})$$

- Can be solved similar to logistic regression objective using numerical optimization techniques, e.g., gradient descent

Word2Vec Optimization Problem

- gradient descent step

$$u_o, v_c = \arg\min \frac{1}{V} \sum_{w=1}^{V} -u_o^T v_c + \log(\sum_{w'=1}^{V} e^{u_o^T v_c})$$

$$v_c^{i+1} = v_c^i - \eta \nabla f(v_c^i)$$

$$\nabla f(v_c^i) = -u_o + \sum_{w=1}^{v} \frac{e^{u_o^T v_c}}{\sum_{w'=1}^{V} e^{u_o^T v_c}} u_o$$

$$\nabla f(v_c^i) = -u_o + \sum_{w=1}^{v} p(v_o|u_c) u_o = -u_o(1 - E(u_o))$$

- Tutorial: https://rare-technologies.com/word2vec-tutorial/

Negative Sampling

- Word2Vec optimization is a highly computationally intensive problem: the **shallow** network has a large number of weights and we will have billions of pairs
- Because we use one-hot vectors, each training pair (c,o) contributes minimally to updating the weights
- Negative sampling: for each positive pair, we randomly generate negative pairs, for which the network output should be 0.

$$u_o, v_c = \arg\min \sigma(u_o^T v_c) + \sum_{k=1}^{K} E_{v_{w_i} \sim P(w_c)} \log(-\sigma(v_{w_i}^T v_c))$$

- This is an instance of naïve data augmentation

Neural vs Traditional Embeddings

- Comparison is challenging (Levy and Goldberg, NeurIPS 2014):
  - **Hyperparamters**
  - Factorization algorithm
  - Amount of data
- A particular word embedding approach is unlikely to be state-of-the-art for all applications
- Hyperparameters appear to have the largest impact in performance.
- Neural models are less sensitive with respect to hyperparameters, and training data preparation is more straightforward and systematic