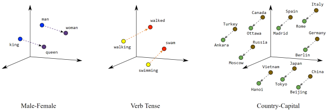
Basics

Word Embeddings



Learning goals

- Understand what word embeddigns are
- Learn the main methods for creating them

MOTIVATION (1)

- How to represent words/tokens in a neural network?
- Possible solution: one-hot encoded indicator vectors of length $|V|$.

$$\vec{w}^{(\text{the})} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \end{bmatrix} \quad \vec{w}^{(\text{cat})} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \end{bmatrix} \quad \vec{w}^{(\text{dog})} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \end{bmatrix}$$

- **Question:** Why is this a bad idea?
  - Parameter explosion ($|V|$ might be $> 1M$)
  - All word vectors are orthogonal to each other
    $\rightarrow$ no notion of word similarity

MOTIVATION (2)

- Learn one word vector $\vec{w}^{(i)} \in \mathbb{R}^D$ ("word embedding") per word i
- Typical dimensionality: $50 \leq D \leq 1000 \ll |V|$
- Embedding matrix: $\mathbf{W} \in \mathbb{R}^{|V| \times D}$
- **Question:** Advantages of using word vectors?
  - We can express similarities between words, e.g., with cosine similarity:

$$\cos(\vec{w}^{(i)}, \vec{w}^{(j)}) = \frac{\mathbf{w}^{(i)T}\mathbf{w}^{(j)}}{\|\mathbf{w}^{(i)}\|_2 \cdot \|\mathbf{w}^{(j)}\|_2}$$

  - Since the embedding operation is a *lookup operation*, we only need to update the vectors that occur in a given training batch

MOTIVATION (3)

Supervised training?

- *Training from scratch*:
  $\rightarrow$ Initialize embeddings randomly and learn it during training phase
  $\rightarrow$ Words that play similar roles w.r.t. task get similar embeddings
- *From sentiment classification*:
  $\rightarrow$ We might expect $\vec{w}^{(\text{great})} \approx \vec{w}^{(\text{awesome})}$
- **Question:** What could be a problem at test time?
  - If training set is small, many words are unseen during training and therefore have random vectors
- We typically have more unlabeled than labeled data.
  $\rightarrow$ Can we learn embeddings from the unlabeled data?

MOTIVATION (4)

- Distributional hypothesis:
  "*A word is characterized by the company it keeps*"
  (J.R. Firth, 1957)
- Idea: Learn similar vectors for words that occur in similar contexts
- Three different (milestone) methods:
  - Word2Vec ▸ (Mikolov et al., 2013)
  - GloVe (not covered) ▸ (Pennington et al., 2014)
  - FastText ▸ (Bojanowski et al., 2016)

WORD2VEC AS A BIGRAM LANGUAGE MODEL

- Words in our vocabulary are represented as two sets of vectors:
  - $\vec{w}^{(i)} \in \mathbb{R}^D$ if they are to be predicted
  - $\vec{v}^{(i)} \in \mathbb{R}^D$ if they are conditioned on as context
- Predict word i given previous word j:

$$P(i|j) = f(\vec{w}^{(i)}, \vec{v}^{(j)})$$

- **Question:** What is a possible function $f(\cdot)$?

A SIMPLE NEURAL NETWORK BIGRAM LANGUAGE MODEL

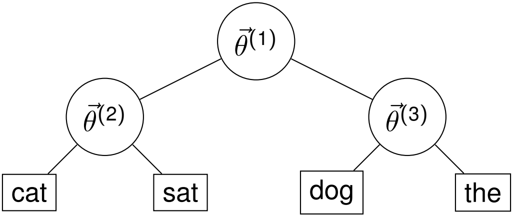
- Softmax!

$$P(i|j) = \frac{exp(\mathbf{w}^{(i)T}\mathbf{v}^{(j)})}{\sum_{k=1}^{|V|} exp(\mathbf{w}^{(k)T}\mathbf{v}^{(j)})}$$

- **Question:** Problem with training softmax?
  $\rightarrow$ *SLOW*
  (Needs to compute dot products with the whole vocabulary in the denominator for every single prediction)

HIERARCHICAL SOFTMAX

- Context vectors $\vec{v}$ are defined like before.
- Word vectors $\vec{w}$ are replaced by a binary tree:



HIERARCHICAL SOFTMAX

- Each tree node l has parameter vector $\vec{\theta}^{(l)}$
- Probability of going left at node l given context word j: $p(\text{left}|l, j) = \sigma((\vec{\theta}^{(l)})^T \ \vec{v}^{(j)})$
- Probability of going right: $p(\text{right}|l, j) = 1 - p(\text{left}|l, j)$
- Probability of word i given j: product of probabilities on the path from root to i

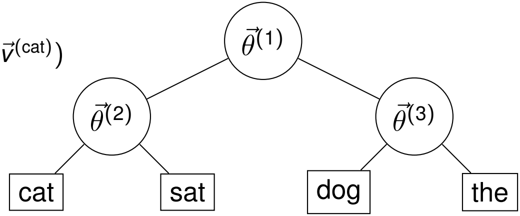
EXAMPLE

Calculate $p(\text{sat}|\text{cat})$.

$\vec{v}^{(\text{cat})}$

$\sigma((\vec{\theta}^{(1)})^T\ \vec{v}^{(\text{cat})})$

$1 - \sigma((\vec{\theta}^{(2)})^T\ \vec{v}^{(\text{cat})})$



$$p(\text{sat}|\text{cat}) = \sigma((\vec{\theta}^{(1)})^T\ \vec{v}^{(\text{cat})})[1 - \sigma((\vec{\theta}^{(2)})^T\ \vec{v}^{(\text{cat})})]$$

QUESTIONS

- **Question:** How many dot products do we need to calculate to get to $p(i|j)$? How does this compare to the naive softmax?
  - $\log_2 |V| \ll |V|$
- **Question:** Show that $\sum_{i'} p(i'|j)$ sums to 1.

SPEEDING UP TRAINING: NEGATIVE SAMPLING

- Another trick: **negative sampling**
  (a variant of *noise contrastive estimation*)
- This changes the objective function, and the resulting model is not a language model anymore!
- **Idea**: Instead of predicting the probability distribution over whole vocabulary, make binary decisions for a small number of words.
- “Positive“ samples: Bigrams seen in the corpus.
- “Negative“ samples: Random bigrams (not seen in the corpus).

NEGATIVE SAMPLING: LIKELIHOOD

- Given:
  - Positive training set: $\mathrm{pos}(\mathcal{O})$
  - Negative training set: $\mathrm{neg}(\mathcal{O})$

$$L = \prod_{(i,j)\in \mathrm{pos}(\mathcal{O})} P(\mathrm{pos}|\vec{w}^{(i)}, \vec{v}^{(j)}) \prod_{(i',j')\in \mathrm{neg}(\mathcal{O})} P(\mathrm{neg}|\vec{w}^{(i')}, \vec{v}^{(j')})$$

- $P(\mathrm{pos}|\vec{w}, \vec{v}) = \sigma(\vec{w}^T \vec{v})$
- $P(\mathrm{neg}|\vec{w}, \vec{v}) = 1 - P(\mathrm{pos}|\vec{w}, \vec{v})$
- **Question:** Why not just maximize $\prod_{(i,j)\in \mathrm{pos}(\mathcal{O})} P(\mathrm{pos}|\vec{w}^{(i)}, \vec{v}^{(j)})$?
  - Trivial solution: make all $\vec{w}$, $\vec{v}$ identical

WORD2VEC WITH NEGATIVE SAMPLING AS CLASSIFICATION

- Maximize likelihood of training data:

$$\mathcal{L}(\theta) = \prod_i P(y^{(i)}|x^{(i)};\theta)$$

- $\Leftrightarrow$ minimize negative log likelihood:

$$NLL(\theta) = -\log \mathcal{L}(\theta) = -\sum_i \log P(y^{(i)}|x^{(i)};\theta))$$

- **Question:** What do these components stand for in Word2Vec with negative sampling?
  - $x^{(i)}$ Word pair, from corpus OR randomly created
  - $y^{(i)}$ Label: 1 $=$ word pair is from positive training set, 0 $=$ word pair is from negative training set
  - θ Parameters $\vec{v}$, $\vec{w}$
  - $P(...)$ Logistic sigmoid: $P(1|\cdot) = \sigma(\vec{w}^T\vec{v})$, resp. $P(0|\cdot) = 1 - \sigma(\vec{w}^T\vec{v})$.

SPEEDING UP TRAINING: NEGATIVE SAMPLING

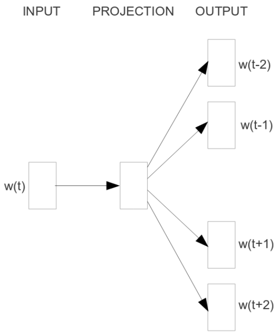
- Constructing a good negative training set can be difficult
- Often it is some random perturbation of the training data (e.g. replacing the second word of each bigram by a random word).
- The number of negative samples is often a multiple (1x to 20x) of the number of posisive samples
- Negative sets are often constructed per batch

QUESTIONS

- **Question:** How many dot products do we need to calculate for a given word pair? How does this compare to the naive and hierarchical softmax?
  - $M + 1 \approx \log_2 |V| \ll |V|$
  - (for $M = 20, |V| = 1,000,000$)

SKIP-GRAM (WORD2VEC)

- Idea: Learn many bigram language models at the same time.
- Given word $w_{[t]}$, predict words inside a window around $w_{[t]}$:
  - One position before the target word: $p(w_{[t-1]}|w_{[t]})$
  - One position after the target word: $p(w_{[t+1]}|w_{[t]})$
  - Two positions before the target word: $p(w_{[t-2]}|w_{[t]})$
  - ... up to a specified window size c.
- Models share all $\vec{w}$, $\vec{v}$ parameters!



Skip-gram

SKIP-GRAM: OBJECTIVE

- Optimize the joint likelihood of the $2c$ language models:

$$p(w_{[t-c]} \dots w_{[t-1]} w_{[t+1]} \dots w_{[t+c]} | w_{[t]}) = \prod_{\substack{i \in \{-c \dots c\} \\ i \neq 0}} p(w_{[t+i]} | w_{[t]})$$

- Negative Log-likelihood for whole corpus (of size N):

$$NLL = -\sum_{t=1}^{N} \sum_{\substack{i \in \{-c \dots c\} \\ i \neq 0}} \log p(w_{[t+i]} | w_{[t]})$$

- Using negative sampling as approximation:

$$\approx -\sum_{t=1}^{N} \sum_{\substack{i \in \{-c \dots c\} \\ i \neq 0}} \Big[\log \sigma(\vec{w}_{[t+i]}^T \vec{v}_{[t]}) + \sum_{m=1}^{M} \log[1 - \sigma(\vec{w}^{(*)^T} \vec{v}_{[t]})] \Big]$$

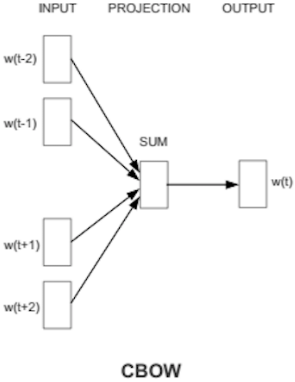
- $\vec{w}^{(*)}$ is the word vector of a random word, M is the number of negatives per positive sample

C(ONTINUOUS) B(AG) O(F) W(ORDS)

- Like Skipgram, but...
- Predict word $w_{[t]}$, given the words inside the window around $w_{[t]}$:

$$p\big(w_{[t]} \big| w_{[t-c]} \dots w_{[t-1]} w_{[t+1]} \dots w_{[t+c]}\big)$$

$$\propto \vec{w}_{[t]}^{T} \sum_{\substack{i \in -c \dots c \\ i \neq 0}} \vec{v}_{[t+i]}$$



FASTTEXT (1)

- Even if we train Word2Vec on a very large corpus, we will still encounter unknown words at test time
- Orthography can often help us:
- $\vec{w}^{(\text{remuneration})}$ should be similar to
  - $\vec{w}^{(\text{remunerate})}$ (same stem)
  - $\vec{w}^{(\text{iteration})}, \vec{w}^{(\text{consideration})} \dots$ (same suffix $\approx$ same POS)

FASTTEXT (2)

$$\text{known word: } \vec{w}^{(i)} = \frac{1}{|\mathrm{ngrams}(i)| + 1} \left[\vec{u}^{(i)} + \sum_{n \in \mathrm{ngrams}(i)} \vec{u}^{(n)} \right]$$

$$\text{unknown word: } \vec{w}^{(i)} = \frac{1}{|\mathrm{ngrams}(i)|} \sum_{n \in \mathrm{ngrams}(i)} \vec{u}^{(n)}$$

$$\mathrm{ngrams}(\text{remuneration}) = \{\text{\$re}, \text{rem}, \text{\$rem}, \ldots \text{ration}, \text{ation\$}\}$$

FASTTEXT TRAINING

- ngrams typically contains character 3- to 6-grams
- Replace $\vec{w}$ in Skipgram objective with its new definition
- During backpropagation, loss gradient vector $\frac{\partial J}{\partial \vec{w}^{(i)}}$ is distributed to word vector $\vec{u}^{(i)}$ and associated n-gram vectors $\vec{u}^{(n)}$

SUMMARY

- Word2Vec as a bigram Language Model
- Hierarchical Softmax
- Negative Sampling
- Skipgram: Predict words in window given word in the middle
- CBOW: Predict word in the middle given words in window
- fastText: N-gram embeddings generalize to unseen words
- Any questions?

USING PRETRAINED EMBEDDINGS

- Knowledge *transfer* from unlabelled corpus
- Design choice: Fine-tune embeddings on task or freeze them?
  - Pro: Can learn/strengthen features that are important for task
  - Contra: Training vocabulary is small subset of entire vocabulary $\rightarrow$ we might overfit and mess up topology w.r.t. unseen words

INITIALIZING NN WITH PRETRAINED EMBEDDINGS

Model		MR	SST-1	SST-2	Subj	TREC	CR	MPQA
CNN-rand	(randomly initialized)	76.1	45.0	82.7	89.6	91.2	79.8	83.4
CNN-static	(pretrained+frozen)	81.0	45.5	86.8	93.0	92.8	84.7	**89.6**
CNN-non-static	(pretrained+fine-tuned)	**81.5**	48.0	87.2	93.4	93.6	84.3	89.5
CNN-multichannel	(combination)	81.1	47.4	**88.1**	93.2	92.2	**85.0**	89.4

Table from Kim 2014: Convolutional Neural Networks for Sentence Classification.

RESOURCES

- `https://fasttext.cc/docs/en/crawl-vectors.html`
  - Embeddings for 157 languages, trained on big web crawls, up to 2M words per language
- `https://nlp.stanford.edu/projects/glove/`
  - GloVe word vectors: Co-occurrence-count objective, not n-gram based

ANALOGY MINING (1)

country-capital

$$\vec{w}^{(\text{Tokio})} - \vec{w}^{(\text{Japan})} + \vec{w}^{(\text{Poland})} \approx \vec{w}^{(\text{Warsaw})}$$

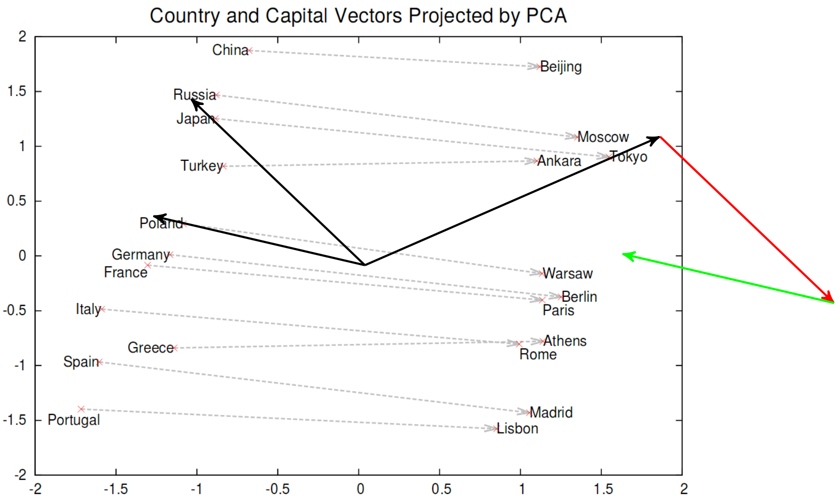
opposite

$$\vec{w}^{(\text{unacceptable})} - \vec{w}^{(\text{acceptable})} + \vec{w}^{(\text{logical})} \approx \vec{w}^{(\text{illogical})}$$

Nationality-adjective

$$\vec{w}^{(\text{Australian})} - \vec{w}^{(\text{Australia})} + \vec{w}^{(\text{Switzerland})} \approx \vec{w}^{(\text{Swiss})}$$

ANALOGY MINING (2)



ANALOGY MINING (3)

$$\vec{w}^{(a)} - \vec{w}^{(b)} + \vec{w}^{(c)} = \vec{w}^{(?)}$$

$$\vec{w}^{(d)} = \underset{\vec{w}^{(d')} \in \mathbf{W}}{\operatorname{argmax}} \quad \cos(\vec{w}^{(?)}, \vec{w}^{(d')})$$

Table 8: *Examples of the word pair relationships, using the best word vectors from Table 4 (Skip-gram model trained on 783M words with 300 dimensionality).*

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

SUMMARY

- Applications of Word Embeddings
  - Word vector initialization in neural networks for NLP tasks
    - E.g., sentiment classification of reviews, topical classification of news
  - Analogy mining
  - Information retrieval: semantic search, query expansion
  - Simple and fast aggregations of sentence representations
  - ...
- Any questions...?