Word2Vec

Benjamin Roth

CIS LMU München

November 22, 2017

Language Models

- What is a language Model?
- What is an n-gram language Model?

Language Models

- What is a language Model (LM)?
  $\Rightarrow$ Function to assign probability to a sequence of words.
- What is an n-gram language Model?
  $\Rightarrow$ Markov asumption
  Probability of word only depends on no more than $n-1$ other (previous) words:

$$P(w_{[1]} \dots w_{[T]}) = \prod_{t=1}^{T} P(w_{[t]} | w_{[t-1]} ... w_{[t-n+1]})$$

A Simple Neural Network Bigram Language Model

- Words w_i, w_j are represented as vectors[1]:
  - $\mathbf{w^{(i)}}$ if they are to be predicted.
  - $\mathbf{v^{(j)}}$ if they are conditioned on as context.
- Predict word w_i given previous word w_j:

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

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

[1] Note on notation: In this context $\mathbf{w^{(i)}}$ stands for the vector for lexicon item with index i, not for an indexed vector element.

A Simple Neural Network Bigram Language Model

- Softmax:

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

- Problem with training softmax?
- Possible advantages of having word vectors?

A Simple Neural Network Bigram Language Model

- Softmax:

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

- Problem with training softmax?
  - $\Rightarrow$ Slow! Needs to sum over whole vocabulary for computing log-likelihood.
- Possible advantages of having word vectors?
  - You can calculate similarities between words, e.g. using cosine similarity:

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

  - You can use them as pre-trained embeddings for more complex NN architectures.

Speeding up Training: Hierarchical Softmax

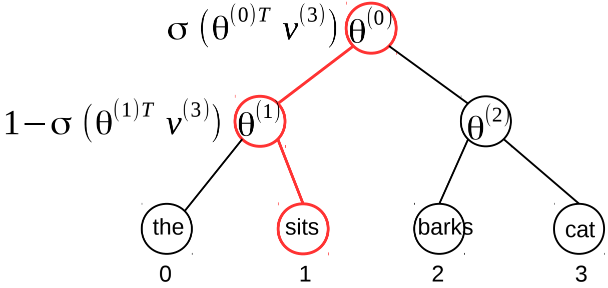
- Problem: Softmax needs to normalize over all output symbols in order to compute log-likelihood for correct output word w_i
- Hierarchical softmax:
  - Structure words in a tree.
  - Compute probability as a product of binary decisions, going down the the tree branches that lead to output word.
    $\Rightarrow$ binary logistic sigmoid.
  - Tree nodes have vector parameters.
  - Probabilities for going left or right are computed using tree node vectors and context word vector.
  - Tree node vectors and context word vector can be updated using gradient methods (only for relevant path).
    $\Rightarrow$ No need to separately normalize over vocabulary.
- Tree depth usually of order $O(\log_2 |V|)$

Example

- Calculate $P(\text{sits}|\text{cat})$
- Vocabulary is indexed by tree. Tree node to reach is node 1 (for *"sits"*), Context vector is $\mathbf{v}^{(3)}$ (for *"cat"*).
- For each node l (and context word w_j) we calculate the probability of going left:

$$P_{lj}(\text{left}) = \sigma(\boldsymbol{\theta}^{(l)T}\mathbf{v}^{(j)})$$

- Probability of *sits* is the product of probabilities on the tree path.
- It is easy to verify that the resulting distribution is normalized.



Speeding up Training: Negative Sampling

- If we are only interested in the resulting vectors, we can apply another trick: **negative sampling** (aka *noise contrastive estimation*)
- This changes the objective function, and the resulting model is not a language model anymore!
- Idea:
  Instead of predicting probability distribution over whole vocabulary (expensive normalization), make binary decisions (trivial normalization).
- Binary decision:
  Given a bigram, is it *good* (like those seen in the training corpus) or is t it *bad* (like those in a negative training set)

Negative Sampling: Likelihood

$$\mathcal{L} = \prod_{(v,w)\in\mathcal{O}} P(\text{good}|w,v) \cdot \prod_{(v',w')\in\text{neg}(\mathcal{O})} P(\text{bad}|w',v')$$

- $\mathcal{O}$: Observed bigrams
- neg($\mathcal{O}$): Negative set
- $P(\text{good}|w,v) = \sigma(\mathbf{w}^T\mathbf{v})$
- $P(\text{bad}|w,v) = 1 - P(\text{good}|w,v)$

$\Rightarrow$ Why not just optimize for $\prod_{(v,w)\in\mathcal{O}} P(\text{good}|w,v)$?

Speeding up Training: Negative Sampling

- How to construct a good negative training training set often requires some experimentation.
- 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 positive data.
- Negative sets are often constructed per batch.

Questions?

Skip-gram (Word2Vec)

- Task: Learn several bigram language models at the same time.
- Each of the bigram models takes the target word (*context word*) from a different relative position to a center word[2]:
  - 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.
    $p(w_{[t-2]}|w_{[t]})$
  - Two positions after the target word.
    $p(w_{[t+2]}|w_{[t]})$
  - ... up to a specified maximal window size c.
- Language models are parametrized as before:
  $$p(w_{[t+i]}|w_{[t]}) = \frac{exp(\mathbf{w}^{[t+i]T}\mathbf{v}^{[t]})}{\sum_{k=1}^{|V|} exp(\mathbf{w}^{(k)T}\mathbf{v}^{[t]})}$$
- Language models share parameters.

---

[2]Notation: $w_{[t]}$ is the word at position t, $\mathbf{w}^{[t]}$ is the vector indexed by target word $w_{[t]}$, $\mathbf{v}^{[t+i]}$ are the vectors indexed by context words $w_{[t+i]}$

Skip-gram: Objective

- Optimize the joint likelihood of these language models:

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

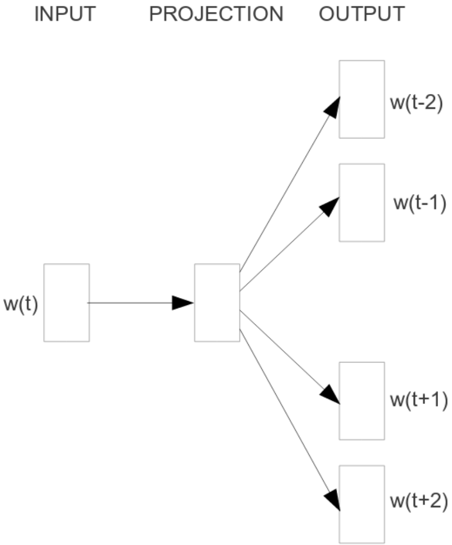
$$= p(w_{[t-c]}|w_{[t]}) \dots p(w_{[t-1]}|w_{[t]}) \cdot p(w_{[t+1]}|w_{[t]}) \dots p(w_{[t+c]}|w_{[t]})$$

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

$$NLL = -\log \prod_{t=1}^{N} p(w_{[t-c]} \dots w_{[t-1]} w_{[t+1]} \dots w_{[t+c]} | w_{[t]})$$

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

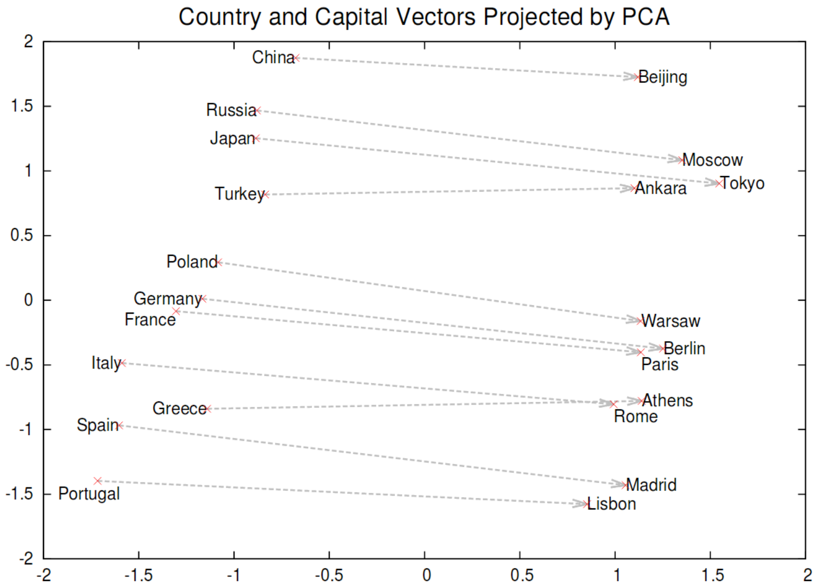
Skipgram: *"Word in the middle predicts all words in context."*



Skipgram: Training

- Either: Hierarchical softmax.
- Or: Negative sampling.
  - Default choice when good word vectors are important rather than distribution over vocabulary.
  - $\mathcal{O}$: All bigrams in all context windows.
  - $\text{neg}(\mathcal{O})$: Random bigrams. Context word of elements in $\mathcal{O}$ is replaced by random word in vocabulary.

Skipgram: Relational Regularities in Vector Space



"Windows is to Microsoft as Android is to Google"

- *v(Bejing) - v(China)* $\sim$ *v(Warsaw) - v(Poland)*
- Add vector on both side:
  v(Bejing) - v(China) $+$ *v(Poland)* $\sim$ *v(Warsaw)*
- Apply the same logic to more triples, and always retrieve the most similar word (cosine):

$$a - b + b^* = ?$$

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