Lecture 2

Andrei Arsene Simion

Columbia University

January 25, 2023

Outline

- Basic Word Embeddings
- CBOW Model
- Skip-Gram Model
- Doc2Vec
- PyTorch Examples

Outline

- Basic Word Embeddings
- CBOW Model
- Skip-Gram Model
- Doc2Vec
- PyTorch Examples

Word Embeddings Motivation

- **Given**: A corpus S with sentences
- **Want**: A vector representation for each word a_w such that the vector a_w captures some of the "meaning" of w for each w
- **Ideally**: These vectors have nice mathematical properties
  - If two words w_1, w_2 are related, their vectors are related in the sense that $a_{w_1}^T a_{w_2}$ is high
  - Or, $||a_{w_1} - a_{w_2}||$ is low: we can cluster similar words!
  - Maybe $a_{w_1} - a_{w_2}$ has a "meaning"; $a_{doctor} - a_{man} \sim$ "medical profession person"

Word Embeddings Motivation

- Tokenize S in some way
  - Possible recipe: Take the tokens as the words
  - Take S and split it on spaces
  - Eliminate rare words?
  - Collapse some words together (running, run, runs)?
  - We now have a vocabulary set V with a set of words {a, and, the, cat, mat, ...}
  - Each word can be assigned a unique integer, and we have a hash map $stoi = \{w \to i\}$
  - From this map, we can easily get get a reverse hash map $itos = \{i \to w\}$
- For each $w \in V$ we can assign a unique index i
- Naively, we can represent each word as a one-hot 1-0 vector e in $\mathbf{R}^{|V|}$ so that each element of e is 0 except the i^{th} element which is 1
- Thus, we have $e[stoi[w]] = 1$ and e is zero for all other indices

Word Embeddings Motivation: One-Hot Example

- Suppose S = "The cat was blue."
- Suppose $V = \{$"the", "cat", "was", "blue", "."$\}$ so $|V| = 5$
- Map each word to a unique index so that stoi = $\{$"the" $\rightarrow$ 0, "cat" $\rightarrow$ 1, "was" $\rightarrow$ 2, "blue" $\rightarrow$ 3, "." $\rightarrow$ 4$\}$
- Then, a one-hot representation might be
  - $e_{the} = [1, 0, 0, 0, 0]$
  - $e_{cat} = [0, 1, 0, 0, 0]$
  - $e_{was} = [0, 0, 1, 0, 0]$
  - $e_{blue} = [0, 0, 0, 1, 0]$
  - $e_{.} = [0, 0, 0, 0, 1]$

Word Embeddings Motivation: One-Hot Problems

- If V is a large set, the vectors we get for each word are very large
- The notion of "similarity" is lost, each vector is independent
- Specifically, for any two different a_i and a_j we have

$$a_{w_i}^{\mathsf{T}} a_{w_j} = 0$$

- **Idea**: Try to reduce the size of this space from $\mathbf{R}^{|V|}$ to something smaller
- **Idea**: Hope that we can find a subspace that encodes the relationships between words so that

$$a_{w_i}^{\mathsf{T}} a_{w_j} = \textit{relationship} \text{ (a number)}$$

Word Embeddings Motivation: SVD

- Idea of SVD: express your data as a matrix and "factorize" the matrix so that you capture the "latent" dimensions of some subspace
- Given some (data summary) matrix X, rewrite $X = A\Sigma B^{\intercal}$
  - Can use the co-occourrence matrix X
  - What does it mean for something to co-occur? To in a certain window or distance from each other?
  - Creating X is not easy!

Word Embeddings Motivation: SVD

- Fix a *window size* of 1
- Given the 3 sentences below so $S = \{s_1, s_2, s_3\}$, we can tokenize them into words and get a square matrix of dimension $|V| \times |V|$
  - s_1 = "I like driving."
  - s_2 = "I think NLP is cool ."
  - s_3 = "Deep learning NLP is cool ."
- $V = \{i, like, driving, ., think, nlp, is, cool, deep, learning\}$
- $|V| = 10$
- Notice that "." is a valid token

Word Embeddings Motivation: SVD

- For any word, again set a unique word integer i via $stoi[w] = i$
- For any two words w_i and w_j let $X[i,j] =$ number of times w_i and w_j are within a window size 1 of each other
- We get the symmetric matrix X:

$$\text{X} = \begin{array}{c} \\ \textit{i} \\ \textit{like} \\ \textit{driving} \\ . \\ \textit{think} \\ \textit{nlp} \\ \textit{is} \\ \textit{cool} \\ \textit{deep} \\ \textit{learning} \end{array} \begin{array}{c} \begin{array}{cccccccccc} \textit{i} & \textit{like} & \textit{driving} & . & \textit{think} & \textit{nlp} & \textit{is} & \textit{cool} & \textit{deep} & \textit{learning} \end{array} \\ \left(\begin{array}{cccccccccc} 2 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 4 & 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 & 2 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 2 & 2 & 2 & 0 & 1 \\ 0 & 0 & 0 & 2 & 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \end{array}\right) \end{array}$$

Word Embeddings Motivation: SVD

- Perform SVD on X so that $X = A\Sigma B^\intercal$
  - A, Σ, B is $|V| \times |V|$
  - Σ is diagonal and has singular values σ_i on it's diagonal
  - Pick some $K < |V|$ so that $\frac{\sum_{i=1}^{K} \sigma_i}{\sum_{i=1}^{|V|} \sigma_i}$ is high enough

Word Embeddings Motivation: SVD

- Approximate X by $\hat{X}$ where we approximate
  - A by $\hat{A}$, where we use only the first K columns
  - Σ by $\hat{\Sigma}$ where we only use the first K rows and K columns
  - B by $\hat{B}$, where we use only the first K rows
- We get an Approximation $X \sim \hat{X} = \hat{A}\hat{\Sigma}\hat{B}^{\intercal}$
- **Note that the** $|V|$ K**-dimensional** rows of $\hat{A}$ or/and $\hat{B}$ can be viewed as vector representations for each word
  - Each row of $\hat{A}$ gives us a K-dimensional vector a_w
  - Each row of $\hat{B}$ gives us a K-dimensonal vector b_w
  - We have a vector representation with dimension $K < |V|$
  - **Maybe a_w is what we want? Maybe these vectors have the nice properties we wanted?**

Word Embeddings Motivation: SVD

- Some problems:
  - X is very sparse
  - Expensive cost to train, $O(|V|^3)$
  - New words imply that the dimension of the matrix changes very often
  - Very large matrix!
  - X usually is not just co-occurrence counts, we usually need more than this and some hacks to make this work best (so the rows of A capture as much semantic and syntactic information as possible)
- Can use Pearson correlation instead of counts
- Remove function words like "he", "has", "the", "a", etc.
- Use a ramp window: weigh co-occurence matrix by distance instead of just counts
- *Instead of this, let's try something else!*
  - Interestingly, we'll see (next class) that what follows can be viewed as matrix factorization (2019!) ...

Word Embeddings Motivation: Towards Word2Vec

- **Idea:** Instead of computing global statistics, we create a model that learns at each iteration and refines what it learns as it progresses
- Two papers below have the core ideas we look at next
  - Efficient Estimation of Word Representations in Vector Space
  - Distributed Representations of Words and Phrases and their Compositionally

Aside: Softmax and Cross Entropy

- **Softmax**: given a $|V|$ dimensional vector x, we want to turn x into a probability
- This pre-probability is usually called a logit
- We can map $softmax(x) \rightarrow (\frac{e^{x_1}}{\sum_{i=1}^{|V|} e^{x_i}}, \ldots, \frac{e^{x_i}}{\sum_{i=1}^{|V|} e^{x_i}})$
- This preserves the order among the components, and is differentiable!
- $max(x)$ is NOT differentiable!
- **Sigmoid**: given a real number, we can map it to a probability via $\sigma(x) \rightarrow \frac{1}{1+e^{-x}}$
- This is used in logistic regression, and we will need it here as well

Aside: Softmax and Cross Entropy

- **Cross-Entropy**: Give two distributions q and p, how do we measure how close they are?
- The cross entropy can be used measure the dissimilarity $CE = -\sum_{i=1}^{|V|} q(i) \log p(i)$
- If q is fixed, what is the optimal p? You'll get $p = q$
- To see this, minimize $L(p) = -\sum_{i=1}^{|V|} q(i) \log p(i)$ subject to $sum(p) = 1$
- Typically for us q is the y, a one-hot vector and p is the softmax of some logit that a model predicts (a vector)

Outline

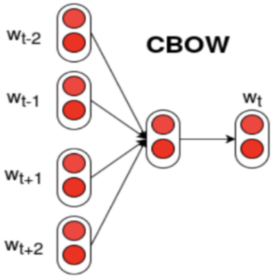
- Basic Word Embeddings
- CBOW Model
- Skip-Gram Model
- Doc2Vec
- PyTorch Examples

Word Embeddings Motivation: Word2Vec

- A high level summary:
  - 2 models: CBOW and Skip-Gram
    - CBOW: predict a center word from the words around it
    - Skip-Gram: predict words around a center word
  - 2 tricks: Negative Sampling and Hierarchical Softmax
    - Both these methods try to solve for the issue that $|V|$ is a very large and leads to slow optimization as we'll see
- Both models are built on the idea: "You shall know a word by the company it keeps" (Distributional Hypothesis - Harris 1954)

Word Embeddings: CBOW

- The goal of CBOW is to predict the missing center word from the neighboring context words



Word Embeddings: CBOW

- We need a model that will assign a probability to a given sentence
- "I big am I if I gym I go now" = low probability
- "All your base are belong to us!" = medium probability
- "The student studied NLP into the night." = high probability
- Given T tokens in a sentence $s = [w_1, w_2, \ldots, w_T]$ we have

$$P(s) = P(w_1, w_2, \ldots, w_T)$$

Word Embeddings: CBOW

- If we use the chain rule (as before), we have that (generally)

$$P(s) = \prod_{i=1}^{T} P(w_i|w_{i-1}, .., w_1)$$

- $P(w_1|w_0)$ is special and w_0 can be a special "BOS" token
- The above is a general formula, but typically we consider a *bi-gram* approximation so that it becomes

$$P(s) \sim \prod_{i=1}^{T} P(w_i|w_{i-1})$$

Word Embeddings: CBOW

- We need to model $P(w_{output}|w_{context})$
- Given a sequence "student studied NLP into the" we can let *NLP* be the output word and "student, studied, into, the" be the context
- Define two matrices $B \in \mathbf{R}^{|V| \times d}$ and $A \in \mathbf{R}^{|V| \times d}$ where d represents the embedding space
- If we have the one-hot representation of a word, e_w, then $B^\intercal e_w = b_w$ can be viewed as dimensional vector representation of w
  - We also have $A^\intercal e_w = a_w$

Word Embeddings: CBOW Algorithm

- Given a set of documents C, break up C into sentences and tokenize C to get the vocabulary V
- Fix a window size m so for each word we look m words to the right and left
- Example: if $s =$ "This is a short sentence." and $m = 1$ we get
  - [*"this"*, *"is"*, *"a"*]
  - [*"is"*, *"a"*, *"short"*]
  - [*"a"*, *"short"*, *"sentence"*]
  - [*"short"*, *"sentence"*, *"."*]

Word Embeddings: CBOW Algorithm

- For each center out word w_o and context words, gather (x, y) pairs like

$$((w_{o-m}, \dots, w_{o-1}, w_{o+1}, \dots, w_{o+m}), w_o)$$

  - $["a", "short", "sentence"]$ becomes $[("a", "sentence"), "short"]$ and $w_o = "short"$
- For each context word, get the one-hot representations e_w
- For each context word, get the vector representations $(A^\intercal e_{w_{o-m}}, A^\intercal e_{w_{o-1}}, A^\intercal e_{w_{o+1}}, \dots, A^\intercal e_{w_{o+m}})$ so that

$$A^\intercal e_{w_{o-m}} = a_{w_{o-m}} \in \mathbf{R}^d$$

etc.

Word Embeddings: CBOW Algorithm

- Average the vectors $a_{avg} = \frac{a_{w_{c-m}} + \ldots + a_{w_{c+m}}}{2m} \in \mathbf{R}^d$
- Generate logit scores $l = Ba_{avg} \in \mathbf{R}^{|V|}$
- Turn the scores into probabilities $p = \text{softmax}(l) \in \mathbf{R}^{|V|}$
  - $softmax(l_1, l_2) = (\frac{\exp l_1}{\exp l_1 + \exp l_2}, \frac{\exp l_2}{\exp l_1 + \exp l_2})$
- We hope that these probabilities p match the true probabilities y and we have the one-hot representation of the output word $\mathbf{e}_{w_o} = y$
- I.e. We hope that $p_{w_o} \sim 1$ and p is zero elsewhere

Word Embeddings: CBOW Algorithm

- Given that we want $p \sim y$, we can focus on minimizing the loss (Cross-Entropy)

$$Cross - Entropy = -\sum_{i=1}^{|V|} e_i \log p_i$$

- Note that, since e is the one-hot vector of w_o; the above is actually just one term of the type $-\log p_j$ where j is the index of w_o in V; $stoi[w_o] = j$
- If $p_j \sim 1$, as it should be, we get a 0 penalty
- If $p_j \sim 0$, we get a $+\infty$ penalty!
- If $p_j \sim \frac{1}{10}$, we get a loss of $-\log \frac{1}{10} = 4.605$
  - *In either of the last two cases, we've given too low a probability to something that should have a high probability so the loss will be large*

Word Embeddings: CBOW Algorithm

- The loss of a general output word w_o and fixed window around it is:

$$
\begin{aligned}
Cross - Entropy &= -\log P(w_o | w_{o-m}, \ldots, w_{o-1}, w_{o+1}, \ldots, w_{o+m}) \\
&= -\log P(w_o | a_{avg}) \\
&= -\log \frac{\exp b_{w_o}^\intercal a_{avg}}{\sum_{w \in V} \exp b_w^\intercal a_{avg}} \\
&= -b_{w_o}^\intercal a_{avg} + \log \sum_{w \in V} \exp b_w^\intercal a_{avg}
\end{aligned}
$$

- The loss of this model is the loss across the entire training data by windowing in this way and averaging over the number of windows
- The parameters of this model are A and B
- We can use SGD to update the parameters of this model
  - $A_{new} \leftarrow A_{old} - \alpha \nabla_A L$
  - $B_{new} \leftarrow B_{old} - \alpha \nabla_B L$

Word Embeddings: CBOW Gradients

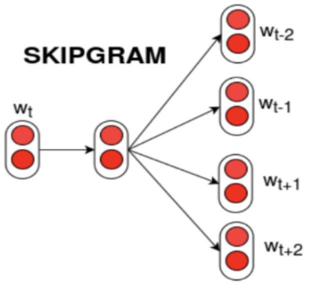
- The loss of a general model looks like an average of $Cross-Entropy$, where we divide by the number of windows (we'll see how to get the training data when we look at code)
- I'll ask you to write this down in the next HW

Outline

- Basic Word Embeddings
- CBOW Model
- Skip-Gram Model
- Doc2Vec
- PyTorch Examples

Word Embeddings: Skip-Gram Model

- Another approach takes the reverse of the CBOW model: given a center word, predict the words around this word
- As with CBOW, you have parameter embedding matrix A, B as before



Word Embeddings: Skip-Gram Model

- Again: given a set of sentences S, tokenize at the word level and get the vocabulary V and fix a window size m
- Get, from all sentences, all center (context) input words c and output words o pairs, gathering pairs like

$$(w_c, w_{c-m}), \ldots, (w_c, w_{c+m})$$

  so that output words are within m distance of the context word w_c (on either side)
  - Note that in this case the center word is a feature (x) and the output word is the target (y)
- For each context center word, get the one-hot representations e_{w_c}
- For each context center word, get the vector representations

$$a_{w_c} = A^\intercal e_{w_c}$$

Word Embeddings: Skip-Gram Model

- Generate a score $l = Ba_{w_c} \in \mathbf{R}^{|V|}$
- Turn the scores into probabilities $p = \text{softmax}(l) \in \mathbf{R}^{|V|}$
- We have that $p_{w_{c-m}}, p_{w_{c-m+1}}, \ldots, p_{w_{c+m-1}}, p_{w_{c+m}}$, are the probabilities of observing the context words given the center word w_c
- As with CBOW, we want the probabilities above to match the one-hot vectors $e_{w_{c-m}}, e_{w_{c-m+1}}, \ldots, e_{w_{c+m-1}}, e_{w_{c+m}}$
- Specifically, for a fixed center word w_c we have loss terms equal to

$$-\sum_{i=-m, i\neq 0}^{m} e_{w_{c+i}} \log p_{w_{c+i}}$$

Word Embeddings: Skip-Gram Model

- We can spell-out the above more by noticing that in our model we want to minimize:

$$
\begin{aligned}
Cross - Entropy &= -\log P(w_{c-m}, w_{c-m+1}, \ldots, w_{c+m-1}, w_{c+m} | w_c) \\
&= -\log \prod_{i=-m, i\neq 0}^{m} P(w_{c+i}|w_c) \\
&= -\log \prod_{i=-m, i\neq 0}^{m} \frac{\exp b_{w_{c+i}}^{\mathsf{T}} a_{w_c}}{\sum_{j=1}^{|V|} \exp b_{w_j}^{\mathsf{T}} a_{w_c}} \\
&= -\sum_{i=-m, i\neq 0}^{m} b_{w_{c+i}}^{\mathsf{T}} a_{w_c} + 2m \log \sum_{j=1}^{|V|} \exp b_{w_j}^{\mathsf{T}} a_{w_c}
\end{aligned}
$$

Word Embeddings: Skip-Gram Model

- Note that in the above, we've used the *Naive Bayes's* assumption $P(A,B|C) = P(A|C)P(B|C)$ repeatedly
- Note that in actuality we have that

$$- \sum_{i=-m, i \neq m}^{m} \log P(b_{w_{c-m+i}} | a_{w_c})$$

is the sum of $2m$ terms

$$\sum_{i=-m, i \neq 0}^{m} Cross - Entropy(e_{c-m+i}, softmax(Ba_{w_c}))$$

Word Embeddings: Skip-Gram Gradients

- So, a general term of the Skip-Gram objective looks like

$$Cross-Entropy = -\log P(b_{w_o}|a_{w_c})) = -b_{w_o}^{\intercal} a_{w_c} + \log \sum_{w \in V} \exp b_w^{\intercal} a_{w_c}$$

- We have (HW!)
  - $\frac{\partial L}{\partial b_{w_o}} = -a_{w_c} + \frac{a_{w_c} \exp b_{w_o}^{\intercal} a_{w_c}}{\sum_{u \in V} \exp b_u^{\intercal} a_{w_c}}$
  - $\frac{\partial L}{\partial a_{w_c}} = -b_{w_o} + \sum_{w \in V} b_w \frac{\exp b_w^{\intercal} a_{w_c}}{\sum_{u \in V} \exp b_u^{\intercal} a_{w_c}}$
    - Notice that this is $-b_{w_o} + E[b_w]$ where the expectation is using $P(w) = \frac{\exp b_w^{\intercal} a_{w_c}}{\sum_{u \in V} \exp b_u^{\intercal} a_{w_c}}$
    - Maybe Gradient = - Hard Guess + Expected Guess

Word Embeddings: Skip-Gram and CBOW issues

- One problem with the Skip-Gram and the CBOW model is the (normalizing) summation $\sum_{w \in V} \exp b_w^\top a_{w_c}$ which is due to two issues
- First, this term is the softmax denominator so if we want to use the softmax as is we need it
- **Idea:** Instead of being exact, approximate

Word Embeddings: Negative Sampling

- Consider the Skip-Gram model and a center word and center word w_c and some context word w_o which we know that $P(w_o|w_c)$ should be nonzero
- In this setting, we have that (w_c, w_o) is a positive $(+1)$ sample, so we can use the sigmoid to approximate

$$P(w_o|w_c) \sim \frac{1}{1 + \exp -b_{w_o}^{\intercal} a_{w_c}}$$

- Similarly, we sample K words w_k that are not in the context and we know these have a negative label (-1), so

$$P(w_k|w_c) \sim \frac{1}{1 + \exp b_{w_k}^{\intercal} a_{w_c}}$$

Word Embeddings: Negative Sampling

- In particular, we replace

$$P(w_o|w_c) = \frac{\exp \mathbf{u}_{w_o}^{\mathsf{T}} \mathbf{v}_{w_c}}{\sum_{j=1}^{|V|} \exp \mathbf{u}_{w_j}^{\mathsf{T}} \mathbf{v}_{w_c}}$$

  by

$$P(w_o|w_c) = (\frac{1}{1 + \exp -\mathbf{u}_{w_o}^{\mathsf{T}} \mathbf{v}_{w_c}}) E_{w_k \sim P(w)} [\prod_{k=1}^{K} \frac{1}{1 + \exp \mathbf{u}_{w_k}^{\mathsf{T}} \mathbf{v}_{w_c}}]$$

- We sample each word w_k from the empirical distribution of the words in the corpus
- With high probability, for a given w_c, these words will not be in the context of w_c

Word Embeddings: Negative Sampling

- With this approximation, we replace the original term in the objective

$$-\mathbf{u}_{w_o}^\intercal \mathbf{v}_{w_c} + \log \sum_{j=1}^{|V|} \exp \mathbf{u}_{w_j}^\intercal \mathbf{v}_{w_c}$$

by

$$-\log \frac{1}{1 + \exp -\mathbf{u}_{w_o}^\intercal \mathbf{v}_{w_c}} - E_{w_k \sim P(w)}[\sum_{k=1}^{K} \log \frac{1}{1 + \exp \mathbf{u}_{w_k}^\intercal \mathbf{v}_{w_c}}]$$

- **Instead of having $O(|V|)$ complexity, we now have $O(K)$**
- Note that this is easily written with sigmoids (σ) as

$$-\log \sigma(\mathbf{u}_{w_o}^\intercal \mathbf{v}_{w_c}) - E_{w_k \sim P(w)}[\sum_{k=1}^{K} \log \sigma(-\mathbf{u}_{w_k}^\intercal \mathbf{v}_{w_c})]$$

Word Embeddings: Negative Sampling + Dropout

- Sampling the K negative samples is an art
- The authors of Skip-Gram used $P_{sample}(w) \propto freq(w)^{\frac{3}{4}}$
- The 3/4 has the effect of pushing less frequent words up more than frequent words
  - "cactus": $0.09 \rightarrow 0.09^{3/4} = 0.16$
  - "the": $0.9 \rightarrow 0.9^{3/4} = 0.92$
- Before doing anything through, the authors also drop words from the corpus
- $P_{keep}(w) = (\sqrt{\frac{z(w)}{0.001}} + 1)\frac{0.001}{z(w)}$ where $z(w) \propto freq(w)$ in the original corpus
- After you drop words, you compute P_{sample} as above

Hierarchical Softmax : Another trick

- Hierarchical softmax is an efficient way of computing softmax. The model uses a binary tree to represent all words in the vocabulary. The V words must be leaf units of the tree
- For each leaf unit, there exists a unique path from the root to the unit; and this path is used to estimate the probability of the word represented by the leaf unit.
- Main advantage: we replace the $O(|V|)$ softmax cost with $O(\log |V|)$
- The speed of this method depends on how the binary tree is built and words are associated with the leaf nodes; the paper uses a binary Huffman tree
- Good project?

Hierarchical Softmax

- Neat thing about thes vectors is they see to have vector properties
- $Italy - Rome + Lisbon = Portugal$!
- But they also have issues: $Doctor - Man + Women = Nurse$ (Bias!)

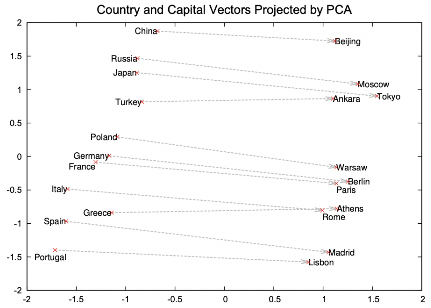


Figure 2: Two-dimensional PCA projection of the 1000-dimensional Skip-gram vectors of countries and their capital cities. The figure illustrates ability of the model to automatically organize concepts and learn implicitly the relationships between them, as during the training we did not provide any supervised information about what a capital city means.

Outline

- Basic Word Embeddings
- CBOW Model
- Skip-Gram Model
- Doc2Vec
- PyTorch Examples

Doc2Vec from Word2Vec

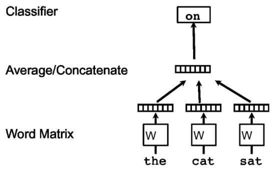


Figure 1. A framework for learning word vectors. Context of three words ("the," "cat," and "sat") is used to predict the fourth word ("on"). The input words are mapped to columns of the matrix W to predict the output word.

- Task: How do we learn a vector representation for a paragraph?
- One idea: take the word vectors for each word in the document, and average them
- This produces one vector, a summary of the document
- Can we do better?

Doc2Vec - Model

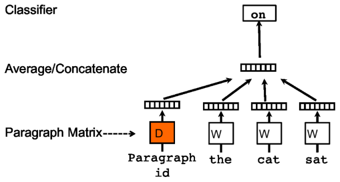


Figure 2. A framework for learning paragraph vector. This framework is similar to the framework presented in Figure 1; the only change is the additional paragraph token that is mapped to a vector via matrix D. In this model, the concatenation or average of this vector with a context of three words is used to predict the fourth word. The paragraph vector represents the missing information from the current context and can act as a memory of the topic of the paragraph.

- Simple idea: add a paragraph "token" and add this to the context information used to predict the "next" word.
- The only difference between this model and the (CBOW) Word2Vec type models is that now you have a new set of vectors D which you are inserting
- The parameters of this model are the document vectors D and the usual other parameters word vectors A, B
- A can be viewed as the word vector

Doc2Vec - Inference

- What happens at inference? How do we get an embedding for a "new" paragraph?
- Add new rows to the D matrix.
- Consider the likelihood with all parameters being fixed except the new ones.
- Minimize the log likelihood across these new parameters.
- Use these new vectors as the paragraph level vectors!

Doc2Vec - SkipGram version

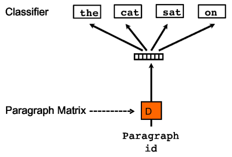


Figure 3. Distributed Bag of Words version of paragraph vectors. In this version, the paragraph vector is trained to predict the words in a small window.

- Another alternative is to predict the document words from the document, kind of like Skip-Gram!
- This model has less parameters: we just have the document vectors and the "out" word vectors due to the softmax.
- The "document" vector can be the average of the vector gotten from the CBOW and Skip-Gram Doc2Vec variants.

Doc2Vec - Experiments

- They test their method on Sentiment Analysis tasks and tasks having to do with information retrieval (Which document is relevant to this query = "Chocolate milk")
- They do tests where documents are just 1 sentence (SST) vs a few sentences (IMDB reviews)

Doc2Vec - Experiments

- From this, they learn the document and word vectors and then fit a logistic classifier (using the document vector) to predict the sentiment for each document
- At inference time, they learn the new document vectors by optimizing the Doc2Vec likelihood to get the document vectors with everything else frozen
- Then, they feed the (new) document vector to the classifier to get the prediction

Doc2Vec - Experiments

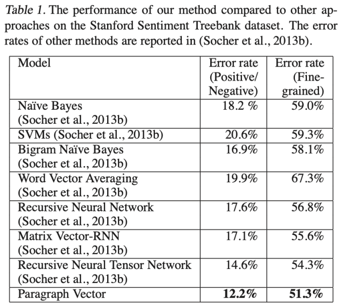


Table 1. The performance of our method compared to other approaches on the Stanford Sentiment Treebank dataset. The error rates of other methods are reported in (Socher et al., 2013b).

Model	Error rate (Positive/ Negative)	Error rate (Fine-grained)
Naïve Bayes (Socher et al., 2013b)	18.2 %	59.0%
SVMs (Socher et al., 2013b)	20.6%	59.3%
Bigram Naïve Bayes (Socher et al., 2013b)	16.9%	58.1%
Word Vector Averaging (Socher et al., 2013b)	19.9%	67.3%
Recursive Neural Network (Socher et al., 2013b)	17.6%	56.8%
Matrix Vector-RNN (Socher et al., 2013b)	17.1%	55.6%
Recursive Neural Tensor Network (Socher et al., 2013b)	14.6%	54.3%
Paragraph Vector	**12.2%**	**51.3%**

Doc2Vec - Experiments

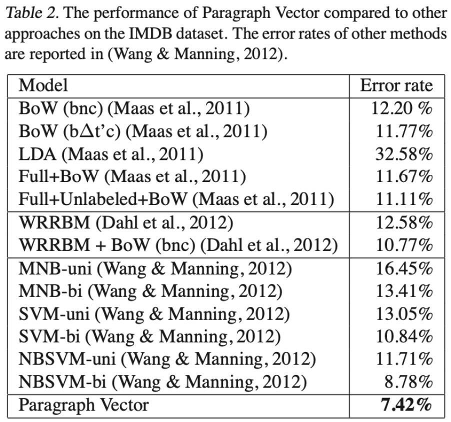


Table 2. The performance of Paragraph Vector compared to other approaches on the IMDB dataset. The error rates of other methods are reported in (Wang & Manning, 2012).

Model	Error rate
BoW (bnc) (Maas et al., 2011)	12.20 %
BoW (bΔt'c) (Maas et al., 2011)	11.77%
LDA (Maas et al., 2011)	32.58%
Full+BoW (Maas et al., 2011)	11.67%
Full+Unlabeled+BoW (Maas et al., 2011)	11.11%
WRRBM (Dahl et al., 2012)	12.58%
WRRBM + BoW (bnc) (Dahl et al., 2012)	10.77%
MNB-uni (Wang & Manning, 2012)	16.45%
MNB-bi (Wang & Manning, 2012)	13.41%
SVM-uni (Wang & Manning, 2012)	13.05%
SVM-bi (Wang & Manning, 2012)	10.84%
NBSVM-uni (Wang & Manning, 2012)	11.71%
NBSVM-bi (Wang & Manning, 2012)	8.78%
Paragraph Vector	**7.42%**

Doc2Vec - Experiments

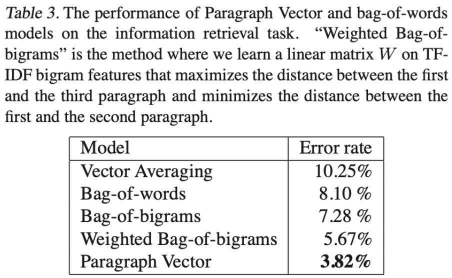


Table 3. The performance of Paragraph Vector and bag-of-words models on the information retrieval task. "Weighted Bag-of-bigrams" is the method where we learn a linear matrix W on TF-IDF bigram features that maximizes the distance between the first and the third paragraph and minimizes the distance between the first and the second paragraph.

Model	Error rate
Vector Averaging	10.25%
Bag-of-words	8.10 %
Bag-of-bigrams	7.28 %
Weighted Bag-of-bigrams	5.67%
Paragraph Vector	**3.82%**

- For Information retrieval, they have a sort of Siamese model where they sample 3 documents and the similarity between the first two should be higher than the similarity between the first and the third
- This also has some nice positive results

Outline

- Basic Word Embeddings
- CBOW Model
- Skip-Gram Model
- Doc2Vec
- PyTorch Examples

References

- Efficient Estimation of Word Representations in Vector Space
- Distributed Representations of Words and Phrases and their Compositionally
- Doc2Vec
- Chris McCormick's BLOG on Word2Vec
- CBOW Negative Sampling