Introduction to Deep Learning (IT3320E)

9 - Deep Learning for Natural Language Processing

Hung Son Nguyen

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Nov. 22, 2022

Agenda

Section 1

Introduction

Examples of low-level NLP problems/tasks

Sequence tagging

- POS-tagging (part of speech)

Example

Time flies like an arrow.

Fruit flies like a banana.

Example

Time$_{NN}$ flies$_{VBZ}$ like$_{IN}$ an$_{DT}$ arrow$_{NN}$.

Fruit$_{NN}$ flies$_{NN}$ like$_{VB}$ a$_{DT}$ banana$_{NN}$.

IN = Preposition or subordinating conjunction (conjunction here); VBZ = Verb, 3rd person singular present; DT = determiner; NN = singular noun

Low-level NLP problems/tasks

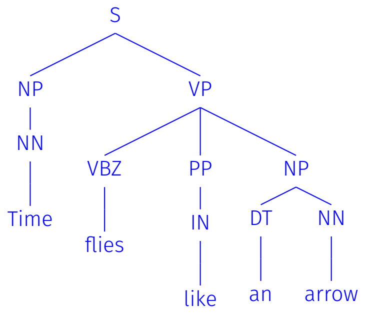
Sequence tagging

- POS-tagging (part of speech)

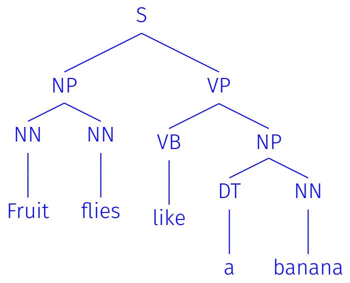
Structure prediction

- Chunking/Parsing

Example



Example



Examples of low-level NLP problems/tasks

Sequence tagging

- POS-tagging (part of speech)

Structure prediction

- Chunking/Parsing

Semantics

- Word sense disambiguation

Example

Time flies like an arrow.

Fruit flies like a banana.

High-level NLP tasks

- Information Extraction
  - search, event detection, textual entailment
- Writing Assistance
  - spell checking, grammar checking, auto-completion
- Text Classification
  - spam, sentiment, author, plagiarism
- Natural language understanding
  - metaphor analysis, argumentation mining, question-answering
- Natural language generation
  - summarization, tutoring systems, chat bots
- Multilinguality
  - machine translation, cross-lingual information retrieval

Example: Named Entity Recognition (NER) 1

Example

"... chancelor$_{O}$ Angela$_{B\text{-}PER}$ Merkel$_{I\text{-}PER}$ said$_{O}$ ..."

"BIO"-tagging

- B = Begin of entity, e.g., $B\text{-}PER$ (person), $B\text{-}LOC$ (location)
- I = "Inside" entity, e.g, $I\text{-}PER$, $I\text{-}LOC$
- O = Other (no entity)

Traditional feature engineering

Example – determining tag for *Angela*

"... chancelor$_{O}$ **Angela**$_{B-PER}$ Merkel$_{I-PER}$ said$_{O}$ ..."

Each token is represented by a feature vector

Feature type	Value
uppercase	1
is Noun	1
previousWord DET	0
previousWord uppercased	0
following word N	1
following word uppercased	1
...	...

Traditional feature engineering (cont.)

Example – determining tag for *Merkel*

"… chancelor$_{O}$ Angela$_{B-PER}$ **Merkel**$_{I-PER}$ said$_{O}$ …"

Each token is represented by a feature vector

Feature type	Value
uppercase	1
is Noun	1
previousWord DET	0
previousWord uppercased	1
following word N	0
following word uppercased	0
…	…

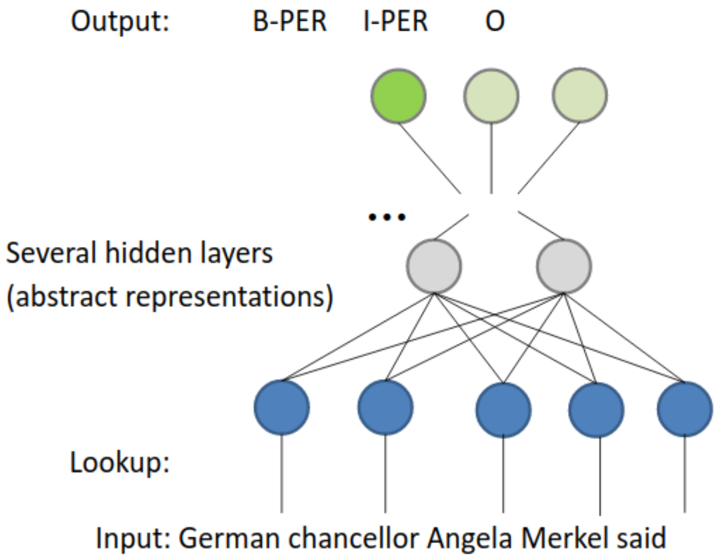
Traditional feature engineering (cont.)

Word specific features

- previous word is minister
- previous word is chancellor
- previous word is president
- previous word is company
- previous word is product
- following word is says
- following word is declares
- following word is claims

Problem: Similar words are represented as distinct features.

DL for NER


Output:
B-PER
I-PER
O
...
Several hidden layers
(abstract representations)
Lookup:
Input: German chancellor Angela Merkel said

Architecture of neural networks: Lecture 3/8-

Mapping words into vector representations: Lecture 4-7

Section 2

Language models

Language Models (LMs)

Language modeling = assigning probability to a sentence in a language

Example: $\Pr(\text{‘The cat sat on the mat .’})$

Ideal performance at language modeling: predict the next token in a sequence with a number of guesses that is the identical to or lower than the number of guesses required by a human

Crucial component in real-world NLP applications: conversational AI, machine-translation, text summarization, etc.

Language Models (LMs)

Assume a sequence of words $w_{1:n} = w_1 w_2 \ldots w_{n-1} w_n$

$$\begin{aligned}\Pr(w_1, w_2, \ldots, w_{n-1}, w_n) = & \Pr(w_1)\cdot \\ & \Pr(w_2|w_1)\cdot \\ & \Pr(w_3|w_1, w_2)\cdot \\ & \Pr(w_4|w_1, w_2, w_3)\cdot \\ & \ldots \cdot \Pr(w_n|w_{1:n-1})\end{aligned}$$

Each word is predicted conditioned on the preceding words

Probability of the last token conditioned on $n-1$ preceding words

Markov assumption

kth order Markov-assumption: next word in a sequence depends only on the last k words

$$\Pr(w_i | w_{1:i-1}) \approx \Pr(w_i | w_{i-k:i-1})$$

Probability of a sequence of tokens $w_{1:n}$

$$\Pr(w_{1:n}) \approx \prod_{i=1}^{n} \Pr(w_i | w_{i-k:i-1})$$

Computationally-friendly version (What is the problem with the above formula?)

$$\log_2 \Pr(w_{1:n}) \approx \sum_{i=1}^{n} \log_2 \left(\Pr(w_i | w_{i-k:i-1})\right)$$

Evaluating language models

Perplexity: How well a LM predicts likelihood of unseen sentence

$$\text{Perp}_{w_{1:n}}(\text{LM}) = 2^{-\frac{1}{n}\sum_{i=1}^{n} \log_2 \text{LM}(w_i | w_{1:i-1})}$$

Low perplexity values $\rightarrow$ better language model (assigns high probabilities to the unseen sentences)

We can compare several language models with one another

Perplexities of two language models are only comparable with respect to the same evaluation dataset

Estimating Probabilities

Count-based: The estimates can be derived from corpus counts

Let $\#(w_{i:j})$ be the count of the sequence of words $w_{i:j}$ in a corpus

The maximum likelihood estimate (MLE) of $\Pr(w_i | w_{i-1-k:i-1})$

$$\Pr(w_i | w_{i-1-k:i-1}) = \frac{\#(w_{i-1-k:i})}{\#(w_{i-1-k:i-1})}$$

Example: $w_1 w_2 w_3 =$ the cat sat

$$\Pr(w_3 | w_{1:2}) = \frac{\#(\text{the cat sat})}{\#(\text{the cat})}$$

What if $\#(w_{i:j}) = 0$?

$\rightarrow$ infinite perplexity

One way of avoiding zero-probability N-grams is to use smoothing techniques

Pro and Cons of Discussed LMs

+ Easy to train, scale to large corpora, and work well in practice
- Scaling to larger N-grams is a problem for MLE-based language models.
- Large number of words in the vocabulary means that statistics for larger N-grams will be sparse
- MLE-based language models suffer from lack of generalization across contexts
~ Having observed “black car” and “blue car” does not influence our estimates of the sequence “red car” if we haven’t seen it before

Neural Language Models

Use neural networks to estimate probabilities of a LM

We can overcome the shortcomings of the MLE-based LMs because neural networks

- enable conditioning on increasingly large context sizes with only a linear increase in the number of parameters
- support generalization across different contexts

We focus on the neural LM that was introduced by (**Bengio.et.al.2003.JMLR**)

Neural Language Models

Let $w_{1:k}$ be the given context

We want to estimate $\Pr(w_{k+1}|w_{1:k})$

Design an MLP neural net, which takes $w_{1:k}$ as input and returns $\Pr(w_{k+1})$ over all words in vocabulary V as output

$$
\begin{aligned}
x &= [v(w_1), v(w_2), \ldots, v(w_k)] \\
h^{(1)} &= g(xW^{(1)} + b^{(1)}) \\
\Pr(w_{k+1}) &= \text{softmax}(h^{(1)}W^{(2)} + b^{(2)})
\end{aligned}
$$

Training examples: word k-grams from the training set, where the identities of the first $k-1$ words are used as features. Last word: target label for the classification

Loss function: cross-entropy loss

(Neural) LMs for language generation

Assume that we are given $w_{1:k}$ as context

Predict the next word w_{k+1} from vocabulary V

Feed $w_{1:k}$ to our trained MLP-based LM

Our LM returns $\Pr(w_{k+1})$ of each word in V

Pick the word with the maximum probability to generate the next word

Add the the predicted word to the context and repeat the above procedure

Section 3

Word embeddings

What’s the problem

$$\begin{tikzcd} x_{1:n} \arrow[dr] & & \\ y_{1:n} \arrow[r] & \boxed{\text{learning algorithm}} \arrow[r] & f() \end{tikzcd}$$

Input to neural models $f()$: numerical vector but NLP work with texts

How can we represent texts via numerical vectors?

This lecture

Text Representation

Vector representation of a text should ideally reflect various linguistic properties of the text

$$\begin{tikzcd}\text{text} \arrow[r] & \boxed{\text{feature function}} \arrow[r] & x \arrow[r] & f()\end{tikzcd}$$

In this lecture we focus on feature functions rather than learning algorithms

Terminology

Letters: smallest units in a language $\longrightarrow$ a,b,c,...

Tokens and **words**: tokens are outputs of a tokenizer and words are meaning-bearing units

- note: in this course we use the terms “word” and “token” interchangeably

Lemma: the dictionary entry of the word $\longrightarrow$ “book” is the lemma of “booking”, “booked”, and “books”

- How to obtain lemmas? Use morphological analyzers
- Available for many languages
- Lemmatization may not work for any sequence of letters, e.g., for mis-spelling

Stems: a shorter venison of a word defined based on some language-specific heuristic $\longrightarrow$ “pictur” is the stem of “pictures”, “pictured”, and “picture”

- Stemmer’s output need not be a valid word

lexical resources

Lexical resources: provide information about words and their relations

Dictionaries

WordNet for English

- Semantic knowledge about words
- Each word is associated with one or several synsets
- Synsets are linked to each other according to semantic relations between their words
- Semantic relations are: hypernym (more specific), hyponym (less specific), antonyms, etc.
- Contains nouns, verbs, adjectives, and adverbs
- Manually created

Outline

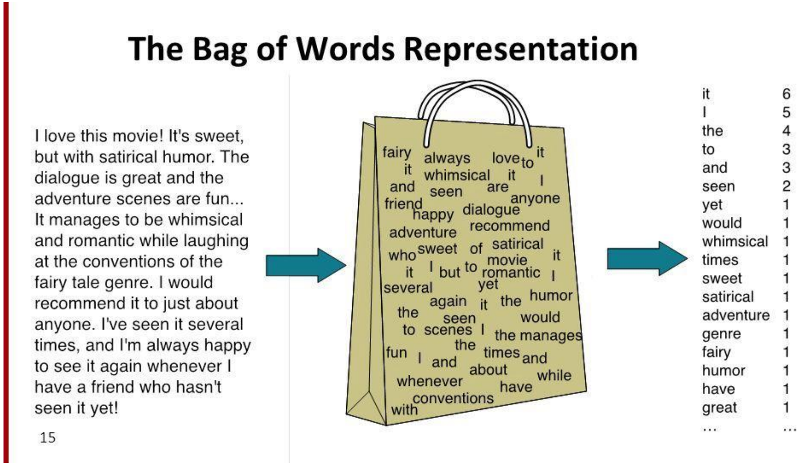
Common Features for Textual Data

What information can we extract directly from a word

- Letters comprising a word and their order
- Length of word
- Is the first letter capitalized?
- Does word include hyphen?

Bag-of-Words (BoW): histogram of words as a feature vector.

- BOW does not care about the order of words



The Bag of Words Representation

I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!

15

it	6
I	5
the	4
to	3
and	3
seen	2
yet	1
would	1
whimsical	1
times	1
sweet	1
satirical	1
adventure	1
genre	1
fairy	1
humor	1
have	1
great	1
...	...

(*Taken from:* *https://www.programmersought.com/article/4304366575/)*

TF-IDF

TF-IDF: Term Frequency - Inverse Document Frequency

Let d be a document from a given corpus D

Map word w of d to a number as follows:

$$\frac{\#(w, d)}{\sum_{w'} \#(w', d)} \times \log \frac{|D|}{|\{d \in D : w \in d\}|}$$

N-grams: instead of using the frequency of a word, we use the frequency of N sequence of words

N=2 $\rightarrow$ bi-gram; N=3 $\rightarrow$ tri-gram

Common Features for Textual Data

Context: each piece of a text occurs within a larger text which is known as context

How to encode contextual information?

– Using position of a word within a sentence or a document

– Using the words that appear in an immediate context of a word

- Immediate context: a window of k words surrounding the target word
- "the cat sat on the mat" , k=2 $\rightarrow$ { word-minus-2=the, word-minus-1=cat, word-plus-1=on, word-plus-2=the }

Common Features for Textual Data

what information can we extract from the relation of text with external source of information?

- is a word in a list of common person names in Germany?
- is a word a female or male person name?
- what is the lemma of the word?
- what is the stem of the word?
- what information do lexical resources give us about the word?

Different Types of Features

Numerical features

- Value of a feature is a number, e.g., the frequency of a word in a text

Categorical features

- Value of a feature is from a set of values
- ”What is the POS of a word? ”

How to encode categorical features?

- One-hot encoding
- Dense embedding

One-hot Encoding

Assume that values of a feature is in categories $\{v_0, v_1, v_2\}$

Encode the space of feature values via vectors in which

- each entry is either 0 or 1
- each entry is associated with one of the categories
- only one item in a vector can be 1, the rest is 0

Example:

- $v_0 = [1, 0, 0]$
- $v_1 = [0, 1, 0]$
- $v_2 = [0, 0, 1]$

Example: Categorical Labels

Fet $f()$ take a vector x which encodes a text and also return a vector $\hat{y}$ representing the topic of the text which can be from $\{$news, sport, science, politic$\}$

How can we represent topic labels using one-hot encoding?

- news = $[1, 0, 0, 0]$
- sport = $[0, 1, 0, 0]$
- science = $[0, 0, 1, 0]$
- politic = $[0, 0, 0, 1]$

What is the length of one-hot vectors for a feature with k categories?

Example: Word Encodings

Let V be vocabulary of a language

We take V as categories a word can take in a text

Let $V = \{v_0, v_1, v_2, ..., v_{|V|-1}\}$

- $v_0 = [1, 0, 0, ..., 0]$
- $v_1 = [0, 1, 0, ..., 0]$
- $v_2 = [0, 0, 1, ..., 0]$
- $\ldots$
- $v_{|V|-1} = [0, 0, 0, ..., 1]$

We can easily represent a text via BOW and then represent each word in BOW with its one-hot vector

How can we define vocabulary V given a corpus D?

One-hot Encodings

- word vectors are very sparse
- semantic relations between words are not encoded in word vectors
- it’s better to use one-hot representations for a few distinct features where we expect no correlation between features

Dense Encodings

- a categorical feature is embedded as a vector in a d dimensional space
- assuming categories $C = \{c_0, c_1, ..., c_{|C|-1}\}$, $d = 4$
  - $c_0 = [+0.1, -0.2, +0.3, +0.5]$
  - $c_1 = [-0.2 - 0.1, +0.1, +0.2]$
  - ...
  - $c_{|C|-1} = [+0.2, -0.2, -0.1, +0.3]$

Example: Word Encodings

- assuming vocabulary $V = \{v_0, v_1, ..., v_{|V|-1}\}$, $d = 4$
  - $v_0 = [+0.1, -0.2, +0.3, +0.5]$
  - $v_1 = [-0.2 - 0.1, +0.1, +0.2]$
  - ...
  - $v_{|V|-1} = [+0.2, -0.2, -0.1, +0.3]$
- in the one-hot method we represent each word with a sparse vector of size $|V|$
- in dense encoding method we represent each word with a dense vector with a small size d

Dense Encodings

- dimensionality of vectors is d
- model training will cause similar features to have similar vectors
- it's mainly useful when we expect some correlations between features $\rightarrow$ "cat" and "dog" are semantically related
- is also useful when we have a large number of features $\rightarrow$ for example vocabulary

Outline

Word Embeddings

- dense representations of words in an embedding space is known as word embeddings
- how to obtain word embeddings?

Random Initizalization

- we initialize the embedding vectors to random values
- values are uniformly sampled from numbers in the range
  - $[-\frac{1}{2d}, +\frac{1}{2d}]$ where d is the number of dimensions (Mikolov et al., 2013)
  - $[-\frac{\sqrt{6}}{\sqrt{d}}, +\frac{\sqrt{6}}{\sqrt{d}}]$ (xavier initialization)

Word Embeddings

- dense representations of words in an embedding space is known as word embeddings
- these vectors can be obtained by random initialization and tuned for any NLP task
- however, we as human beings can define semantic relations between words independent of any NLP task
  - e.g., “blue” and ’black” are colors
  - e.g., “dog” is more similar to “cat” than to “chair”
  - e.g., “easy” is the opposite of “difficult”
- how can we find word embeddings such that vectors of words with similar meaning be close to each other in the embedding space?

Distributional Hypothesis

- words that occur in the same contexts tend to have similar meanings (Harris, 1945)
- how can we model the distributional hypothesis?

Word-Context Matrix

- we count how often a word has occurred with a context in texts from a large corpus
- context is defined by a window over words
- let V be the set of words in vocabulary and C be the set of possible contexts
- word-context matrix M is a two dimensional matrix whose rows are associated with V and columns with C
- each entry of the matrix indicates how often a word co-occurs with a context in the given corpus
- this matrix is also known as co-occurrence matrix

Outline

Word-Context Matrix

- corpus:
  - I like DL
  - I like NLP
  - I love ML
  - I love NLP
- window size = 1

	I	like	love	DL	NLP	ML
I	0	2	2	0	0	0
like	2	0	0	1	1	0
love	2	0	0	0	1	1
DL	0	1	0	0	0	0
NLP	0	1	1	0	0	0
ML	0	0	1	0	0	0

Word-Context Matrix

- corpus:
  - I like DL
  - I like NLP
  - I love ML
  - I love NLP
- window size = 1

	I	like	love	DL	NLP	ML
I	0	2	2	0	0	0
like	2	0	0	1	1	0
love	2	0	0	0	1	1
DL	0	1	0	0	0	0
NLP	0	1	1	0	0	0
ML	0	0	1	0	0	0

Count-based Embeddings

- for a large corpus, the size of matrix is very large
- the matrix could be sparse too
- to encounter this, we may use dimensionality reduction techniques
- however, for adding a new word we need to enlarge the matrix and apply dimensionality reduction again

CBoW Method

- CBoW stands for Continuous Bag of Words
- task: given a context $\rightarrow$ predict a missing word from the context

input: “I _ NLP” $\rightarrow$ output: “like”

input: (“I _ NLP”, “like”) $\rightarrow$ output: 1 and
input: (“I _ NLP”, “apple”) $\rightarrow$ output: 0

Outline

CBoW Method

- given a corpus, we create two sets:
  - positive set (D): consisting of pairs (c, w) where c is a context and w is the correct value for the missing word
  - negative set (D'): consisting of pairs (c, w) where c is a context and w is a random value for the missing word
- we compute the score estimating similarity between context c and word w given context-word pair (c, w)

$$s(c, w) = e(w) \sum_{w_i \in c} e(w_i)$$

where e is a function that maps each word to its embeddings

- Example

$$s(\text{I \_ NLP}, \text{like}) = e(\text{like})(e(\text{I}) + e(\text{NLP}))$$

CBoW Method

Use the sigmoid function to map the score to a probability

$$P(y = 1|(c, w)) = \frac{1}{1 + e^{-s(c,w)}}$$

Minimize the following loss

$$L(\Theta) = -\frac{1}{|D|} \sum_{(c,w)\in D} \log \Pr(y = 1|(c, w)) \\ - \frac{1}{|D'|} \sum_{(c,w)\in D'} \log \Pr(y = 0|(c, w))$$

Skip-Gram Method

Treat words of a context independent from each other

$$P(y=1|(c,w)) = \prod_{c_i \in c} P(y=1|(w,c_i)) = \prod_{c_i \in c} \frac{1}{1+e^{-e(w)e(c_i)}}$$

Loss in Skip-Gram is identical to that in CBoW

We fine-tune parameters (word embeddings) using SGD (Stochastic Gradient Descent)

CBoW and Skip-Gram

- CBoW loses the order information between context's elements
- CBoW allows the use of variable-length contexts
- Skip-Gram decouples the dependence between context elements even further than CBoW
- Skip-Gram treats each context's element as an independent context

Context Matters!

- context of a word translates to its surrounding words in a sentence or paragraph
- window-based context: for a word we consider all words in a window of length $2m + 1$ centred on the word as context
- the size of window: large windows tend to capture more topical similarities and small windows capture syntactic similarities
- contexts can be defined based on text units, for example, words that occur in the same sentence, paragraph, or text
- contexts can also be defined based on syntax of a sentence for example using parse tree or dependency tree of a sentence
- contexts of a word can be foreign words that are aligned to the word in multilingual corpora

Outline

Evaluating Word Embeddings

- intrinsic
    - word similarity tasks
    - word analogy tasks
- extrinsic
    - on a downstream NLP task, we compare the performance of two models that differ only in the word embeddings they use
    - named entity recognition (NER): accuracy
    - machine translation (MT): BLEU score
    - summarization: ROUGE score
    - information retrieval (IR): precision - recall - F1 score

Word Similarity Tasks

- similar words should have similar representations
- dataset: *http://alfonseca.org/eng/research/wordsim353.html*
- word-1 and word-2 $\rightarrow$ similarity score $\in$ [0, 10]
- our function $f()$ should map two words to a similarity score using the distance between the word vectors of the words
- cosine similarity

$$\text{sim}(w_i, w_j) = \frac{e(w_i) \cdot e(w_j)}{\|e(w_i)\| \|e(w_j)\|}$$

Word Analogy Tasks

- A is to B as C to ?
- Germany is to Berlin as France is to ?
- dataset: *http://download.tensorflow.org/data/questions-words.txt*
- capital-common-countries
  - Athens Greece Baghdad $\longrightarrow$ Iraq
  - Athens Greece Berlin $\longrightarrow$ Germany
- family
  - boy girl brother $\longrightarrow$ sister
  - brother sister dad $\longrightarrow$ mom
- currency, adj-to-adverb, comparative, ...

Finding the Prototype of a Group of Words

- if we have a group of words $g = \{w_1, w_2, ..., w_k\}$
- the prototype of this group can be computed as follows

$$\text{proto}(g) = \frac{1}{k} \sum_{i \in 1..k} e(w_i)$$

where e maps a word to its embeddings

Finding Similar Words

- cosine similarity

$$\text{sim}(w_i, w_j) = \frac{e(w_i) \cdot e(w_j)}{\|e(w_i)\| \|e(w_j)\|}$$

- words found by embedding-based similarities can be filtered with other types of word similarities

Short Text Similarities

$d_1 = \{w_1^1, w_2^1, ..., w_m^1\}$ and $d_2 = \{w_1^2, w_2^2, ..., w_n^2\}$

$$\text{sim}(d_1, d_2) = \frac{1}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} \cos\left(e(w_i^1), e(w_j^2)\right)$$

Pre-Trained Embeddings

- Word2Vec: the implementations of CBoW and Skip-Gram methods.
  - trained on Google News (100 billion tokens)
  - *https://code.google.com/archive/p/word2vec/*
  - Skip-Gram is more effective in practice
- GloVe: Global Vectors for Word Representation
  - GloVe is an unsupervised method for obtaining word embeddings
  - GloVe aims at reconciling the advantages of corpus-wide co-occurrence counts and local context windows
  - *https://nlp.stanford.edu/projects/glove/*
  - trained on Wikipedia (6 billion tokens)
  - trained on CommonCrawl (42 and 840 billion tokens)
  - trained on Twitter (27 million tokens)
- many other pre-trained embeddings for different languages
  - *https://fasttext.cc/docs/en/crawl-vectors.html*

Practical Hints

- always try out different word embeddings (consider them as a hyperparameter)
- results may vary drastically with different embeddings
- consider source of corpora used to train word embeddings (larger is not always better, a smaller but more domain-focused can be more effective)
- consider what contexts were used to define similarities
- it's better to use the same tokenization and text normalization methods that were used for creating word embeddings

Limitations

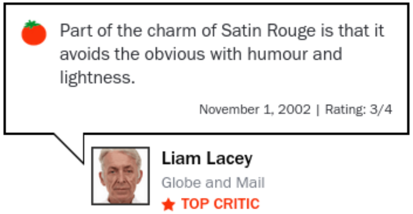
- the algorithms discussed provide very little control over the kind of similarity they include
    - “cat” is more similar to “dog” than to “tiger” as they both are pets
    - “cat” is more similar to “tiger” than to “dog” as they both as felines
- many of the trivial properties of words are ignored because people are less likely to mention known information than they are to mention novel one
- text corpora can easily be biased for better or worse $\longrightarrow$ so word embeddings can become biased too
    - gender and racial biases are very common
- these word vectors are context independent
    - in reality, there is no such a thing to have context independent word meaning
    - some words have multiple sense e.g., “bank” may refer to a financial institution or to the side of a river

Section 4

CNN for NLP

Classifying sentences or documents

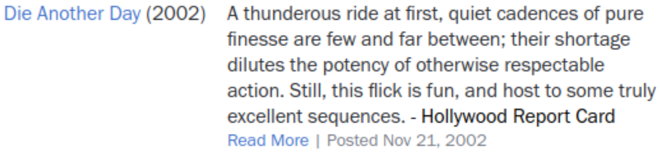
Predicting the sentiment (positive, negative, or neutral)[1]



Part of the charm of Satin Rouge is that it avoids the obvious with humour and lightness.

November 1, 2002 | Rating: 3/4

Liam Lacey
Globe and Mail
TOP CRITIC



Die Another Day (2002) A thunderous ride at first, quiet cadences of pure finesse are few and far between; their shortage dilutes the potency of otherwise respectable action. Still, this flick is fun, and host to some truly excellent sequences. - Hollywood Report Card

Read More | Posted Nov 21, 2002

- Some of the words are **very informative** of the sentiment (charm, fun, excellent) other words are less informative (Still, host, flick, lightness, obvious, avoids)

- Informative clue is informative regardless of its position in the sentence

[1] https://www.rottentomatoes.com

Motivation

Problem 1: Variable-sized input

- standard MLP always expect the same input size

Problem 2: Relevance of words

- “to” and “a” are not very informative, but content words like “kidnapping” are important for most tasks independent of their position in the input

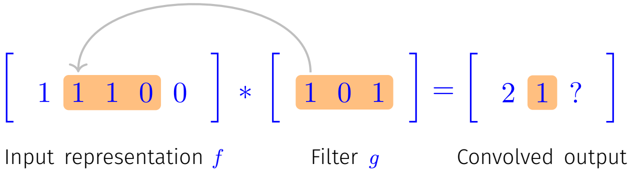
Problem 3: MLP may have too many parameters (“too complex models”) in certain situations

Convolution and pooling for NLP

Idea of convolution (– Yoav Goldberg)

“A convolutional neural network is designed to identify indicative local predictors in a large structure, and combine them to produce a fixed size vector representation of the structure, capturing these local aspects that are most informative for the prediction task at hand.“

1-D convolution operation example



$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 1 & ? \end{bmatrix}$
Input representation f
Filter g
Convolved output

2-D convolution operation example

The goal is to **learn** good kernels (filter parameters)

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 4 & 3 & 4 \\ 2 & 4 & 3 \\ 2 & 3 & 4 \\ 2 & 3 & 1 \end{bmatrix}$$

Input representation f

Filter g (also 'kernel')

Convolved output

How to define convolution in texts

- Sentiment classification

  *The **movie** was **really good**.*

  *We saw this **really good movie**.*

  *The **movie**, which we saw yesterday with all the colleagues in this tiny movie theater next to the bridge, was (despite my expectations) **really good**.*

- For this task, position information does not really matter.
  Advantages of text flow
  - Usually only one dimension
  - as opposed to two dimensions (or even three) in images
- Convolutional networks in NLP are also called time-delay neural networks (TDNN)

Outline

Convolutional layer in NLP

- Input sentence $\mathbf{x}_{i:i+n}$ are stacked word embeddings of dimensionality d

- Convolutional filter $\mathbf{w} \in \mathbb{R}^{h \times d}$

where h is the filter size (how many neighboring words are convoluted)

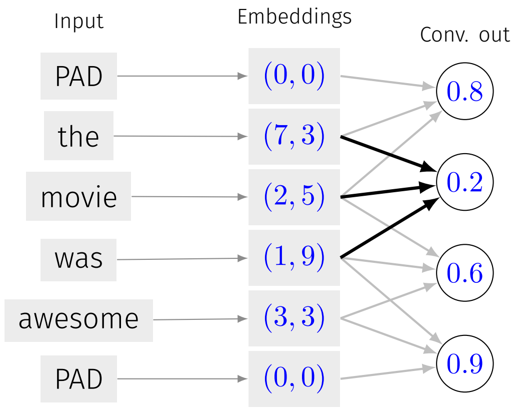
- Convolution operation $\mathbf{w} * \mathbf{x}_{i:i+n}$

- Non-linear operation

$$c_i = \mathrm{ReLU}(\mathbf{w} * \mathbf{x}_{i:i+n} + b)$$

Example

$d = 2 \qquad h = 3$



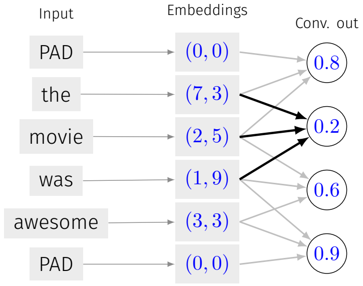
Properties of convolutional networks

Not every input is connected to every output in the following layer

- **sparse connectivity** (vs fully-connected/dense layers)

For each window, we use the same weights and bias values

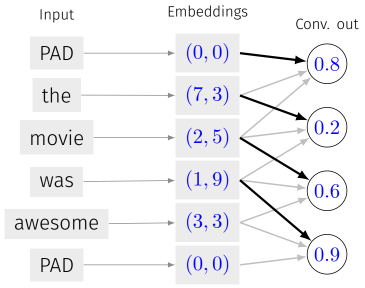
- **parameter sharing**



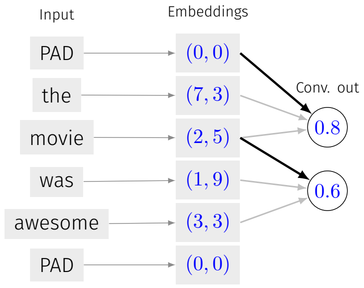
Stride

Stride: the step size for moving over the sentence

- Stride 1 common in NLP; other values in comp. vision



Stride = 1



Stride = 2

Dense layer vs. Convolutional Layer

In principle, a convolutional layer could handle variable-sized inputs

- But in practice, it handles fixed-sized input, just like in an MLP

We usually pad with zeros so that all sequences in our data have the same length

- Sometimes we also truncate

Outline

Pooling layer

Another new building block: pooling layer

- Idea: capture the most important activation

Let $c_1, c_2, \ldots, \in \mathbb{R}$ denote the output values of the convolutional filter

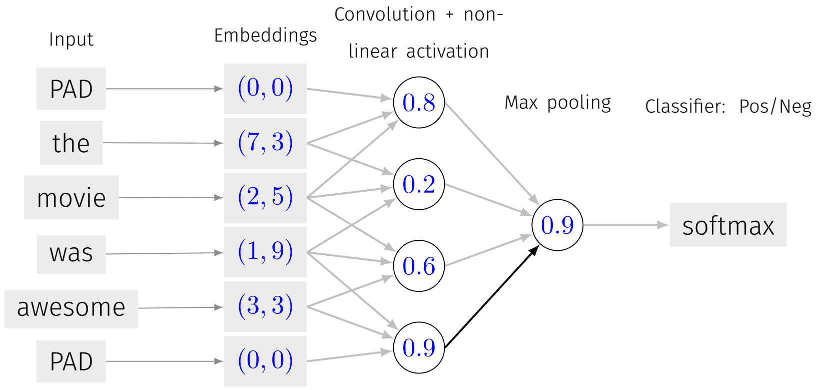
Compute output o for a **max-over-time pooling** layer as

$$o = \max_i c_i$$

Max-over-time pooling is most common in NLP. You can also find min-pooling and mean-pooling in other areas. Could also use some other averaging

Note that there are no associated weights

Classification with convolution and pooling

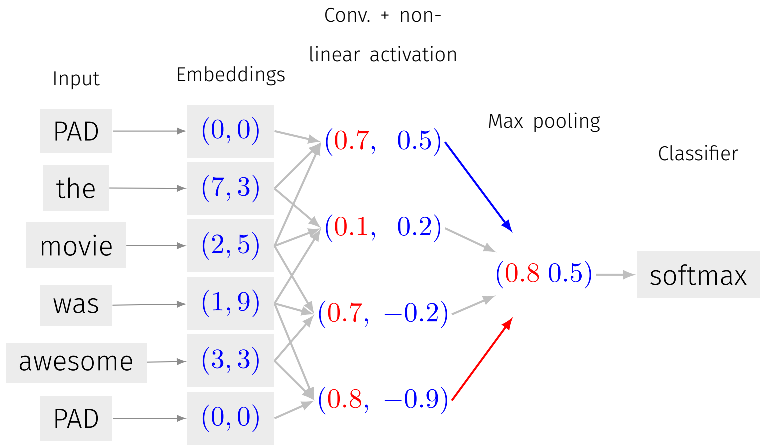


Multiple filters

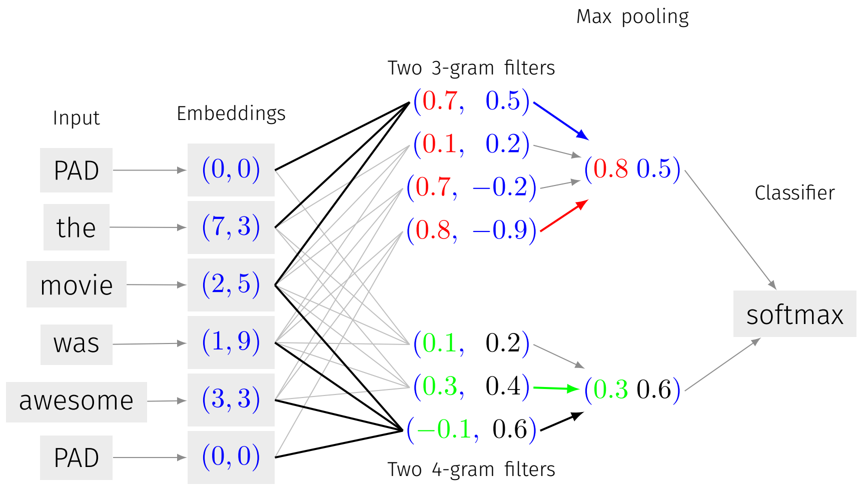
Usually we have **many** filters (hundreds or thousands), not just one

They may be of same or of different sizes

Multiple filters



Multiple filters



Properties of pooling

Idea: Extracting relevant features independent of their position in the input

- Problems:

Output remains the same if a feature occurs once or multiple times

The music was great, but the cast was horrible, the plot was horrible and the costumes were horrible.

Outline

Investigating CNNs[2]

In computer vision

- Effectiveness of deep CNNs can be very well explained

- Primary conv. layers detect the edges of an object

- As we go deeper, more complex features of an image are learnt

In NLP

- Not much understanding behind their success

[2] Madasu.AnveshRao.2019.EMNLP

Open questions

- Popular CNN architectures use convolution with a fixed window size over the words in a sentence.
  - Is sequential information preserved using convolution across words?
  - What will be the effect if word sequences are randomly shuffled and convolution is applied on them?
- Previously proposed CNN architectures use max pooling operation across convolution outputs to capture most important features[3]
  - Will the feature selected by max pooling always be the most important feature of the input or otherwise?

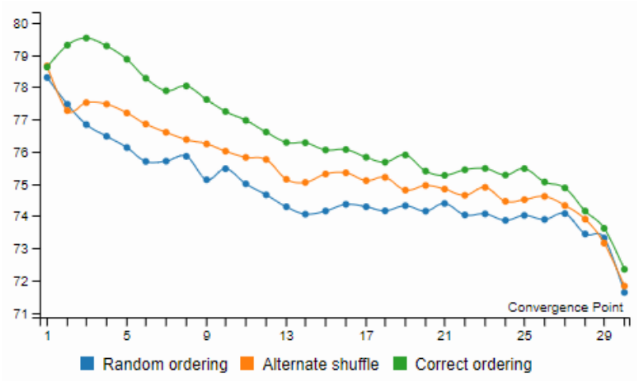
[3] Kim.2014.EMNLP

Experiments

Train a CNN with convolution applied on words

- Fixed window size varying from one to maximum sentence length
- Repeat this experiment with randomly shuffling
    - **Random ordering**: All the words in an input sentence
    - **Alternate shuffle**: Swap every two consecutive words, e.g. ‘read book forget movie’ $\rightarrow$ ‘book read movie forget’

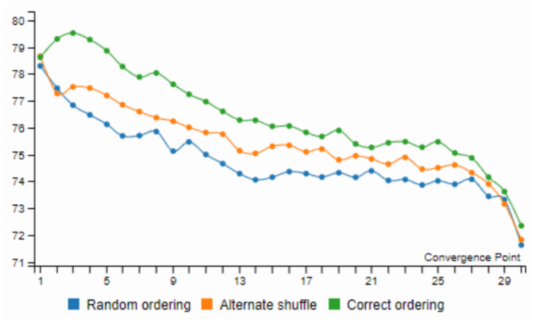
Analysis



CNNs fail to fully incorporate sequential information – performance on random ordering and correct ordering are marginally near each other

Performance with correct ordering on window size 7 $\approx$ random ordering on window size 1

Analysis



Increasing window size $\rightarrow$ worse in capturing sequential information

But: Performance on random ordering is still higher than other context blind algorithms like bag-of-words

CNNs still learn something valuable! What?

Experiment 2: What it learns

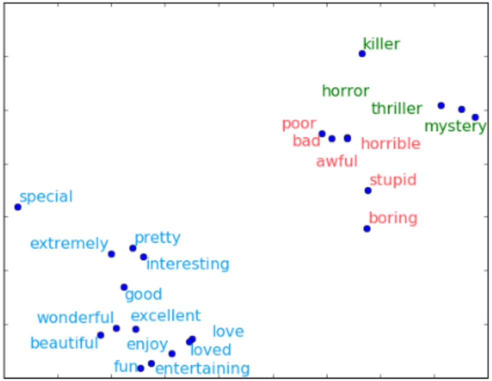
Train a CNN with window size 1 on sentiment classification

- Convolution acts over a single word $\rightarrow$ no ability to capture sequential information

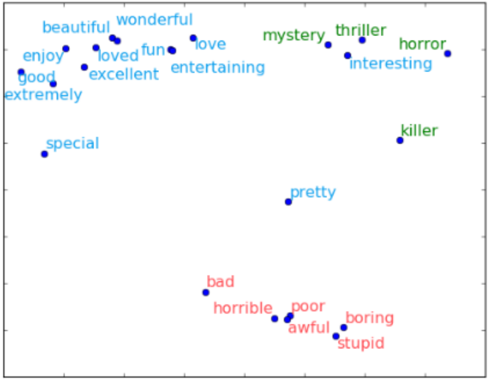
- Words will always have same respective convolution output

- Convolution layer here acts like an embedding transformation, where input embedding space is transformed into another space

Analysis



(a) Original Embeddings



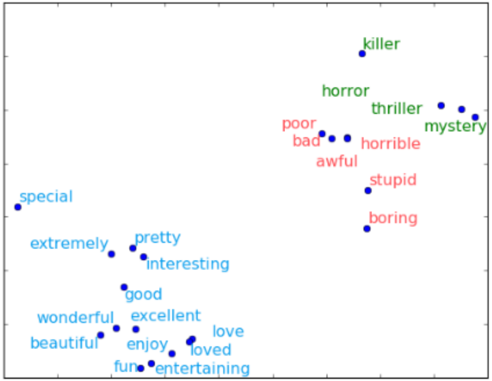
(b) Convolution Transformation

Figure 1: t-SNE projection of original embeddings and after convolution transformation

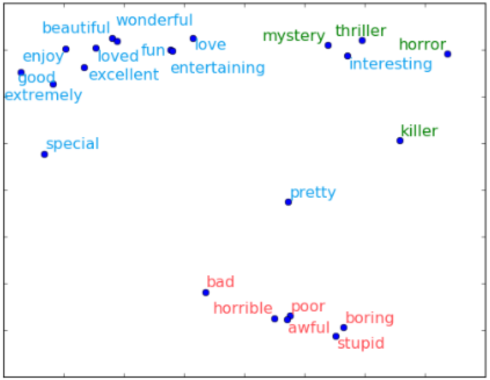
Blue = positive sentiment

Red = negative sentiment

Analysis



(a) Original Embeddings

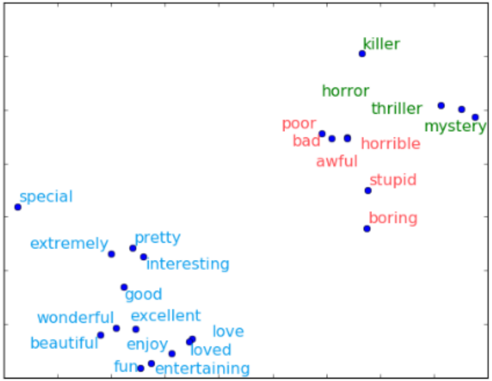


(b) Convolution Transformation

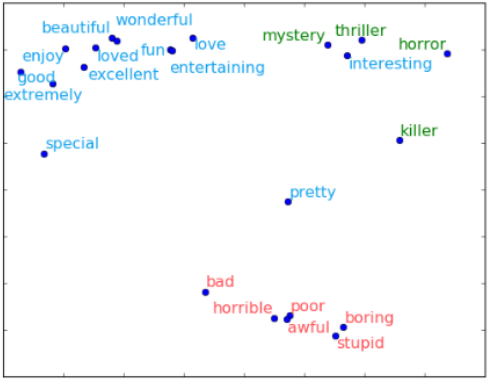
Figure 1: t-SNE projection of original embeddings and after convolution transformation

Convolution layer tunes input embeddings $\rightarrow$ closer to the positive sentiment cluster than to their original semantic cluster

Analysis: the world “killer”



(a) Original Embeddings



(b) Convolution Transformation

Figure 1: t-SNE projection of original embeddings and after convolution transformation

- in the original space: very close to negative words “bad” and “awful” (Glove embeddings)
- in the sentiment dataset: is often used to describe a movie very positively

Take-home message

Single convolution filter output value captures a weighted summation over all the features of the original word embedding.

This enables the network to learn more task appropriate features as many as the number of filters.

Madasu.AnveshRao.2019.EMNLP

Summary

Convolutional networks can deal with variable sized input

- Sparse connectivity, parameter sharing - Narrow vs wide convolution

Pooling enables focus on most relevant features

- Max-over-time pooling

Convolutional networks for NLP

- Sentence classification