Text Analysis and Retrieval

6. Neural NLP – Recurrent models

Martin Tutek, Josip Jukić

University of Zagreb
Faculty of Electrical Engineering and Computing (FER)

Academic Year 2022/2023

Creative Commons Attribution–NonCommercial–NoDerivs 3.0

v3.0

Outline

Outline

Learning word embeddings

"Words are discrete objects. We have to figure out a way to represent them in a feature vector for a machine learning model."

- **Goal:** learn an embedding for each word in a vocabulary that encapsulates semantic and syntactic properties of that word
- Three key components:
  1. What is our **model**?
  2. What is our **loss function**?
  3. What **data** will we train our model on?

Data

- Labeled text data is sparse and expensive to create
- Unlabeled text data is abundant:
    - Wikipedia
    - News portals
    - Message boards
    - The internet
- **Idea:** can we use unlabeled data to construct a supervised classification task

Training task

"You will know a word by the company it keeps" – Firth, 1957

- Firth's distributional hypothesis: based on the *context* of a word, we should be able to determine the word itself

"anarchism is a political ________ which considers the state"

- The **context** of width k indicates the number of words to the *left and right* of a **target** word
- **Self-supervised** classification task

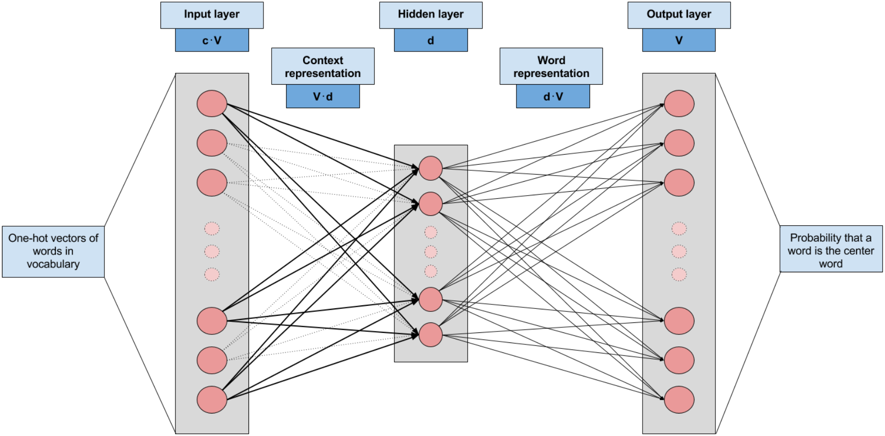
Continuous bag-of-words (CBOW)

"The $\boxed{\text{quick brown } \boxed{\text{[fox]}} \text{ jumps over}}$ the lazy dog"

- The **model** is a single-layer neural network
  1. Form a vocabulary
  2. We assign each word a random initial word embedding, forming an *embedding matrix*
  3. Compute the *average* of the context word embeddings, obtain probabilities of each word in vocabulary being the target
  4. Backpropagate the *cross-entropy* classification loss

$$x_i = \boxed{\text{quick, brown, jumps, over}}$$
$$y_i = \boxed{\text{fox}}$$

Continuous bag-of-words (CBOW)



Skip-gram (SG)

"The $\boxed{\text{quick brown } \boxed{\text{[fox]}} \text{ jumps over}}$ the lazy dog"

- The **model** is (again) a single-layer neural network
  1. Form a vocabulary
  2. Create random initial embedding for each word to form an *embedding matrix*
  3. Perform classification tasks, one for every context word

$$x^i = \boxed{\text{fox}}$$
$$y^i = \boxed{\text{quick, brown, jumps, over}}$$

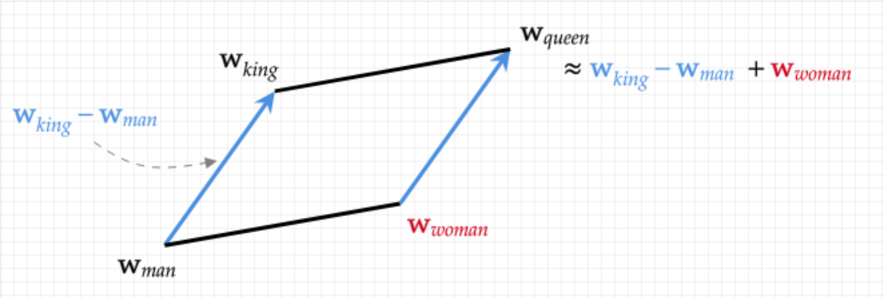
Skip-gram (SG)

- Optimizing the SG model over a large text corpus yields **higher quality** word embeddings **faster** (compared to CBOW)
  - Both tasks (CBOW, SG) are impossible to solve perfectly with finite context size
  - But, optimizing those tasks yields quality word embeddings
- Can we do better? Yes, we can be more model-efficient

Negative sampling

- Instead of computing the probability of every possible word, can we get away with computing only the probabilities of the **correct** word and a small subset of words from the vocabulary called the *negative samples*?
- A new, binary classification task for each negative example and the correct word
- Typically, if N is the size of the negative sample and V is the size of the vocabulary, $N \ll V$

Embedding space



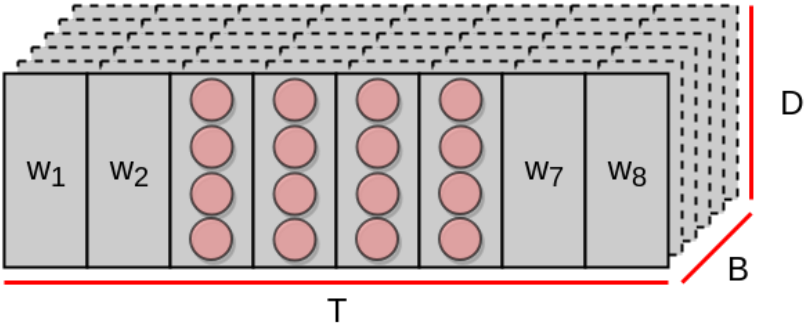
Learning outcomes 1

1. Explain what word embeddings are and what they can be used for
2. Describe the Skip-gram training setup and provide an example of a training instance
3. Describe the CBOW training setup, compare it to Skip-gram, and provide an example of a training instance
4. Define negative sampling and explain what we use it for

Outline

The problem of variable length input

A typical input instance of an NLP model consists of a sequence of words of length T. Each token is represented by a d-dimensional word embedding. Sequences are organized in mini-batches of size B.



- Issue: The length of a word sequence T usually varies across and within batches!

Semantic composition

We have to, at some point in our model, reduce the variable-length dimension to a fixed-size representation $\rightarrow$ compose the meaning of individual words to a single representation

- max, $mean$, sum, $weighted\_sum$...
- Something better?

Recurrent Neural Networks

Issues with simple methods:

- Invariant to word order (poor semantic composition)
- No token-level representations in output

Define s as the fixed-size *sequence* representation, x_i as word embeddings of our input tokens.

- We want a function that can summarize $f : (x_0, \dots, x_i, \dots, x_T) \to s$.
- We also want the function to produce token-level outputs y_i.

A **Recurrent Neural Network** (RNN) computes an intermediate output for each input in a sequence:

$$h^{(t)} = \mathsf{RNN}(h^{(t-1)}, x^{(t)}) \tag{1}$$

Vanilla recurrent neural networks

Vanilla recurrent neural networks (Elman RNNs) implement the following recurrence relation:

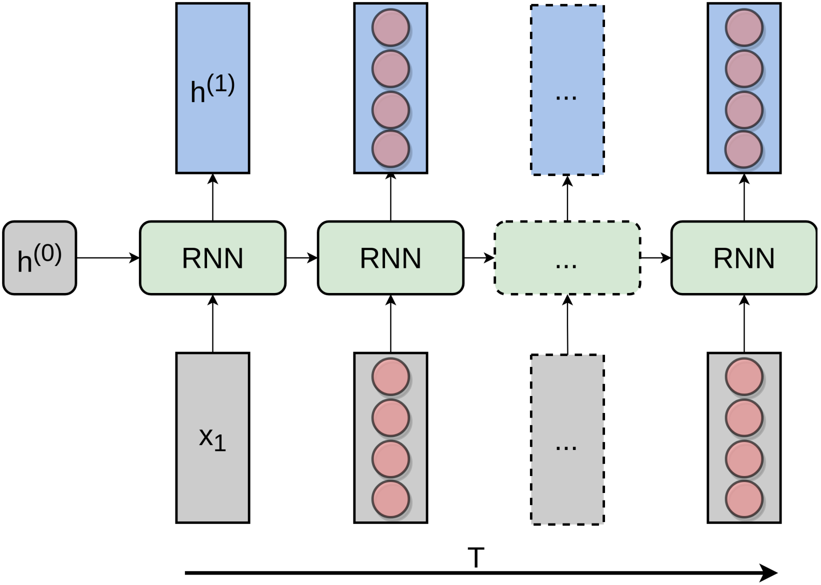
$$h^{(t)} = \sigma(a^{(t)}) = \sigma(W_{hh}h^{(t-1)} + W_{xh}x^{(t)} + b) \tag{2}$$

Where σ is the sigmoid nonlinearity, W the weight matrices and b the bias vector.

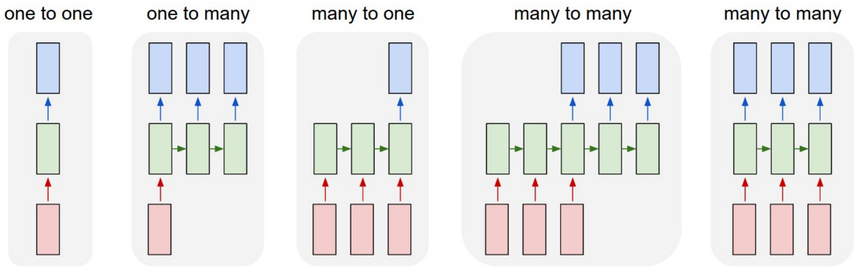
Note that:

- We can set $s = h^{(T)}$
- We obtain an intermediate representation $h^{(t)}$ for every input $x^{(t)}$
- The weight matrices and bias vector are *shared* between timesteps
- One step of a recurrent network is a single-layer neural network

Unrolling the recurrent network



Types of sequential processing problems



1. Fixed size input tasks
2. **One-to-many**: text generation, music generation
3. **Many-to-one**: text classification, sentiment analysis
4. **Sequence-to-sequence**: machine translation, text summarization
5. **Sequence labeling**: POS tagging, named entity recognition

Issues with recurrent networks

Despite all their benefits, RNNs are prone to problems:

1. **Learning long-term dependencies**
   - "I grew up in France... I speak fluent [French]."
2. **Exploding and vanishing gradients**
   - During backpropagation, the gradient is repeatedly multiplied with the recurrent weight matrix W_{hh}
   - Dependent on the largest eigenvalues of W_{hh}, the values of the gradient will either tend towards infinity (explode) or towards zero (vanish)

Long Short-Term Memory

How to mitigate the issues of vanilla RNNs? We can use the mechanism of selective memory

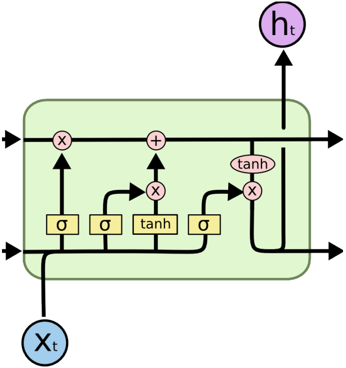
Long Short-Term Memory (LSTM) is a popular variant whose recurrent relations are defined as:

$$h^{(t)} = o^{(t)} \odot tanh(c^{(t)}) \tag{3}$$

- Separation of responsibility with the dual cell state: $h^{(t)}$ holds the hidden representation by combining output $o^{(t)}$ with memory $c^{(t)}$
- Restrict access to the memory through **gates**

LSTM gates

- **Forget** gate – determines which information, if any, should be *erased* from memory.
- **Input** gate – propagates information from the input representation to memory.
- **Output** gate – determines which information from the cell state is relevant for the current output.



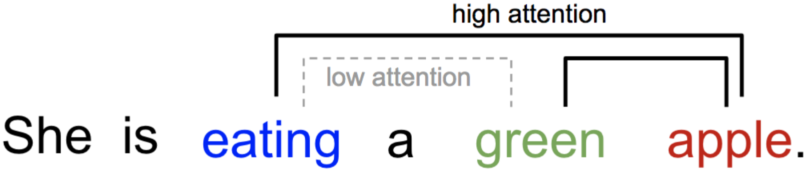
LSTM cell

LSTM: issues

LSTMs still have **issues** with learning long-term dependencies!

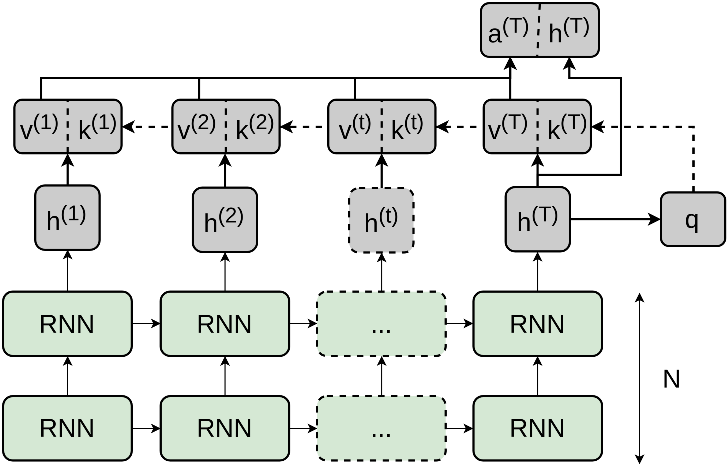
1. **Issue 1**: Memory capacity
   - "You can't cram the meaning of a whole %&$# sentence into a single $&# vector!" – Raymond Mooney
2. **Issue 2**: Uncertainty
   - At timestep t, we are unaware of what the future inputs hold $\rightarrow$ a vast amount of possible options

Attention in recurrent networks



- Reduce the burden of a state in recurrent networks if we allow our networks a *glimpse* into the previous hidden states, which will **augment** the information in the current hidden state
- The number of previous hidden states is variable, and not every previous state is equally important

Attention in recurrent networks



A sketch of the attention mechanism

Learning outcomes 2

1. Explain why traditional neural models cannot be used on variable-length input sequences
2. Sketch four RNN use patterns and name a prototypical NLP task for each
3. List two main issues vanilla RNNs face and exemplify them
4. List the key ideas behind an LSTM cell and explain how these address the problems of vanilla RNNs
5. Motivate the attention mechanism and define its general case

Outline

Contextualized word representations

Word embeddings learned by word2vec-style models are *context-invariant* (static) – the embedding represents the word in isolation. Can we learn *context-aware* (dynamic) embeddings of words (tokens)? We can!

Self-supervised setups (on unlabeled corpora):

1. **(Causal) language modeling** (CLM): predict next token given history.
2. **Masked language modeling** (MLM): randomly replace a proportion of input tokens with the special "$[MASK]$" token. The network has to reconstruct the masked tokens

ELMo: Deep contextualized word representations

ELMo: Deep contextualized word representations

Peters et al., NAACL **2018.**

- ELMo stands for "Embeddings from Language Models"

Idea:

1. Train a large bidirectional LSTM language model on a large text corpus (Billion word benchmark)
   - Forward LM predicts the next token given past context
   - Backward LM predicts the previous token given future context
2. Use the trained network as a **sentence encoder** for other tasks
3. Profit (*significant* performance gains across tasks)

Learning outcomes 3

1. List and compare two self-supervised setups for learning contextualized representations
2. Describe ELMo in terms of its purpose, the underlying neural model, and the prediction task

Study assignment

1. Watch the first two parts of TAR "Neural NLP" video lectures
   - Video Part I
   - Video Part II
2. Read the first two sections (The Unreasonable Effectiveness of Recurrent Neural Networks; Understanding LSTM Networks):
   - Reading
3. Read Section 1 (Intro) from "Deep contextualized word representations":
   - ELMo
4. Read the article on attention in RNNs:
   - Attention in RNNs
5. Self-check against learning outcomes!