Lecture 9
Transformers

LTAT.01.001 – Natural Language Processing
Kairit Sirts (kairit.sirts@ut.ee)
12.04.2022
Plan for today

- Word senses
- Contextual word embeddings
- Transformers
- BERT
Word senses
Word Senses

- One lemma can have several meanings.

- Consider the word **bank**
  
  Sense 1: • ...a bank can hold the investments in a custodial account...
  Sense 2: • “…as agriculture burgeons on the east bank the river will shrink even more”

- **Sense** (or **word sense**)  
  - A discrete representation of an aspect of a word’s meaning.

- The lemma **bank** here has two senses
Homonymy

**Homonyms**: words that share a form but have unrelated, distinct meanings:

- **bank**<sub>1</sub>: financial institution,  **bank**<sub>2</sub>: sloping land
- **bat**<sub>1</sub>: club for hitting a ball,  **bat**<sub>2</sub>: nocturnal flying mammal

1. **Homographs**
   - In English: bank/bank, bat/bat
   - In Estonian: tee/tee

2. **Homophones**:
   - In English: write and right, piece and peace
   - In Estonian: baar and paar, ball and pall
Polysemy

● 1. The **bank** was constructed in 1875 out of local red brick.
● 2. I withdrew the money from the **bank**

● Are those the same sense?
  ● Sense 2: “A financial institution”
  ● Sense 1: “The building belonging to a financial institution”

● A **polysemous** word has **related** meanings
  ● Most non-rare words have multiple meanings
Sense information in word embeddings

- Embedding systems like Word2Vec or Glove include one embedding per word
- All different meanings are conflated into a single vector
- This can result in semantically unrelated words being pulled together

Source: Camacho-Collados and Pilehvar, 2018. From word to sense embeddings: A survey on vector representations of meaning
Solving the sense problem in word embeddings

- Option I: just ignore it
- Option II: Devise methods to learn sense embeddings, i.e. one vector per sense. This is related to various problems:
  - How many senses does each word have?
  - For instance, the word “bass” has 9 senses in English WordNet

- Wordnet: a hierarchically organized lexical database that includes concepts and their relations (synonyms, antonyms, is-a relation, is-part-of relation etc)
Noun

- **S:** (n) **bass** (the lowest part of the musical range)
- **S:** (n) **bass, bass part** (the lowest part in polyphonic music)
- **S:** (n) **bass, basso** (an adult male singer with the lowest voice)
- **S:** (n) **sea bass, bass** (the lean flesh of a saltwater fish of the family Serranidae)
- **S:** (n) **freshwater bass, bass** (any of various North American freshwater fish with lean flesh (especially of the genus Micropterus))
- **S:** (n) **bass, bass voice, basso** (the lowest adult male singing voice)
- **S:** (n) **bass** (the member with the lowest range of a family of musical instruments)
- **S:** (n) **bass** (nontechnical name for any of numerous edible marine and freshwater spiny–finned fishes)

Adjective

- **S:** (adj) **bass, deep** (having or denoting a low vocal or instrumental range) "a deep voice"; "a bass voice is lower than a baritone voice"; "a bass clarinet"
Solving the sense problem in word embeddings

- Option I: just ignore it
- Option II: Devise methods to learn sense embeddings, i.e. one vector per sense. This is related to various problems:
  - How many senses does each word have?
  - For instance, the word “bass” has 9 senses in English WordNet
  - Wordnet: a hierarchically organized lexical database that includes concepts and their relations (synonyms, antonyms, is-a relation, is-part-of relation etc)
  - Can group senses into larger clusters:
    - bass – music related
    - bass – fish related
- Option III: contextual word embeddings
Contextual word embeddings
Contextualized word embeddings

- Computed by large language models
- Pretrained on large amounts of unannotated data
- The representation of each word is constructed based on its context.
- The word “bank” in the following two sentences will have a different representation:
  - …a **bank** can hold the investments in a custodial account…
  - “…as agriculture burgeons on the east **bank** the river will shrink even more”
- Thus, contextual word embeddings address the problem of different senses
Contextualized word embeddings

- Recent trend in NLP
- Pre-train deep contextualized word embeddings
- Nice illustrated tutorials:
  - https://jalammar.github.io/illustrated-transformer/
  - http://jalammar.github.io/illustrated-bert/
- Mostly based on transformer architecture
Transformer
Attention is all you need  
Vaswani et al., 2017

- A sequence to sequence model based on attention
- Strong results on standard machine translation benchmark datasets
- Fast (???) because it does not include recurrent networks

![Diagram of attention model](image)
Transformer architecture

- Although initially introduced in the context of encoder-decoder architecture, it can be also used as a basis for simply an encoder ...
- ... for replacing and RNN

- Transformer is based on self-attention blocks

- The model is sequential but it does not contain recurrence
  - i.e. it is basically fully feedforward
Essential components in Transformer

- **Self-attention**: each layer combines words with each other
- **Multi-headed attention**: several attention heads learn independently
- **Positional encodings**: as the model does not use an RNN, a mechanism is needed to distinguish different input positions
Recall scaled dot-product attention

\[ a(q, k) = \frac{q \cdot k}{\sqrt{d_k}} \]

\[ \alpha(q, k_i) = \text{softmax}(a(q, k_i)) \]

\[ c = \sum_i \alpha_i h_i^e \]

- q – query – current decoder state
- k – key – encoder representation
- h – encoder representation
In matrix notation

\[ c = \text{attention}(q, K, V) = \text{softmax}(\frac{qK^T}{\sqrt{d_k}})V \]

- V – value matrix
- consists of value vectors v
- In our current case, these are the same encoder representations
- v = h^e
- V = H^e
Self-attention

- What if attention is used to tie together words in the encoder
  - query – an encoder representation
  - keys – encoder representations
  - values – encoder representations

- Self-attention
  \[ \text{selfattention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \]

- In self-attention $Q == K == V$
  - because each word is a query in turn
Self-attention

● To model the different roles in self-attention...
  ● query
  ● key
  ● value

● ... the encoder representations are projected into different subspaces:
  ● $q_i = h_i W^q$
  ● $k_i = h_i W^k$
  ● $v_i = h_i W^v$

● In matrix notation
  ● $Q = HW^q$
  ● $K = HW^k$
  ● $V = HW^v$
Self-attention

- Each layer combines words with others with scaled dot product attention
  \[ a(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \]

- In retain the dimensionality of the representations:
  - The dimensionality of all projection matrices is \( R^{d_k \times d_k} \)
Multi-head attention

- Self-attention creates new representation for each input token by computing a weighted combination over all tokens (projected into values role)

- Weights are computed via softmax --> each token can attend closely only to a single other token.

- What if there are more than one relevant input tokens in the context that a token should attend to?
  - syntactic relations
  - semantic relations
  - morphological agreements
  - coreferential relations
  - ...

Multi-head attention

- Solution: use multiple attention heads in parallel
- Each head has its own projection matrices
- Typically, the projection matrices reduce the dimensionality
  - If the number of heads is $n$, then the output dimension of the projections in $d_k/n$
- Later, the results of all attention heads are concatenated, restoring the original dimensionality
Multi-head attention

- Project keys, queries and values into several different sub-spaces

\[ Q_i = QW_i^Q, K_i = KW_i^K, V_i = VW_i^V \]

- For each attention \( i \) compute a head

\[ \text{head}_i = a(Q_i, K_i, V_i) \]

\[ \text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O \]
Transformer block
Because the transformer architecture is feedforward, there must be some way to encode the ordering of the tokens in the input. Currently used solution: add positional embeddings to input token embeddings.
Positional embeddings

- Positional embeddings can be learned:
  - Learn a different embedding for each token position: 0, 1, 2 etc
  - Drawback: embeddings with high indices will get less training than embeddings with low indices

- Positions can be encoded deterministically with sine and cosine functions of different frequencies

- In the original Vaswani paper:

\[
P_{E_{pos,2i}} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right)
\]

\[
P_{E_{pos,2i+1}} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right)
\]
Transformer architecture

- Transformer is a powerful attention-based feedforward building block
- Typically several transformer layers are stacked on top of each other
- Each layer has its own multi-head attention with respective projection matrices for queries, keys and values
- The trainable parameters in the transformer model are:
  - multi-head attention projection matrices
  - linear layers in the transformer layers
  - word/token embeddings in the input
  - classification layers on top of the model
BERT
BERT: Bidirectional Encoder Representations from Transformer

- Devlin et al., 2018. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*

- Main characteristics
  - Bidirectional
  - Very deep

- Key components
  - Masked language model
  - Auxiliary “next sentence prediction” task
### Input to the BERT

<table>
<thead>
<tr>
<th>Input</th>
<th>[CLS]</th>
<th>my</th>
<th>dog</th>
<th>is</th>
<th>cute</th>
<th>[SEP]</th>
<th>he</th>
<th>likes</th>
<th>play</th>
<th>#ing</th>
<th>[SEP]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Token Embeddings</strong></td>
<td>$E_{[CLS]}$</td>
<td>$E_{my}$</td>
<td>$E_{dog}$</td>
<td>$E_{is}$</td>
<td>$E_{cute}$</td>
<td>$E_{[SEP]}$</td>
<td>$E_{he}$</td>
<td>$E_{likes}$</td>
<td>$E_{play}$</td>
<td>$E_{#ing}$</td>
<td>$E_{[SEP]}$</td>
</tr>
<tr>
<td></td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td><strong>Segment Embeddings</strong></td>
<td>$E_{A}$</td>
<td>$E_{A}$</td>
<td>$E_{A}$</td>
<td>$E_{A}$</td>
<td>$E_{A}$</td>
<td>$E_{B}$</td>
<td>$E_{B}$</td>
<td>$E_{B}$</td>
<td>$E_{B}$</td>
<td>$E_{B}$</td>
<td>$E_{B}$</td>
</tr>
<tr>
<td></td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td><strong>Position Embeddings</strong></td>
<td>$E_{0}$</td>
<td>$E_{1}$</td>
<td>$E_{2}$</td>
<td>$E_{3}$</td>
<td>$E_{4}$</td>
<td>$E_{5}$</td>
<td>$E_{6}$</td>
<td>$E_{7}$</td>
<td>$E_{8}$</td>
<td>$E_{9}$</td>
<td>$E_{10}$</td>
</tr>
</tbody>
</table>
Input tokenization

- Typically, BERT vocabulary consists of sub-words instead of full words.
- Several algorithms for creating vocabulary: BPE, wordpiece, sentencepiece.
- They all achieve roughly the same goal:
  - Create a vocabulary of reasonable size (usually 30-100K) such that
  - Most frequent words get their own entry.
  - Less frequent words get split into segments.
  - The vocabulary also contains single characters to enable representing totally novel or very rare words.
Byte-pair encoding algorithm (BPE)

- Start from a vocabulary of characters
- At each step of the algorithm:
  - find the most frequent merging operation
  - Add the resulting token to the vocabulary
  - Replace all occurrences in the text with the merged segment
- Continue until a predefined number of merges has been done
BPE: example

corpus
5 low_
2 lowest_
6 newer_
3 wider_
2 new_

corpus
4 ten_
3 ten_
5 ten_
2 ten_
1 ten_
2 ten_
3 ten_
4 ten_
5 ten_

vocabulary
__, d, e, i, l, n, o, r, s, t, w
BPE example

- Merge: e r → er

<table>
<thead>
<tr>
<th>corpus</th>
<th>vocabulary</th>
</tr>
</thead>
<tbody>
<tr>
<td>low _</td>
<td>_, d, e, i, l, n, o, r, s, t, w, er</td>
</tr>
<tr>
<td>lowest _</td>
<td></td>
</tr>
<tr>
<td>newer _</td>
<td></td>
</tr>
<tr>
<td>wider _</td>
<td></td>
</tr>
<tr>
<td>new _</td>
<td></td>
</tr>
</tbody>
</table>
BPE example

- Merge: er _ → er_

<table>
<thead>
<tr>
<th>corpus</th>
<th>vocabulary</th>
</tr>
</thead>
<tbody>
<tr>
<td>5  low _</td>
<td><strong>, d, e, i, l, n, o, r, s, t, w, er, er</strong></td>
</tr>
<tr>
<td>2  lowest _</td>
<td></td>
</tr>
<tr>
<td>6  newer _</td>
<td></td>
</tr>
<tr>
<td>3  wider _</td>
<td></td>
</tr>
<tr>
<td>2  new _</td>
<td></td>
</tr>
</tbody>
</table>
BPE example

- Merge: n e → ne

<table>
<thead>
<tr>
<th>corpus</th>
<th>vocabulary</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 low _</td>
<td><strong>, d, e, i, l, n, o, r, s, t, w, er, er</strong>, ne</td>
</tr>
<tr>
<td>2 lowest _</td>
<td></td>
</tr>
<tr>
<td>6 new er__</td>
<td></td>
</tr>
<tr>
<td>3 wider _</td>
<td></td>
</tr>
<tr>
<td>2 new w _</td>
<td></td>
</tr>
</tbody>
</table>
BPE example

- and so on ...

<table>
<thead>
<tr>
<th>Merge</th>
<th>Current Vocabulary</th>
</tr>
</thead>
<tbody>
<tr>
<td>(ne,  w)</td>
<td><strong>, d, e, i, l, n, o, r, s, t, w, er, er</strong>, ne, new</td>
</tr>
<tr>
<td>(l,  o)</td>
<td><strong>, d, e, i, l, n, o, r, s, t, w, er, er</strong>, ne, new, lo</td>
</tr>
<tr>
<td>(lo,  w)</td>
<td><strong>, d, e, i, l, n, o, r, s, t, w, er, er</strong>, ne, new, lo, low</td>
</tr>
<tr>
<td>(new, er__)</td>
<td><strong>, d, e, i, l, n, o, r, s, t, w, er, er</strong>, ne, new, lo, low, newer__</td>
</tr>
<tr>
<td>(low, __)</td>
<td><strong>, d, e, i, l, n, o, r, s, t, w, er, er</strong>, ne, new, lo, low, newer__, low__</td>
</tr>
</tbody>
</table>
Input tokenization

- The BPE (or another sub-word segmentation) algorithm is applied to a training set to create a vocabulary
- Later, the vocabulary is applied in a greedy manner to new texts to get sub-word tokenization
Bidirectional language model

- The language models we have seen so far are causal: they predict the next word given the previous context (in the left).
- BERT is bidirectional language model, predicting words based on both left AND right context.
- Thus, it is not a causal language model and it cannot be used for generation.
- BUT, it can be used to encode input texts.
Masked language model

- BERT uses the masked language model objective, which is based on the cloze task.
- Cloze task: predict, which word fits into the blank
- For example:

Today, I went to the ___________ and bought some milk and eggs. I knew it was going to rain, but I forgot to take my __________, and ended up getting wet on the way.
Masked language model

- If the input text is given, then predicting the word in the blank is trivial

  Today, I went to the store and bought some milk and eggs.

- Therefore, BERT replaces the blank word with a special [MASK] token

  Today, I went to the [MASK] and bought some milk and eggs.
Masked language model

Use the output of the masked word’s position to predict the masked word
Masked language model

During training, BERT chooses randomly 15% of training tokens for masking

From those:
- Mask the token 80% of the time
- replace the token with another random word 10% of the time
- Leave the token unchanged 10% of the time
Next sentence prediction task

- In order to facilitate tasks that rely on two input sequences:
  - question answering
  - textual entailment

- Sentences are chosen into pretraining so that 50% of the time the sequence B is an actual next sentence to the sequence A
- 50% of the time sequence B is not an actual sequence to sentence A, but a random sentence from the input corpus
- A special [SEP] token is used to separate the two texts

- The sequence-initial [CLS] token representation is used to train a binary classifier to discriminate between these two cases
Input to the BERT
Next sentence prediction task

Predict likelihood that sentence B belongs after sentence A.

Input

[CLS] the man [MASK] to the store [SEP] penguin [MASK] are flightless birds [SEP]

Sentence A  Sentence B
Training BERT model

- Training a BERT model from scratch is typically called **pretraining**
- During training, BERT optimizes simultaneously:
  - masked language model loss
  - next sentence prediction loss
- The pretraining should be done on large amounts of texts:
  - generic domains (news, wikipedia, web texts)
  - specific domains (clinical or biomedical domains, twitter, reddit etc)
- Pretraining on GPUs or TPUs (typically on several)
- Pretraining can take weeks
Fine-tuning

• The pretrained BERT is typically used as a basis for **fine-tuning** some downstream task

• This is a form of **transfer learning:**
  • the downstream task can use all the knowledge BERT accumulated during pretraining
  • fine-tuning trains a special classifier on top of BERT specific to the downstream task
Fine-tuning options

- Freeze the BERT model parameters, only train the classifier
  - Quickest but might not work so well
- Train the classifier and also fine-tune all BERT parameters
  - Slowest but probably adapts the best
- Train the classifier and fine-tune only part of the BERT parameters, for instance in the higher layers
  - Probably optimal in terms for training time and overall performance
  - Need to some tweaking inside the model by yo
Task-specific fine-tuning: input

- One input sequence for:
  - text classification
  - morphological/syntactic tasks
  - NER etc

- Two input sequences for tasks like:
  - question answering
  - textual entailment
  - paraphrase pairs etc
Task-specific fine-tuning: output

- For classification tasks:
  - Fit a dense classification layer + softmax on top of the CLS representation
  - The classifier can be also more complex: several dense layers with non-linearity in between
  - Can construct the document representation also by averaging the token representations

- For sequence tagging tasks:
  - Fit a dense classification layer + softmax on top of each token representation
  - Use the first piece of each token to represent the word (remember: the input was subword tokenized)
Task-specific fine-tuning: text classification
Text classification based on two inputs

- For instance – the task of textual entailment
- Decide if and how the two sentences are related

- Positive entailment:
  - text: If you help the needy, God will reward you.
  - hypothesis: Giving money to a poor man has good consequences.

- Negative entailment – contradiction:
  - text: If you help the needy, God will reward you.
  - hypothesis: Giving money to a poor man has no consequences.

- No entailment – no relation
  - If you help the needy, God will reward you.
  - Giving money to a poor man will make you a better person.
Text classification based on two inputs
Task-specific fine-tuning: token classification
Available BERT models

- Huggingface transformers library offers good access to available BERT models: https://huggingface.co/transformers/

- Various versions of pretrained BERT: cased and uncased, smaller and larger models

- RoBERTa
  - A BERT model trained on even larger dataset and longer
  - Did not use the next sentence prediction task
  - Often a better basis than BERT

- ALBERT, TinyBERT – smaller models
Multilingual BERT models

Several multilingual BERT models are available via huggingface Bert base multilingual cased:

- trained on 104 largest Wikipedia languages
- 110 shared WordPiece vocabulary
- includes languages like Estonian, Russian, Ukraine

There are also other BERT-like models:

- XLM masked LM trained on 100 languages
- DistilmBERT distilled from multilingual BERT model
- XLM-RoBERTa large trained on 100 languages

XLM-RoBERTa seems to be the most competitive
Non-English BERT models

- BERT models are available for many languages now, including:
  - Chinese
  - German
  - Japanese
  - Finnish
  - Dutch

- These models are also available via huggingface transformers library

- We have also pretrained models for Estonian: [https://huggingface.co/tartuNLP](https://huggingface.co/tartuNLP)

- [https://bertlang.unibocconi.it](https://bertlang.unibocconi.it) – a web page that consolidates info about language-specific BERT models
Limitations of BERT

● Computational complexity
  ● Deep models, thus lots of computational resources are needed both for pretraining, but also fine-tuning and simply for making predictions

● The input size is limited
  ● Most models are limited to the input length of 128 or 512
  ● This is the maximum number of subword tokens and not word tokens
  ● Thus, in order to process longer texts, more complex handling is necessary
Summary

- Pretrained language models offer a general representational component for many NLP tasks.
- They help to improve the models especially in low resource setting.
- Major results in NLP during the last decade:
  - static word embeddings
  - recurrent neural networks
  - encoder-decoder networks
  - attention mechanism
  - Transformer architecture
  - Pre-trained large language models based on transformers architecture
- These advances have opened an area for lots of research and experiments for many languages and tasks.