Natural Language Processing

Lecture : Contextual Word Embeddings

Master Degree in Computer Engineering
University of Padua
Lecturer : Giorgio Satta

Contextual word embeddings

Magda Ehlers on Pexels

Contextual word embeddings

Word embeddings such as word2vec or GloVe learn a single vector for each **type** (unique word) in the vocabulary V. These are also called **static embeddings**.

However, in natural language words have rich linguistic relationships with other words that may be far away.

See also the discussion on the limitations of N-gram models.

Example :
I walked along the **pond**, and noticed one of the trees along the **bank**

Example :
The **chicken** didn't cross the road because **it** was too tired
The chicken didn't cross the **road** because **it** was too wide

Contextual word embeddings

By contrast, **contextual word embeddings** represent each **token** (word occurrence) by a different vector, according to the context the token appears in.

Also called dynamic embeddings or token embeddings.

Incorporating context into word embeddings has proven to be a **watershed idea** in NLP, opening the way to transfer learning.

We discuss transfer learning and fine-tuning in a later lecture.

Contextual word embeddings

How can we learn to represent words along with the context they occur in?

We train a neural network combining ideas from word embeddings and language models (in historical order):

- predict a word from left and right context **independently**

  Bidirectional LSTM (ELMo)

- predict a word from left and right context **jointly**

  Transformer encoder (BERT)

- predict a word from left context only

  Transformer decoder (GPT-*n*)

ELMo (embeddings from language model) looks at the entire sentence before assigning each word its embedding.

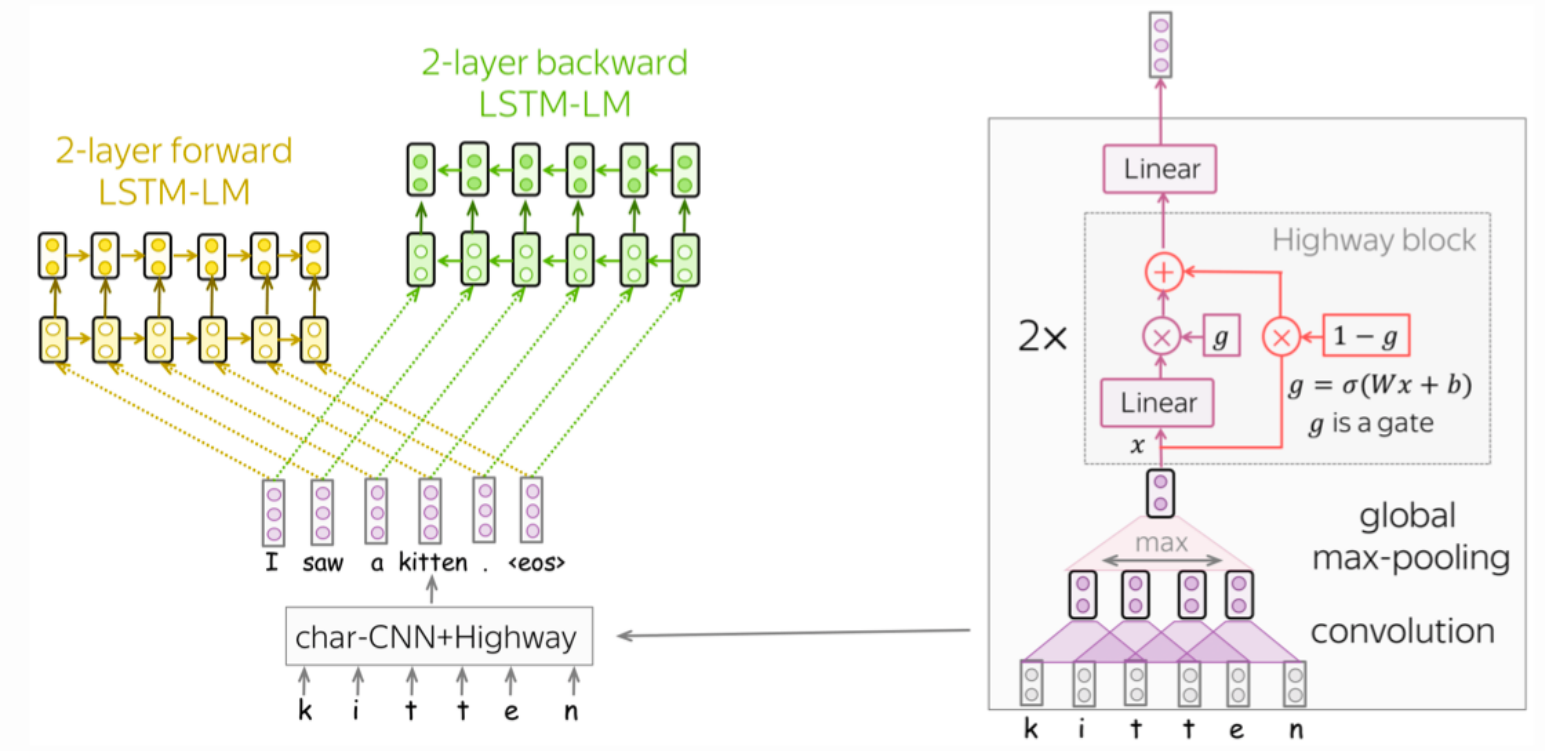
Created by researchers at the Allen Institute for Artificial Intelligence (AI2) and University of Washington, 2018.

Tokens are processed by a character-level convolutional neural network (CNN) producing word-level embeddings.

This captures word morphology and resolves OOV words.

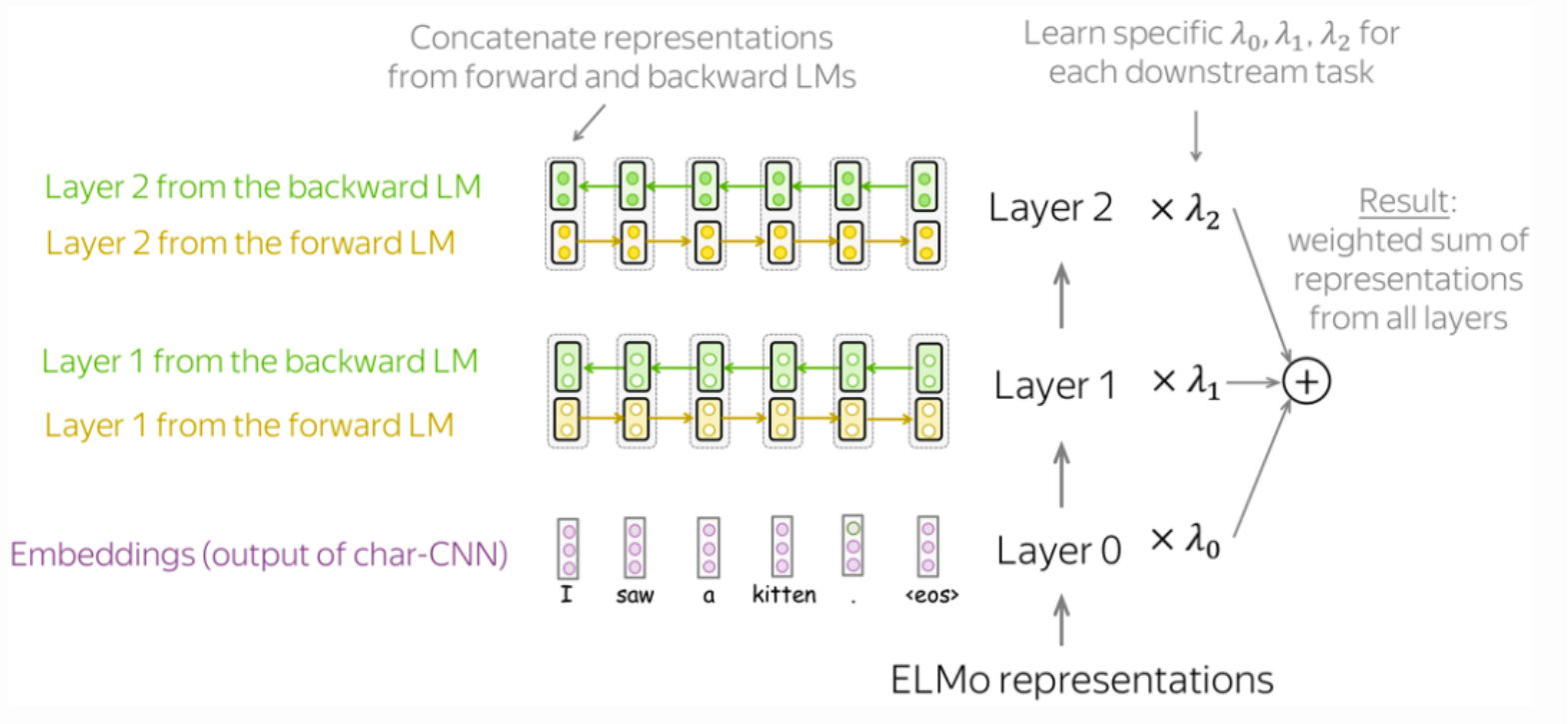
These embeddings are processed by a left-to-right and a right-to-left, 2-layers LSTM.

This is also called a bi-directional LSTM.



https://lena-voita.github.io/nlp_course/transfer_learning.html

For each word, the output embeddings at each layer (including the CNN layer) are combined, producing contextual embeddings.



https://lena-voita.github.io/nlp_course/transfer_learning.html

When **training**, add output layer with

- linear transformation to $|V|$ dimension
- softmax

and use cross-entropy as in neural language models.

The training objective is **bidirectional** causal language modeling, i.e., predicting the next word in both forward and backward directions.

Also recommended: parameter dropout and L2 regularization.

When **encoding** the input

- ignore output layer
- retain the word embedding represented by the last hidden layer.

BERT

©PBS/HBO Sesame Street

BERT

See Jurafsky & Martin §8 for Transformer neural networks. See also the introduction in the outlet in the Moodle of this course.

BERT (bidirectional encoder representations from transformers) produces word representations by **jointly** conditioning on both left and right context.

Created by researchers at Google AI Language, 2018.

The model is based on the **encoder** component of the transformer neural network. Tokenizer uses WordPiece algorithm.

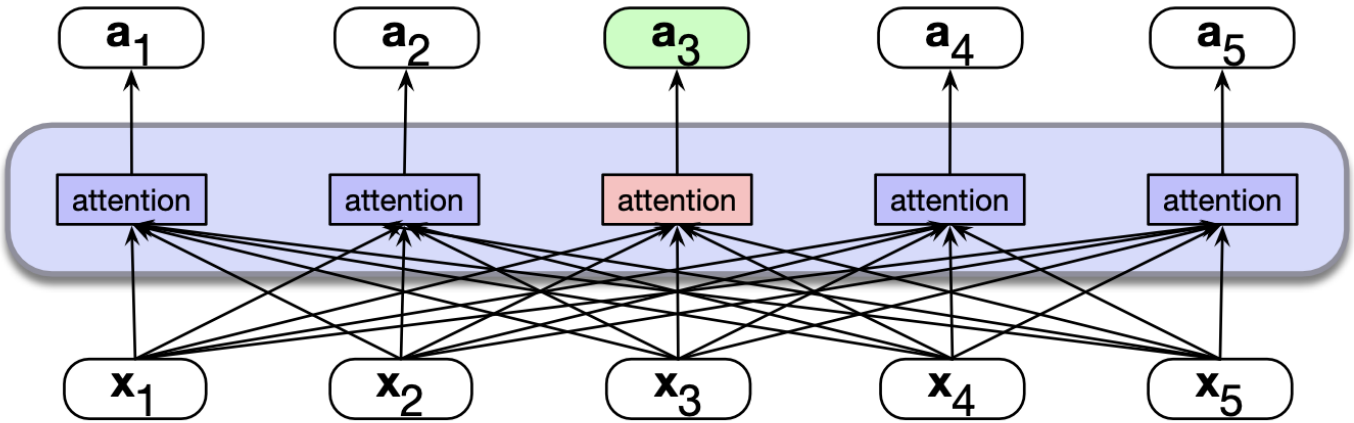
Two main versions

- BERT Base: 12 layers, 12 attention heads, 768 embedding size, 110 million parameters
- BERT Large: 24 layers, 16 attention heads, 1024 embedding size, 340 million parameters

BERT

The bidirectional encoding in BERT is inherited from the self-attention mechanism of the transformer encoder.

Recall that the encoder does not use masking for the computation of attention.



Masked language modeling

The model learns to perform a fill-in-the-blank task, technically called the **cloze task**.

Example : Please turn _____ homework in.

The approach is called **masked language modeling** (MLM).
A random sample of 15% of the input tokens are

- replaced with the unique vocabulary token `[MASK]` (80%)
- replaced with another token randomly sampled with unigram probabilities (10%)
- left unchanged (10%)

Masked language modeling

Why the three possible manipulations?

Adding the `[MASK]` token creates a mismatch between training and inference.

If we just replaced tokens with the `[MASK]`, the model might only predict tokens when it sees a `[MASK]`.

However we want the model to try to always predict the input token.

Masked language modeling

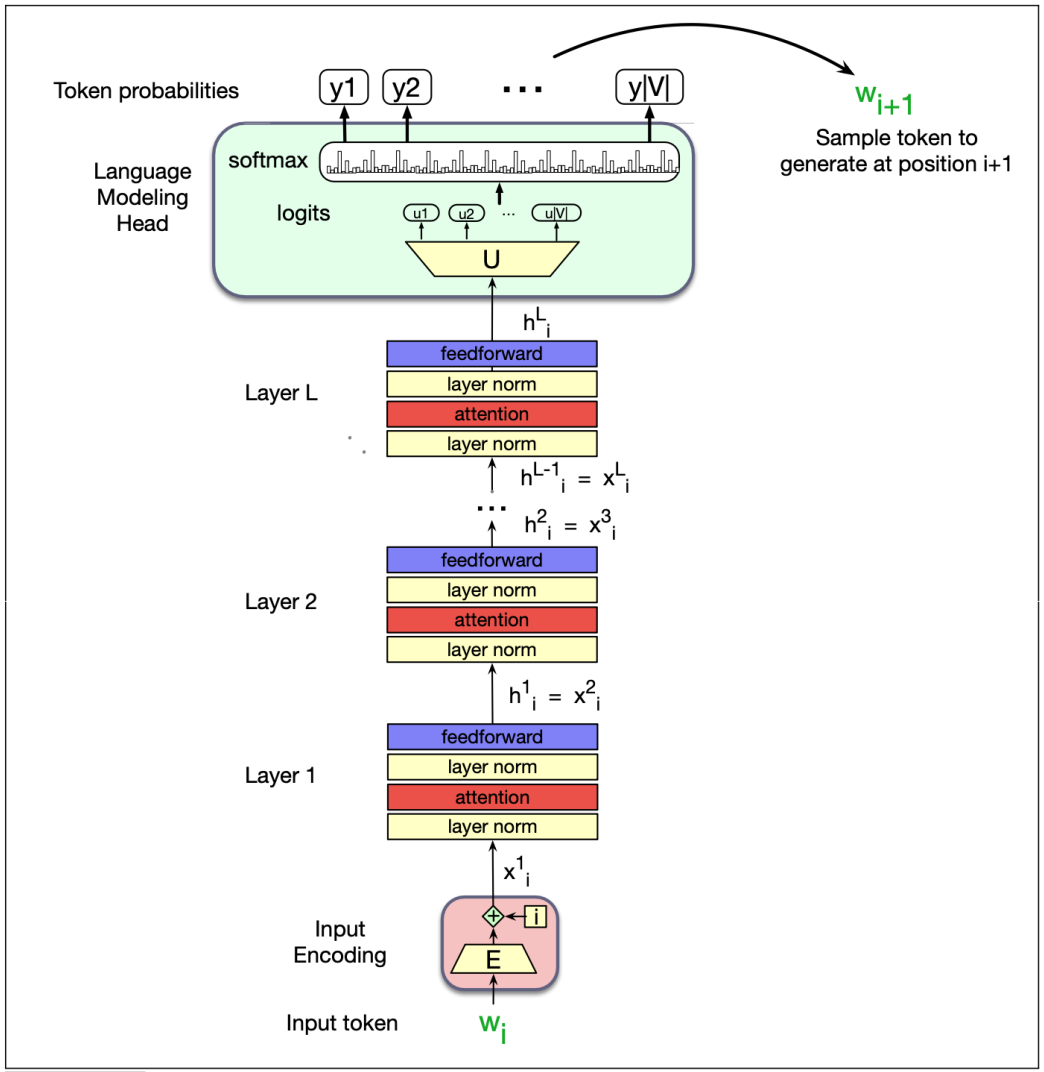
We combine masked language modeling with a **language modeling head**, which

- takes the output vector $\mathbf{h}_i^L$ for each of the masked tokens
- multiplies it by the unembedding layer $\mathbf{E}^\top$ to produce the logits $\mathbf{u}$; in other words, language modeling head is tied to the embedding matrix $\mathbf{E}$.
- uses softmax to turn the logits into probabilities $\mathbf{y}$ over the vocabulary, to **predict** the masked word

$$\begin{aligned} \mathbf{u}_i &= \mathbf{E}^\top \mathbf{h}_i^L \\ \mathbf{y}_i &= \text{softmax}(\mathbf{u}_i) \end{aligned}$$

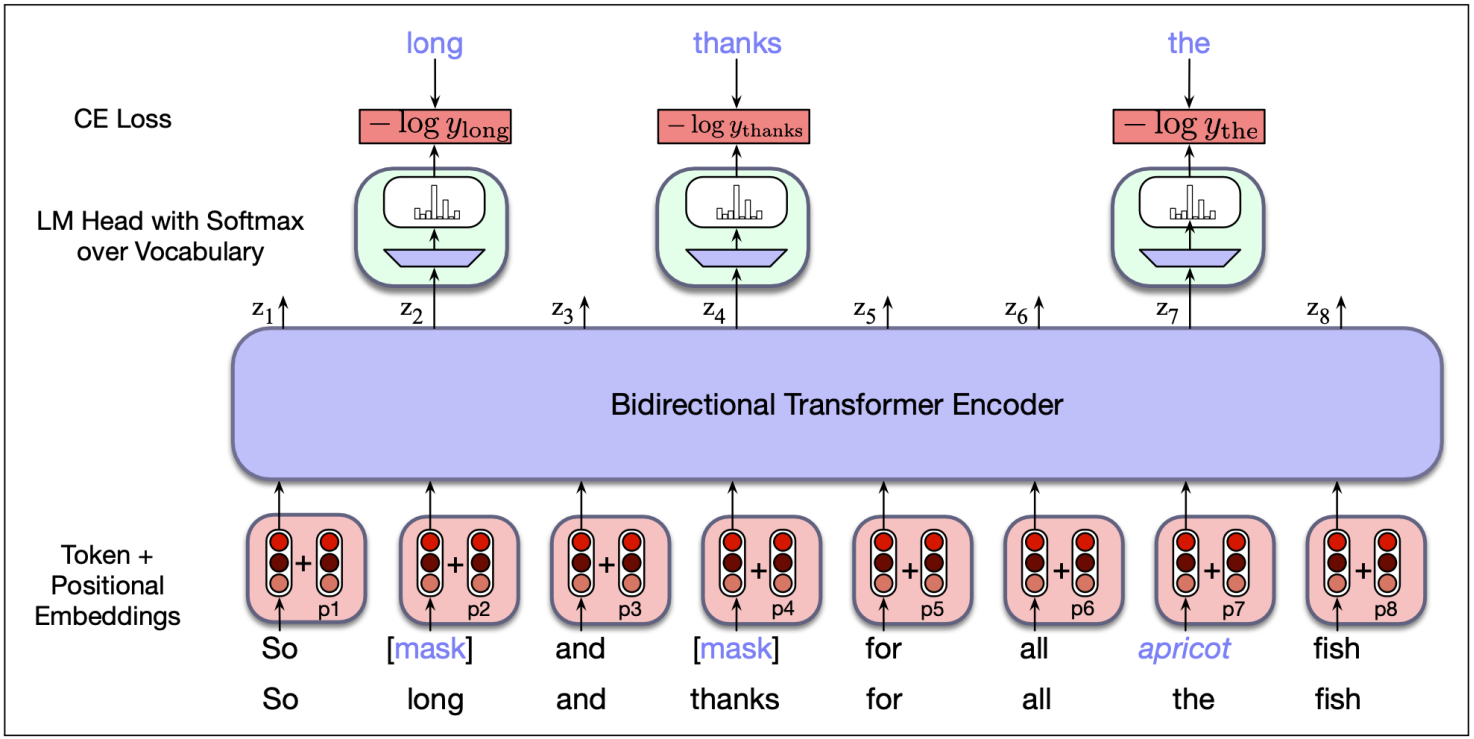
Masked language modeling

Full stack view of the language modeling head.



Masked language modeling

Training is carried out as usual for language modeling, using **cross-entropy** as loss function.



Masked language modeling

Let x_i be an input token that has been masked. The loss at position i is the $-\log$ of the probability of the correct token x_i, given the hidden vector $\mathbf{h}_i^L$

$$L_{MLM}(x_i) = -\log P(x_i \mid \mathbf{h}_i^L)$$

Let also M be the set of token positions in a batch that have been masked. The loss function is the average loss over the sampled learning items

$$L_{MLM}(M) = -\frac{1}{|M|} \sum_{i \in M} \log P(x_i \mid \mathbf{h}_i^L)$$

Masked language modeling

Masked language modeling can be generalized to any method that corrupts the training input and asks to recover the original.

Examples include

- masking of several adjacent words
- reorderings of adjacent words
- insertion of extraneous words

The general name for this kind of training is **denoising**.

Next sentence prediction

Many important downstream tasks involve relationship between pairs of sentences:

- **paraphrase detection**: detecting if two sentences have similar meanings, also called paraphrase identification
- **semantic textual similarity**: detecting the degree of semantic similarity between two sentences
- **sentence entailment**: detecting if the meanings of two sentences entail or contradict each other

This is not directly captured by language modeling.

Next sentence prediction

In order to train a model that understands sentence relationships, BERT introduces a second learning objective called **next sentence prediction** (NSP).

The model is asked to predict whether a pair of sentences can be adjacent or are totally unrelated.

NSP produces **sentence embeddings** that can be used for tasks involving relationship between pairs of sentences.

Next sentence prediction

NSP is a binary decision task that can be trained from **pairs** of sentences sampled from any monolingual corpus.

Each pair consists of

- an actual pair of adjacent sentences from the training corpus (50%)
- a pair of unrelated sentences randomly selected (50%)

  Used as in contrastive learning.

Next sentence prediction

NSP introduces two special tokens

- token `[CLS]` prepended to the first sentence
- token `[SEP]` placed between the two sentences and after the rightmost token of the second sentence

In NSP an additional **segment** embedding is added to the token embedding and the positional embedding, marking the first and second sentences.

See later slides for a picture.

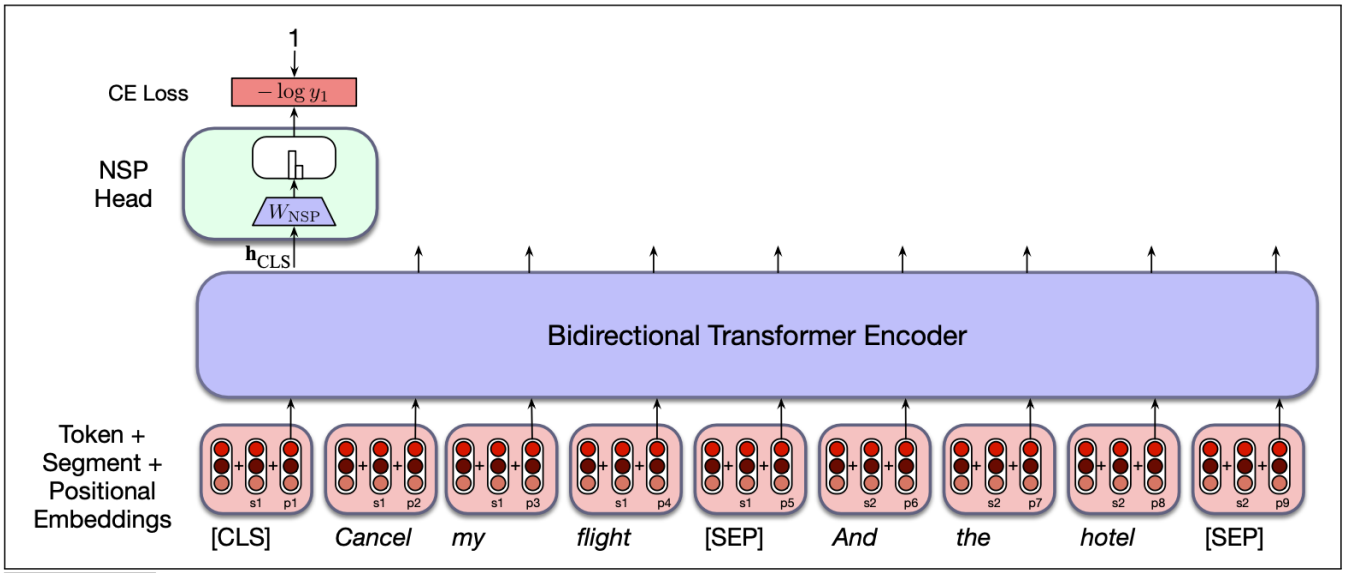
Next sentence prediction

We add a special NSP head to hidden vector $\mathbf{h}^L_{\text{CLS}}$, using learnable matrix $\mathbf{W_{NSP}} \in \mathbb{R}^{d \times 2}$ for two-class prediction

$$\mathbf{y}_i \quad = \quad \text{softmax}(\mathbf{W_{NSP}} \mathbf{h}^L_{\text{CLS}})$$

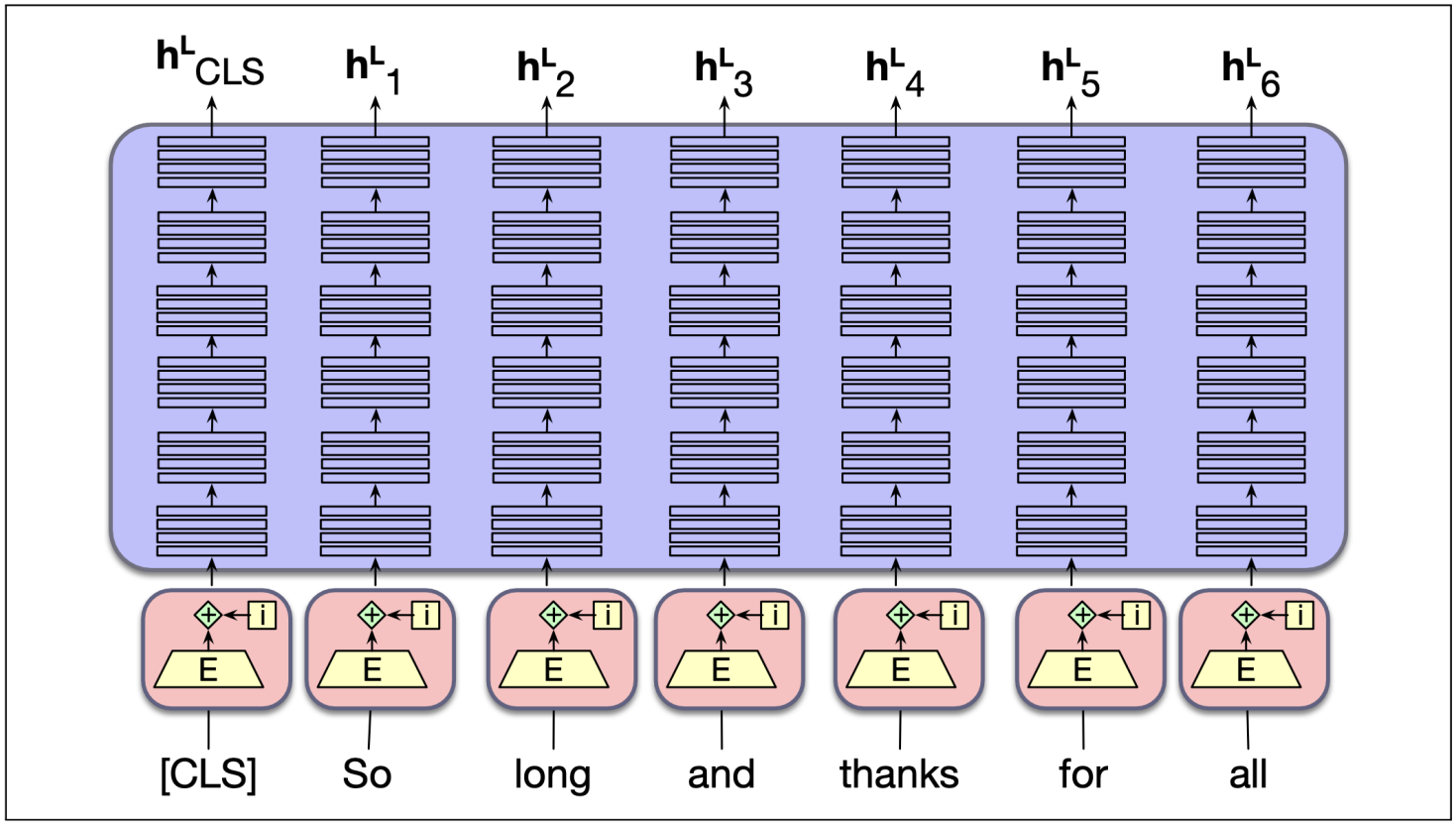
Next sentence prediction

Cross entropy is used to compute the NSP loss for each sentence pair presented to the model.



Contextual embeddings

At inference time, the output of BERT is a **contextual word embedding** vector $\mathbf{h}_i^L$ for input token x_i.



Contextual embeddings

For BERT, it is common to compute a representation for token w_t by **averaging** the output embeddings at t from each of the topmost layers of the model.

`[CLS]` embedding also used as **sentence embedding** for tasks involving single sentence classification.

Contextual embeddings

We can use the contextual embeddings for several tasks

- word-sense disambiguation (WSD)
- text classification, e.g. sentiment analysis and spam detection
- natural language inference (NLI)
- named entity recognition (NER)

We now discuss some of these tasks individually.

Sentiment analysis

Assume a three-way sentiment classification task: positive, negative, neutral.

Let $\mathbf{W}_C \in \mathbb{R}^{3\times d}$ and $\mathbf{b} \in \mathbb{R}^3$ be arrays of learnable parameters, where d is the size of BERT embeddings.

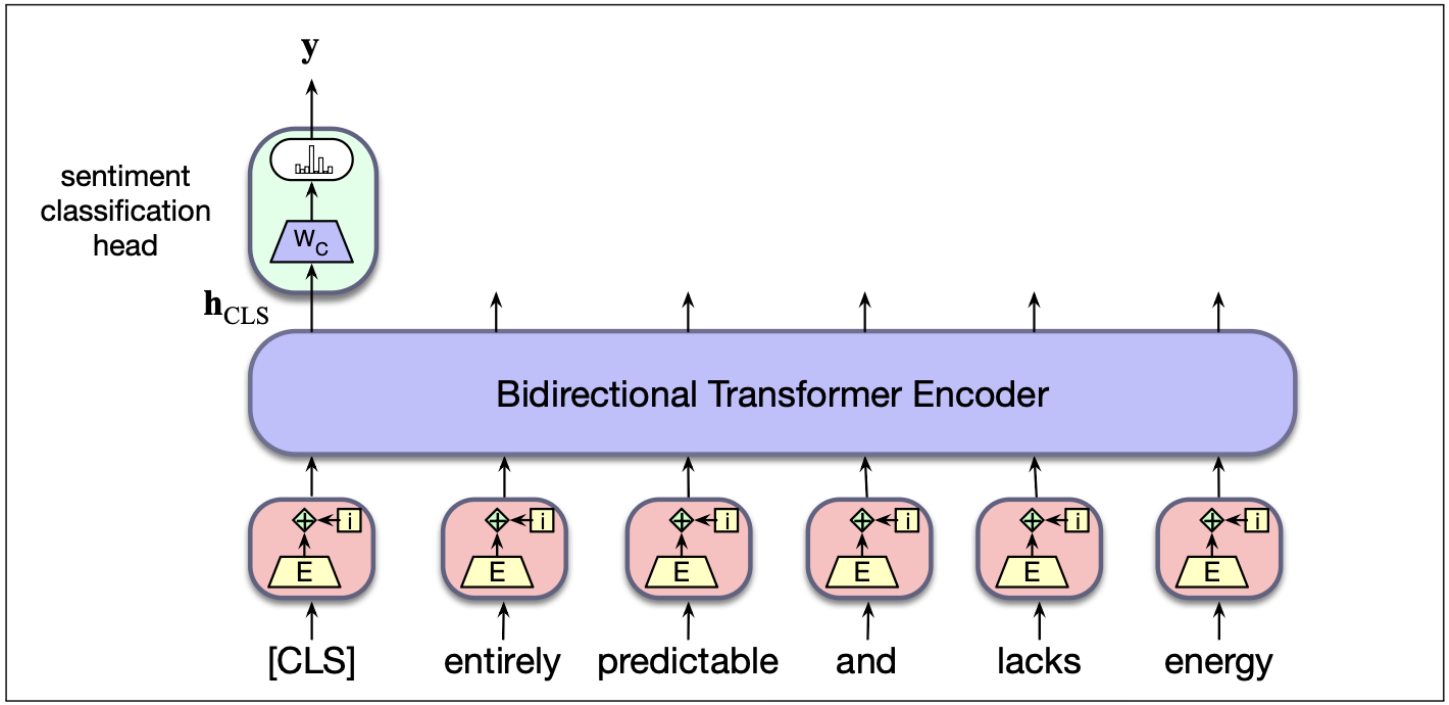
We set a **classification head** for the `[CLS]` token and map $\mathbf{h}^L_{CLS}$ to three logits, and pass through the softmax

$$\mathbf{y} = \text{softmax}(\mathbf{W}_C\,\mathbf{h}^L_{CLS} + \mathbf{b})$$

This requires supervised training on data consisting of input sequences labeled with the appropriate sentiment class.

Sentiment analysis

Example :



Named entity recognition

A **named entity** is anything that can be referred to with a proper name: a person, a location, an organization, a geographical location, etc.

The task of named entity recognition is to find spans of text that constitute named entities, and to tag them with the proper class.

Example : $[_{\text{PER}}$ Jane Villanueva] of $[_{\text{ORG}}$ United], a unit of $[_{\text{ORG}}$ United Airlines Holding], said the fare applies to the $[_{\text{LOC}}$ Chicago] route.

Named entity recognition

The standard approach for span-recognition is BIO tagging.

In **BIO tagging**

- tokens that begin a span are marked with label B
- tokens that occur inside a span in a position other than the leftmost one are marked with label I
- tokens outside of any span of interest are marked with O

Spans never overlap.

Named entity recognition

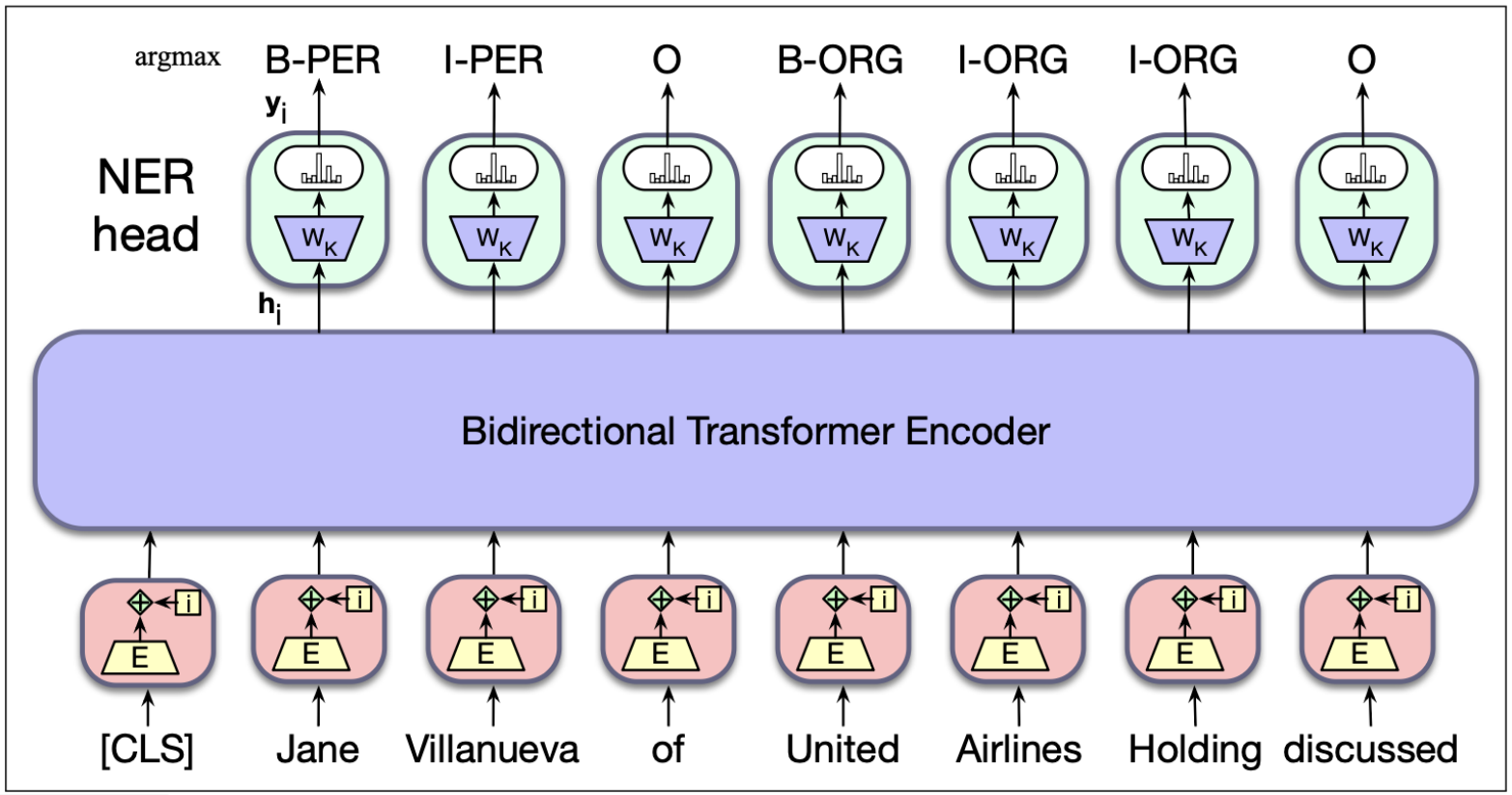
BIO tagging along with alternative tagging techniques for span.

Words	IO Label	BIO Label	BIOES Label
Jane	I-PER	B-PER	B-PER
Villanueva	I-PER	I-PER	E-PER
of	O	O	O
United	I-ORG	B-ORG	B-ORG
Airlines	I-ORG	I-ORG	I-ORG
Holding	I-ORG	I-ORG	E-ORG
discussed	O	O	O
the	O	O	O
Chicago	I-LOC	B-LOC	S-LOC
route	O	O	O
.	O	O	O

Note the difference between the sequences BI and BB, indicating one 2-word expression vs. two 1-word expressions.

Named entity recognition

Example :



BERT family

BERT was significantly undertrained.

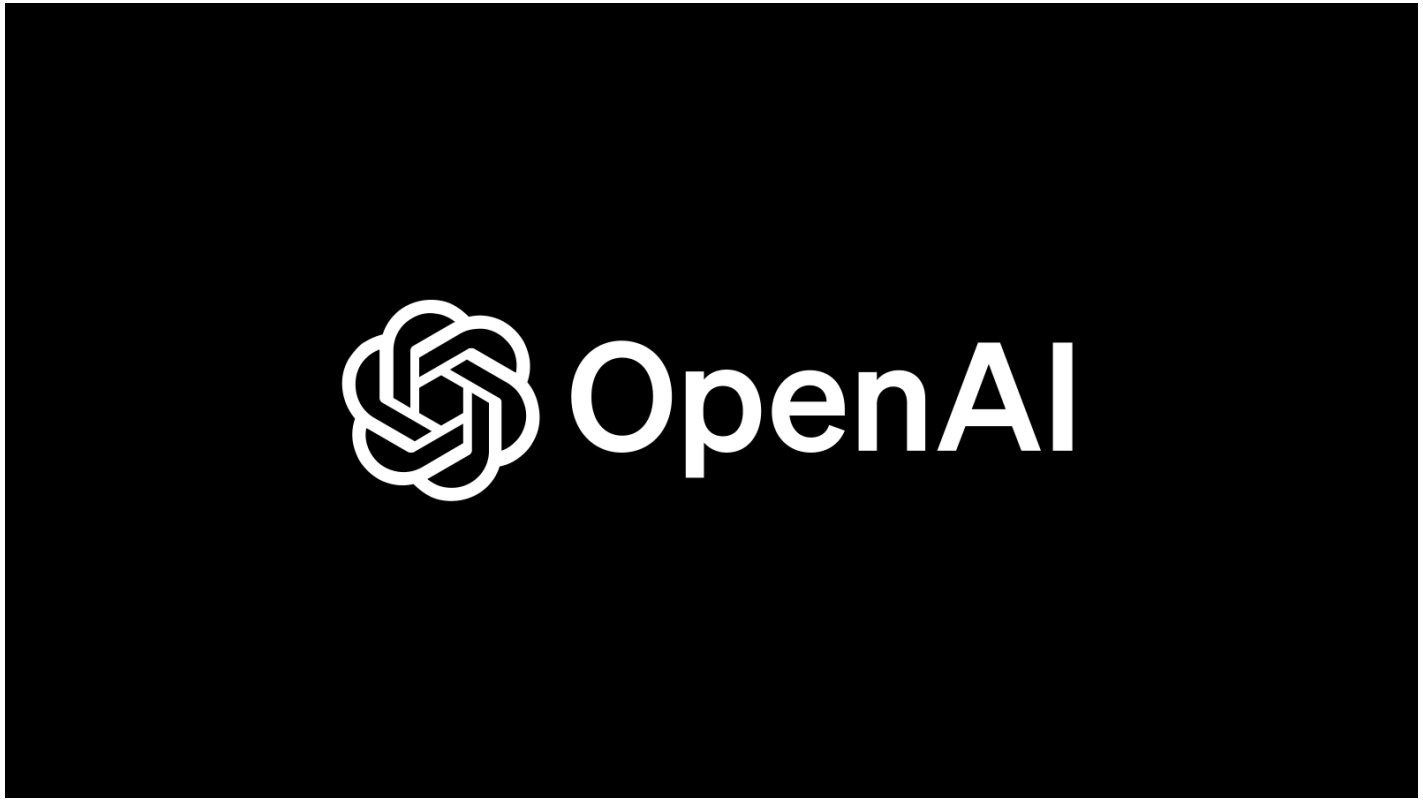
RoBERTa (Robustly Optimized BERT Approach) is based on a novel recipe for training BERT

- longer training over more data ($\times 10$) with larger batches
- removing NSP pre-training objective
- dynamically changing the masking pattern

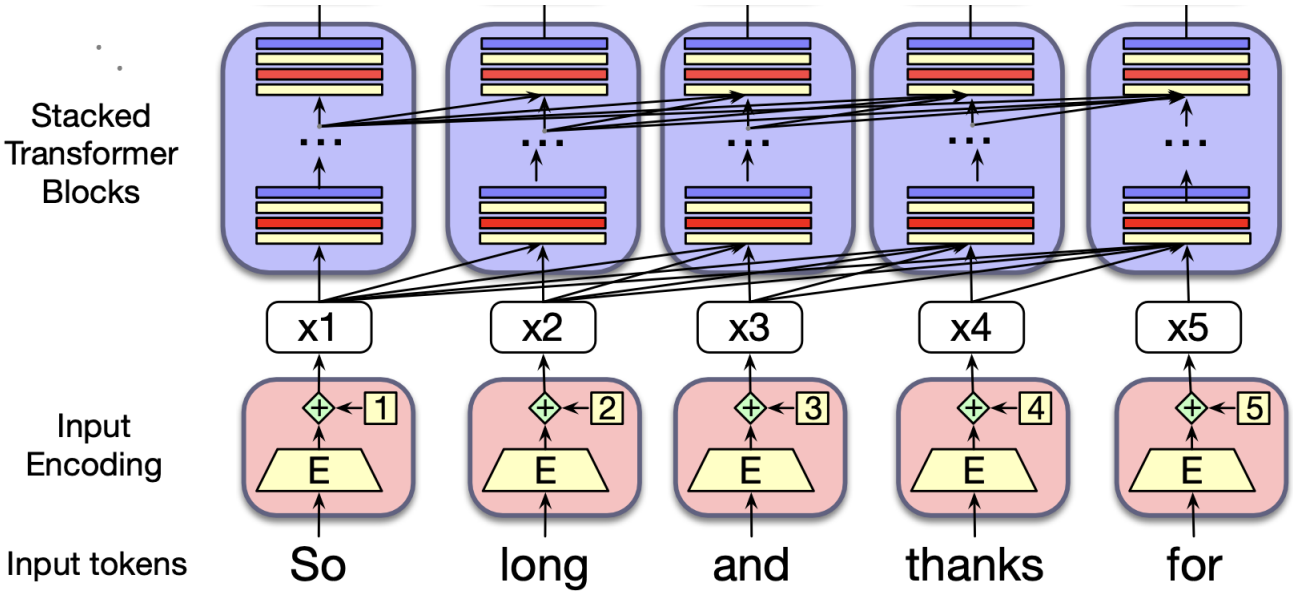
  BERT pre-training data are masked once for all
- using a byte level BPE tokenizer with larger vocabulary

Several more variants of BERT: ALBERT, BART, SpanBERT, SBERT, mBERT.

GPT-*n*



GPT (Generative Pre-training Transformer) produces contextual word embeddings conditioning on left context only.



The architecture is based on the transformer **decoder**.

SBERT

Samuel Ferrara on Unsplash

SBERT

In some sentence-pair regression tasks, we need to derive **sentence embeddings** such that semantically similar sentences are close in vector space.

Sentence embeddings can be computed through BERT by

- using the output of the `[CLS]` token, or else
- mean/max pooling BERT output layer

The above methods result in **bad** sentence embeddings, often worse than averaged static embeddings.

This is so because embeddings produced by BERT are not **directly** optimized for the above semantic task.

SBERT

Alternatively, one could

- compute semantic similarity for a pair of sentences by passing through BERT cross-encoder
- iterate for each pair to find the semantically closest pair

This setup **does not scale** due to

- the computational cost of cross-encoder
- the quadratic number of possible sentence combinations

SBERT

Sentence-BERT, or SBERT for short, is an efficient and effective method to compute sentence embeddings on the basis of pre-trained BERT model.

The basic idea underlying SBERT

- encode sentences independently
- compare embeddings via some objective function
- adjust BERT parameters, to a minimum extent, using **siamese neural network**

The approach avoids processing each pair through cross-encoding.

SBERT

Sentence embeddings are obtained from BERT word embeddings through several **pooling strategies**

- mean pooling: the average of all output token vectors. This usually yields the best results for similarity tasks.
- max-over-time pooling: taking the maximum value across all token vectors for each dimension, across all tokens in the sequence.
- `[CLS]` token: using the representation of the special classification token.

SBERT

SBERT is fine-tuned on labeled data (like SNLI or MultiNLI) using different **objective functions**, depending on the task

- classification objective: concatenate vectors u, v, and their element-wise difference $|u - v|$, then pass to a softmax classifier

$$\text{softmax}(\mathbf{W}[u; v; |u - v|])$$

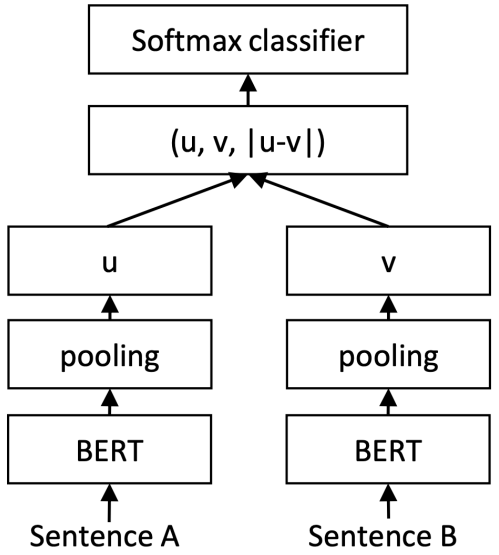
- regression objective: directly minimize the difference between the cosine similarity of (u, v) and a gold Standard similarity score
- triplet objective: an anchor sentence is pushed closer to a positive sentence and further from a negative one

SBERT

SBERT uses a **siamese neural network**, a special network that contains two identical subnetworks sharing their parameters.

Siamese networks are most commonly used to compute **similarity scores** for the inputs, and have many applications in computer vision.

SBERT training



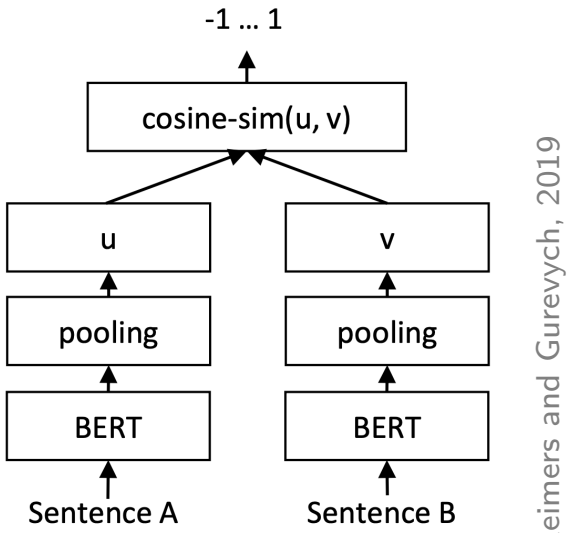
Reimers and Gurevych, 2019

Assume n the embeddings dimension and k the number of labels.

Classification objective function concatenates vectors $u, v \in \mathbb{R}^n$ and the element-wise difference $|u - v|$, and uses trainable matrix $\mathbf{W} \in \mathbb{R}^{3n \times k}$ to compute

$$\text{softmax}(\mathbf{W}[u; v; |u - v|])$$

SBERT training



Reimers and Gurevych, 2019

Regression objective function compares sentence embeddings u and v using a cosine similarity, which will output a semantic similarity score.

SBERT training

Assume an anchor sentence a, a positive sentence p, and a negative sentence n.

For $x \in \{a, p, n\}$ let s_x be the sentence embedding for x.

Triple objective function minimizes the following loss function

$$\max(|s_a - s_p| - |s_a - s_n| + \epsilon, 0)$$

In words, we tune the network such that the distance between a and p is smaller than the distance between a and n, modulo some margin ϵ.

SBERT training

During optimization, BERT is **fine-tuned**: this means we update only few of BERT parameters, typically those at the uppermost layers (see later lectures).

Parameter updating is **mirrored** (shared) across both sub-networks. This ensures vectors u and v live in the same semantic space.

SBERT applications

The fine-tuned BERT can then be used to compute semantically meaningful sentence embeddings for several scenarios

- semantic textual similarity (STS): comparing how similar two sentences are (e.g., for plagiarism detection)
- clustering: grouping thousands of documents into topics in seconds
- information retrieval: dense vector search where a query finds the most meaningful match, not just keyword matches

SBERT maintains BERT's accuracy while being fast enough for real-time applications.