Multilingual BERT

Timothee Mickus

October 28th, 2020

Outline

Contextual Word Embeddings

BERT as a Transformer

How is BERT trained?

Many languages at once

Exercise

Where do contextual embeddings come from?

Outline

Contextual Word Embeddings

A very general timeline

The general idea has always been to turn a word into a dense vector of real value. Theoretical works generally stress a connection with the distributional hypothesis (Harris, 1954; Firth, 1957)

Contextual Word Embeddings

A very general timeline

The general idea has always been to turn a word into a dense vector of real value. Theoretical works generally stress a connection with the distributional hypothesis (Harris, 1954; Firth, 1957)

- stems from information retrieval ('70s)

Contextual Word Embeddings

A very general timeline

The general idea has always been to turn a word into a dense vector of real value. Theoretical works generally stress a connection with the distributional hypothesis (Harris, 1954; Firth, 1957)

- stems from information retrieval ('70s)
- popularization of word vectors as "distributional semantics" in the '90s

Contextual Word Embeddings

A very general timeline

The general idea has always been to turn a word into a dense vector of real value. Theoretical works generally stress a connection with the distributional hypothesis (Harris, 1954; Firth, 1957)

- stems from information retrieval ('70s)
- popularization of word vectors as “distributional semantics” in the '90s
- first *neural* embeddings in 2003

Contextual Word Embeddings

A very general timeline

The general idea has always been to turn a word into a dense vector of real value. Theoretical works generally stress a connection with the distributional hypothesis (Harris, 1954; Firth, 1957)

- stems from information retrieval ('70s)
- popularization of word vectors as "distributional semantics" in the '90s
- first *neural* embeddings in 2003
- wide-spread use of embeddings from 2013 onward

Contextual Word Embeddings

A very general timeline

The general idea has always been to turn a word into a dense vector of real value. Theoretical works generally stress a connection with the distributional hypothesis (Harris, 1954; Firth, 1957)

- stems from information retrieval ('70s)
- popularization of word vectors as "distributional semantics" in the '90s
- first *neural* embeddings in 2003
- wide-spread use of embeddings from 2013 onward
- first *contextualized* neural embeddings 2017

Contextual Word Embeddings

The Rise of Contextual Embeddings

Embeddings for words in context. The trend mostly caught on in 2018

Contextual Word Embeddings

The Rise of Contextual Embeddings

Embeddings for words in context. The trend mostly caught on in 2018

- CoVe McCann et al. (2017)
- ELMo Peters et al. (2018)
- OpenAI GPT Radford (2018)
- BERT Devlin et al. (2018)
- and now many more variants of these

Contextual Word Embeddings

The Rise of Contextual Embeddings

Embeddings for words in context. The trend mostly caught on in 2018

- CoVe McCann et al. (2017)
- ELMo Peters et al. (2018)
- OpenAI GPT Radford (2018)
- BERT Devlin et al. (2018)
- and now many more variants of these

Explosive gains across multiple NLP tasks

Contextual Word Embeddings

The Rise of Contextual Embeddings

Embeddings for words in context. The trend mostly caught on in 2018

- COVE McCann et al. (2017)
- ELMO Peters et al. (2018)
- OpenAI GPT Radford (2018)
- BERT Devlin et al. (2018)
- and now many more variants of these

Explosive gains across multiple NLP tasks

- Work has been going on to understand how they function

Contextual Word Embeddings

The Rise of Contextual Embeddings

Embeddings for words in context. The trend mostly caught on in 2018

- CoVe McCann et al. (2017)
- ELMo Peters et al. (2018)
- OpenAI GPT Radford (2018)
- BERT Devlin et al. (2018)
- and now many more variants of these

Explosive gains across multiple NLP tasks

- Work has been going on to understand how they function
- Much work remains to be done

Contextual Word Embeddings

What changed: from words to sentences

> *Unlike most widely used word embeddings, [...] [contextual] word representations are functions of the entire input sequence*

Peters et al. (2018)

Contextual Word Embeddings

What changed: from words to sentences

> *Unlike most widely used word embeddings, [...] [contextual] word representations are functions of the entire input sequence*
>
> Peters et al. (2018)

- Contextualized representations guarantee a bijection between sequences of words and sequences of vectors, not between words and vectors individually.

Contextual Word Embeddings

What changed: from words to sentences

> *Unlike most widely used word embeddings, [...] [contextual] word representations are functions of the entire input sequence*
>
> Peters et al. (2018)

- Contextualized representations guarantee a bijection between sequences of words and sequences of vectors, not between words and vectors individually.
  - Has interesting consequences, such as the fact that the sum of all vectors for a sentence is sensitive to order ($\neq$ BoW)

Contextual Word Embeddings

What changed: from words to sentences

> *Unlike most widely used word embeddings, [...] [contextual] word representations are functions of the entire input sequence*
>
> Peters et al. (2018)

- Contextualized representations guarantee a bijection between sequences of words and sequences of vectors, not between words and vectors individually.
  - Has interesting consequences, such as the fact that the sum of all vectors for a sentence is sensitive to order ($\neq$ BoW)
  - An equivalent way of stating this: contextualized embeddings are word-*token* vectors, non-contextualized embeddings are word-*type* vectors

Contextual Word Embeddings

What changed: from words to sentences

> *Unlike most widely used word embeddings, [...] [contextual] word representations are functions of the entire input sequence*

Peters et al. (2018)

- Contextualized representations guarantee a bijection between sequences of words and sequences of vectors, not between words and vectors individually.
  - Has interesting consequences, such as the fact that the sum of all vectors for a sentence is sensitive to order ($\neq$ BoW)
  - An equivalent way of stating this: contextualized embeddings are word-*token* vectors, non-contextualized embeddings are word-*type* vectors
  - Also entails that existing word embeddings benchmarks may not be appropriate

Contextual Word Embeddings

What changed: from words to sentences

> *Unlike most widely used word embeddings, [...] [contextual] word representations are functions of the entire input sequence*

Peters et al. (2018)

- Contextualized representations guarantee a bijection between sequences of words and sequences of vectors, not between words and vectors individually.
  - Has interesting consequences, such as the fact that the sum of all vectors for a sentence is sensitive to order ($\neq$ BoW)
  - An equivalent way of stating this: contextualized embeddings are word-*token* vectors, non-contextualized embeddings are word-*type* vectors
  - Also entails that existing word embeddings benchmarks may not be appropriate
- Unlike sentence encoders, which merge together in a single vector all the semantics of the sentence, contextualized embedding algorithm assign to each token a representation that is a function of the entire input sentence.

Contextual Word Embeddings

What changed: fine-tuning vs. feature-based models

Devlin et al. (2018) suggest two ways of using embeddings

1. Fine-tuning: Achieving state-of-the-art performance on multiple tasks by simply fine-tuning the embeddings model.

   NB: 'catastrophic forgetting' can occur

Contextual Word Embeddings

What changed: fine-tuning vs. feature-based models

Devlin et al. (2018) suggest two ways of using embeddings

1. Fine-tuning: Achieving state-of-the-art performance on multiple tasks by simply fine-tuning the embeddings model.
   NB: 'catastrophic forgetting' can occur

2. Contrasts with previous non-contextualized embeddings which were most of the time used as additional features for more complex, often task-specific models
   NB: still possible with contextualized representations

Outline

BERT as a Transformer

The BERT hype

- BERT is a contextualized embedding algorithm designed to assign a sequence of vectors to a sequence of words

BERT as a Transformer

The BERT hype

- BERT is a contextualized embedding algorithm designed to assign a sequence of vectors to a sequence of words
- BERT is designed to be used as generally as possible

BERT as a Transformer

The BERT hype

- BERT is a contextualized embedding algorithm designed to assign a sequence of vectors to a sequence of words
- BERT is designed to be used as generally as possible
- BERT is based on the Transformer architecture, which is trendy but pretty much not understood

BERT as a Transformer

The BERT hype

- BERT is a contextualized embedding algorithm designed to assign a sequence of vectors to a sequence of words
- BERT is designed to be used as generally as possible
- BERT is based on the Transformer architecture, which is trendy but pretty much not understood
- BERT is trained on two tasks at once:
  - word-level MLM, derived from a standard psychology test
  - sentence-level Next Sentence Prediction, which allows for sentence relationship awareness

BERT as a Transformer

The BERT hype

- BERT is a contextualized embedding algorithm designed to assign a sequence of vectors to a sequence of words
- BERT is designed to be used as generally as possible
- BERT is based on the Transformer architecture, which is trendy but pretty much not understood
- BERT is trained on two tasks at once:
  - word-level MLM, derived from a standard psychology test
  - sentence-level Next Sentence Prediction, which allows for sentence relationship awareness
- BERT has dominated many benchmarks

BERT as a Transformer

The BERT hype

- BERT is a contextualized embedding algorithm designed to assign a sequence of vectors to a sequence of words
- BERT is designed to be used as generally as possible
- BERT is based on the Transformer architecture, which is trendy but pretty much not understood
- BERT is trained on two tasks at once:
  - word-level MLM, derived from a standard psychology test
  - sentence-level Next Sentence Prediction, which allows for sentence relationship awareness
- BERT has dominated many benchmarks
- So many studies on BERT that we coined the term “Bertology”

BERT as a Transformer

BERT (Devlin et al., 2018) is a simplified encoder from a Transformer (Vaswani et al., 2017)

BERT as a Transformer

BERT (Devlin et al., 2018) is a simplified encoder from a Transformer (Vaswani et al., 2017)

- A Transformer encoder is a stack of L layers divided into two sub-layers, each using residual connection and layer-normalization:

$$\text{SubLayer} = \text{LNorm}(\vec{x} + F(\vec{x}))$$

BERT as a Transformer

BERT (Devlin et al., 2018) is a simplified encoder from a Transformer (Vaswani et al., 2017)

- A Transformer encoder is a stack of L layers divided into two sub-layers, each using residual connection and layer-normalization:

$$\text{SubLayer} = \text{LNorm}(\vec{x} + F(\vec{x}))$$

- Informally, residual connections allow the upper layers to still retain some information from the input, whereas normalization ensure that intermediate representations have a similar scale

BERT as a Transformer

BERT (Devlin et al., 2018) is a simplified encoder from a Transformer (Vaswani et al., 2017)

- A Transformer encoder is a stack of L layers divided into two sub-layers, each using residual connection and layer-normalization:

$$\text{SubLayer} = \text{LNorm}(\vec{x} + F(\vec{x}))$$

- Informally, residual connections allow the upper layers to still retain some information from the input, whereas normalization ensure that intermediate representations have a similar scale

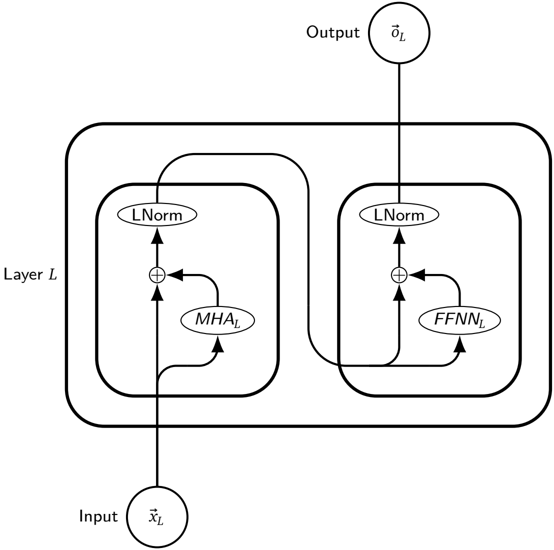
- Layer-normalization employs two learned parameters, a bias $\vec{b}$ and a gain $\vec{g}$:

$$\text{LayerNorm}(\vec{x}) = \vec{g} \odot \frac{\vec{x} - \mu_{\vec{x}}}{\sigma_{\vec{x}}} + \vec{b}$$

  where $\mu_{\vec{x}}$ is the mean of the components of $\vec{x}$, and $\sigma_{\vec{x}}$ the standard deviation, and $\odot$ is element-wise multiplication

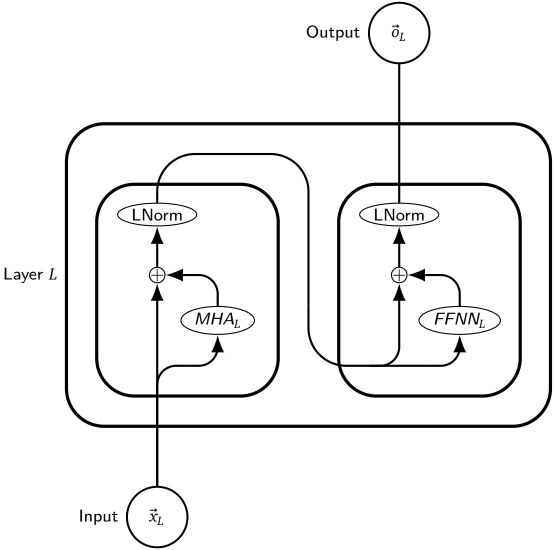
BERT as a Transformer

Transformer Layer at a glance



BERT as a Transformer

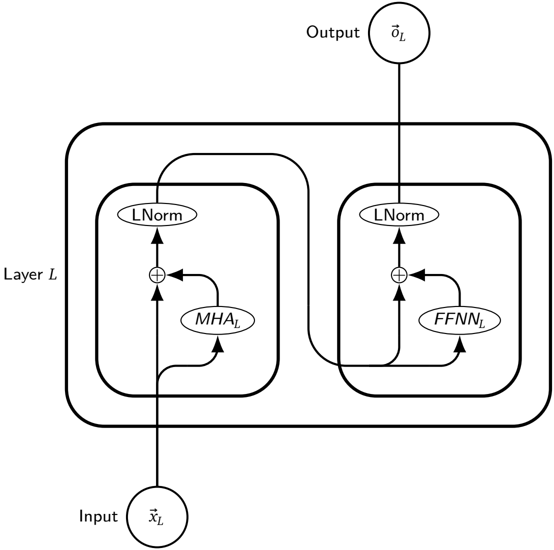
Transformer Layer at a glance



- Many implementation apply layer-norm before the layer function (MHA or FFN)

BERT as a Transformer

Transformer Layer at a glance



- Many implementation apply layer-norm before the layer function (MHA or FFN)
- In any event, there exists a path from input to output passing only by residual connections and layer norms

BERT as a Transformer

Multi-Head Attention

- The first sub-layer applies scaled-dot self-attention:

$$\text{Attention}(Q,K,V) = \text{Softmax}(\frac{Q \cdot K^T}{\sqrt{d_K}})V$$

NB: dot product can be seen as a measure of similarity: $\vec{u} \cdot \vec{v} = \|\vec{u}\|_2 \ \|\vec{v}\|_2 \ \cos(\vec{u}, \vec{v})$

BERT as a Transformer

Multi-Head Attention

- The first sub-layer applies scaled-dot self-attention:

$$\text{Attention}(Q,K,V) = \text{Softmax}\left(\frac{Q \cdot K^T}{\sqrt{d_K}}\right)V$$

NB: dot product can be seen as a measure of similarity: $\vec{u} \cdot \vec{v} = \|\vec{u}\|_2 \; \|\vec{v}\|_2 \; \cos(\vec{u}, \vec{v})$

- ... combined with multi-head attention, ie. each attention sublayer has A learned linear projections for queries Q, keys K and values V

$$\text{MultiHead}(Q,K,V) = W_{\text{MHA}} \cdot \Big(\bigoplus_{a}^{A} \text{Attention}(W_q^a Q, W_k^a K, W_v^a V)\Big)$$

where $\oplus$ denotes concatenation

BERT as a Transformer

Multi-Head Attention

- The first sub-layer applies scaled-dot self-attention:

$$\text{Attention}(Q,K,V) = \text{Softmax}(\frac{Q \cdot K^T}{\sqrt{d_K}})V$$

NB: dot product can be seen as a measure of similarity: $\vec{u} \cdot \vec{v} = \|\vec{u}\|_2 \; \|\vec{v}\|_2 \; \cos(\vec{u}, \vec{v})$

- ... combined with multi-head attention, ie. each attention sublayer has A learned linear projections for queries Q, keys K and values V

$$\text{MultiHead}(Q,K,V) = W_{\text{MHA}} \cdot \Big(\bigoplus_{a}^{A} \text{Attention}(W_q^a Q, W_k^a K, W_v^a V) \Big)$$

where $\oplus$ denotes concatenation

- Queries Q, keys K and values V correspond to the previous layer's output.

BERT as a Transformer

Multi-Head Attention

- The first sub-layer applies scaled-dot self-attention:

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{Q \cdot K^T}{\sqrt{d_K}}\right)V$$

NB: dot product can be seen as a measure of similarity: $\vec{u} \cdot \vec{v} = \|\vec{u}\|_2 \; \|\vec{v}\|_2 \; \cos(\vec{u}, \vec{v})$

- ... combined with multi-head attention, ie. each attention sublayer has A learned linear projections for queries Q, keys K and values V

$$\text{MultiHead}(Q, K, V) = W_{\text{MHA}} \cdot \Big(\bigoplus_{a}^{A} \text{Attention}(W_q^a Q, W_k^a K, W_v^a V) \Big)$$

where $\oplus$ denotes concatenation

- Queries Q, keys K and values V correspond to the previous layer's output.
- Think of query vectors as questions, value vectors as possible answers, key vectors as how fitting a given answer is for a given type of questions
    - $\vec{q}$ encodes information

BERT as a Transformer

Multi-Head Attention

- The first sub-layer applies scaled-dot self-attention:

$$\text{Attention}(Q,K,V) = \text{Softmax}(\frac{Q \cdot K^T}{\sqrt{d_K}})V$$

NB: dot product can be seen as a measure of similarity: $\vec{u} \cdot \vec{v} = ||\vec{u}||_2 \; ||\vec{v}||_2 \; \cos(\vec{u}, \vec{v})$

- ... combined with multi-head attention, ie. each attention sublayer has A learned linear projections for queries Q, keys K and values V

$$\text{MultiHead}(Q,K,V) = W_{\text{MHA}} \cdot \Big(\bigoplus_{a}^{A} \text{Attention}(W_q^a Q, W_k^a K, W_v^a V)\Big)$$

where $\oplus$ denotes concatenation

- Queries Q, keys K and values V correspond to the previous layer's output.
- Think of query vectors as questions, value vectors as possible answers, key vectors as how fitting a given answer is for a given type of questions
  - $\vec{q}$ encodes information
  - $\vec{q} \cdot \vec{k_1}, \ldots, \vec{q} \cdot \vec{k_n}$ quantify how similar each item is to $\vec{q}$

BERT as a Transformer

Multi-Head Attention

- The first sub-layer applies scaled-dot self-attention:

$$\text{Attention}(Q, K, V) = \text{Softmax}(\frac{Q \cdot K^T}{\sqrt{d_K}})V$$

NB: dot product can be seen as a measure of similarity: $\vec{u} \cdot \vec{v} = \|\vec{u}\|_2 \, \|\vec{v}\|_2 \, \cos(\vec{u}, \vec{v})$

- ... combined with multi-head attention, ie. each attention sublayer has A learned linear projections for queries Q, keys K and values V

$$\text{MultiHead}(Q, K, V) = W_{\text{MHA}} \cdot \Big(\bigoplus_{a}^{A} \text{Attention}(W_q^a Q, W_k^a K, W_v^a V) \Big)$$

where $\oplus$ denotes concatenation

- Queries Q, keys K and values V correspond to the previous layer's output.
- Think of query vectors as questions, value vectors as possible answers, key vectors as how fitting a given answer is for a given type of questions
  - $\vec{q}$ encodes information
  - $\vec{q} \cdot \vec{k_1}, \ldots, \vec{q} \cdot \vec{k_n}$ quantify how similar each item is to $\vec{q}$
  - Softmax$(\ldots)V$ is a weighted average that strongly favors the most similar elements

BERT as a Transformer

Layer Wiring

- The second sub-layer is a feed forward network, composed of two linear transformations with a rectified linear unit activation in between :

$$FFN(\vec{x}) = \mathrm{ReLU}(\vec{x}W_1 + \vec{b_1})W_2 + \vec{b_2}$$

BERT as a Transformer

Layer Wiring

- The second sub-layer is a feed forward network, composed of two linear transformations with a rectified linear unit activation in between :

$$FFN(\vec{x}) = \mathrm{ReLU}(\vec{x}W_1 + \vec{b_1})W_2 + \vec{b_2}$$

- The systems uses learned embeddings to convert the input tokens.

BERT as a Transformer

Layer Wiring

- The second sub-layer is a feed forward network, composed of two linear transformations with a rectified linear unit activation in between :

$$FFN(\vec{x}) = \mathrm{ReLU}(\vec{x}W_1 + \vec{b_1})W_2 + \vec{b_2}$$

- The systems uses learned embeddings to convert the input tokens.

- To provide the model with information relative to the position of a word in a sequence, position encoding vectors are added to the corresponding embeddings :

$$\mathrm{PositionEncoding}(\mathrm{pos}) = \langle\overrightarrow{c(\mathrm{pos}, 1),\ \ldots,\ c(\mathrm{pos}, d_e)}\rangle$$

BERT as a Transformer

Layer Wiring

- The second sub-layer is a feed forward network, composed of two linear transformations with a rectified linear unit activation in between :

$$FFN(\vec{x}) = \mathrm{ReLU}(\vec{x}W_1 + \vec{b_1})W_2 + \vec{b_2}$$

- The systems uses learned embeddings to convert the input tokens.

- To provide the model with information relative to the position of a word in a sequence, position encoding vectors are added to the corresponding embeddings :

$$\mathrm{PositionEncoding}(\mathrm{pos}) = \langle\overrightarrow{c(\mathrm{pos}, 1),\ \ldots,\ c(\mathrm{pos}, d_e)}\rangle$$

where each component of the position encoding vector is defined using :

$$c(\mathrm{pos}, \mathrm{dim}) = \begin{cases} \sin(\frac{\mathrm{pos}}{10000^{\mathrm{dim}/d_e}}) \text{ if } \mathrm{dim} = 2k \\ \cos(\frac{\mathrm{pos}}{10000^{\mathrm{dim}/d_e}}) \text{ otherwise.} \end{cases}$$

BERT as a Transformer

Layer Wiring

- The second sub-layer is a feed forward network, composed of two linear transformations with a rectified linear unit activation in between :

$$FFN(\vec{x}) = \mathrm{ReLU}(\vec{x}W_1 + \vec{b_1})W_2 + \vec{b_2}$$

- The systems uses learned embeddings to convert the input tokens.
- To provide the model with information relative to the position of a word in a sequence, position encoding vectors are added to the corresponding embeddings :

$$\mathrm{PositionEncoding}(\mathrm{pos}) = \langle\overrightarrow{c(\mathrm{pos}, 1), \ \ldots, \ c(\mathrm{pos}, d_e)}\rangle$$

where each component of the position encoding vector is defined using :

$$c(\mathrm{pos}, \mathrm{dim}) = \begin{cases} \sin(\frac{\mathrm{pos}}{10000^{\mathrm{dim}/d_e}}) \text{ if } \mathrm{dim} = 2k \\ \cos(\frac{\mathrm{pos}}{10000^{\mathrm{dim}/d_e}}) \text{ otherwise.} \end{cases}$$

- In other words the position encoding vectors are **fixed**.

BERT as a Transformer

Hyperparameters

The Transformer (more precisely its encoder) depends mostly on three hyperparameters :

BERT as a Transformer

Hyperparameters

The Transformer (more precisely its encoder) depends mostly on three hyperparameters :

- L, the number of layers

BERT as a Transformer

Hyperparameters

The Transformer (more precisely its encoder) depends mostly on three hyperparameters :

- L, the number of layers
- A, the number of attention heads

BERT as a Transformer

Hyperparameters

The Transformer (more precisely its encoder) depends mostly on three hyperparameters :

- L, the number of layers
- A, the number of attention heads
- H, the dimensionality of the hidden representations

BERT as a Transformer

Hyperparameters

The Transformer (more precisely its encoder) depends mostly on three hyperparameters :

- L, the number of layers
- A, the number of attention heads
- H, the dimensionality of the hidden representations

Various transformers have various hyperparameters settings:

BERT as a Transformer

Hyperparameters

The Transformer (more precisely its encoder) depends mostly on three hyperparameters :

- L, the number of layers
- A, the number of attention heads
- H, the dimensionality of the hidden representations

Various transformers have various hyperparameters settings:

- the original transformer by Vaswani et al. (2017) was $L = 6$, $H = 512$, $A = 8$

BERT as a Transformer

Hyperparameters

The Transformer (more precisely its encoder) depends mostly on three hyperparameters :

- L, the number of layers
- A, the number of attention heads
- H, the dimensionality of the hidden representations

Various transformers have various hyperparameters settings:

- the original transformer by Vaswani et al. (2017) was $L = 6$, $H = 512$, $A = 8$
- BERT-Base is $L = 12$, $H = 768$, $A = 12$

BERT as a Transformer

Hyperparameters

The Transformer (more precisely its encoder) depends mostly on three hyperparameters :

- L, the number of layers
- A, the number of attention heads
- H, the dimensionality of the hidden representations

Various transformers have various hyperparameters settings:

- the original transformer by Vaswani et al. (2017) was $L = 6$, $H = 512$, $A = 8$
- BERT-Base is $L = 12$, $H = 768$, $A = 12$
- BERT-Large is $L = 24$, $H = 1024$, $A = 16$

BERT as a Transformer

Hyperparameters

The Transformer (more precisely its encoder) depends mostly on three hyperparameters :

- L, the number of layers
- A, the number of attention heads
- H, the dimensionality of the hidden representations

Various transformers have various hyperparameters settings:

- the original transformer by Vaswani et al. (2017) was $L = 6$, $H = 512$, $A = 8$
- BERT-Base is $L = 12$, $H = 768$, $A = 12$
- BERT-Large is $L = 24$, $H = 1024$, $A = 16$
- ...

Outline

How is BERT trained?

Other than dropping the decoder altogether, BERT has very few amendments to the original Transformer algorithm

How is BERT trained?

Other than dropping the decoder altogether, BERT has very few amendments to the original Transformer algorithm

- the most important change is its learned sentence-specific embeddings (or 'segment' embeddings), which are used for the sentence-level objective (we'll get to it later)

How is BERT trained?

Other than dropping the decoder altogether, BERT has very few amendments to the original Transformer algorithm

- the most important change is its learned sentence-specific embeddings (or ‘segment’ embeddings), which are used for the sentence-level objective (we’ll get to it later)

- Some other minor changes involve the **systematic** use of word-piece to tokenize the input text.

How is BERT trained?

Other than dropping the decoder altogether, BERT has very few amendments to the original Transformer algorithm

- the most important change is its learned sentence-specific embeddings (or ‘segment’ embeddings), which are used for the sentence-level objective (we’ll get to it later)

- Some other minor changes involve the **systematic** use of word-piece to tokenize the input text.

- BERT uses GELU rather than ReLU activation

How is BERT trained?

Other than dropping the decoder altogether, BERT has very few amendments to the original Transformer algorithm

- the most important change is its learned sentence-specific embeddings (or ‘segment’ embeddings), which are used for the sentence-level objective (we’ll get to it later)

- Some other minor changes involve the **systematic** use of word-piece to tokenize the input text.

- BERT uses GELU rather than ReLU activation

Training procedure is deeply tied to input format.

How is BERT trained?

BERT input format

- To convert a sequence of tokens to a BERT input format:
  1. tokens are first embedded
  2. 'positional encodings' $\vec{p(i)}$ mark the position i of the token
  3. 'Segment encodings' $\vec{\mathtt{seg}}_A$, $\vec{\mathtt{seg}}_B$ mark which sentence tokens belong to
  4. 3 special tokens: `[SEP]` for sentence boundaries, `[CLS]` and `[MASK]` for performing the actual training
- Given the example "*My dog barks. It is a pooch.*", the actual input would be:

$$\begin{aligned}
\vec{\mathtt{[CLS]}} + \vec{p(0)} + \vec{\mathtt{seg}}_A, &\quad \vec{My} + \vec{p(1)} + \vec{\mathtt{seg}}_A,\\
\vec{dog} + \vec{p(2)} + \vec{\mathtt{seg}}_A, &\quad \vec{barks} + \vec{p(3)} + \vec{\mathtt{seg}}_A,\\
\vec{.} + \vec{p(4)} + \vec{\mathtt{seg}}_A, &\quad \vec{\mathtt{[SEP]}} + \vec{p(5)} + \vec{\mathtt{seg}}_A,\\
\vec{It} + \vec{p(6)} + \vec{\mathtt{seg}}_B, &\quad \vec{is} + \vec{p(7)} + \vec{\mathtt{seg}}_B,\\
\vec{a} + \vec{p(8)} + \vec{\mathtt{seg}}_B, &\quad \vec{pooch} + \vec{p(9)} + \vec{\mathtt{seg}}_B,\\
\vec{.} + \vec{p(10)} + \vec{\mathtt{seg}}_B, &\quad \vec{\mathtt{[SEP]}} + \vec{p(11)} + \vec{\mathtt{seg}}_B
\end{aligned}$$

- During training, some tokens at random will be masked using $\vec{\mathtt{[MASK]}}$.
- Positional encodings are only there for technical reasons, so we can safely ignore them.

How is BERT trained?

BERT is trained on two objectives simultaneously

1. A word-level objective

How is BERT trained?

BERT is trained on two objectives simultaneously

1. A word-level objective

2. A sentence-level objective

How is BERT trained?

MLM, aka. Cloze Test

The word-level objective for BERT comes from psychology (Taylor, 1953)

How is BERT trained?

MLM, aka. Cloze Test

The word-level objective for BERT comes from psychology (Taylor, 1953)

- The “Cloze Test”, also known as “Gap-Fill”, “Cloze deletion test”, “Fill in the blanks”...

How is BERT trained?

MLM, aka. Cloze Test

The word-level objective for BERT comes from psychology (Taylor, 1953)

- The “Cloze Test”, also known as “Gap-Fill”, “Cloze deletion test”, “Fill in the blanks”...
- In a given sentence a word (or a group of words) will be blanked out

How is BERT trained?

MLM, aka. Cloze Test

The word-level objective for BERT comes from psychology (Taylor, 1953)

- The “Cloze Test”, also known as “Gap-Fill”, “Cloze deletion test”, “Fill in the blanks”...
- In a given sentence a word (or a group of words) will be blanked out
- Subjects will then be tasked with filling in said blanks.

How is BERT trained?

MLM, aka. Cloze Test

The word-level objective for BERT comes from psychology (Taylor, 1953)

- The "Cloze Test", also known as "Gap-Fill", "Cloze deletion test", "Fill in the blanks"...
- In a given sentence a word (or a group of words) will be blanked out
- Subjects will then be tasked with filling in said blanks.
- It is mostly used as learning exercises to assess **reading proficiency** and **mastery of grammar**.

How is BERT trained?

MLM, aka. Cloze Test

The word-level objective for BERT comes from psychology (Taylor, 1953)

- The “Cloze Test”, also known as “Gap-Fill”, “Cloze deletion test”, “Fill in the blanks”...
- In a given sentence a word (or a group of words) will be blanked out
- Subjects will then be tasked with filling in said blanks.
- It is mostly used as learning exercises to assess **reading proficiency** and **mastery of grammar**.
- It has also been used jointly with eye-tracking.

How is BERT trained?

Implementing the Cloze Test as an objective

- The idea behind BERT is to train the Transformer architecture to do well on Cloze Test: if it can find the correct parameters to solve a reading exercise, then it's probably a decent textual representation.

How is BERT trained?

Implementing the Cloze Test as an objective

- The idea behind BERT is to train the Transformer architecture to do well on Cloze Test: if it can find the correct parameters to solve a reading exercise, then it's probably a decent textual representation.
- To do so, we need to formulate the Cloze as a task

How is BERT trained?

Implementing the Cloze Test as an objective

- The idea behind BERT is to train the Transformer architecture to do well on Cloze Test: if it can find the correct parameters to solve a reading exercise, then it's probably a decent textual representation.

- To do so, we need to formulate the Cloze as a task

- The task will be predict correctly an item that has been 'blanked out'.

How is BERT trained?

Implementing the Cloze Test as an objective

- The idea behind BERT is to train the Transformer architecture to do well on Cloze Test: if it can find the correct parameters to solve a reading exercise, then it's probably a decent textual representation.
- To do so, we need to formulate the Cloze as a task
- The task will be predict correctly an item that has been 'blanked out'.
- The prediction can be done using a simple softmax layer to which is fed the embedding of the blanked-out item.

How is BERT trained?

Implementing the Cloze Test as an objective

- The idea behind BERT is to train the Transformer architecture to do well on Cloze Test: if it can find the correct parameters to solve a reading exercise, then it's probably a decent textual representation.
- To do so, we need to formulate the Cloze as a task
- The task will be predict correctly an item that has been 'blanked out'.
- The prediction can be done using a simple softmax layer to which is fed the embedding of the blanked-out item.

This use of the Cloze Test as a training task was dubbed by the authors the 'Masked Language Model' task, or MLM for short.

How is BERT trained?

MLM, concretely

More concretely:

How is BERT trained?

MLM, concretely

More concretely:

- The model first randomly selects 15% of the input tokens, which will be fed to the softmax prediction layer.

How is BERT trained?

MLM, concretely

More concretely:

- The model first randomly selects 15% of the input tokens, which will be fed to the softmax prediction layer.
- 80% of the randomly selected items (= 12% of the word-pieces in total) will be replaced by a special token `[MASK]` representing a blank

How is BERT trained?

MLM, concretely

More concretely:

- The model first randomly selects 15% of the input tokens, which will be fed to the softmax prediction layer.
- 80% of the randomly selected items (= 12% of the word-pieces in total) will be replaced by a special token `[MASK]` representing a blank
- 10% of the randomly selected word-pieces (= 1.5% of the word-pieces in total) will be replaced by a word at random.

  NB: This is done to mitigate the mismatch between training and down-stream applications, where the special token `[MASK]` will never be encountered.

How is BERT trained?

MLM, concretely

More concretely:

- The model first randomly selects 15% of the input tokens, which will be fed to the softmax prediction layer.
- 80% of the randomly selected items (= 12% of the word-pieces in total) will be replaced by a special token `[MASK]` representing a blank
- 10% of the randomly selected word-pieces (= 1.5% of the word-pieces in total) will be replaced by a word at random.

  NB: This is done to mitigate the mismatch between training and down-stream applications, where the special token `[MASK]` will never be encountered.
- 10% of the randomly selected word-pieces (= 1.5% of the word-pieces in total) will be replaced by a word at random.

  NB: This is done to "bias the representation towards the actual observed word".

How is BERT trained?

Sentence-level objective

- We mentioned earlier that BERT had two objectives, the second being sentence-level

How is BERT trained?

Sentence-level objective

- We mentioned earlier that BERT had two objectives, the second being sentence-level
- This second objective is to predict whether a sentence immediately another in the corpus; it has been prosaically dubbed the “next sentence prediction” task

How is BERT trained?

Sentence-level objective

- We mentioned earlier that BERT had two objectives, the second being sentence-level
- This second objective is to predict whether a sentence immediately another in the corpus; it has been prosaically dubbed the “next sentence prediction” task
- This objective entails that BERT can only be trained on a corpus of coherent documents, and not on corpora composed of shuffled sentences

How is BERT trained?

Sentence-level objective

- We mentioned earlier that BERT had two objectives, the second being sentence-level
- This second objective is to predict whether a sentence immediately another in the corpus; it has been prosaically dubbed the “next sentence prediction” task
- This objective entails that BERT can only be trained on a corpus of coherent documents, and not on corpora composed of shuffled sentences
- This second objective supposedly helps on QA and NLI downstream tasks.

How is BERT trained?

Sentence-level objective

- We mentioned earlier that BERT had two objectives, the second being sentence-level
- This second objective is to predict whether a sentence immediately another in the corpus; it has been prosaically dubbed the “next sentence prediction” task
- This objective entails that BERT can only be trained on a corpus of coherent documents, and not on corpora composed of shuffled sentences
- This second objective supposedly helps on QA and NLI downstream tasks.
- There are serious concerns about the usefulness of this task, newer implementations of BERT don’t necessarily use it.

How is BERT trained?

Next sentence prediction

- NSP consists in a binary classification of paired sentences $\langle S_A, S_B \rangle$ between two labels `IsNext` and `NotNext`.

How is BERT trained?

Next sentence prediction

- NSP consists in a binary classification of paired sentences $\langle S_A, S_B \rangle$ between two labels `IsNext` and `NotNext`.
  - The first label `IsNext` corresponds to when S_A is immediately followed by S_B in the training corpus

How is BERT trained?

Next sentence prediction

- NSP consists in a binary classification of paired sentences $\langle S_A, S_B \rangle$ between two labels `IsNext` and `NotNext`.
  - The first label `IsNext` corresponds to when S_A is immediately followed by S_B in the training corpus
  - The second label `NotNext` corresponds to when S_A and S_B were just randomly and separately sampled from the corpus and paired together.

How is BERT trained?

Next sentence prediction

- NSP consists in a binary classification of paired sentences $\langle S_A, S_B \rangle$ between two labels `IsNext` and `NotNext`.
  - The first label `IsNext` corresponds to when S_A is immediately followed by S_B in the training corpus
  - The second label `NotNext` corresponds to when S_A and S_B were just randomly and separately sampled from the corpus and paired together.

- Sentences are presented as a contiguous span of text to the system, using two special tokens `[CLS]` and `[SEP]` as separators.

  NB: More concretely, if $S_A = w_1^A, \ldots, w_n^A$ and $S_B = w_1^B, \ldots, w_m^B$, the system will receive the following sequence as input: `[CLS]`, $w_1^A, \ldots, w_n^A$, `[SEP]`, $w_1^B, \ldots, w_m^B$, `[SEP]`

How is BERT trained?

Next sentence prediction

- NSP consists in a binary classification of paired sentences $\langle S_A, S_B \rangle$ between two labels `IsNext` and `NotNext`.
  - The first label `IsNext` corresponds to when S_A is immediately followed by S_B in the training corpus
  - The second label `NotNext` corresponds to when S_A and S_B were just randomly and separately sampled from the corpus and paired together.

- Sentences are presented as a contiguous span of text to the system, using two special tokens `[CLS]` and `[SEP]` as separators.
  NB: More concretely, if $S_A = w_1^A, ..., w_n^A$ and $S_B = w_1^B, ..., w_m^B$, the system will receive the following sequence as input: `[CLS]`, $w_1^A, \ldots, w_n^A$, `[SEP]`, $w_1^B, \ldots, w_m^B$, `[SEP]`

- To further facilitate the models ability to distinguish two sentences, learned sentences markers $\texttt{seg}_A$ for S_A and $\texttt{seg}_B$ for S_B are added respectively to `[CLS]`, $w_1^A, \ldots, w_n^A$, `[SEP]` and to $w_1^B, \ldots, w_m^B$, `[SEP]`

How is BERT trained?

Next sentence prediction

- NSP consists in a binary classification of paired sentences $\langle S_A, S_B \rangle$ between two labels `IsNext` and `NotNext`.
  - The first label `IsNext` corresponds to when S_A is immediately followed by S_B in the training corpus
  - The second label `NotNext` corresponds to when S_A and S_B were just randomly and separately sampled from the corpus and paired together.

- Sentences are presented as a contiguous span of text to the system, using two special tokens `[CLS]` and `[SEP]` as separators.

  NB: More concretely, if $S_A = w_1^A, \ldots, w_n^A$ and $S_B = w_1^B, \ldots, w_m^B$, the system will receive the following sequence as input: `[CLS]`, $w_1^A, \ldots, w_n^A$, `[SEP]`, $w_1^B, \ldots, w_m^B$, `[SEP]`

- To further facilitate the models ability to distinguish two sentences, learned sentences markers $\texttt{seg}_A$ for S_A and $\texttt{seg}_B$ for S_B are added respectively to `[CLS]`, $w_1^A, \ldots, w_n^A$, `[SEP]` and to $w_1^B, \ldots, w_m^B$, `[SEP]`

- Although not specified in the original paper, the sentence prediction only uses the `[CLS]` token for its prediction.

How is BERT trained?

Next sentence prediction

- NSP consists in a binary classification of paired sentences $\langle S_A, S_B \rangle$ between two labels `IsNext` and `NotNext`.
  - The first label `IsNext` corresponds to when S_A is immediately followed by S_B in the training corpus
  - The second label `NotNext` corresponds to when S_A and S_B were just randomly and separately sampled from the corpus and paired together.
- Sentences are presented as a contiguous span of text to the system, using two special tokens `[CLS]` and `[SEP]` as separators.

  NB: More concretely, if $S_A = w_1^A, \ldots, w_n^A$ and $S_B = w_1^B, \ldots, w_m^B$, the system will receive the following sequence as input: `[CLS]`, $w_1^A, \ldots, w_n^A$, `[SEP]`, $w_1^B, \ldots, w_m^B$, `[SEP]`
- To further facilitate the models ability to distinguish two sentences, learned sentences markers $\texttt{seg}_A$ for S_A and $\texttt{seg}_B$ for S_B are added respectively to `[CLS]`, $w_1^A, \ldots, w_n^A$, `[SEP]` and to $w_1^B, \ldots, w_m^B$, `[SEP]`
- Although not specified in the original paper, the sentence prediction only uses the `[CLS]` token for its prediction.
- The prediction is done using a simple linear classifier.

Outline

Many languages at once

BERT can be trained on multiple language at once.

- May help if there's no BERT for the specific language

Many languages at once

BERT can be trained on multiple language at once.

- May help if there's no BERT for the specific language
- May be less useful than language-specific BERT models (ZH, FR, FI, DE, NL, EN…)

Many languages at once

BERT can be trained on multiple language at once.

- May help if there's no BERT for the specific language
- May be less useful than language-specific BERT models (ZH, FR, FI, DE, NL, EN…)
- There are *many* language-specific models, so it's worth reading on the subject.

Many languages at once

BERT can be trained on multiple language at once.

- May help if there’s no BERT for the specific language
- May be less useful than language-specific BERT models (ZH, FR, FI, DE, NL, EN...)
- There are *many* language-specific models, so it’s worth reading on the subject.

"Multilingual BERT" model was trained on the 100 largest wikipedia dumps

Many languages at once

BPE

Input in BERT is tokenized using Byte-Pair Encoding (BPE) word-pieces.

Many languages at once

BPE

Input in BERT is tokenized using Byte-Pair Encoding (BPE) word-pieces.

To compute BPE:

Many languages at once

BPE

Input in BERT is tokenized using Byte-Pair Encoding (BPE) word-pieces.

To compute BPE:

1. start with all characters as possible tokens T

Many languages at once

BPE

Input in BERT is tokenized using Byte-Pair Encoding (BPE) word-pieces.

To compute BPE:

1. start with all characters as possible tokens T
2. tokenize the text according to T and whitespaces

Many languages at once

BPE

Input in BERT is tokenized using Byte-Pair Encoding (BPE) word-pieces.

To compute BPE:

1. start with all characters as possible tokens T
2. tokenize the text according to T and whitespaces
3. find the most frequent pair of tokens $\langle t_1, t_2 \rangle$

Many languages at once

BPE

Input in BERT is tokenized using Byte-Pair Encoding (BPE) word-pieces.

To compute BPE:

1. start with all characters as possible tokens T
2. tokenize the text according to T and whitespaces
3. find the most frequent pair of tokens $\langle t_1, t_2 \rangle$
4. add their concatenation $t_1 t_2$ to T

Many languages at once

BPE

Input in BERT is tokenized using Byte-Pair Encoding (BPE) word-pieces.

To compute BPE:

1. start with all characters as possible tokens T
2. tokenize the text according to T and whitespaces
3. find the most frequent pair of tokens $\langle t_1, t_2 \rangle$
4. add their concatenation $t_1 t_2$ to T
5. restart from 2.; loop until T reaches a certain size

Many languages at once

BPE in Multilingual BERT

In BERT multilingual, all data from all languages are merged into a single corpus, and a vocabulary of 110K BPE word-pieces is then computed.

Many languages at once

BPE in Multilingual BERT

In BERT multilingual, all data from all languages are merged into a single corpus, and a vocabulary of 110K BPE word-pieces is then computed.

- Chinese characters in multilingual BERT are considered as distinct tokens, since Chinese doesn't use whitespaces.

Many languages at once

BPE in Multilingual BERT

In BERT multilingual, all data from all languages are merged into a single corpus, and a vocabulary of 110K BPE word-pieces is then computed.

- Chinese characters in multilingual BERT are considered as distinct tokens, since Chinese doesn’t use whitespaces.
- all other language characters are lowercased and accent diacritics are removed

Many languages at once

BPE in Multilingual BERT

In BERT multilingual, all data from all languages are merged into a single corpus, and a vocabulary of 110K BPE word-pieces is then computed.

- Chinese characters in multilingual BERT are considered as distinct tokens, since Chinese doesn't use whitespaces.
- all other language characters are lowercased and accent diacritics are removed
- to avoid over- or under-representing languages when training the model and computing BPE, a sampling ratio per language is defined:

$$\hat{P}(L) = \frac{P(L)^S}{\sum_{L'} P(L')^S}$$

with $P(L)$ the original magnitude of the language L in the corpus, and S a smoothing factor (in the case of multilingual BERT, $S = 0.7$).

NB: The result is that more frequent languages are sampled less, whereas less frequent languages are sampled more.

Outline

Bertology 101: Studying the Stability of the BERT Semantic Space

Testing BERT on a similarity benchmark

In this exercise, you will be comparing BERT embeddings drawn from two different corpora on a word-type similarity benchmark.

1. Download data

- **Python code for loading BERT:** Download the original repository using `git clone https://github.com/google-research/bert.git`. *It is highly recommended to use a dedicated virtual environment, eg. using `python3 -m venv bert-venv`.*
- **Download the model itself:** All information pertaining to this step should be on the github you installed in the previous step.
- **Retrieving a similarity benchmark:** Download the MEN dataset from https://staff.fnwi.uva.nl/e.bruni/MEN, and retrieve the file `MEN_dataset_natural_form_full`. It contains space-separated triples, composed of two words and a similarity rating.
- **Retrieve sentence corpora:** BERT embeddings are computed "on the fly", so there is no file containing the exact embeddings (unlike word2vec for instance). As a result, you need sentences corpora to compute embeddings from: retrieve the two pre-parsed corpora from the lecture's github (derived from Wikipedia and OpenSubtitles).

2. Retrieve word type representation

- **Retrieve word token embeddings:** Use the script `extract_features.py` from Google's BERT github. **Carefully read the README file for this github**. You only need the output from the last layer (use `--layers=-1`. This script should produce a JSON output that contains all the required information.
- **Compute word type representations:** Parse the output file from the previous step to retrieve individual tokens paired with their embeddings. You can then compute the average embedding for a given word type to retrieve a word type representation.

Tip: *do not retrieve all embeddings before computing the average for a given word type: instead, compute the sum of token embeddings as you parse them, and divide by the number of tokens for that word type.*

3. Compare embeddings from the two corpora

- **Using the MEN benchmark:**
  - Make sure you computed word-token representations: the script `extract_features.py` produces outputs associated to *word-pieces*.
  - For each triple from the MEN dataset, retrieve the word-type representations of the two paired words (as derived from the first of the two corpora)
  - Compute their cosine. This should result in a series of similarity measurements.
  - Compute the Spearman correlation of cosine measurements and human similarity ratings from MEN (i.e., compare the cosine with the third item from the triple).
  - Repeat the process, this time using embeddings from the second of the two copora.
- **Directly compare the embeddings of the two corpora:**
  - This time, you can directly compare embeddings of word-pieces.
  - Compute the average type representations for each word-piece embedding in the first corpus; repeat on the second corpus.
  - Compare directly the vectors component-wise using a related sample t-test over the two sets of vectors (use the scipy function scipy.stats.ttest_rel).
  - Probe the structure of the two vector spaces for the two corpora: a) select a random sub-sample of word-pieces, b) compute their pairwise distances (Euclidean or cosine) in both corpora, c) compute a t-test using these two related series of measurements.
- **What do you conclude from these experiments?**
  - How do you interpret them?
  - Are there unclear/uncertain points remaining?
  - How would you try to clarify them?

References I

Devlin, Jacob et al. (2018). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *CoRR* abs/1810.04805. arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805.

Firth, J. R. (1957). "A synopsis of linguistic theory 1930-55.". In: *Studies in Linguistic Analysis (special volume of the Philological Society)* 1952-59, pp. 1–32.

Harris, Zellig (1954). "Distributional structure". In: *Word* 10.23, pp. 146–162.

McCann, Bryan et al. (2017). "Learned in Translation: Contextualized Word Vectors". In: *CoRR* abs/1708.00107. arXiv: 1708.00107. URL: http://arxiv.org/abs/1708.00107.

Peters, Matthew E. et al. (2018). "Deep contextualized word representations". In: *CoRR* abs/1802.05365. arXiv: 1802.05365. URL: http://arxiv.org/abs/1802.05365.

Radford, Alec (2018). "Improving Language Understanding by Generative Pre-Training". In:

Taylor, Wilson (1953). "Cloze Procedure: A New Tool for Measuring Readability". In: *Journalism Quarterly* 30, pp. 415–433.

Vaswani, Ashish et al. (2017). "Attention Is All You Need". In: *CoRR* abs/1706.03762. arXiv: 1706.03762. URL: http://arxiv.org/abs/1706.03762.