Attention in Neural Networks

Nina Poerner

December 11, 2019

Why attention?

- Limitation of CNN
  - Can only capture local dependencies
- Limitations of LSTM, GRU, etc.:
  - Can capture long-range dependencies, but may find them difficult to learn
  - Sequential processing not parallelizable
  - In Sequence2Sequence (Encoder-Decoder) architectures: Must fit all source sentence info into a single fixed-size vector
    - Fails when source sentence is very long
    - Major issue in early NMT architectures
    - Attention was first proposed for Sequence2Sequence / NMT (Bahdanau et al. 2015)

Attention: The basic formula

- **Ingredients:**
- One query vector: $\mathbf{q} \in \mathbb{R}^{d_q}$
- T key vectors: $\mathbf{K} \in \mathbb{R}^{T \times d_k}$
- T value vectors: $\mathbf{V} \in \mathbb{R}^{T \times d_v}$
- Scoring function $f : \mathbb{R}^{d_q} \times \mathbb{R}^{d_k} \rightarrow \mathbb{R}$
  - Maps a query-key pair to a scalar ("score")
  - f may be parametrized by parameters θ_f

Attention: The basic formula

- **Step 1**: Apply f to $\mathbf{q}$ and all keys $\mathbf{k}_t$ to get T scores (one per key):

$$\mathbf{e} = \begin{bmatrix} e_1 \\ \vdots \\ e_T \end{bmatrix} = \begin{bmatrix} f(\mathbf{q}, \mathbf{k}_1) \\ \vdots \\ f(\mathbf{q}, \mathbf{k_T}) \end{bmatrix}$$

  - **Question:** What is the range of e_t? $(-\infty, \infty)$
- **Step 2**: Turn $\mathbf{e}$ into probabilities... how? Softmax!

$$\alpha_t = \frac{\exp(e_t)}{\sum_{t'} \exp(e_{t'})}$$

- **Step 3**: α-weighted sum over $\mathbf{V}$

$$\mathbf{o} = \sum_{t=1}^{T} \alpha_t \mathbf{v}_t$$

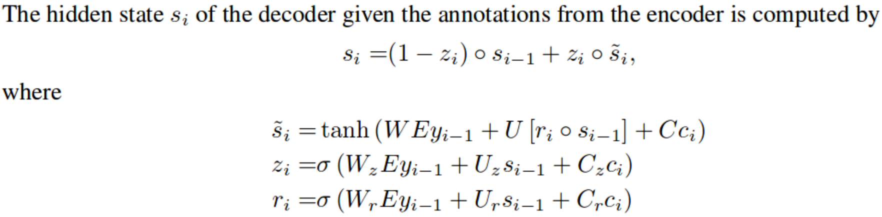
  - **Question:** What is the shape of $\mathbf{o}$? $\mathbb{R}^{d_v}$

Any questions?

Bahdanau et al. (2015)

- Machine Translation
- Source sentence: $[\mathbf{x}_1 \ldots \mathbf{x}_{T_x}]$
- Target sentence: $[\mathbf{y}_1 \ldots \mathbf{y}_{T_y}]$
- Encode $[\mathbf{x}_1 \ldots \mathbf{x}_{T_x}]$ with encoder RNN: $[\mathbf{h}_1 \ldots \mathbf{h}_{T_x}]$

Bahdanau et al. (2015)



The hidden state s_i of the decoder given the annotations from the encoder is computed by

$$s_i = (1 - z_i) \circ s_{i-1} + z_i \circ \tilde{s}_i,$$

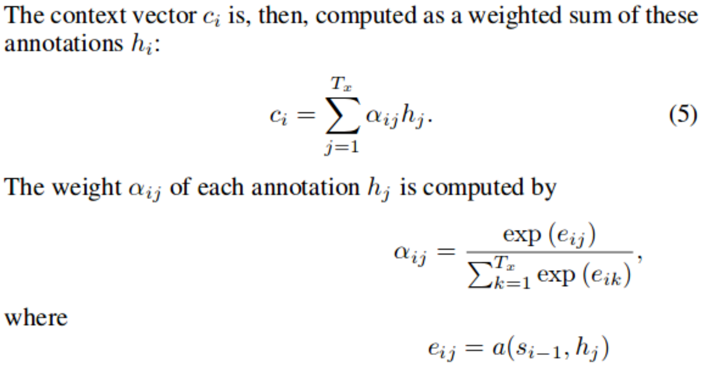
where

$$\begin{aligned}
\tilde{s}_i &= \tanh\left(W E y_{i-1} + U\left[r_i \circ s_{i-1}\right] + C c_i\right) \\
z_i &= \sigma\left(W_z E y_{i-1} + U_z s_{i-1} + C_z c_i\right) \\
r_i &= \sigma\left(W_r E y_{i-1} + U_r s_{i-1} + C_r c_i\right)
\end{aligned}$$

Bahdanau et al. (2015), ICLR

- What architecture is this? GRU
- Which variables are gates? $\mathbf{z}_i, \mathbf{r}_i$
- Which variable is the candidate vector? $\tilde{\mathbf{s}}_i$
- Which variables are the trainable parameters? $\mathbf{E}, \mathbf{W}, \mathbf{U}, \mathbf{C}, \mathbf{W}_z, \mathbf{U}_z, \mathbf{C}_z, \mathbf{W}_r, \mathbf{U}_r, \mathbf{C}_r$
- What is $\mathbf{s}_{i-1}$? Previous hidden state
- What is y_{i-1}? Previous correct word (teacher forcing)
- Extra term: $\mathbf{c}_i$

Bahdanau et al. (2015)



The context vector c_i is, then, computed as a weighted sum of these annotations h_i:

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j. \tag{5}$$

The weight α_{ij} of each annotation h_j is computed by

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})},$$

where

$$e_{ij} = a(s_{i-1}, h_j)$$

Bahdanau et al. (2015), ICLR

- Which equation corresponds to which step from the basic formula?
- Which variable corresponds to the query vector? $\mathbf{q} = \mathbf{s}_{i-1}$
- Which variables are the key vectors? $\mathbf{K} = [\mathbf{h}_1 \ldots \mathbf{h}_{T_x}]$
- Which variables are the value vectors? $\mathbf{V} = [\mathbf{h}_1 \ldots \mathbf{h}_{T_x}]$
- Which variable is the output? $\mathbf{o} = \mathbf{c}_i$

Bahdanau et al. (2015)

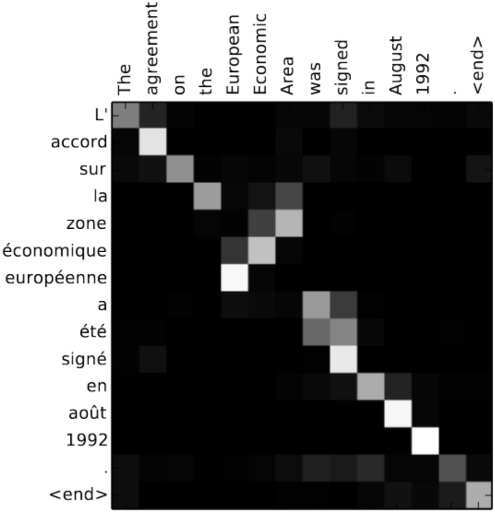
- Scoring function:

$$a(s_{i-1}, h_j) = v_a^\top \tanh\left(W_a s_{i-1} + U_a h_j\right),$$

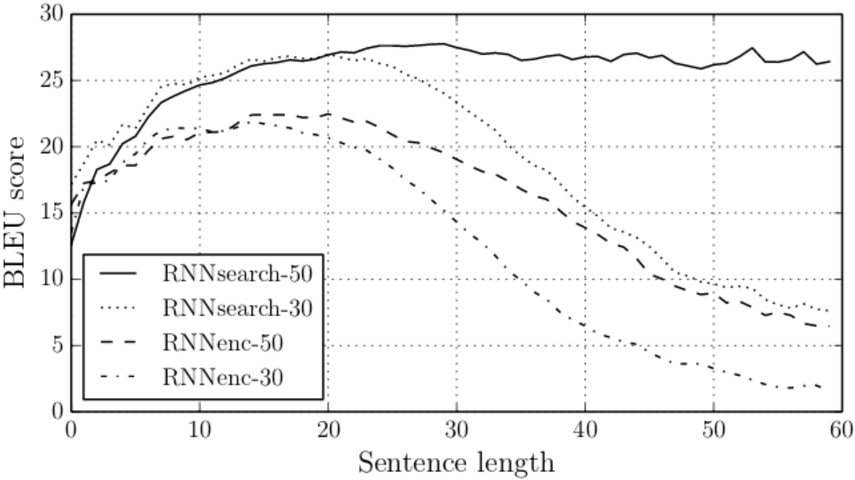
Bahdanau et al. (2015), ICLR (appendix)

- With additional trainable parameters $\mathbf{v}_a, \mathbf{U}_a, \mathbf{W}_a$

What does attention do?



Bahdanau et al. (2015), ICLR, Figure 3. Black is $\alpha_{i,j} = 0$, white is $\alpha_{i,j} = 1$



Bahdanau et al. (2015), ICLR, Figure 2

- Important to note: The Bahdanau model is still an RNN, just with attention on top.
- Do we actually need the RNN?

Any questions?

Self-Attention

- Input $\mathbf{X} \in \mathbb{R}^{T \times d_x}; \mathbf{X} = [\mathbf{x}_1 \ldots \mathbf{x}_T]$
- Trainable linear layers (parameters) $\theta = (\mathbf{W}^{(q)}, \mathbf{W}^{(k)}, \mathbf{W}^{(v)})$
- Transform $\mathbf{X}$ into
  - query matrix $\mathbf{Q} \in \mathbb{R}^{T \times d_q}; \mathbf{Q} = \mathbf{X}\mathbf{W}^{(q)}$
  - key matrix $\mathbf{K} \in \mathbb{R}^{T \times d_k}; \mathbf{K} = \mathbf{X}\mathbf{W}^{(k)}$
  - value matrix $\mathbf{V} \in \mathbb{R}^{T \times d_v}; \mathbf{V} = \mathbf{X}\mathbf{W}^{(v)}$

Self-Attention with a loop

- $\mathbf{Q} \in \mathbb{R}^{T \times d_q}, \mathbf{K} \in \mathbb{R}^{T \times d_k}, \mathbf{V} \in \mathbb{R}^{T \times d_v}$
- For every time step t:
  - Apply the basic attention formula to $(\mathbf{q}_t, \mathbf{K}, \mathbf{V})$
  - Let's call the output $\mathbf{o}_t$
- Stack all $\mathbf{o}_t$ into output matrix $\mathbf{O}$
- **Question:** What is the shape of $\mathbf{O}$? $\mathbb{R}^{T \times d_v}$

Any questions?

Self-Attention parallelized

- $\mathbf{o}_t$ does not depend on $\mathbf{o}_{t-1}$ (or any other $\mathbf{o}_{t' \neq t}$)
- We can parallelize the loop (unlike an RNN!)
- Scaled dot product scoring function (instead of Bahdanau's complicated function):

$$f(\mathbf{q}, \mathbf{k}) = \frac{\mathbf{q}^T \mathbf{k}}{\sqrt{d_k}}$$

- d_q must be equal to d_k... why?
- Note: We could also use more complicated scoring functions in parallel, it would just be more difficult to write down.

Self-Attention parallelized

- **Step 1**: $\mathbf{E} = \frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}$

$$\begin{array}{cc} & \rightarrow keys \rightarrow \\ \begin{array}{c}\downarrow\\ queries\\ \downarrow\end{array} & \begin{bmatrix} e_{1,1} & \dots & e_{1,T}\\ \vdots & \ddots & \vdots \\ e_{T,1} & \dots & e_{T,T}\end{bmatrix}\end{array} = \frac{1}{\sqrt{d_k}} \begin{bmatrix} - & \mathbf{q}_1 & - \\ & \vdots & \\ - & \mathbf{q}_T & -\end{bmatrix} \begin{bmatrix} | & & | \\ \mathbf{k}_1 & \dots & \mathbf{k}_T \\ | & & | \end{bmatrix}$$

- What is the dimensionality of **E**? $\mathbb{R}^{T \times T}$
- **Step 2**: Softmax
- Which axis of **E** should we normalize over? The rows (i.e., the keys)

$$\alpha_{t,t'} = \frac{\exp(e_{t,t'})}{\sum_{t''=1}^{T} \exp(e_{t,t''})}$$

- Let's call this new normalized matrix $\mathbf{A} = \mathrm{softmax}(\mathbf{E})$
- The rows $\boldsymbol{\alpha}_t$ of **A** are probability distributions

- **Step 3**: Weighted sum

$$\mathbf{O} = \mathbf{A}\mathbf{V}$$

$$\begin{array}{cc} & \rightarrow d_v\text{(value dims)}\rightarrow \\ \begin{matrix}\downarrow\\ \textit{queries}\\ \downarrow\end{matrix} & \begin{bmatrix} o_{1,1} & \dots & o_{1,d_v} \\ \vdots & \ddots & \vdots \\ o_{T,1} & \dots & o_{T,d_v}\end{bmatrix}\end{array} = \begin{bmatrix} - & \boldsymbol{\alpha}_1 & - \\ & \vdots & \\ - & \boldsymbol{\alpha}_T & - \end{bmatrix} \begin{bmatrix} | & & | \\ \mathbf{v}_{:,1} & \dots & \mathbf{v}_{:,d_v} \\ | & & | \end{bmatrix}$$

- Scaled dot-product Self-Attention as a one-liner:

$$\mathbf{O} = \mathrm{softmax}\Big(\frac{(\mathbf{X}\mathbf{W}^{(q)})(\mathbf{X}\mathbf{W}^{(k)})^T}{\sqrt{d_k}}\Big)(\mathbf{X}\mathbf{W}^{(v)})$$

- (where the softmax is over the second axis)

Any questions?

Is (Self-)Attention all you need?

- A Neural Network takes as input a sequence of word2vec vectors (as matrix $\mathbf{X}$) and transforms them with self-attention into a matrix $\mathbf{O}$
- We feed the NN with $\mathbf{X}_1 = [\mathbf{w}^{(\text{space})}, \mathbf{w}^{(\text{ship})}]$ and get $\mathbf{O}_1$
- We feed the NN with $\mathbf{X}_2 = [\mathbf{w}^{(\text{ship})}, \mathbf{w}^{(\text{space})}]$ and get $\mathbf{O}_2$
    - Is there a difference between $\mathbf{x}_{1,1}$ and $\mathbf{x}_{2,2}$?
    - No, because $\mathbf{x}_{1,1} = \mathbf{w}^{(\text{space})} = \mathbf{x}_{2,2}$

    - **Question:** Is there a difference between $\mathbf{o}_{1,1}$ and $\mathbf{o}_{2,2}$?

    - **Question:** Would it help to apply another layer of self attention?

Is (Self-)Attention all you need?

Position embeddings

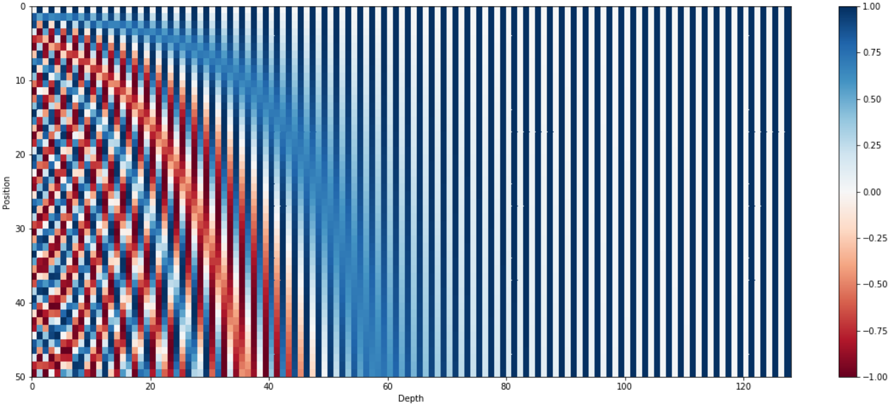
- Add to every input word embedding a position embedding $\mathbf{p}$:
- Representation of word "space" in position t: $\mathbf{x}_t = \mathbf{w}^{(\text{space})} + \mathbf{p}_t$

$$\mathbf{w}^{(\text{space})} + \mathbf{p}_1 \neq \mathbf{w}^{(\text{space})} + \mathbf{p}_2$$

- Option 1: Trainable position embeddings: $\mathbf{P} \in \mathbb{R}^{T^{\max} \times d}$
  - Disadvantage: Cannot deal with inputs longer than $T^{\max}$
- Option 2: Sinusoidal position embeddings (deterministic):

$$p_{t,i} = \begin{cases} \sin(w_k \cdot t) & \text{if } i = 2k \text{ (even)} \\ \cos(w_k \cdot t) & \text{if } i = 2k+1 \text{ (odd)} \end{cases} ; w_k = \frac{1}{10000^{\frac{2k}{d}}}$$

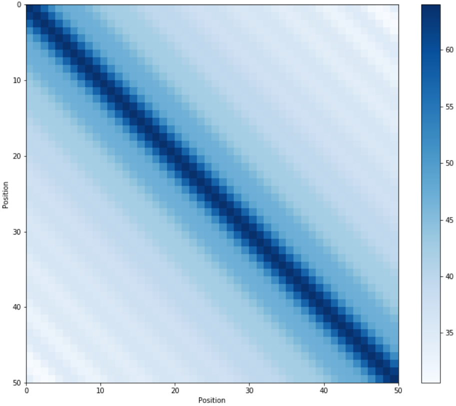
Sinusoidal position embeddings



https://kazemnejad.com/blog/transformer_architecture_positional_encoding

Sinusoidal position embeddings

Pairwise dot products of sinusoidal position embeddings



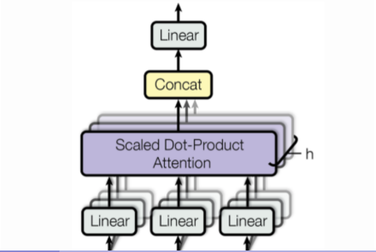
https://kazemnejad.com/blog/transformer_architecture_positional_encoding

Any questions?

Multi-head self-attention

- Before:
  - We have parameters $\theta = (\mathbf{W}^{(Q)}, \mathbf{W}^{(K)}, \mathbf{W}^{(V)})$
  - We use θ to transform $\mathbf{X}$ into $(\mathbf{Q}, \mathbf{K}, \mathbf{V})$
  - $\mathbf{O} = \mathrm{selfattention}(\mathbf{Q}, \mathbf{K}, \mathbf{V})$
- Now:
  - N sets of parameters $\{\theta_1, \ldots, \theta_N\}$, with $\theta_n = (\mathbf{W}_n^{(Q)}, \mathbf{W}_n^{(K)}, \mathbf{W}_n^{(V)})$
  - For every $1 \leq n \leq N$ (every "head"):
    - ★ Use θ_n to transform $\mathbf{X}$ into $(\mathbf{Q}_n, \mathbf{K}_n, \mathbf{V}_n)$
    - ★ $\mathbf{O}_n = \mathrm{selfattention}(\mathbf{Q}_n, \mathbf{K}_n, \mathbf{V}_n)$
  - Concatenate all $\mathbf{O}_n$ along last axis into output matrix $\mathbf{O}$
  - Final linear layer $\mathbf{W}^{(o)} \in \mathbb{R}^{Nd_v \times d}$

- (In reality, all heads are calculated in parallel)
- Conceptually like single filter vs. multiple filters in CNN



Masked Self-Attention

- RNNs are “causal” models: they cannot look at future inputs
- Self-Attention (in its basic form) is not causal
- Without causal modeling, our models will cheat when doing Language Modeling or MT Decoding

y	y_1 =the	y_2=cat	y_3=sits	y_4=on
	$\mathbf{o}_1$	$\mathbf{o}_2$	$\mathbf{o}_3$	$\mathbf{o}_4$
	self-attention			
x	x_1 =<s>	x_2 =the	x_3 =cat	x_4 =sits

- For instance, we don’t want $\mathbf{o}_3$ to get information about x_4
- “Getting information” means an attention weight > 0
- **Question:** How can we set $\alpha_{3,4} = 0$?
- By setting $e_{3,4} = -\infty$ (in practice: $e_{3,4} = -10000$)

Masked Self-Attention

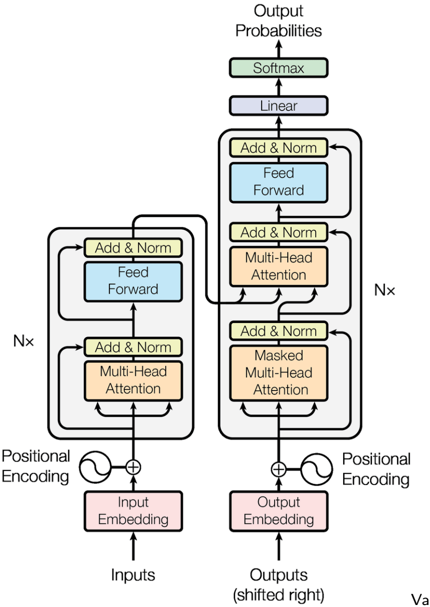
- Calculate **E** like you usually would
- Set $e_{i,j} = -10000$ for all illegal connections

$$\begin{bmatrix} e_{1,1} & e_{1,2} & e_{1,3} \\ e_{2,1} & e_{2,2} & e_{2,3} \\ e_{3,1} & e_{3,2} & e_{3,3} \end{bmatrix} \rightarrow \begin{bmatrix} e_{1,1} & -10000 & -10000 \\ e_{2,1} & e_{2,2} & -10000 \\ e_{3,1} & e_{3,2} & e_{3,3} \end{bmatrix}$$

- Not just useful for decoding, but also for ignoring padded inputs

Any questions?

Transformer Architecture



Vaswani et al (2017), NeurIPS

Any questions?

NLP since 2018

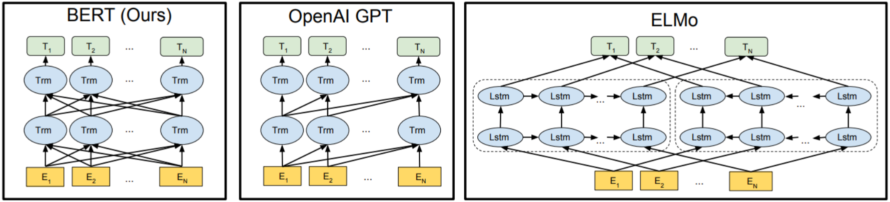
- Pre-train (some sort of) Language Model on a big unlabeled corpus
- Give the model the name of a Sesame Street character
    - So far: ELMo, BERT, ERNIE, ERNIE 2.0, Kermit, Grover
- Use it as initialization or feature extractor for other models

BERT

- Devlin et al. (2019): BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding
- Almost 3000 citations in one year (that's 8 a day)
- Best long paper at NAACL 2019
- Has been integrated into Google Search `https://www.blog.google/products/search/search-language-understanding-bert/`
- Transformer Encoder Masked Language Model (with Next Sentence Prediction) pre-trained on Wikipedia and some books
- Masked Language Model $\neq$ Masked Self-Attention

The problem with bidirectional LMs

- We said before that Language Models must be "causal" (i.e., unidirectional) so that they do not cheat
- But we want a bidirectional model...
- Option 1:
  - Use two unidirectional models (left-to-right, right-to-left) and combine $\overleftarrow{\mathbf{o}}_{t-1}, \overrightarrow{\mathbf{o}}_{t+1}$ to predict y_t (c.f., ELMo)
  - Problem: The unidirectional models cannot communicate with each other at their lower layers (without violating the causality), so this is "shallow" bidirectionality



Devlin et al. (2019), NAACL

- Option 2:
  - Use a fully bidirectional model and only predict one word t per sentence, while setting $e_{:,t} = -10000$
  - Problem: This is inefficient. If our sentence has length 512, we must see it 512 times to predict all words

Masked Language Modeling (MLM)

- Use a fully bidirectional model
- In the input, replace some randomly chosen words (15%) with a special [MASK] token.
- Predict the identity of the [MASK] tokens
- $x =$ The cat [MASK] on the [MASK] .
- $y =$ sat, mat
- Pro: No need to set any attention weights to zero.
- Contra: Cannot learn conditional probabilities $p(x_1|x_2), p(x_2|x_1)$ when both x_1 and x_2 are masked
- ... but this does not seem to be an issue in practice, as the masking patterns differ between epochs

Next Sentence Prediction (NSP)

- Second Loss function of BERT:
- Given sentence s_1 and s_2, predict whether s_2 follows s_1
- A bit like word2vec with negative sampling, just for sentences!

  [CLS] The cat sat on the [MASK] . [SEP] Then it got up and [MASK] a mouse. [SEP]

  → positive sample

  [CLS] The cat [MASK] on the mat. [SEP] The police [MASK] . [SEP]

  → random (negative) sample
- $L_{\mathrm{bert}} = L_{\mathrm{mlm}} + L_{\mathrm{nsp}}$

Using BERT

- Pre-trained BERT is available through different libraries (huggingface, tensorflow-hub)
- BERT-base: 12 layers, 12 heads, hidden size 768
- BERT-large: 24 layers, 16 heads, hidden size 1024
- Usual workflow:
  - Extract the embedding layer and 12 (or 24) Transformer layers
  - Put a smaller model (e.g., a feed-forward layer) on top of layer 12 (or 24) to do some specific task (e.g., sentiment analysis, POS tagging...)
  - Either: freeze BERT and train only your own model
  - Or: finetune BERT and your model together
- Assumption: Some of the features that were useful for language modeling are also useful for your target task. BERT already knows how to extract these features, so you don't have to learn them from scratch

```
from transformers import BertForSequenceClassification, BertTokenizer
sentences = ["[CLS] Aweful movie! [SEP]"]
label_tensor = torch.tensor([0])

model = BertForSequenceClassification.from_pretrained("bert-base-cased",
    num_labels = 5)
params = list(model.parameters())
# len(params) 201

tokenizer = BertTokenizer.from_pretrained("bert-base-cased")
tokenized = [tokenizer.tokenize(sentence) for sentence in sentences]
# tokenized [[’[CLS]’, ’A’, ’##we’, ’##ful’, ’movie’, ’!’, ’[SEP]’]]

input_ids = [tokenizer.convert_tokens_to_ids(tokens) for tokens in tokenized]
input_ids_tensor = torch.tensor(input_ids)
#input_ids_tensor tensor([[ 101,  138, 7921, 2365, 2523,  106,  102]])

logits = model(input_ids_tensor)[0]
# logits tensor([[-0.4342,  0.7886, -0.6013,  1.0922, -0.1007]], grad_fn=<AddmmBackw

loss = torch.nn.CrossEntropyLoss()(logits, label_tensor)
# loss tensor(2.2308, grad_fn=<NllLossBackward>)
loss.backward()

#params[6].grad.max() tensor(0.0292)
```

Since BERT

Rank	Name	Model	Score
1	ERNIE Team - Baidu	ERNIE	90.1
2	Microsoft D365 AI & MSR AI & GATECH	MT-DNN-SMART	89.9
3	T5 Team - Google	T5	89.7
+ 4	王玮	ALICE v2 large ensemble (Alibaba DAMO NLP)	89.5
5	XLNet Team	XLNet (ensemble)	89.5
6	ALBERT-Team Google Language	ALBERT (Ensemble)	89.4
7	Microsoft D365 AI & UMD	FreeLB-RoBERTa (ensemble)	88.8
8	Facebook AI	RoBERTa	88.5
+ 9	Microsoft D365 AI & MSR AI	MT-DNN-ensemble	87.6
10	GLUE Human Baselines	GLUE Human Baselines	87.1

Relative position embeddings

- $\mathbf{A}^{(k)} \in \mathbb{R}^{(2T+1)\times d_k}$ (for keys)
- $\mathbf{A}^{(v)} \in \mathbb{R}^{(2T+1)\times d_v}$ (for values)
    - Trainable lookup tables

$\mathbf{A}^{(*)} \in \mathbb{R}^{(2T+1)\times d_*}$

$\mathbf{a}_1 \dots \mathbf{a}_T$	$\mathbf{a}_{T+1}$	$\mathbf{a}_{T+2} \dots \mathbf{a}_{2T+1}$
$t - t' < 0$	$t - t' = 0$	$t - t' > 0$
“query before key”	“query is key”	“key before query”

- $e_{t,t'} = f(\mathbf{q}_t, \mathbf{k}_{t'}, \mathbf{a}^{(k)}_{(T+1+t-t')})$
- With scaled dot product:

$$e_{t,t'} = \frac{\mathbf{q}_t^T(\mathbf{k}_{t'} + \mathbf{a}^{(k)}_{(T+1+t-t')})}{\sqrt{d_k}}$$

- $\mathbf{o}_t = \sum_{t'=1}^{T} \alpha_{t,t'}(\mathbf{v}_{t'} + \mathbf{a}^{(v)}_{(T+1+t-t')})$

- In practice: Limit T to some “clipping distance”.
- If $t - t' < -T$, use $\mathbf{a}_1$.
- If $t - t' > T$, use $\mathbf{a}_{(2T+1)}$.