Transformer: Neural machine translation without recurrence

- It is possible to replace recurrence with **self-attention** within the encoder and decoder, as in the **transformer architecture**

$$z_m^{(i)} \in \sum_{n=1}^{M^{(s)}} \alpha_{m\to n}^{(i)} (\Theta_v \boldsymbol{h}_n^{(i-1)})$$

For each token m at level i, we co ion over the entire source sentence. The keys, valu all projections of the vector $\boldsymbol{h}^{(i-1)}$.

- The attention scores $\alpha_{m\to n}^{(i)}$ are computed using a scaled form of softmax attention,

$$\alpha_{m\to n} \propto \exp(\psi_\alpha(m,n)/M)$$

"Self-attention"

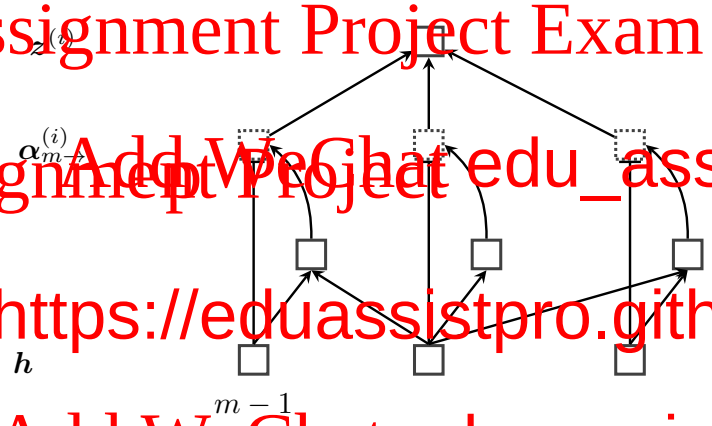

Figure 18.7: The transformer encoder's computation of $_{m}$ $^{-1)}$. The **k**ey, **v**alue, and **q**uery are shown for token $m-1$. For example, $\psi_{\alpha}^{(i)}(m, m-1)$ is computed from the key $\Theta_k \boldsymbol{h}_{m-1}^{(i-1)}$ and the query $\Theta_q \boldsymbol{h}_{m}^{(i-1)}$, and the gate $\alpha_{m\rightarrow m-1}^{(i)}$ operates on the value $\Theta_v \boldsymbol{h}_{m-1}^{(i-1)}$. The figure shows a minimal version of the architecture, with a single attention head. With multiple heads, it is possible to attend to different properties of multiple words.

"Self-attention"

"Self-attention"

"Self-attention"

"Self-attention"

"Self-attention"

"Self-attention"

- Attention is an effective mechanism to model dependencies.
- By varying the keys, values, and queries, we get variants of attention. We get self-attention value and quer sentence.

- Intuitively self-attention contextual sentence (or any sequence of word token interpreting the word token, which will in turns helps other tasks that build on the understanding of words.

Multiple Attention "heads"

Multiple Attention "heads"

Multiple Attention "heads"

Multiple Attention "heads"

What does self-attention do?

- Contextualization
- why is contextualization a good thing?
  - "Work p
  - "Heat th
- Having the same embedding for both inst doesn't make sense

The Transformer architecture in (almost) its entirety

Figure 1: The Transformer - model architecture.

Position Embedding

The original Transfor position embedding.

- For word w at position $pos \in [0, L-1]$ in the sequence $\boldsymbol{w} = \{w_0, \cdots, w_{L-1}\}$, with a 4-dimensi e_w, and $d_{model} = 4$, the operation would be:

$$e_{w'} = e_w + \left[\sin\left(\frac{pos}{10000^{2/4}}\right), \cos\left(\frac{pos}{10000^{2/4}}\right)\right]$$
$$= e_w + \left[\sin\left(\frac{pos}{10000^{2/4}}\right), \cos\left(\frac{pos}{10000^{2/4}}\right)\right]$$

- The formula to calculate the position em

$$PE(pos, 2i) = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$
$$PE(pos, 2i+1) = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

for $i \in [0, d_{model}/2 - 1]$

Position-wise feedforward neural network

- A feed-forward network is applied to each position separately and identically, i.e., with shared weights acr the sequence

$$) + b_2$$

- This is equivalent to applying two one-di convolutional network with one kernel t
- Question to think about: why not just use a one-dimensional convolution?

Layer normalization

- Given a minibatch $B = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m\}$, after applying lay h, we get a transformed minibatch $B' = \{\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_m\}$, where $\boldsymbol{y}_i = LN_{\gamma,\beta}(\boldsymbol{x}_i)$
- Specifically, layer normalization needs th
  - Compute the mean and variance of each sa

$$\mu_i = \frac{1}{m}\sum_{k=1}^{m} x_{i,k}, \quad \sigma_i^2 = \frac{1}{m}\sum_{k=1}^{m}(x_{i,k} - \mu_i)^2.$$

  - Normalize each sample such that the e have zero mean and unit variance:

$$\hat{x}_{i,k} = \frac{x_{i,k} - \mu_i}{\sqrt{\sigma_i^2 + \varepsilon}}$$

  - Finally scaling and shifting with γ and β:

$$\boldsymbol{y}_i = \boldsymbol{\gamma} \odot \hat{\boldsymbol{x}}_i + \boldsymbol{\beta} \triangleq LN_{\gamma,\beta}(\boldsymbol{x}_i)$$

Reducing the output space with BPE

BPE is a simple data compression technique

- Initialize the vocabulary with the character vocabulary, and segment each words into a sequence of chara end-of-word symbol.

- Iteratively co frequent pa produces a new symbol which represent . Frequent character ngrams are eventu gle symbol.

- The final vocabulary size equals to the initial vocabulary size plus the number of merge operations
- The number of merge operations is a hyperparameter

Python implementation

Python implementation

Tutorials

- NMT with Tensorflow: https://github.c nmt
- Attention visualization: https://jal l-machine-translation- h-attention/
- Illustrated transformer: http://jalammar.github.io/illust er/

BERT: Bidirectional Encoder Representations from Transformers

Input Representations:

- The input can be both a single sentence or a pair of s
- "Sentence" here just means a sequence of wo
- Sentences ar cab size= 30K).
- The first token of the input sequence is a spe
- Sentence pars are packed together into o separated with a special token [SEP]
- Separate token embeddings for the first sentence and the second sentence in a sentence pair.

The pretrain-fine tune paradigm

Pre-Training BERT

- The Transformer is trained with a Masked La (MLM) task, and a next sentence prediction (sentence pair
- The MLM tas 15% of the tokens that are randomly mask
- The final hidden vectors correspondin s are fed into a softmax over the token vocabulary.

BERT Pretraining with MLM

BERT next sentence prediction

BERT: fine tune to specific tasks