Seq2Seq. Transformers

ML4SE

Denis Litvinov

January 12, 2022

Table of Contents

Problem Statement

Let $x_1..x_M$ - source sequence of tokens from vocabulary Y
Let $y_1..y_T$ - target sequence of tokens from vocabulary X
We want to maximize probability of the target sequence given the source sequence:

$$P(y_1..y_T|x_1..x_M) = \prod_{t=1}^{T} P(y_t|y_{<t}, x_1..x_M)$$

Loss function is

$$Loss = -\frac{1}{|T|}\sum_{i=1}^{|T|}\sum_{j=1}^{|Y|} I[y_i = j]\log \hat{y}_{ij}$$

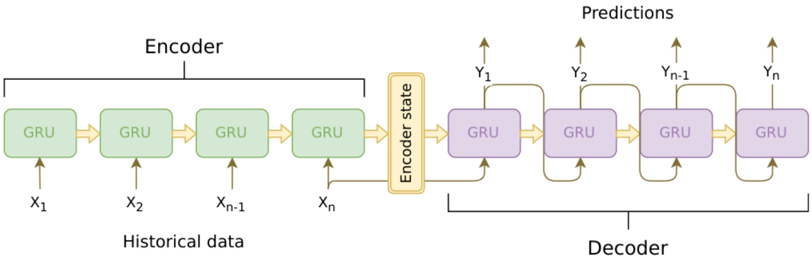
where $\hat{y}_{ij}$ - probability of j-th token at i-th place.

Problem Statement

There are some nuances:

- If the predicted sequence is shorter than the target sequence, it should be padded to the target sequence length.
- If the predicted sequence is longer than the target sequence, it should be cutted to the target sequence length.
- Because you train your model with mini-batches, you have to pad your target sequences to have common length.
- Padding value should not be counted as an error.

RNN Seq2Seq Example



RNN Seq2Seq Example

Encoder:
$h_t = LSTM(h_{t-1}, x_t)$ - encoder hidden state
$e_t = out_e(h_t)$ - output of encoder at time t
Decoder:
$s_t = LSTM(s_{t-1}, y_{t-1})$ - decoder hidden state
$g_t = out_g(s_t)$ - output of decoder at time t
$p_t = softmax(g_t)$ - probabilities of tokens at time t
$y_t = argmax(p_t)$ - predicted token at time t
$s_o = h_T$ - initial decoder hidden state is the last encoder hidden state.

Applications

- Neural Machine Translation
- Open Domain Question Answering
- Code Generation
- Docstring Generation
- Function Name Generation
- ..

Similarity with LM

Remember LM Task:

$$P(y_1..y_T) = \prod_{t=1}^{T} P(y_t|y_{<t})$$

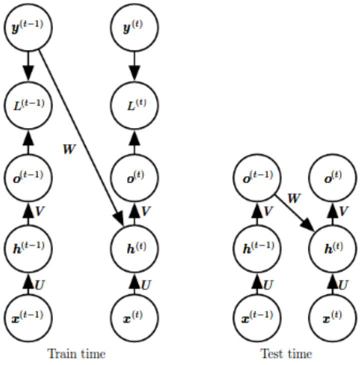
It means you can reduce Seq2seq task to LM task!
What are pros and cons of that?

Similarity with LM

- We need a joint vocabulary, which significantly increase model size
- We do unnecessary work of modeling source sequence
- We cannot exploit full self-attention for source sequences
- But we can exploit pretrained LM model for seq2seq purposes
- Allows more efficient batch utilization

Teacher Forcing

There is a problem with training the decoder.
Because y_t depends on $y_{t-1}, ..., y_0$, if at some timestamp a wrong token is predicted, the rest of the sequence will be wrong too.



Training/Inference Discrepancy

At the training phase, at every timestep you give the decoder **true previous token** y_{t-1} to predict the current one y_t.
At the inference phase, at every timestep you give the decoder **predicted previous token** $\hat{y}_{t-1}$ to predict the current one y_t.

BLEU

aka Bilingual Evaluation Understudy
Suppose you have a several translation hypothesis and reference sentences.
Input: "Un gato se sienta en una silla"
Reference 1: "A cat sits on a chair"
Reference 2: "Cat is on an armchair"
Hypothesis 1: "Dog sits on a chair"
Hypothesis 2: "A cat sat on a mat"
Let g_n - n-gram
Let $count_{clip}(g_n)$ be bounded above by highest count of the n-gram in any reference sentence
Then a modified precision is

$$p_n = \frac{\sum_{C \in hyp} \sum_{g_n \in C} count_{clip}(g_n)}{\sum_{C \in hyp} \sum_{g_n \in C} count(g_n)}$$

BLEU

Compute brevity penalty for short hypothesis.

$$BP = \begin{cases} e^{1-\frac{|ref|}{|hyp|}} & |ref| > |hyp| \\ 1 & else \end{cases}$$

Final score

$$BLEU = BP * \exp(\sum_{n=1}^{N} w_n p_n)$$

where w_p -weights assosiated with p_n
Usually,

$$BLEU = BP * \exp(\frac{1}{N} \sum_{n=1}^{N} p_n)$$

BLEU

- BLEU is a Corpus-based Metric. Because n-gram statistics for individual sentences are less meaningful, they are accumulated over an entire corpus when computing the score.
- No distinction between content and function words
- Not good at capturing meaning and grammaticality of a sentence. (Negation, longer dependencies)
- Prior to computing the BLEU score, both the reference and candidate translations are normalized and tokenized.

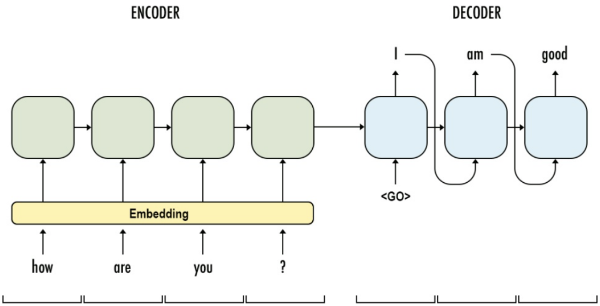
ROUGE Score

Overlap between n-grams of hypothesis and reference sentences

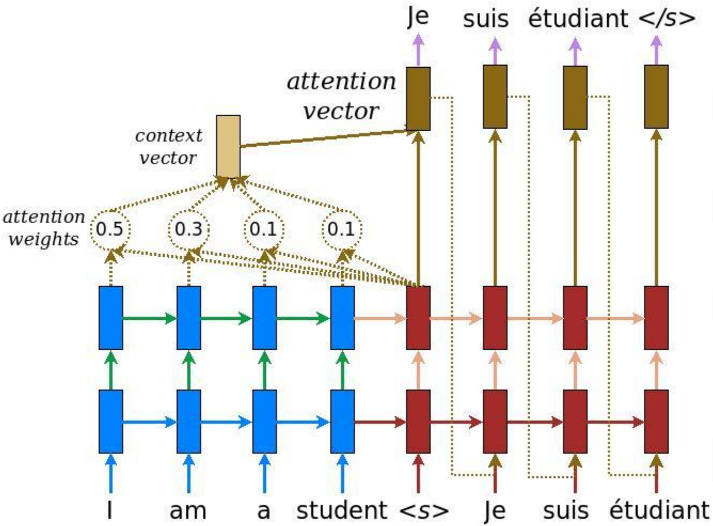
$$ROUGE - N = \frac{\sum_{C \in hyp} \sum_{g_n \in C} count_{match}(g_n)}{\sum_{C \in hyp} \sum_{g_n \in C} count(g_n)}$$

Problems with Vanilla Encoder-Decoder Architecture

1. Poor performance on long sentences
2. Bias towards shorter candidates
3. Fluent but inadequate output
4. Bottleneck in the context embedding



Attention


Je
suis
étudiant
</s>
attention vector
context vector
attention weights
0.5
0.3
0.1
0.1
I
am
a
student
<s>
Je
suis
étudiant

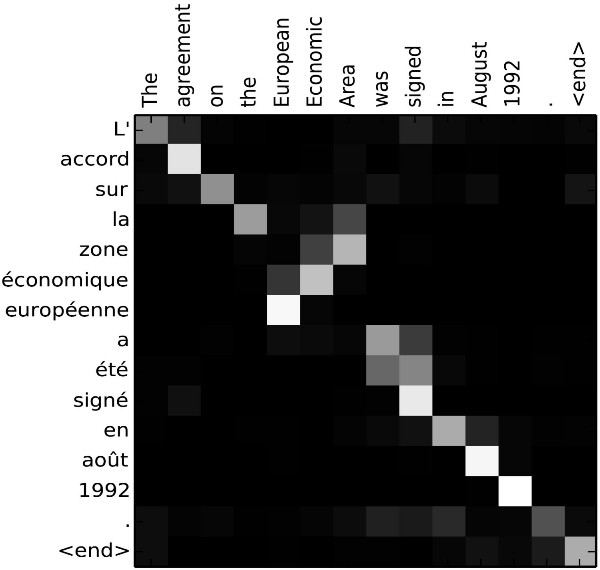
Attention

Attention is a mechanism of conditioning of every output on a weighted sum of source inputs.
Introduce attention through new function f:
$\alpha_{t'} = f(g_{t-1}, e_{t'})$ - weights of source tokens.
$\bar{\alpha} = softmax(\alpha)$ - normalize weights.
$c_t = \sum_{t'=0}^{T} \bar{\alpha}_{t'} e_{t'}$ - context vector as a weighted sum
Encoder:
$h_t = LSTM(h_{t-1}, x_t)$ - encoder hidden state
$e_t = out_e(h_t)$ - output of encoder at time t
Decoder:
$s_t = LSTM(s_{t-1}, [y_{t-1}, c_t])$ - decoder hidden state
$g_t = out_g(s_t)$ - output of decoder at time t
$p_t = softmax(g_t)$ - probabilities of tokens at time t
$y_t = argmax(p_t)$ - predicted token at time t
$s_o = h_T$ - initial decoder hidden state is the last encoder hidden state.

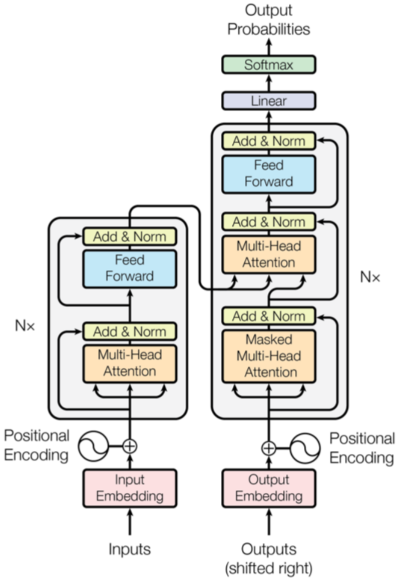
Attention

Usual choices for attention functions:
$f(h, e) = h^T e$ - dot
$f(h, e) = h^T We$ - general, W - trainable
$f(h, e) = v^T tanh(W[h, e])$ concat, , W - trainable

Token alignment in NMT



General Architecture



General Architecture

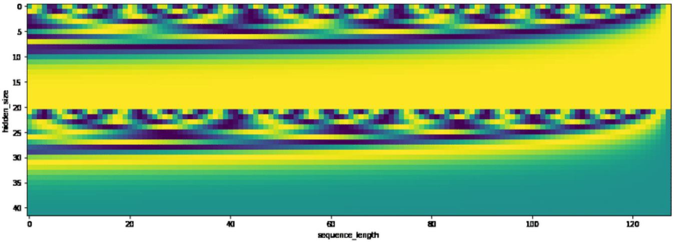
Described by

- *d* model "hidden size"
- numbed of encoder and decoder layers
- *h* number of heads
- *n* - max sequence length - used for precomputed positional encoding
- dropout rate
- source and target vocab size
- etc...

Positional Encoding

$$PE(pos, 2i) = \sin(pos/10000^{\frac{2i}{d}})$$

$$PE(pos, 2i+1) = \cos(pos/10000^{\frac{2i}{d}})$$



Review of BatchNorm

$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{ij}$$

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{ij} - \mu_j)^2$$

$$\hat{x}_{ij} = \frac{x_{ij} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}$$

$$\hat{x}_{ij} = \alpha \hat{x}_{ij} + \beta$$

Problems with BatchNorm

- It puts a lower limit on the batch size
- It makes batch normalization difficult to apply to recurrent connections in recurrent neural network

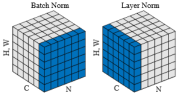
LayerNorm

$$\mu_i = \frac{1}{C}\sum_{j=1}^{C} x_{ij}$$

$$\sigma_i^2 = \frac{1}{C}\sum_{i=1}^{C}(x_{ij} - \mu_i)^2$$

$$\hat{x}_{ij} = \frac{x_{ij} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}}$$

$$\hat{x}_{ij} = \alpha\hat{x}_{ij} + \beta$$



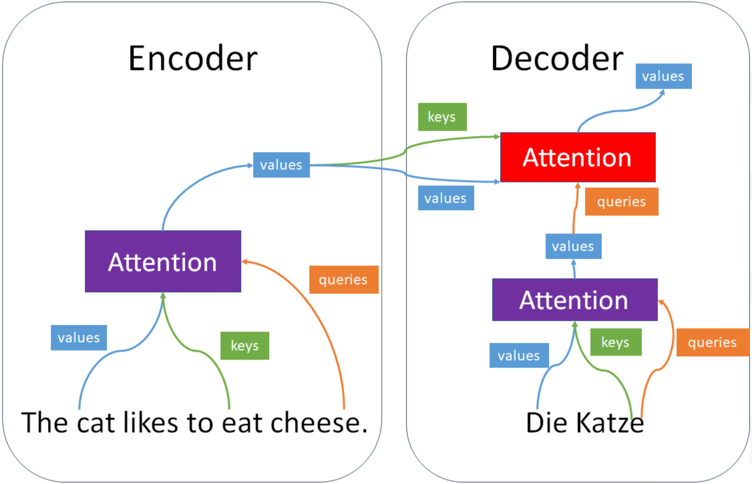
Self-Attention

Let $X \in R^{nxd}$

$$attention = softmax(\frac{QK^T}{\sqrt{d}})V \in R^{nx\frac{d}{h}}$$

$Q = XW^Q$
$K = XW^K$
$V = XW^V$
where $W^Q, W^K, W^V \in R^{dx\frac{d}{h}}$
Complexity $O(n^2 d)$

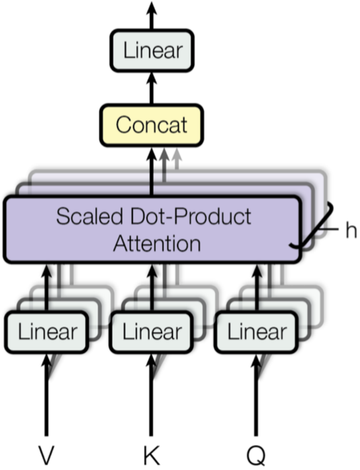
Cross-Attention



Multi-Head Attention

$$MultiHead(Q, K, V) = Concat(head_1, ..., head_h)W^O$$

where $W^O \in R^{dxd}$



Masked Attention in Decoder