Transformers

Marco Moresi
Dialog Systems and Machine Learning Group

22.05.2020
Content

- Recurrent Neural Network (RNN)
  - What is a RNN?
  - Long term dependencies
- Transformers
  - Self Attention Mechanism
  - Multi Head Attention
- Transformer-based models
  - BERT
  - XLNet
  - Electra
- Applications in dialogue systems
- Conclusion
Recurrent Neural Network (RNN)

- The idea behind RNNs is to make use of sequential information.
- Computation takes into account historical information using the recurrence.
- Weights are shared across time.
- Different architectures.
Recurrent Neural Network

Recurrent Neural Network (RNN)

- The idea behind RNNs is to make use of sequential information.
- Computation takes into account historical information using the recurrence.
- Weights are shared across time.
- Different architectures.

Unfolded recurrent neural network diagram.
The idea behind RNNs is to make use of sequential information.

Computation takes into account historical information using recurrence.

Weights are shared across time.

Different architectures.
Recurrent Neural Network

- **Loss Function**

\[ \mathcal{L}(\hat{y}, y) = \sum_{t=1}^{T_y} \mathcal{L}(\hat{y}^{<t>}, y^{<t>}) \]

- **Backpropagation**

\[ \frac{\partial \mathcal{L}^{(T)}}{\partial W} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}^{(T)}}{\partial W} \bigg|_{(t)} \]
Recurrent Neural Network

Vanishing gradient problem

- The gradient value becomes small during backpropagation so does not affect the values at the beginning of the network.

- Many local influences. A value is mainly influenced by inputs that are somewhere close.

- Lack of long term dependencies

*The cat, which ate a lot of food ...*, was full.
*The cats, which ate a lot of food ...*, were full.
Transformers

Architecture

[The diagram shows an input node labeled "The cat walks" connected to a transformer node, which is then connected to an output node labeled "El gato camina.

[Vaswani, et al. Attention is all you need, 2017]
Transformers

Architecture

,input The cat walks  

ENCODERS

DECODERS

El gato camina

[Vaswani, et al. Attention is all you need, 2017]
Transformers

Architecture

[ Vaswani, et al. Attention is all you need, 2017 ]
The full model architecture of the transformer. (Image source: Fig 1 & 2 in "Attention is all you need" Vaswani, et al., 2017)
Self-Attention

- Sequence to Sequence Operation
- input vectors $x_1, x_2, \ldots, x_t$
- output vectors $y_1, y_2, \ldots, y_t$

$$y_i = \sum_j w_{ij} x_j$$

$$w_{ij}' = x_i^\top x_j$$

$$w_{ij} = \frac{\exp w_{ij}'}{\sum_j \exp w_{ij}'}$$
Self-Attention in action

- We have the following sequence as input:
  - " the, cat, walks, on, the, street 

- Assign each word $t$ in our sequence its corresponding embedding
  - $V_{\text{the}}, V_{\text{cat}}, V_{\text{walks}}, V_{\text{on}}, V_{\text{the}}, V_{\text{street}}$

- Feed this sequence into a self-attention layer and the output looks as follow:
  - $Y_{\text{the}}, Y_{\text{cat}}, Y_{\text{walks}}, Y_{\text{on}}, Y_{\text{the}}, Y_{\text{street}}$

  Where $Y_{\text{cat}}$ is a weighted sum over all embeddings vectors in the first sequence, weighted by their dot-product with $V_{\text{cat}}$
Transformer

Queries, Keys and Values

- Every input vector $\mathbf{x}_i$ is used in three different ways in the self attention operation:
  - Query: It is compared to every other vector to establish the weights for its own output $\mathbf{y}_i$
  - Key: It is compared to every other vector to establish the weights for the output of the $j$-th vector $\mathbf{y}_j$
  - Value: It is used as part of the weighted sum to compute each output vector once the weights have been established.
In order to calculate the query, keys and values vectors we incorporate $W_q, W_k, W_v$ matrices

$$q_i = W_q x_i, \quad k_i = W_k x_i, \quad v_i = W_v x_i$$

$$w'_{ij} = \frac{q_i^T k_j}{\sqrt{k}}$$

$$w_{ij} = \text{softmax}(w'_{ij})$$

$$y_i = \sum_j w_{ij} v_j$$
Transformer

Multi-Head Attention

- Given the sequence:
  - “Juan gave roses to Susan”

- Words like ”gave” has different relations to different parts of the sentence.
- In a single Self-Attention operation all the information just gets summed together.
  - If “Susan gave roses to Juan” instead, the output vector $Y_{gave}$ would be the same even though the meaning has changed.

- We can give the self attention greater power of discrimination, by combining several self attention mechanisms
Transformer

Multi-Head Attention

- Multi-head attention allows the model to jointly attend to information from different representation subspaces at different position.
- Each head $i$ has its own matrices $W^i_V$, $W^i_K$, $W^i_Q$ to do the projections into different subspaces.
- Scaled Dot-Product Attention are calculated in parallel.
- All the outputs are concatenated.
- Finally the concatenated output is projected with a weight matrix $W^0$ that was trained jointly with the model.
Transformer

Multi-Head Attention (recap)

1) This is our input sentence*
2) We embed each word*
3) Split into 8 heads.
   We multiply $X$ or $R$ with weight matrices
4) Calculate attention using the resulting $Q/K/V$ matrices
5) Concatenate the resulting $Z$ matrices,
   then multiply with weight matrix $W^o$ to produce the output of the layer

* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one

https://jalammar.github.io/illustrated-transformer/
Positional Encoding

- As attention system does not take into account the order of the sequence as RNN does, it is necessary to add extra information.

- Positional Encoding

\[
PE_{pos, 2i} = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)
\]

\[
PE_{pos, 2i+1} = \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)
\]

Example:
The cat walks
The residuals

- Each sub-layer in each encoder has a residual connection around it, and is followed by a layer-normalization step.

- Normalization and residual connections are standard tricks used to help deep neural networks train faster and more accurately.

- The layer normalization is applied over the embedding dimension only.
The self-attention layer is only allowed to attend to earlier positions in the output sequence. This is done by masking future positions (setting them to -inf) before the softmax step in the self-attention calculation.

The Linear layer is a simple fully connected neural network that projects the vector produced by the stack of decoders, into a much, much larger vector called a logits vector.

The softmax layer then turns scores of logits vector into probabilities.
Transformer

Results

Architecture Details:
- Transformer base model
  - Stack of 6 encoder-decoder
  - 8 Attention heads
  - 512 hidden dimensions
- Transformer big
  - Stack of 6 encoder-decoder
  - 16 Attention heads
  - 1024 hidden dimensions

<table>
<thead>
<tr>
<th>Model</th>
<th>BLEU</th>
<th>Training Cost (FLOPs)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>EN-DE</td>
<td>EN-FR</td>
</tr>
<tr>
<td>ByteNet [15]</td>
<td>23.75</td>
<td>39.2</td>
</tr>
<tr>
<td>Deep-Att + PosUnk [32]</td>
<td>24.6</td>
<td>39.92</td>
</tr>
<tr>
<td>GNMT + RL [31]</td>
<td>25.16</td>
<td>40.46</td>
</tr>
<tr>
<td>ConvS2S [8]</td>
<td>26.03</td>
<td>40.56</td>
</tr>
<tr>
<td>MoE [26]</td>
<td>26.36</td>
<td><strong>41.29</strong></td>
</tr>
<tr>
<td>Deep-Att + PosUnk Ensemble [32]</td>
<td></td>
<td>40.4</td>
</tr>
<tr>
<td>GNMT + RL Ensemble [31]</td>
<td>26.30</td>
<td>41.16</td>
</tr>
<tr>
<td>ConvS2S Ensemble [8]</td>
<td>26.36</td>
<td><strong>41.29</strong></td>
</tr>
<tr>
<td>Transformer (base model)</td>
<td>27.3</td>
<td>38.1</td>
</tr>
<tr>
<td>Transformer (big)</td>
<td><strong>28.4</strong></td>
<td><strong>41.0</strong></td>
</tr>
</tbody>
</table>

BLEU scores on the English-to-German and English-to-French newstest 2014
Designed to pretrain bidirectional representation from unlabeled text

BERT-base: 12 Transformer encoder blocks, hidden dimension 768 and 12 attention heads.

Fixed input size (512 tokens)

Word embedding + Positional encoding + Sentence embedding

Special tokens ([CLS], [SEP], sentence A/B embedding)

Pre-training tasks:
- Masked LM
- Next Sentence Prediction

[Devlin, et al.. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.]
**Transformer**

**BERT** (Bidirectional Encoder Representation from Transformers)

- Designed to pretrain bidirectional representation from unlabeled text.
- BERT-base: 12 Transformer encoder blocks, hidden dimension 768 and 12 attention heads.
- Fixed input size (512 tokens).
- Word embedding +Positional encoding +Sentence embedding.
- Special tokens ([CLS], [SEP], sentence A/B embedding).
- Pre-training tasks:
  - Masked LM
  - Next Sentence Prediction

**Masked Language Model:**

- Mask some percentage of the input tokens at random, and then predict those masked tokens.
- Last hidden vector corresponding to the masked tokens are fed into an output softmax layer over the vocabulary.

[Devlin, et al.. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.]
**Transformer**

**BERT** (Bidirectional Encoder Representation from Transformers)

- Designed to pretrain bidirectional representation from unlabeled text.
- BERT-base: 12 Transformer encoder blocks, hidden dimension 768 and 12 attention heads.
- Fixed input size (512 tokens).
- Word embedding + Positional encoding + Sentence embedding.
- Special tokens ([CLS], [SEP], A/B embedding).
- Pre-training tasks:
  - Masked LM
  - Next Sentence Prediction

**Next Sentence Prediction:**
- Given a pair of two sentences \((A, B)\)
- Learns to predict if the second sentence in the pair is the subsequent sentence in the original document

[Devlin, et al.. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.]
Masked Language Model

Use the output of the masked word's position to predict the masked word.

Randomly mask 15% of tokens.

Input:

[CLS] Let's stick to improvisation in this skit

Possible classes:
- All English words
- Improvisation
- Zzyzyva

FTNN + Softmax
Transformer

Next Sentence Prediction

Predict likelihood that sentence B belongs after sentence A

1% IsNext
99% NotNext

FFNN + Softmax

Tokenized Input

[CLS] the man [MASK] to the store [SEP] penguin [MASK] are flightless birds [SEP]

Input

Sentence A
Sentence B
Transformer

Fine Tuning

(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG

(b) Single Sentence Classification Tasks: SST-2, CoLA

(c) Question Answering Tasks: SQuAD v1.1

(d) Single Sentence Tagging Tasks: CoNLL-2003 NER
## Benchmarks

<table>
<thead>
<tr>
<th>Model</th>
<th>MNLI</th>
<th>QNLI</th>
<th>QQP</th>
<th>RTE</th>
<th>SST-2</th>
<th>MRPC</th>
<th>CoLA</th>
<th>STS-B</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-OpenAI SOTA</td>
<td>80.6</td>
<td>82.3</td>
<td>66.1</td>
<td>61.7</td>
<td>93.2</td>
<td>86.0</td>
<td>35.0</td>
<td>81.0</td>
<td>74.0</td>
</tr>
<tr>
<td>BiLSTM+ELMo+Attn</td>
<td>76.4</td>
<td>79.8</td>
<td>64.8</td>
<td>56.8</td>
<td>90.4</td>
<td>84.9</td>
<td>36.0</td>
<td>73.3</td>
<td>71.0</td>
</tr>
<tr>
<td>OpenAI GPT</td>
<td>82.1</td>
<td>87.4</td>
<td>70.3</td>
<td>56.0</td>
<td>91.3</td>
<td>82.3</td>
<td>45.4</td>
<td>80.0</td>
<td>75.1</td>
</tr>
<tr>
<td>BERT Base</td>
<td>84.6</td>
<td>90.5</td>
<td>71.2</td>
<td>66.4</td>
<td>93.5</td>
<td>88.9</td>
<td>52.1</td>
<td>85.8</td>
<td>79.6</td>
</tr>
<tr>
<td>BERT Large</td>
<td><strong>86.7</strong></td>
<td><strong>92.7</strong></td>
<td><strong>72.1</strong></td>
<td><strong>70.1</strong></td>
<td><strong>94.9</strong></td>
<td><strong>89.3</strong></td>
<td><strong>60.5</strong></td>
<td><strong>86.5</strong></td>
<td><strong>82.1</strong></td>
</tr>
</tbody>
</table>

GLUE Test results

### SQuAD 1.1 results

<table>
<thead>
<tr>
<th>Model</th>
<th>EM</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>BiDAF+ELMo</td>
<td>-</td>
<td>85.6</td>
</tr>
<tr>
<td>R.M. Reader</td>
<td>81.2</td>
<td>82.3</td>
</tr>
<tr>
<td>BERT Large</td>
<td><strong>84.1</strong></td>
<td><strong>90.9</strong></td>
</tr>
</tbody>
</table>

### SQuAD 2.0 results

<table>
<thead>
<tr>
<th>Model</th>
<th>EM</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>unet</td>
<td>71.4</td>
<td>74.9</td>
</tr>
<tr>
<td>SLQA</td>
<td>71.4</td>
<td>74.4</td>
</tr>
<tr>
<td>BERT Large</td>
<td><strong>80.0</strong></td>
<td><strong>83.1</strong></td>
</tr>
</tbody>
</table>
Transformer

**XLNet (Generalized Autoregressive Pretraining for Language Understanding)**

- Permutation Language Modeling
- Does not rely on data corruption
- Integrates the segment recurrence mechanism and relative encoding.
- Mitigate the problem of Masked Language Model

[Yang, et al. Xlnet: Generalized autoregressive pretraining for language understanding, 2019]
Transformer

**XLNet** *(Generalized Autoregressive Pretraining for Language Understanding)*

- Permutation Language Modeling
- Does not rely on data corruption
- Integrates the segment recurrence mechanism and relative encoding.
- Mitigate the problem of Masked Language Model

Language Model:

\[
P(W) = \prod_{t} P(w_t | w_0, w_1, \ldots w_{t-1})
\]

\[
W^* = \arg\max_{W} P(W)
\]

[XLNet: Generalized autoregressive pretraining for language understanding, 2019]

[Image of diagram showing XLNet - Permutation Language Modeling]
**Transformer**

**XLNet** *(Generalized Autoregressive Pretraining for Language Understanding)*

- Permutation Language Modeling
- Does not rely on data corruption
- Integrates the segment recurrence mechanism and relative encoding.
- Mitigate the problem of Masked Language Model

[Yang, et al. Xlnet: Generalized autoregressive pretraining for languageunderstanding, 2019]
**Transformer**

**XLNet** *(Generalized Autoregressive Pretraining for Language Understanding)*

- Permutation Language Modeling
- Does not rely on data corruption
- Integrates the segment recurrence mechanism and relative encoding.
- Mitigate the problem of Masked Language Model

---

[Yang, et al. Xlnet: Generalized autoregressive pretraining for language understanding, 2019]
Transformer

**XLNet** *(Generalized Autoregressive Pretraining for Language Understanding)*

- Permutation Language Modeling
- Does not rely on data corruption
- Integrates the segment recurrence mechanism and relative encoding.
- Mitigate the problem of Masked Language Model

[Yang, et al. Xlnet: Generalized autoregressive pretraining for language understanding, 2019]
XLNet (Generalized Autoregressive Pretraining for Language Understanding)

- Permutation Language Modeling
- Does not rely on data corruption
- Integrates the segment recurrence mechanism and relative encoding.
- Mitigate the problem of Masked Language Model

[Yang, et al. Xlnet: Generalized autoregressive pretraining for language understanding, 2019]
XLNet (Generalized Autoregressive Pretraining for Language Understanding)

- Permutation Language Modeling
- Does not rely on data corruption
- Integrates the segment recurrence mechanism and relative encoding.
- Mitigate the problem of Masked Language Model

"cats", "than", "I", "more", "dogs", "like"

[Yang, et al. Xlnet: Generalized autoregressive pretraining for language understanding, 2019]
**Transformer**

**XLNet** *(Generalized Autoregressive Pretraining for Language Understanding)*

- Permutation Language Modeling

- Does not rely on data corruption

- Integrates the segment recurrence mechanism and relative encoding.

- Mitigate the problem of Masked Language Model

---

[Yang, et al. Xlnet: Generalized autoregressive pretraining for languageunderstanding, 2019]
Permutation Language Modeling

Does not rely on data corruption

Integrates the segment recurrence mechanism and relative encoding

Mitigate the problem of Masked Language Model

Example:
New York is a city
Target: Predict tokens New and York

\[ J_{\text{BERT}} = \log p(\text{New} | \text{is a city}) + \log p(\text{York} | \text{is a city}) \]

\[ J_{\text{XLNet}} = \log p(\text{New} | \text{is a city}) + \log p(\text{York} | \text{New, is a city}) \]

[Yang, et al. Xlnet: Generalized autoregressive pretraining for language understanding, 2019]
## Benchmarks

<table>
<thead>
<tr>
<th>Model</th>
<th>MNLI</th>
<th>QNLI</th>
<th>QQP</th>
<th>RTE</th>
<th>SST-2</th>
<th>MRPC</th>
<th>CoLA</th>
<th>STS-B</th>
<th>AVERAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>BERT-Large</td>
<td>86.6</td>
<td>92.3</td>
<td>91.3</td>
<td>70.4</td>
<td>93.2</td>
<td>88.0</td>
<td>60.6</td>
<td>90</td>
<td>84.1</td>
</tr>
<tr>
<td>XLNet-Base</td>
<td>86.8</td>
<td>91.7</td>
<td>91.4</td>
<td>74.0</td>
<td>94.7</td>
<td>88.2</td>
<td>60.2</td>
<td>89.5</td>
<td>84.6</td>
</tr>
<tr>
<td>XLNet-Large</td>
<td>89.8</td>
<td>93.9</td>
<td>91.8</td>
<td>83.8</td>
<td>95.6</td>
<td>89.2</td>
<td>63.6</td>
<td>91.8</td>
<td>87.5</td>
</tr>
</tbody>
</table>

GLUE Test results

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SQuAD1.1</td>
<td>EM</td>
<td>F1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BERT-Large</td>
<td>84.1</td>
<td>90.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XLNet-Large</td>
<td>89.7</td>
<td>95.1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SQuAD 1.1 results

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SQuAD2.0</td>
<td>EM</td>
<td>F1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BERT-Large</td>
<td>80.0</td>
<td>83.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XLNet-Large</td>
<td>87.9</td>
<td>90.6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SQuAD 2.0 results
Mitigate the problem of Masked Language Model

Replaced token detection as pre-training task

More efficient pre training task

Bi-directional representation

Less computation consumption

An overview of replaced token detection. After pre-training, we throw out the generator and only fine-tune the discriminator (the ELECTRA model) on downstream tasks.

Transformer

Benchmarks

<table>
<thead>
<tr>
<th>Model</th>
<th>MNLI</th>
<th>QNLI</th>
<th>QQP</th>
<th>RTE</th>
<th>SST-2</th>
<th>MRPC</th>
<th>CoLA</th>
<th>STS-B</th>
<th>AVERAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>BERT</td>
<td>86.7</td>
<td>92.7</td>
<td>89.3</td>
<td>70.1</td>
<td>94.9</td>
<td>85.4</td>
<td>60.5</td>
<td>86.5</td>
<td>83.3</td>
</tr>
<tr>
<td>XLNET</td>
<td>90.9</td>
<td>93.9</td>
<td>90.4</td>
<td>92.5</td>
<td>97.1</td>
<td>90.5</td>
<td>70.2</td>
<td>90.4</td>
<td>89.5</td>
</tr>
<tr>
<td>ELECTRA</td>
<td>91.3</td>
<td>95.8</td>
<td>90.8</td>
<td>92.5</td>
<td>97.1</td>
<td>90.7</td>
<td>71.7</td>
<td>90.8</td>
<td>90.1</td>
</tr>
</tbody>
</table>

GLUE Test results

<table>
<thead>
<tr>
<th></th>
<th>EM</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQuAD1.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BERT-Large</td>
<td>84.1</td>
<td>90.9</td>
</tr>
<tr>
<td>XLNet-Large</td>
<td>89.7</td>
<td>95.1</td>
</tr>
<tr>
<td>ELECTRA</td>
<td>89.7</td>
<td>94.9</td>
</tr>
</tbody>
</table>

SQuAD 1.1 results

<table>
<thead>
<tr>
<th></th>
<th>EM</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQuAD2.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BERT-Large</td>
<td>78.98</td>
<td>81.77</td>
</tr>
<tr>
<td>XLNet-Large</td>
<td>87.9</td>
<td>90.6</td>
</tr>
<tr>
<td>ELECTRA</td>
<td>88.7</td>
<td>91.4</td>
</tr>
</tbody>
</table>

SQuAD 2.0 results
Transformer

Benchmarks

Replaced token detection pre-training consistently outperforms masked language model pre-training given the same compute budget. The left figure is a zoomed-in view of the dashed box.
**Applications in Dialogue Systems**

**Dialog State Tracking**
- TripPy: A Triple Copy Strategy for Value Independent Neural Dialog State Tracking (Heck et al., 2020)
- BERT-DST: Scalable End-to-End Dialogue State Tracking with Bidirectional Encoder Representations from Transformer (Chao et al., 2019)

**Natural Language Generation**
- Few-shot Natural Language Generation for Task-Oriented Dialog (Peng et al., 2020)
- Semantically Conditioned Dialog Response Generation via Hierarchical Disentangled Self-Attention (Chen et al., 2019)

**Evaluation**
- BERTScore: Evaluating Text Generation with BERT (Zhang et al., 2019)
- USR: An Unsupervised and Reference Free Evaluation Metric for Dialog Generation (Mehri et al., 2020)

**Sentiments in Dialog**
- Hierarchical Transformer Network for Utterance-level Emotion Recognition (Li et al., 2020)

... and more.
Conclusion

Pros
- Tackle scarcity data problem
- Bi-directional representation
- Easy to fine-tune for a specific task

Cons
- Hard to start from the scratch
- Catastrophic forgetting
- More expensive to train than RNN
Thanks!
References

- TripPy: A Triple Copy Strategy for Value Independent Neural Dialog State Tracking (Heck et al., 2020)
- BERT-DST: Scalable End-to-End Dialogue State Tracking with Bidirectional Encoder Representations from Transformer (Chao et al., 2019)
- Few-shot Natural Language Generation for Task-Oriented Dialog (Peng et al., 2020)
- Semantically Conditioned Dialog Response Generation via Hierarchical Disentangled Self-Attention (Chen et al., 2019)
- BERTScore: Evaluating Text Generation with BERT (Zhang et al., 2019)
- USR: An Unsupervised and Reference Free Evaluation Metric for Dialog Generation (Mehri et al., 2020)
- Hierarchical Transformer Network for Utterance-level Emotion Recognition (Li et al., 2020)
- Xlnet: Generalized autoregressive pretraining for language understanding (Yang et al., 2019)
- Attention is all you need, (Vaswani et al., 2017)
- Bert: Pre-training of deep bidirectional transformers for language understanding. (Devlin et al., 2018)
- Glue: A multi-task benchmark and analysis platform for natural language understanding. (Wang et al, 2018)