Contextualized word embeddings

Instructor: Thien Huu Nguyen
Recap: word2vec

Context words: windows of size 2 before and after the center word

Continuous Bag of Words (CBOW):
predicting the center words using the context words \( P(w_t|w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2}) \)

Skip-grams (SG):
predicting the context words using the center word \( P(w_{t+i}|w_t), i \in \{-2, -1, 1, 2\} \)
Recap: wovd2vec

• For each position $i = 1, ..., N$, predict the context words within a window of fixed size $m$, given the center word $w_i$:

\[
\text{Likelihood} = L(\theta) = \prod_{i=1}^{N} \prod_{-m \leq j \leq m, j \neq 0} P(w_{i+j} | w_i; \theta)
\]

• The objective/loss function is the (average) negative log likelihood:

\[
\text{loss} = J(\theta) = -\frac{1}{N} \sum_{i=1}^{N} \sum_{-m \leq j \leq m, j \neq 0} \log P(w_{i+j} | w_i; \theta)
\]

• The parameters $\theta$ here involve the word vectors we want to learn.
• Minimizing the loss function amounts to maximizing the predictive accuracy.
Recap: word2vec
Problem with word2vec

• A single vector is used for a word, neglecting its possibly multiple meanings (i.e., polysemy)
  – e.g., “fire”, “bank”, “present”, etc.
  – a simple trick: learn an embedding vector for each meaning (i.e., senses in WordNet) of the words, but this assumes the availability of high-quality word sense disambiguation systems to assign senses to words in the sentences (not reliable)
  – this ignores the contexts of the words in the sentences, thus called uncontextualized word embeddings

• So, we want contextualized word embeddings that can take the context of the words (i.e., their sentences) into account to produce vectors for the words
Contextualized Word Embeddings

- Idea: the vector for a word should be specific to the word’s context, so we can train models that take the word’s context and produce the word vector.

- Questions: How do we train such models? How do we obtain the training data?

- Answer: remember language models with RNN? We can train a RNN language model and use the RNN hidden vectors as the contextualized embeddings for the words in the sentences.
  - We thus need to store the whole language model (i.e., the parameters) so it can be applied to new sentences.
  - The embedding vector for a word will now be conditional on the other words in the sentences.
  - The whole language model can be fine-tuned later for specific downstream tasks.
Recap: Language Modeling

- Language Modeling is the task of predicting what word comes next given a sequence of previous words.

  the students opened their

- More formally, given a sequence of words $x_1, x_2, ..., x_i$, compute the probability distribution of the following word:

  $P(x_{i+1} | x_i, x_{i-1}, ..., x_1)$

- A system that can do this is called a language model
Recap: the RNN language model

output distribution
\[ y_i = \text{softmax}(Uh_t + b_2) \]

hidden states
\[ h_i = \sigma(W_h h_{i-1} + W_e e_i + b_1) \]
\( h_0 \) is the initial hidden state using LSTM or GRU is more common

word embeddings

words/one-hot vectors
\[ x_i \in \{0,1\}^{|V|} \]
Contextualized word embeddings with the RNN language model

Negative log likelihood/probability

- Use the hidden vectors $h_i$ as the word vectors for the words in the input sentence
- Each vector $h_i$ for $x_i$ will involve the context information from the previous words in the sentence, thus being contextualized

words/one-hot vectors

$x_i \in \{0,1\}^{|V|}$

$J_1(\theta) + J_2(\theta) + J_3(\theta) + J_4(\theta) + ... = J(\theta)$
Problem with the RNN-based word vectors

• The word vectors $h_i$ only encode the context information from the words on the left (i.e., the previous words), what’s about the context on the right?

• The RNN-based model needs to keep the input embedding table $E$ that is large and assumes a fixed vocabulary. What’s if we have out-of-vocabulary words?
  — One trick is to reserve an UNKNOWN word for all the out-of-vocabulary words, but this might not lose some useful information from the form of the word (i.e., morphology).

• A single RNN layer might not be sufficient to capture the underlying context information for the input sentences. How’s about making it deeper (i.e., more layers)?
Deep Contextualized Word Embeddings

• ELMo (Embeddings from Language Models) is introduced in (Peters et al., 2018).

• Main ideas:
  – Jointly perform both forward and backward language modeling (i.e., bidirectional language models)
  – Increase the number of RNN layers
  – Employ character-level input representations to alleviate the out-of-vocabulary issue
  – Fine tune the model for downstream tasks
We learn the model parameters by jointly optimizing the forward and backward language model objectives:

$$\sum_{i=1}^{N} (\log \hat{P}(x_{i+1}|x_{i}, ..., x_{1}; \theta_{cnn}, \tilde{\theta}_{lstm}, \theta_{s})$$

$$+ \log \hat{P}(x_{i-1}|x_{i}, ..., x_{N}; \theta_{cnn}, \tilde{\theta}_{lstm}, \theta_{s}))$$

Note that the parameters for the character-based CNN and the softmax layer for distributions are shared.
ELMo: The character CNN

- **Character embeddings**
- **CNN with multiple filters**
- **Highway Networks**
- **Max-pooling**
Fine-tuning with ELMo

• The representation vector for input token $i$ is now:

$$R_k = \{e_i, \tilde{h}_{i,j}^{LM}, \check{h}_{i,j}^{LM} | j = 1, ..., L\}$$

$$= \{h_{i,j} | j = 0, ..., L\}$$

with $h_{i,0} = e_i$ and $h_{i,j} = [\tilde{h}_{i,j}^{LM}, \check{h}_{i,j}^{LM}]$ otherwise.

• We can combine the internal representations via a (trainable, weighted) linear combination:

$$ELMo_i^{task} = \gamma^{task} \sum_{j=0}^{L} s_{j}^{task} h_{i,j}$$

with $s^{task}$ are softmax-normalized weights.

• The ELMo representations can be used as extra token-level features:
  – For input layer: replace the original input vector $x_i$ with $[x_i, ELMo_i^{task}]$
  – For output layer: replace the hidden vector $h_i$ with $[h_i, ELMo_i^{task}]$
ELMo: evaluation

<table>
<thead>
<tr>
<th>Task</th>
<th>Previous SOTA</th>
<th>Our Baseline</th>
<th>ELMo + Baseline</th>
<th>Increase (Absolute/Relative)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQuAD</td>
<td>Liu et al. (2017)</td>
<td>84.4</td>
<td>81.1</td>
<td>85.8</td>
</tr>
<tr>
<td>SNLI</td>
<td>Chen et al. (2017)</td>
<td>88.6</td>
<td>88.0</td>
<td>88.7 ± 0.17</td>
</tr>
<tr>
<td>SRL</td>
<td>He et al. (2017)</td>
<td>81.7</td>
<td>81.4</td>
<td>84.6</td>
</tr>
<tr>
<td>Coref</td>
<td>Lee et al. (2017)</td>
<td>67.2</td>
<td>67.2</td>
<td>70.4</td>
</tr>
<tr>
<td>NER</td>
<td>Peters et al. (2017)</td>
<td>91.93 ± 0.19</td>
<td>90.15</td>
<td>92.22 ± 0.10</td>
</tr>
<tr>
<td>SST-5</td>
<td>McCann et al. (2017)</td>
<td>53.7</td>
<td>51.4</td>
<td>54.7 ± 0.5</td>
</tr>
</tbody>
</table>

*Table 1 from the original paper.*
Problem with ELMo

• The recurrent computation is slow and prevents parallelization. The actual application often fixes the ELMo network to avoid the computational cost, thus lacking the ability to fine tune the network for the downstream tasks.

• Although the gated mechanisms in LSTM and GRU can mitigate the gradient vanishing problem to some extent, it is still very difficult to incorporate the context information from the very far way words into the representation for the current word.

• Question: How do we address such problem?
• Answer: self-attention
  – Avoid the recurrent connections to enable parallelization
  – The far away words have chance to directly contribute to the current word’s representation, depending on their relatedness.
Self-attention

- Similar to RNN, self-attention is a sequence-to-sequence operation: a sequence of vector comes in and a sequence of vectors goes out.

- Let \( x_1, \ldots, x_n \) be the input vector sequence. The output vector sequence \( y_1, \ldots, y_n \) is computed by:

\[
y_i = \sum_j w_{ij} x_j
\]

\[
w'_{ij} = x_i^\top x_j
\]

\[
w_{ij} = \frac{\exp w'_{ij}}{\sum_j \exp w'_{ij}}
\]

http://www.peterbloem.nl/blog/transformers
Some improvements for self-attention

• Query, key and value vectors to differentiate three roles for the input vector $x_i$: computing the weights for the output vector $y_i$, computing the weights for the other output vectors $y_j (j \neq i)$, and serving as the part of the weighted sums for the output vectors:

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

$$w'_{ij} = q_i^T k_j$$

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

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

• Scaling the dot product: to alleviate the large values of the dot product due to the dimensions of the input vectors:

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

• This process is denoted by:

$$y_1, ..., y_n = \text{self\_attention}(x_1, ..., x_n; W_q, W_k, W_v)$$
Some improvements for self-attention

• Multi-head attention
  – A single self-attention operation might only focus on one semantic aspect in the output representation vectors.
  – Multiple self-attention operations might enable greater representation power to cover multiple semantic aspects for the representation.
  – Introducing multiple query, key and value transformation matrices $W^q_r, W^k_r, W^v_r$ (called attention head and indexed by $r$) to compute multiple output representation vectors for each position $i$ in the input. The corresponding representation vectors for each position are concatenated and sent to a feed-forward net to reduce the dimension back those in the original input.

$$y^r_1, ..., y^r_n = \text{self\_attention}(x_1, ..., x_n; W^q_r, W^k_r, W^v_r)$$
$$y_i = W[y^1_i, ..., y^H_i] + b \quad \text{where } H \text{ is the number of attention heads}$$
$$|y_i| = |x_i| = k$$
Transformer

- A composition of many multi-head attention operations, originally designed for machine translation with the encoder and decoder networks (Vaswani et al., 2017).

*Figure from the original paper*
Transformer: Positional encoding

- The self-attention operation is permutation equivariant (i.e., if we permute the input sequence, the output sequence will be exactly the same, except permuted also).
- How can we make the representations sensitive to word order?
- Two possible ways:
  - Positional embeddings: learn vectors to embed the word positions in the sentences as we embed the words (i.e., create an embedding vector for each possible position) (can’t generalize to unseen positions).
  - Positional encoding: **don’t learn** the position vectors, simply choose a function to map the word positions to real-valued vectors (to be added to the word vectors).

\[
PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}})
\]
\[
PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{model}})
\]
**Transformer: Encoder**

- **Encoder** is composed of N layers; each of which has two sublayers (a multi-head attention and feed forward network) with residual connections around them.

```
1  x_encoder = word_embeds + pos_embeds
2  for layer_i in range(N):
3      q, k, v = x_encoder, x_encoder, x_encoder
4      att = MultiHeadAttention(q, k, v)
5      att = LayerNorm(x_encoder + Dropout(att))
6      ffw = Feedfw(att)
7      x_encoder = LayerNorm(att + dropout(ffw))
```
Transformer: Decoder

- Decoder is designed similarly to Encoder except that it has a new sublayer called Masked Multi-Head Attention to ensure that predictions for position \( i \) can only depend on previous positions.
Transformer: Decoder

- The masking schema in Transformer Decoder (multiplied to the attention matrix directly):

http://nlp.seas.harvard.edu/2018/04/03/attention.html
### Transformer for machine translation

**Evaluation**

<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 [18]</td>
<td>23.75</td>
<td>39.2</td>
</tr>
<tr>
<td>Deep-Att + PosUnk [39]</td>
<td>24.6</td>
<td>39.92</td>
</tr>
<tr>
<td>GNMT + RL [38]</td>
<td>25.16</td>
<td>40.46</td>
</tr>
<tr>
<td>ConvS2S [9]</td>
<td>26.03</td>
<td>40.56</td>
</tr>
<tr>
<td>MoE [32]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deep-Att + PosUnk Ensemble [39]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GNMT + RL Ensemble [38]</td>
<td>26.30</td>
<td>41.16</td>
</tr>
<tr>
<td>ConvS2S Ensemble [9]</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>41.8</strong></td>
</tr>
</tbody>
</table>
GPT

• Generative Pretrained Transformer [Radford et al., 2018]
• Similar to ELMo, GPT trains a traditional **left-to-right** language model. However, the decoder architecture from Transformer is used instead of LSTM.
• In particular, the Transformer decoder is pre-trained with the next word prediction task:

\[
L_{\text{pretrain}} = \sum_{i=1}^{K} \log(P(t_i|t_{i-1}, t_{i-2}, \ldots, t_1; \theta_{\text{transformer}}))
\]
GPT: Fine-tuning

- The whole GPT model is often fine-tuned for downstream applications
- Initialize all weights with pretrained weights
- Convert target task’s input to the single sequence format
- Use the last token’s representation to make predictions
- Add an extra softmax layer to make predictions on target task
- Re-train the whole model with the combined loss of the target task and the language model task:
  - Improve generalization of the supervised model
  - Accelerate convergence

\[ L = L_{\text{target}} + \lambda L_{\text{pretrain}} \]
GPT: Input Transformation for Fine-tuning

Classification

Entailment

Similarity

Multiple Choice
## GPT: Evaluation

<table>
<thead>
<tr>
<th>Method</th>
<th>MNLI-m</th>
<th>MNLI-mm</th>
<th>SNLI</th>
<th>SciTail</th>
<th>QNLI</th>
<th>RTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESIM + ELMo [44] (5x)</td>
<td>-</td>
<td>-</td>
<td>89.3</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>CAFE [58] (5x)</td>
<td>80.2</td>
<td>79.0</td>
<td>89.3</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Stochastic Answer Network [35] (3x)</td>
<td>80.6</td>
<td>80.1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>CAFE [58]</td>
<td>78.7</td>
<td>77.9</td>
<td>88.5</td>
<td>83.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GenSen [64]</td>
<td>71.4</td>
<td>71.3</td>
<td>-</td>
<td>-</td>
<td>82.3</td>
<td>59.2</td>
</tr>
<tr>
<td>Multi-task BiLSTM + Attn [64]</td>
<td>72.2</td>
<td>72.1</td>
<td>-</td>
<td>-</td>
<td>82.1</td>
<td>61.7</td>
</tr>
<tr>
<td>Finetuned Transformer LM (ours)</td>
<td><strong>82.1</strong></td>
<td><strong>81.4</strong></td>
<td><strong>89.9</strong></td>
<td><strong>88.3</strong></td>
<td><strong>88.1</strong></td>
<td><strong>56.0</strong></td>
</tr>
</tbody>
</table>
GPT: Evaluation

Just wanted to give you all a heads up, our lab found an amazing breakthrough in language understanding. but we also worry it may fall into the wrong hands. so we decided to scrap it and only publish the regular *ACL stuff instead. Big respect for the team for their great work.

1:08 AM · Feb 16, 2019 · Twitter for Android

112 Retweets 774 Likes

To wit: when GPT-2 was tasked with writing a response to the prompt, “Recycling is good for the world, no, you could not be more wrong,” the machine spat back:

“Recycling is NOT good for the world. It is bad for the environment, it is bad for our health, and it is bad for our economy. I’m not kidding. Recycling is not good for the environment. It is destructive to the earth and it is a major contributor to global warming. Recycling is not good for our health. It contributes to obesity and diseases like heart disease and cancer. Recycling is bad for our economy. It increases the cost of a product, and in turn, the price of everything that is made with that product. Recycling is not good for our nation. We pay a tremendous price for the privilege of having the world’s most advanced and efficient recycling system. Recycling is a huge, colossal waste of time, energy, money, and resources.”

https://techcrunch.com/2019/02/17/openai-text-generator-dangerous/
Problem with GPT

- Still only use the left context to represent the words in the sentences.

Anybody sees BERT?
BERT

• Bidirectional Encoder Representations from Transformer (BERT) (Devlin et al., 2018)

• BERT aims to capture the context over the whole sentence for the word representations based on two different training objectives:
  – Masked language model (i.e., the cloze test)
  – Next sentence prediction

• Use the encoder of Transformer to the network architecture

• Can be fine-tuned for both sentence and word level tasks

• Use the WordPiece tokenization: the vocabulary is initialized with all the individual characters in the language, and then the most frequent/likely combinations of the symbols in the vocabulary are iteratively added to the vocabulary.
BERT: pre-training

- Masked Language Model

> Words are chosen at random for being masked (by a special token [MASK]) or replaced by a random token.

=> force the model to collect bidirectional information to make true predictions.

> Training objective: recover the original tokens from the corrupted version:

\[
\sum_{i=1}^{K} m_i \log(P(t_i|t_1, ..., t_{i-1}, t_{i+1}, ..., t_K))
\]

\(m_i \in \{0, 1\}\) indicates whether \(t_i\) is masked or not.
BERT: pre-training

- Next sentence prediction (binary classification) (done after the masked language model pre-training)
  - Introduce two special tokens [CLS] and [SEP] that are put at the beginning of the first sentence and the end of the sentences respectively.
  - Sample 2 sentences A and B:
    - 50% of the time B is actually next to A (positive examples)
    - 50% of the time B is randomly chosen (negative examples)
BERT: fine-tuning for sentence pair classification tasks

• i.e., natural language inference (MNLI), question pair matching (QQP).

⇒ Use [CLS] for fine-tuning on sentence pair classification tasks.
BERT: fine-tuning for single sentence classification tasks

Consider a single sentence $A$ as a degenerate $<A, \emptyset>$ pair. ⇒ Use $[CLS]$ for fine-tuning as usual.
BERT: Evaluation

- GLUE (General Language Understanding Evaluation) benchmark

<table>
<thead>
<tr>
<th>System</th>
<th>MNLI-(m/mm)</th>
<th>QQP</th>
<th>QNLI</th>
<th>SST-2</th>
<th>CoLA</th>
<th>STS-B</th>
<th>MRPC</th>
<th>RTE</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>392k</td>
<td>108k</td>
<td>67k</td>
<td>8.5k</td>
<td>5.7k</td>
<td>3.5k</td>
<td>2.5k</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Pre-OpenAI SOTA</td>
<td>80.6/80.1</td>
<td>66.1</td>
<td>82.3</td>
<td>93.2</td>
<td>35.0</td>
<td>81.0</td>
<td>86.0</td>
<td>61.7</td>
<td>74.0</td>
</tr>
<tr>
<td>BiLSTM+ELMo+Attn</td>
<td>76.4/76.1</td>
<td>64.8</td>
<td>79.8</td>
<td>90.4</td>
<td>36.0</td>
<td>73.3</td>
<td>84.9</td>
<td>56.8</td>
<td>71.0</td>
</tr>
<tr>
<td>OpenAI GPT</td>
<td>82.1/81.4</td>
<td>70.3</td>
<td>87.4</td>
<td>91.3</td>
<td>45.4</td>
<td>80.0</td>
<td>82.3</td>
<td>56.0</td>
<td>75.1</td>
</tr>
<tr>
<td>BERT_{BASE}</td>
<td>84.6/83.4</td>
<td>71.2</td>
<td>90.5</td>
<td>93.5</td>
<td>52.1</td>
<td>85.8</td>
<td>88.9</td>
<td>66.4</td>
<td>79.6</td>
</tr>
<tr>
<td>BERT_{LARGE}</td>
<td><strong>86.7/85.9</strong></td>
<td><strong>72.1</strong></td>
<td><strong>92.7</strong></td>
<td><strong>94.9</strong></td>
<td><strong>60.5</strong></td>
<td><strong>86.5</strong></td>
<td><strong>89.3</strong></td>
<td><strong>70.1</strong></td>
<td><strong>82.1</strong></td>
</tr>
</tbody>
</table>
BERT: finetuning for question answering

- e.g., the SQuAD dataset
- Determine the answer span by identifying its start and end token.
- Introduce **START** vector $S \in \mathbb{R}^H$ and **END** vector $E \in \mathbb{R}^H$
- Probability of the $i$-th token being the start of the answer:
  \[ P(\text{start} = i) = \frac{e^{S \cdot T_i}}{\sum e^{S \cdot T_i'}} \]
- Probability of the $j$-th token being the end of the answer:
  \[ P(\text{end} = j) = \frac{e^{E \cdot T_j}}{\sum e^{E \cdot T_j'}} \]
- Training objective: maximize
  \[ \log(P(\text{start} = i)) + \log(P(\text{end} = j)) \]
BERT: finetuning for question answering (token level tasks)

- e.g., the SQuAD dataset

- Determine the answer span by identifying its start and end token.
- Introduce **START** vector $S \in \mathbb{R}^H$ and **END** vector $E \in \mathbb{R}^H$
- Probability of the $i$-th token being the start of the answer:
  $$P(\text{start} = i) = \frac{e^{S \cdot T_i}}{\sum e^{S \cdot T_i}}$$
- Probability of the $j$-th token being the end of the answer:
  $$P(\text{end} = j) = \frac{e^{E \cdot T_j}}{\sum e^{E \cdot T_j}}$$
- Training objective: maximize
  $$\log(P(\text{start} = i)) + \log(P(\text{end} = j))$$
- Evaluation: choose the span with highest score:
  $$\text{score}(start = i, end = j) = S \cdot T_i + E \cdot T_j$$
## BERT: Evaluation

### SQuAD v1.1.

<table>
<thead>
<tr>
<th>System</th>
<th>Dev EM</th>
<th>Dev F1</th>
<th>Test EM</th>
<th>Test F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Human</td>
<td>-</td>
<td>-</td>
<td>82.3</td>
<td>91.2</td>
</tr>
<tr>
<td>#1 Ensemble - nlnet</td>
<td>-</td>
<td>-</td>
<td>86.0</td>
<td>91.7</td>
</tr>
<tr>
<td>#2 Ensemble - QANet</td>
<td>-</td>
<td>-</td>
<td>84.5</td>
<td>90.5</td>
</tr>
</tbody>
</table>

### Published

<table>
<thead>
<tr>
<th>System</th>
<th>Dev EM</th>
<th>Dev F1</th>
<th>Test EM</th>
<th>Test F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>BiDAF+ELMo (Single)</td>
<td>-</td>
<td>85.6</td>
<td>-</td>
<td>85.8</td>
</tr>
<tr>
<td>R.M. Reader (Ensemble)</td>
<td>81.2</td>
<td>87.9</td>
<td>82.3</td>
<td>88.5</td>
</tr>
</tbody>
</table>

### Ours

<table>
<thead>
<tr>
<th>System</th>
<th>Dev EM</th>
<th>Dev F1</th>
<th>Test EM</th>
<th>Test F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>BERT_BASE (Single)</td>
<td>80.8</td>
<td>88.5</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>BERT_LARGE (Single)</td>
<td>84.1</td>
<td>90.9</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>BERT_LARGE (Ensemble)</td>
<td>85.8</td>
<td>91.8</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>BERT_LARGE (Sgl.+TriviaQA)</td>
<td><strong>84.2</strong></td>
<td><strong>91.1</strong></td>
<td><strong>85.1</strong></td>
<td><strong>91.8</strong></td>
</tr>
<tr>
<td>BERT_LARGE (Ens.+TriviaQA)</td>
<td><strong>86.2</strong></td>
<td><strong>92.2</strong></td>
<td><strong>87.4</strong></td>
<td><strong>93.2</strong></td>
</tr>
</tbody>
</table>

### SQuAD v2.0.

<table>
<thead>
<tr>
<th>System</th>
<th>Dev EM</th>
<th>Dev F1</th>
<th>Test EM</th>
<th>Test F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Human</td>
<td>86.3</td>
<td>89.0</td>
<td>86.9</td>
<td>89.5</td>
</tr>
<tr>
<td>#1 Single - MIR-MRC (F-Net)</td>
<td>-</td>
<td>-</td>
<td>74.8</td>
<td>78.0</td>
</tr>
<tr>
<td>#2 Single - nlnet</td>
<td>-</td>
<td>-</td>
<td>74.2</td>
<td>77.1</td>
</tr>
</tbody>
</table>

### Published

<table>
<thead>
<tr>
<th>System</th>
<th>Dev EM</th>
<th>Dev F1</th>
<th>Test EM</th>
<th>Test F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>unet (Ensemble)</td>
<td>-</td>
<td>-</td>
<td>71.4</td>
<td>74.9</td>
</tr>
<tr>
<td>SLQA+ (Single)</td>
<td>-</td>
<td>-</td>
<td>71.4</td>
<td>74.4</td>
</tr>
</tbody>
</table>

### Ours

<table>
<thead>
<tr>
<th>System</th>
<th>Dev EM</th>
<th>Dev F1</th>
<th>Test EM</th>
<th>Test F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>BERT_LARGE (Single)</td>
<td><strong>78.7</strong></td>
<td><strong>81.9</strong></td>
<td><strong>80.0</strong></td>
<td><strong>83.1</strong></td>
</tr>
</tbody>
</table>
Problems with BERT

- Discrepancy between pretraining and fine-tuning due to [MASK] tokens

- Assumes the predicted tokens (i.e., the masked ones) are independent of each other given the unmasked tokens (i.e., able to model the joint probability using the product rule due to the avoidance of recurrence)

- The training is done over separated fixed length segments of the input text
  - Cannot capture the longer-term dependency beyond the predefined context length
  - The fixed-length segments are created by selecting a consecutive chunk of tokens/symbols without respecting the sentence or any other semantic boundary (context fragmentation).
Transformer-XL (extra long)

- Introduce the segment-level recurrence into Transformer.
- During training, the hidden state sequence computed for the previous segment is fixed and cached to be reused as an extended context for the next segment computation.
- Let: \( s_\tau = [x_{\tau,1}, \cdots, x_{\tau,L}] \) and \( s_{\tau+1} = [x_{\tau+1,1}, \cdots, x_{\tau+1,L}] \) be the two consecutive segments in training, and \( h^n_\tau \in \mathbb{R}^{L \times d} \) be the hidden state sequence of the n-layer of the model for the \( \tau \) segment \( s_\tau \), then:

\[
\begin{align*}
\tilde{h}^{n-1}_{\tau+1} &= [\text{SG}(h^{n-1}_\tau) \circ h^{n-1}_{\tau+1}], \\
q^n_{\tau+1}, k^n_{\tau+1}, v^n_{\tau+1} &= h^{n-1}_{\tau+1} W^q, \tilde{h}^{n-1}_{\tau+1} W^k, \tilde{h}^{n-1}_{\tau+1} W^v, \\
h^n_{\tau+1} &= \text{Transformer-Layer} (q^n_{\tau+1}, k^n_{\tau+1}, v^n_{\tau+1}).
\end{align*}
\]

where SG stands for stop-gradient and \([h_u \circ h_v]\) is the concatenation operation.
Transformer-XL: segment-level recurrence

• So, different from Transformer, the key and value vectors are also conditioned on the extended context cached from the previous segment.

• This creates the *segment-level recurrence* that allows the effective context to go way beyond just two segments (analogous to truncated BPTT for RNN language models).

• During the inference time, the representations from the previous segments can be reused, instead of being computed from scratch.
Transformer-XL: relative position encodings

• With the standard absolute position encodings: $h_{\tau+1} = f(h_{\tau}, E_{s_{\tau+1}} + U_{1:L})$
  $h_{\tau} = f(h_{\tau-1}, E_{s_{\tau}} + U_{1:L})$
  so no positional difference between $x_{\tau,j}$ and $x_{\tau+1,j}$.

• Instead of using absolute position encodings, use the relative position encodings to inject the temporal bias into the attention scores of the layers (i.e., the distance $i - j$ between the $j$-th key vector $k_{\tau,j}$ and the $i$-th query vector $q_{\tau,i}$).

$$A_{i,j}^{\text{abs}} = \begin{pmatrix} E_{x_{i}}^{T} W_{k} E_{x_{j}}^{T} + & E_{x_{i}}^{T} W_{q} W_{k} U_{j} \\ (a) \\ + U_{i}^{T} W_{q} W_{k} E_{x_{j}}^{T} + & U_{i}^{T} W_{q} W_{k} U_{j} \\ (c) \end{pmatrix} + \begin{pmatrix} E_{x_{i}}^{T} W_{k} E_{x_{j}}^{T} + & E_{x_{i}}^{T} W_{q} W_{k} R_{i-j} \\ (b) \\ + u^{T} W_{k} E_{x_{j}}^{T} + & v^{T} W_{k} R_{i-j} \\ (d) \end{pmatrix}$$

$E_{*}$ is word embedding vector, $U_{*}$ and $R_{*}$ are absolution and relative position embeddings (respectively), and $u$ and $v$ are learnable vectors.

Transformer-XL evaluation on language modeling datasets

<table>
<thead>
<tr>
<th>Model</th>
<th>#Param</th>
<th>PPL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grave et al. (2016b) - LSTM</td>
<td>-</td>
<td>48.7</td>
</tr>
<tr>
<td>Bai et al. (2018) - TCN</td>
<td>-</td>
<td>45.2</td>
</tr>
<tr>
<td>Dauphin et al. (2016) - GCNN-8</td>
<td>-</td>
<td>44.9</td>
</tr>
<tr>
<td>Grave et al. (2016b) - LSTM + Neural cache</td>
<td>-</td>
<td>40.8</td>
</tr>
<tr>
<td>Dauphin et al. (2016) - GCNN-14</td>
<td>-</td>
<td>37.2</td>
</tr>
<tr>
<td>Merity et al. (2018) - QRNN</td>
<td>151M</td>
<td>33.0</td>
</tr>
<tr>
<td>Rae et al. (2018) - Hebbian + Cache</td>
<td>-</td>
<td>29.9</td>
</tr>
<tr>
<td>Ours - Transformer-XL Standard</td>
<td>151M</td>
<td>24.0</td>
</tr>
<tr>
<td>Ours - Transformer-XL Large</td>
<td>257M</td>
<td>18.3</td>
</tr>
</tbody>
</table>

Table 1: Comparison with state-of-the-art results on WikiText-103. ◦ indicates contemporary work.

<table>
<thead>
<tr>
<th>Model</th>
<th>#Param</th>
<th>PPL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shazeer et al. (2014) - Sparse Non-Negative</td>
<td>33B</td>
<td>52.9</td>
</tr>
<tr>
<td>Chelba et al. (2013) - RNN-1024 + 9 Gram</td>
<td>20B</td>
<td>51.3</td>
</tr>
<tr>
<td>Kuchaiev and Ginsburg (2017) - G-LSTM-2</td>
<td>-</td>
<td>36.0</td>
</tr>
<tr>
<td>Dauphin et al. (2016) - GCNN-14 bottleneck</td>
<td>-</td>
<td>31.9</td>
</tr>
<tr>
<td>Jozefowicz et al. (2016) - LSTM</td>
<td>1.8B</td>
<td>30.6</td>
</tr>
<tr>
<td>Jozefowicz et al. (2016) - LSTM + CNN Input</td>
<td>1.04B</td>
<td>30.0</td>
</tr>
<tr>
<td>Shazeer et al. (2017) - Low-Budget MoE</td>
<td>~5B</td>
<td>34.1</td>
</tr>
<tr>
<td>Shazeer et al. (2017) - High-Budget MoE</td>
<td>~5B</td>
<td>28.0</td>
</tr>
<tr>
<td>Shazeer et al. (2018) - Mesh Tensorflow</td>
<td>4.9B</td>
<td>24.0</td>
</tr>
<tr>
<td>Baevski and Auli (2018) - Adaptive Input</td>
<td>0.46B</td>
<td>24.1</td>
</tr>
<tr>
<td>Baevski and Auli (2018) - Adaptive Input ◦</td>
<td>1.0B</td>
<td>23.7</td>
</tr>
<tr>
<td>Ours - Transformer-XL Base</td>
<td>0.46B</td>
<td>23.5</td>
</tr>
<tr>
<td>Ours - Transformer-XL Large</td>
<td>0.8B</td>
<td>21.8</td>
</tr>
</tbody>
</table>

Table 4: Comparison with state-of-the-art results on One Billion Word. ◦ indicates contemporary work.
XLNet (Yang et al. 2019)

- Use Transformer-XL as the network architecture.
- Inheriting the bidirectional context modeling from BERT while addressing its masked token independency assumption and pretraining-finetuning discrepancy, XLNet performs permutation language modeling:
  - Tokens are indexed from 1 to $T$
  - $Z_T$ is the set of all possible permutations of $[1, 2, \ldots, T]$
    
    $z = [z_1, z_2, \ldots, z_t, \ldots, z_T] \in Z_T$
    
    $z_{<t} = [z_1, z_2, \ldots, z_{t-1}]$
  - The loss function:
    
    $$\max_{\theta} E_{z \sim Z_T} \left[ \sum_{t=1}^{T} \log p_{\theta}(x_{z_t} \mid x_{z_{<t}}) \right]$$
  - Only permute the factorization order, not the sequence order: keep the original order, use the positional encodings corresponding to the original sequence, and rely on a proper attention mask in Transformers to achieve permutation of the factorization order.
XLNet: Partial Prediction

- Permutation causes slow convergence in the preliminary experiments.
- So, only predict the last tokens in the factorization order to reduce the optimization difficulty (as the last tokens can assume the longest context in the sequence given the current factorization order)
  - Split the sequence into a non-target subsequence and target subsequence.

\[
\max_{\theta} \mathbb{E}_{z \sim Z_T} \left[ \log p_{\theta}(x_{z>c} | x_{z \leq c}) \right] = \mathbb{E}_{z \sim Z_T} \left[ \sum_{t=c+1}^{|z|} \log p_{\theta}(x_t | x_{z<t}) \right]
\]

* \( c \) is chosen such that \( |z| / (|z| - c) \approx K \)
XLNet: Two-Stream Self-Attention

• In the naïve implementation, the next-token distribution would be:

\[
p_\theta(X_{z|t} = x \mid x_{z<t}) = \frac{\exp(e(x)^\top h_\theta(x_{z<t}))}{\sum_{x'} \exp(e(x')^\top h_\theta(x_{z<t}))}
\]

• This does not depend on which position it will predict, i.e., \(z_t\)

• So, we want to make this distribution to be target position aware:

\[
p_\theta(X_{z|t} = x \mid x_{z<t}) = \frac{\exp(e(x)^\top g_\theta(x_{z<t}, z_t))}{\sum_{x'} \exp(e(x')^\top g_\theta(x_{z<t}, z_t))}
\]

• Idea: using two sets of hidden representations:
  – The context representations \(h_\theta(x_{z|t})\) to encode both the context and \(x_{z|t}\) itself.
  – The query representations \(g_\theta(x_{z|t}, z_t)\) to only encode the contextual information \(x_{z|t}\) and the position \(z_t\).

Modeling this is non-trivial, i.e., need to handle the cases for \(j = t\) and \(j > t\) differently.
XLNet: Two-Stream Self-Attention

The content stream:

\[ h_{zt}^{(m)} \leftarrow \text{Attention}(Q = h_{zt}^{(m-1)}, KV = h_{z \leq t}^{(m-1)}; \theta) \]

This is also integrated with the ideas of segment level recurrence and relative position encodings from Transformer-XL:

\[ h_{zt}^{(m)} \leftarrow \text{Attention}(Q = h_{zt}^{(m-1)}, KV = [\tilde{h}^{(m-1)}, h_{z \leq t}^{(m-1)}]; \theta) \]

The query stream:

\[ g_{zt}^{(m)} \leftarrow \text{Attention}(Q = g_{zt}^{(m-1)}, KV = h_{z \leq t}^{(m-1)}; \theta) \]
XLNet: Two-Stream Self-Attention

+ $g_{zt}^{(M)}$ is used for prediction during pre-training.

$$p_\theta(X_{zt} = x \mid x_{z < t}) = \frac{\exp \left( e(x)^\top g_\theta(x_{z < t}, z_t) \right)}{\sum_{x'} \exp \left( e(x')^\top g_\theta(x_{z < t}, z_t) \right)}$$

+ $h_{zt}^{(M)}$ is the contextualized embedding used for fine-tuning.
XLNet: fine-tuning

• Follow BERT to fine-tune for downstream tasks
• For token level tasks: the same
• For sentence level tasks:
  – also use the [CLS] and [SEQ] tokens as BERT
  – recurrence connection over sentences (segments)
  – Relative segment encoding:
    • $a_{i,j} = (q_i + b)^T s_{i,j}$ is added to:
      $$ A_{i,j}^{rel} = \underbrace{E_{x_i}^T W_k^T W_{q,E} E_{x_j}}_{(a)} + \underbrace{E_{x_i}^T W_q^T W_{k,R} R_{i-j}}_{(b)} $$
      $$ + \underbrace{u^T W_{k,E} E_{x_j}}_{(c)} + \underbrace{v^T W_{k,R} R_{i-j}}_{(d)} $$
    • $s_{i,j} = s_+$ if positions $i$ and $j$ belong to the same sentence; and $s_{i,j} = s_-$ otherwise.

→ XLNet can directly model input with more than two sentences.
XLNet: Evaluation

- The SQuAD question answering datasets

<table>
<thead>
<tr>
<th>SQuAD1.1</th>
<th>EM</th>
<th>F1</th>
<th>SQuAD2.0</th>
<th>EM</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Dev set results without data augmentation</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BERT [10]</td>
<td>84.1</td>
<td>90.9</td>
<td>BERT† [10]</td>
<td>78.98</td>
<td>81.77</td>
</tr>
<tr>
<td>XLNet</td>
<td><strong>88.95</strong></td>
<td><strong>94.52</strong></td>
<td>XLNet</td>
<td><strong>86.12</strong></td>
<td><strong>88.79</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Test set results on leaderboard, with data augmentation (as of June 19, 2019)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Human [27]</td>
<td>82.30</td>
<td>91.22</td>
<td>BERT+N-Gram+Self-Training [10]</td>
<td>85.15</td>
<td>87.72</td>
</tr>
<tr>
<td>ATB</td>
<td>86.94</td>
<td>92.64</td>
<td>SG-Net</td>
<td>85.23</td>
<td>87.93</td>
</tr>
<tr>
<td>BERT* [10]</td>
<td>87.43</td>
<td>93.16</td>
<td>BERT+DAE+AoA</td>
<td>85.88</td>
<td>88.62</td>
</tr>
<tr>
<td>XLNet</td>
<td><strong>89.90</strong></td>
<td><strong>95.08</strong></td>
<td>XLNet</td>
<td><strong>86.35</strong></td>
<td><strong>89.13</strong></td>
</tr>
</tbody>
</table>
### XLNet: Evaluation

- The GLUE benchmark

<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>WNLI</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Single-task single models on dev</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BERT [2]</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>90.0</td>
<td>-</td>
</tr>
<tr>
<td>XLNet</td>
<td><strong>89.8/-</strong></td>
<td><strong>93.9</strong></td>
<td><strong>91.8</strong></td>
<td><strong>83.8</strong></td>
<td><strong>95.6</strong></td>
<td><strong>89.2</strong></td>
<td><strong>63.6</strong></td>
<td><strong>91.8</strong></td>
<td>-</td>
</tr>
<tr>
<td><strong>Single-task single models on test</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BERT [10]</td>
<td>86.7/85.9</td>
<td>91.1</td>
<td>89.3</td>
<td>70.1</td>
<td>94.9</td>
<td>89.3</td>
<td>60.5</td>
<td>87.6</td>
<td>65.1</td>
</tr>
<tr>
<td><strong>Multi-task ensembles on test (from leaderboard as of June 19, 2019)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Snorkel* [29]</td>
<td>87.6/87.2</td>
<td>93.9</td>
<td>89.9</td>
<td>80.9</td>
<td>96.2</td>
<td>91.5</td>
<td>63.8</td>
<td>90.1</td>
<td>65.1</td>
</tr>
<tr>
<td>ALICE*</td>
<td>88.2/87.9</td>
<td>95.7</td>
<td><strong>90.7</strong></td>
<td>83.5</td>
<td>95.2</td>
<td>92.6</td>
<td><strong>68.6</strong></td>
<td>91.1</td>
<td>80.8</td>
</tr>
<tr>
<td>MT-DNN* [18]</td>
<td>87.9/87.4</td>
<td>96.0</td>
<td>89.9</td>
<td><strong>86.3</strong></td>
<td>96.5</td>
<td>92.7</td>
<td>68.4</td>
<td>91.1</td>
<td>89.0</td>
</tr>
<tr>
<td>XLNet*</td>
<td><strong>90.2/89.7</strong></td>
<td><strong>98.6</strong></td>
<td><strong>90.3</strong></td>
<td><strong>86.3</strong></td>
<td><strong>96.8</strong></td>
<td><strong>93.0</strong></td>
<td>67.8</td>
<td><strong>91.6</strong></td>
<td><strong>90.4</strong></td>
</tr>
</tbody>
</table>
## Datasets and Resources

<table>
<thead>
<tr>
<th>Model</th>
<th>Pre-trained datasets</th>
</tr>
</thead>
<tbody>
<tr>
<td>ELMo</td>
<td>- One Billion Word Benchmark</td>
</tr>
<tr>
<td>GPT</td>
<td>- BooksCorpus (800M words)</td>
</tr>
</tbody>
</table>
| BERT   | - BooksCorpus (800M words)  
          - English Wikipedia (2,500M words)  
          (13GB text in total) |
| XLNet  | - BooksCorpus (800M words)  
          - English Wikipedia (2,500M words)  
          - Giga5 (16GB text)  
          - ClueWeb 2012-B (19GB text)  
          - Common Crawl (78GB text) |

More on costs to train these models (about $245,000 for XLNet!):  
Hugging Face – PyTorch Transformers

• Hugging Face (https://huggingface.co/) implements most of the well-known transformers.

• Pretrained model of BERT, GPT, XLNet, ... are ready to be fine-tuned on downstream tasks and available at:

https://github.com/huggingface/pytorch-transformers