Contextualized Word Embeddings

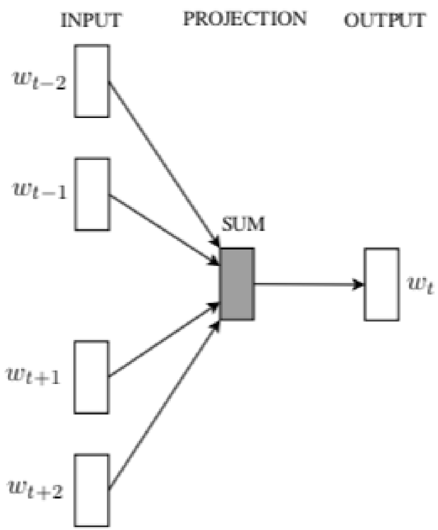
Bonan Min

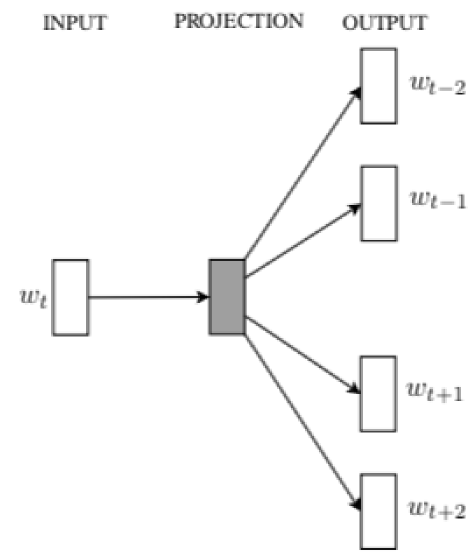
bonanmin@gmail.com

Some slides are based on class materials from Thien Huu Nguyen, Alexander Rush

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: Word2vec

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

$$\text{Likelihood} = L(\theta) = \prod_{i=1}^{N} \prod_{\substack{-m \le j \le m \\ j \neq 0}} P(w_{i+j} | w_i; \theta)$$

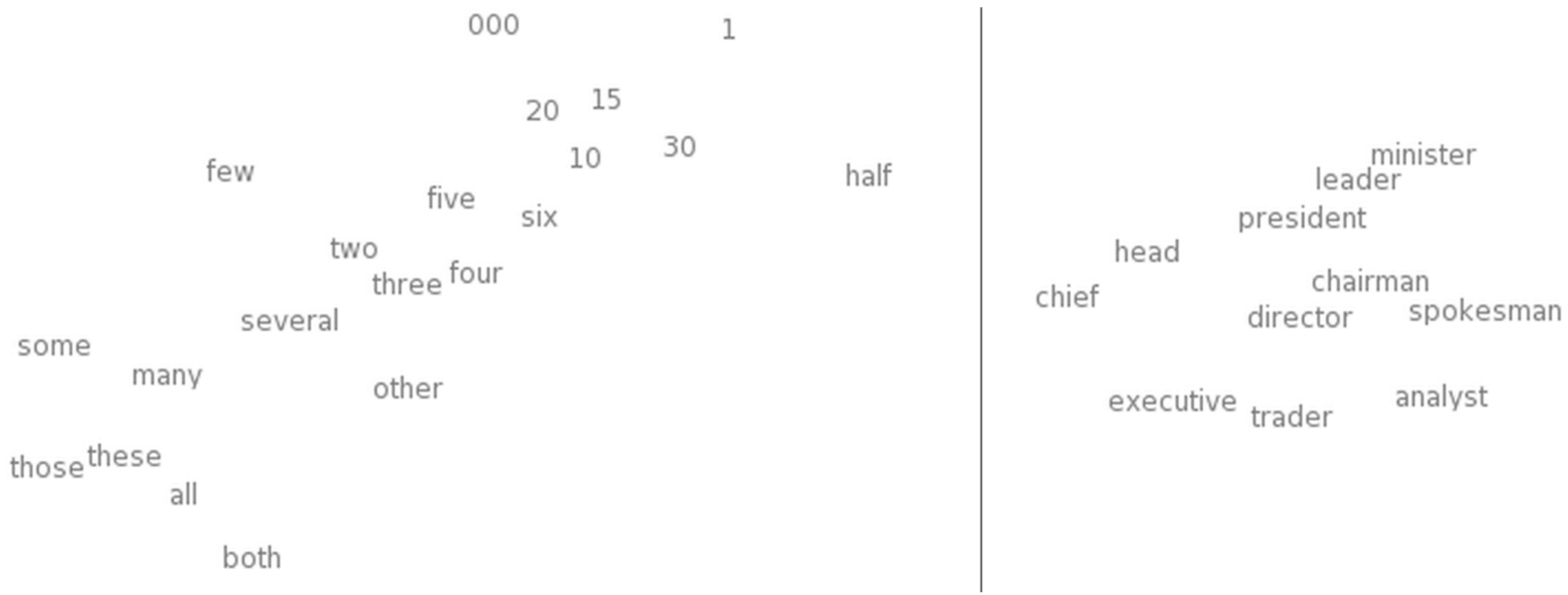
The objective/loss function is the (average) negative log likelihood:

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

The parameters θ here involve the word vectors we want to learn

Minimizing the loss function amounts to maximizing the predictive accuracy

Recap: Wovd2vec



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

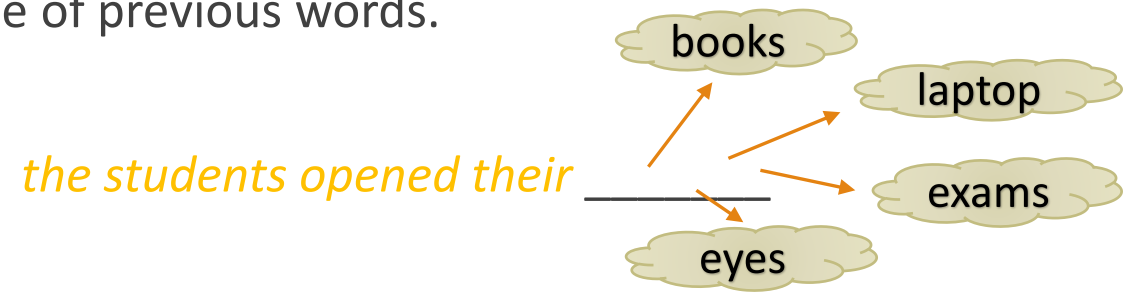
Questions: How do we train such models? How do we obtain the training data?

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.



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

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

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

$h_i = \sigma(\boldsymbol{W}_h h_{i-1} + \boldsymbol{W}_e e_i + b_1)$

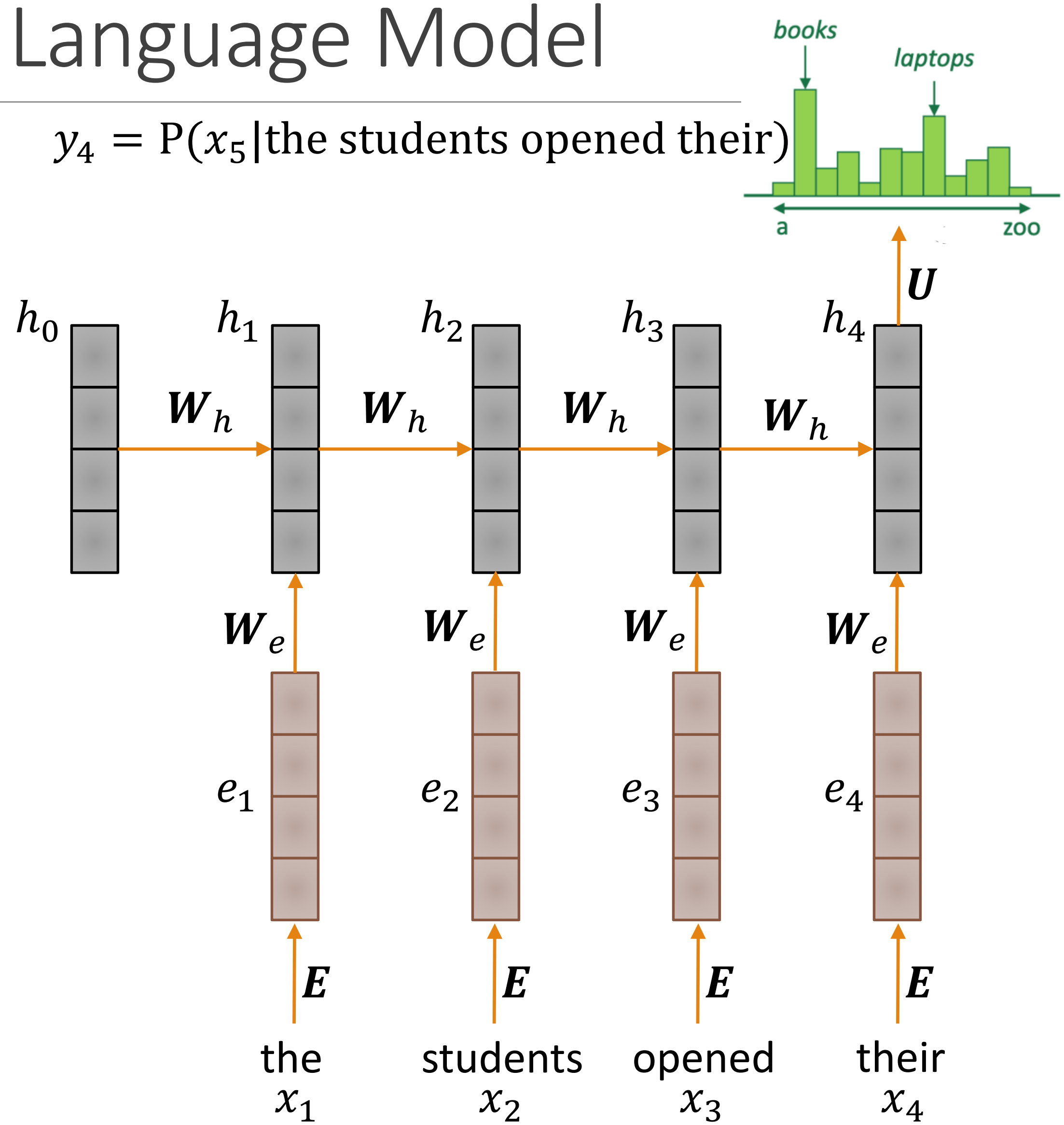
h_0 is the initial hidden state

using LSTM or GRU is more common

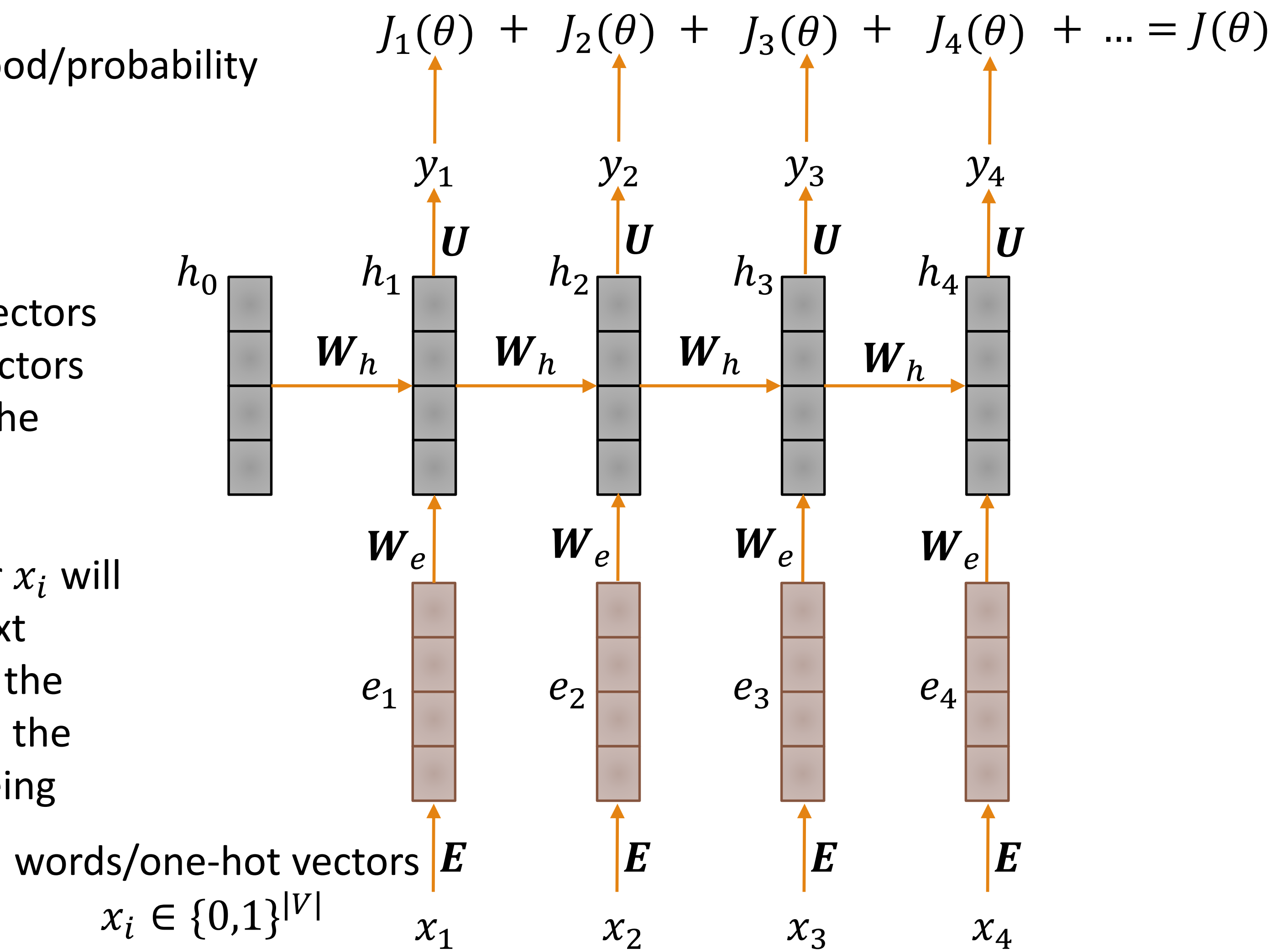
word embeddings

words/one-hot vectors

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



Contextualized Word Embeddings With The RNN Language Model



- 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

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 $\boldsymbol{E}$ 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 loose some useful information from the form of the word (i.e., morphology).

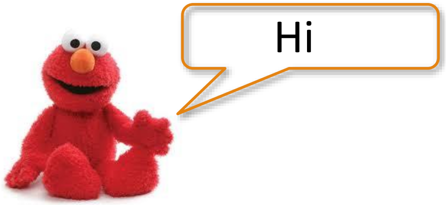
A single RNN layer might not be sufficient to capture the underlying context information for the input sentences. How's about making it deeper (i.e., more layers)?

Deep Contextualized Word Embeddings

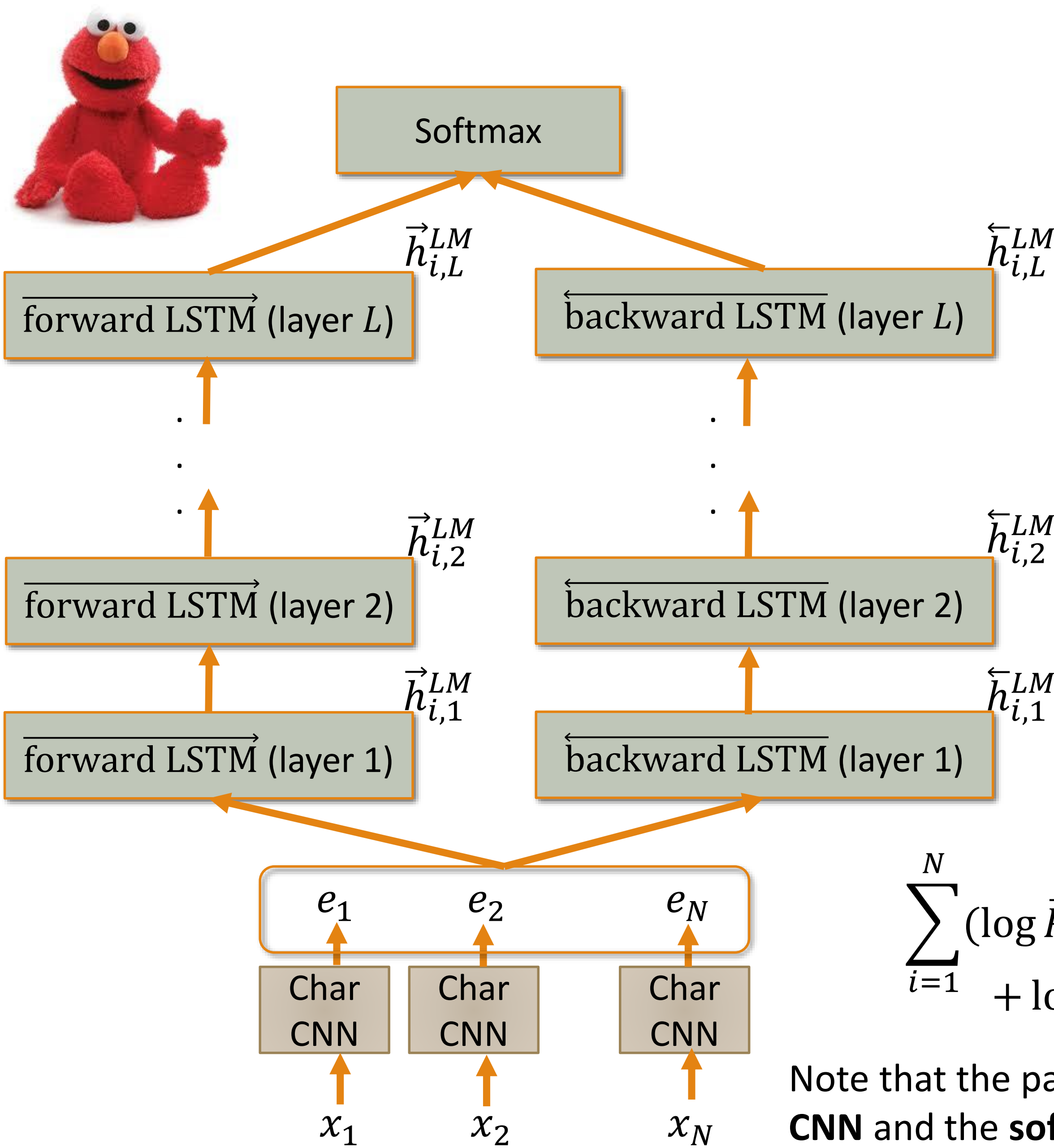
ELMo (Embeddings from Language Models) is introduced in (Peters et al., 2018).

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



ELMo



$\vec{h}_{i,L}^{LM}$ is used to predict the next token x_{i+1}

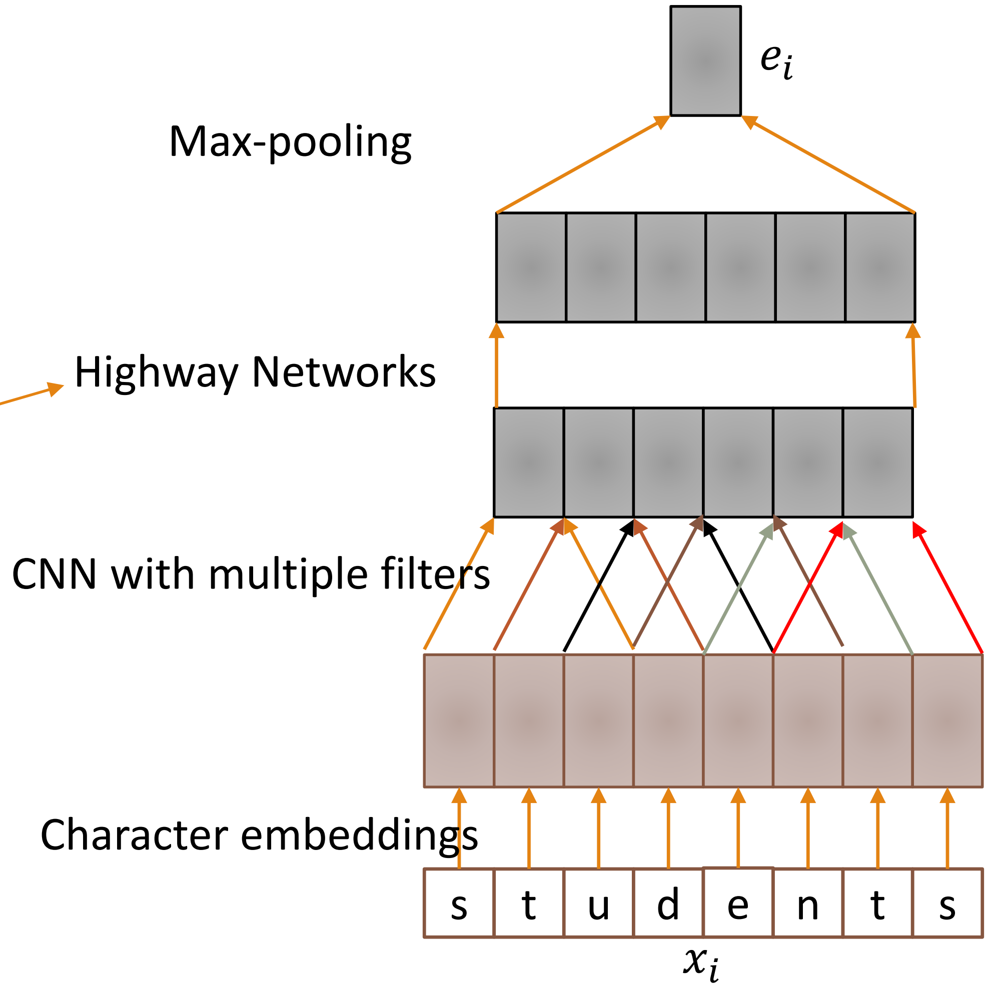
$\overleftarrow{h}_{i,L}^{LM}$ is used to predict the previous token x_{i-1}

We learn the model parameters by jointly optimizing the forward and backward language model objectives:

$$\sum_{i=1}^{N} \left(\log \vec{P}\left(x_{i+1} | x_i, \dots, x_1; \theta_{cnn}, \vec{\theta}_{lstm}, \theta_s\right) + \log \overleftarrow{P}\left(x_{i-1} | x_i, \dots, x_N; \theta_{cnn}, \overleftarrow{\theta}_{lstm}, \theta_s\right)\right)$$

Note that the parameters for the **character-based CNN** and the **softmax layer** for distributions are **shared**

ELMo: The character CNN



Inspired by the gating mechanism in Long Short Term Memory (LSTM) RNN, **Highway Networks** (Srivastava et al., 2015) allows information to either being *transformed* (as usual DNN does) or *carried* through in its layers, so that information flow across layers becomes much easier.

Allows very deep NN to be trained with simple SGD.

Fine-tuning with ELMo

The representation vector for input token i is now:

$$R_k = \left\{e_i, \overrightarrow{h}_{i,j}^{LM}, \overleftarrow{h}_{i,j}^{LM} \middle| j = 1, \dots, L\right\}$$
$$= \{h_{i,j} | j = 0, \dots, L\}$$

with $h_{i,0} = e_i$ and $h_{i,j} = [\overrightarrow{h}_{i,j}^{LM}, \overleftarrow{h}_{i,j}^{LM}]$ 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 inpu 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

Task	Previous SOTA		Our baseline	ELMo + baseline	Increase (absolute/relative)
SQuAD	Liu et al. (2017)	84.4	81.1	85.8	4.7 / 24.9%
SNLI	Chen et al. (2017)	88.6	88.0	88.7 ± 0.17	0.7 / 5.8%
SRL	He et al. (2017)	81.7	81.4	84.6	3.2 / 17.2%
Coref	Lee et al. (2017)	67.2	67.2	70.4	3.2 / 9.8%
NER	Peters et al. (2017)	91.93 ± 0.19	90.15	92.22 ± 0.10	2.06 / 21%
SST-5	McCann et al. (2017)	53.7	51.4	54.7 ± 0.5	3.3 / 6.8%

Table 1 from the original paper.

Problem with ELMo

Problems

- The recurrent computation (sequential) 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

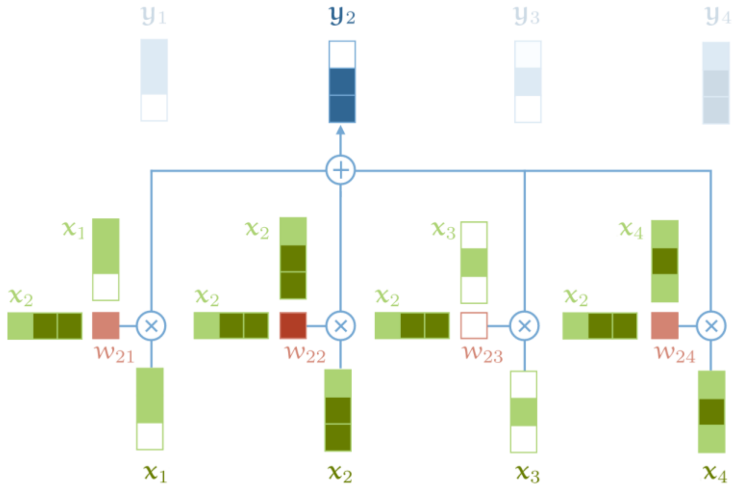
Similar to RNN, self-attention is a sequence-to-sequence operation: a sequence of vector comes in and a sequence of vectors goes out.

Vanilla self-attention: let $x_1, \ldots, x_n$ be the input vector sequence. The output vector sequence $y_1, \ldots, y_n$ is computed by:

$$\mathbf{y}_i = \sum_j w_{ij} \mathbf{x}_j$$

$$w'_{ij} = \mathbf{x}_i^{\mathsf{T}} \mathbf{x}_j$$

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



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**:

$$\mathbf{q}_i = \boldsymbol{W}_q \boldsymbol{x}_i \qquad \mathbf{k}_i = \boldsymbol{W}_k \boldsymbol{x}_i \qquad \boldsymbol{v}_i = \boldsymbol{W}_v \boldsymbol{x}_i$$

$$w'_{ij} = \mathbf{q}_i^{\mathsf{T}} \mathbf{k}_j$$

query

key

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

$$\boldsymbol{y}_i = \sum_j w_{ij} \boldsymbol{v}_j \,.$$

value

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

$$w'_{ij} = \frac{\mathbf{q}_i^{\mathsf{T}} \mathbf{k}_j}{\sqrt{\mathrm{k}}}$$

This process is denoted by:

$$y_1, \dots, y_n = self\_attention(x_1, \dots, x_n; \boldsymbol{W}_q, \boldsymbol{W}_k, \boldsymbol{W}_v)$$

Multi-head Self-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 $\boldsymbol{W}_q^r, \boldsymbol{W}_k^r, \boldsymbol{W}_v^r$ (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.

$$y_1^r, \dots, y_n^r = self\_attention(x_1, \dots, x_n; \boldsymbol{W}_q^r, \boldsymbol{W}_k^r, \boldsymbol{W}_v^r)$$

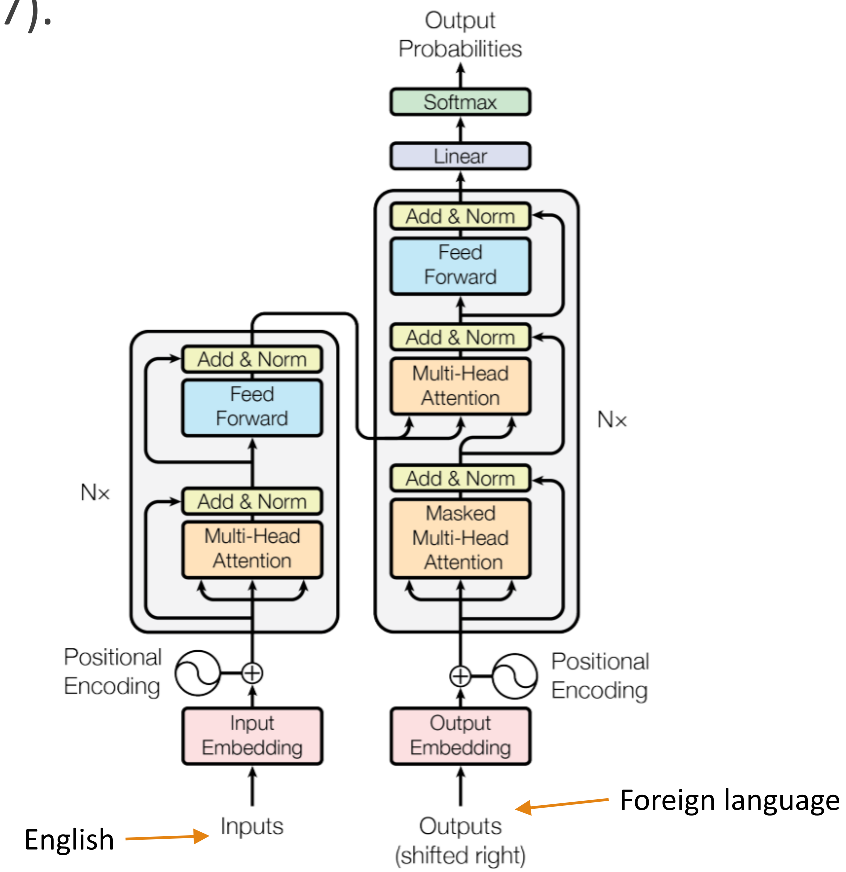
$$y_i = W\left[y_i^1, \dots, y_i^H\right] + b$$

$$|y_i| = |x_i| = k$$

where H is the number of attention heads

Transformer

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



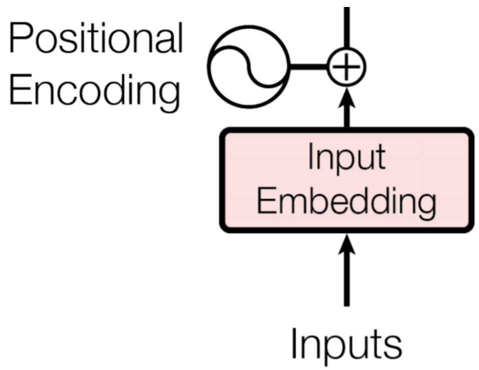
Vaswani et al. Attention Is All You Need. NIPS 2017.

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?

- 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).

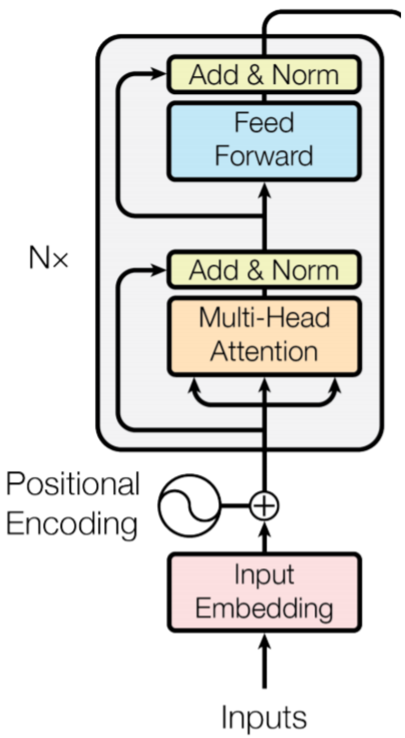


$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

$$pos \xrightarrow[d_{model} = 4]{} \left[sin\left(\frac{pos}{10000^0}\right), cos\left(\frac{pos}{10000^0}\right), sin\left(\frac{pos}{10000^{2/4}}\right), cos\left(\frac{pos}{10000^{2/4}}\right)\right]$$

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.

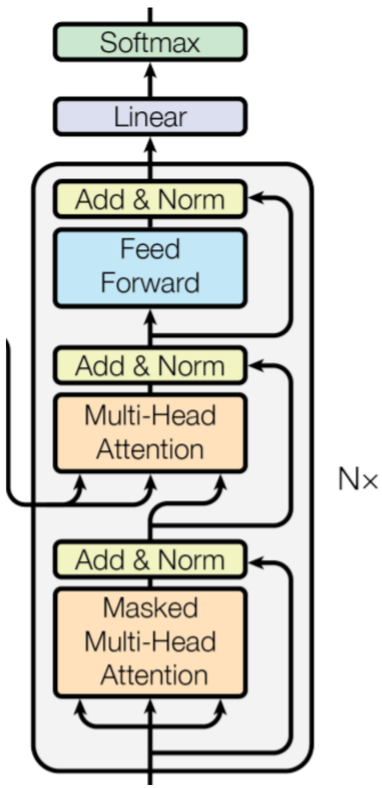



```
x_encoder = word_embeds + pos_embeds
for layer_i in range(N):
  q, k, v = x_encoder, x_encoder, x_encoder
  att = MultiHeadAttention(q, k, v)
  att = LayerNorm(x_encoder + Dropout(att))
  ffw = Feedfw(att)
  x_encoder = LayerNorm(att + dropout(ffw))
```

Vaswani et al. Attention Is All You Need. NIPS 2017.

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 depends on previous positions (for generate a translated sentence in Machine Translation).



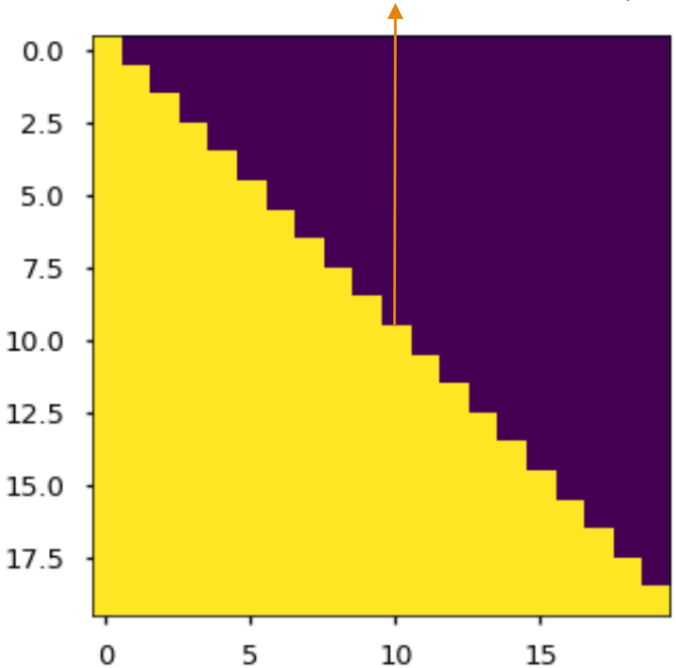

```
x_decoder = word_embeds + pos_embeds

for layer_i in range(N):
    q, k, v = x_decoder, x_decoder, x_decoder
    att = MaskedMultiHeadAttention(q, k, v)
    att = LayerNorm(x_decoder + Dropout(att))

    q = att
    k, v = x_encoder, x_encoder

    att = MultiheadAttention(q, k, v)
    ffw = Feedfw(att)
    x_decoder = LayerNorm(att + dropout(ffw))

logits = Feedfw(x_decoder)
preds = softmax(logits)
```

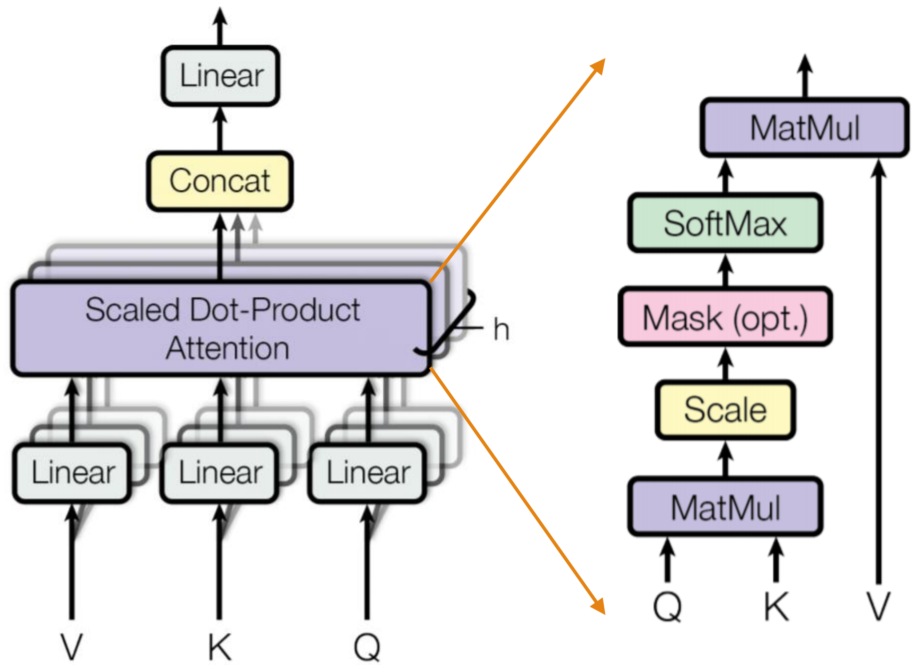
Vaswani et al. Attention Is All You Need. NIPS 2017.

Transformer: Decoder

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

Using the mask, word at position 10 is only allowed to attend to words at position 0-9





Masked Multi-Head Attention

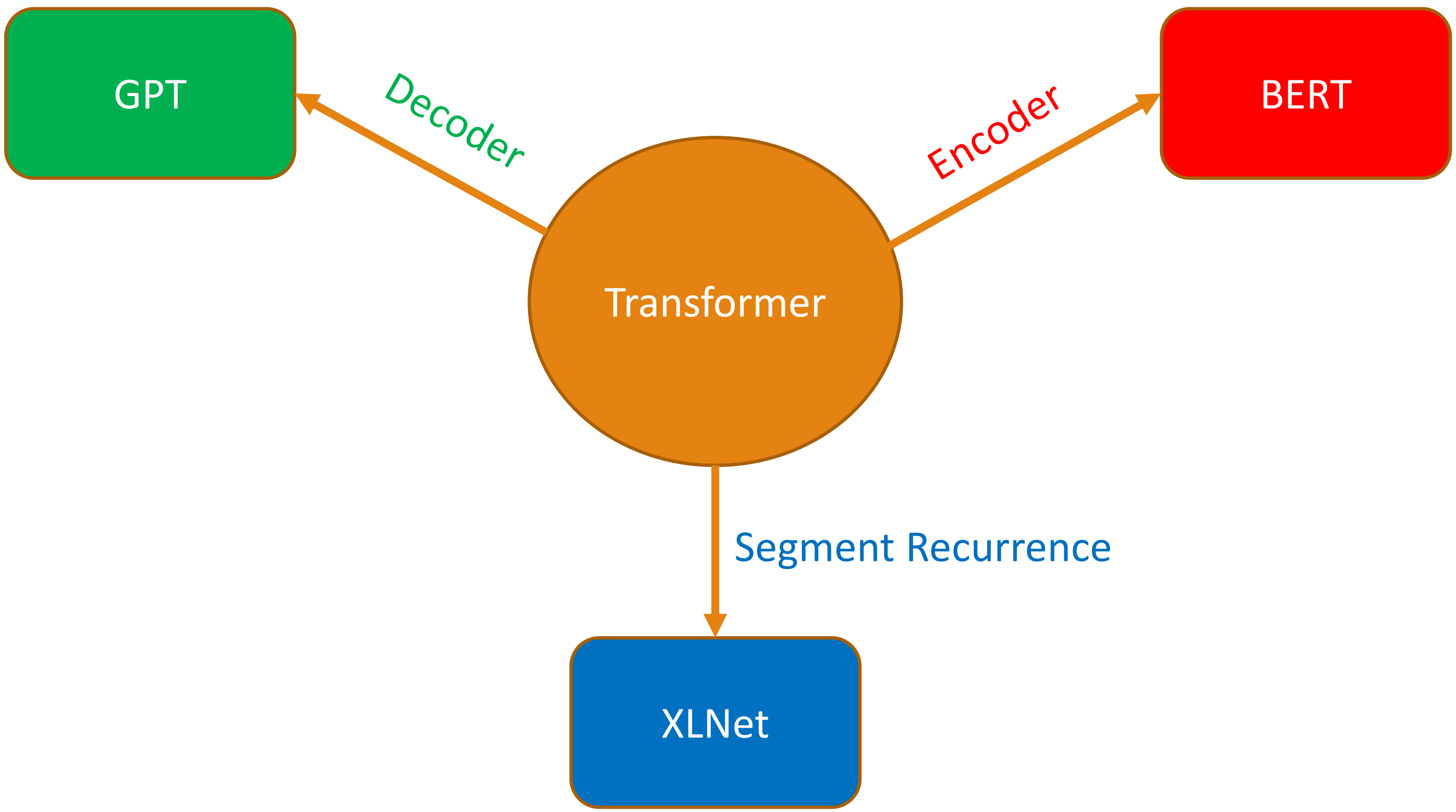
http://nlp.seas.harvard.edu/2018/04/03/attention.html

Transformer For Machine Translation

Evaluation

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$\mathbf{3.3 \cdot 10^{18}}$	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

Transformer For Contextualized Word Embeddings

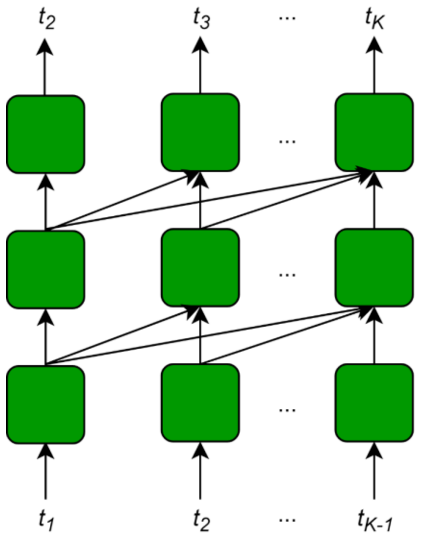


GPT

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:



Maximize:

$$L_{pretrain} = \sum_{i=1}^{K} \log(P(t_i | t_{i-1}, t_{i-2}, ..., t_1; \theta_{transformer}))$$

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

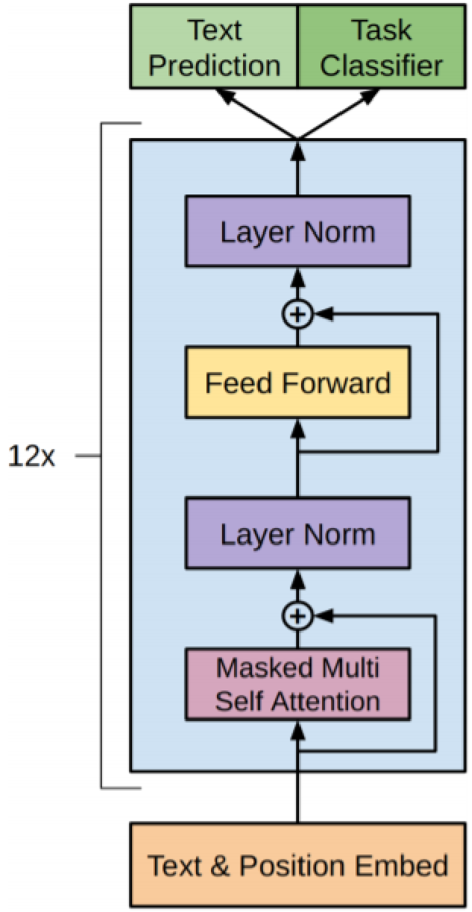
Use the last token's representation to make predictions

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

$$L = L_{target} + \lambda L_{pretrain}$$

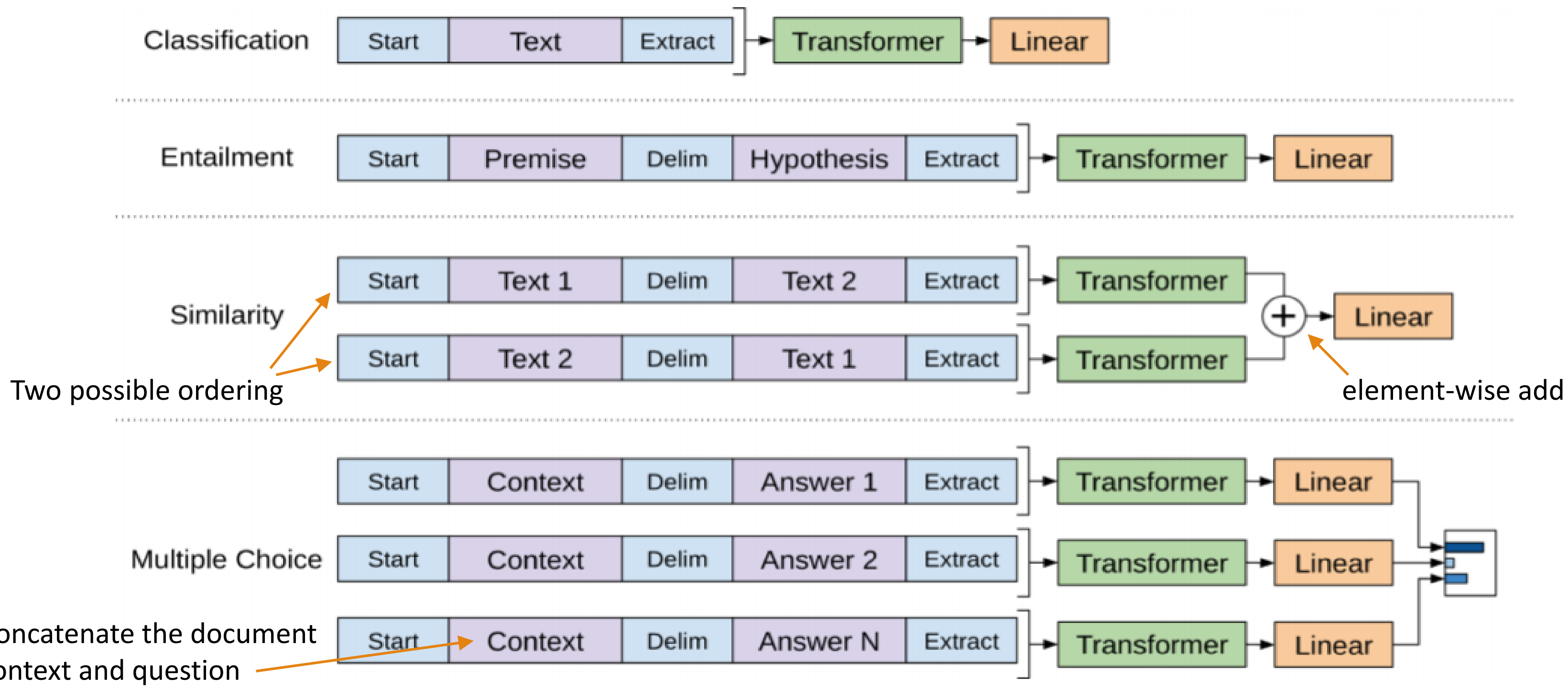


GPT: Input Transformation for Fine-tuning

Convert structured inputs into an ordered sequence that our pre-trained model can process.

- Since the pre-trained model was trained on contiguous sequences of text

These **input transformations** allow us to avoid making extensive changes to the architecture across tasks



GPT: Evaluation

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

GPT: Evaluation

((((yoav' ()ل)()ل)))
@yoavgo

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.

BERT

Let me take it, ELMo

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

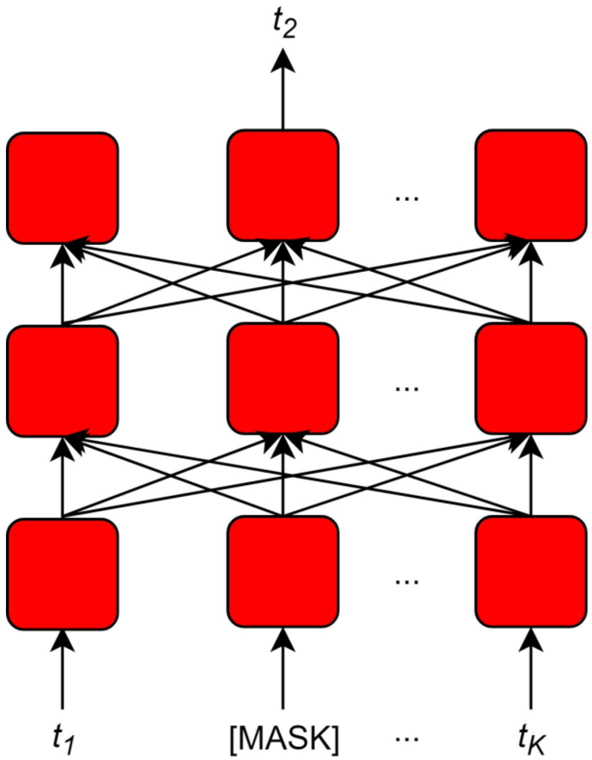
Use the encoder of Transformer to the network architecture

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 tokens.
  => force the model to collect bidirectional information to make true predictions.
- Training objective: recover the original tokens from the corrupted version:

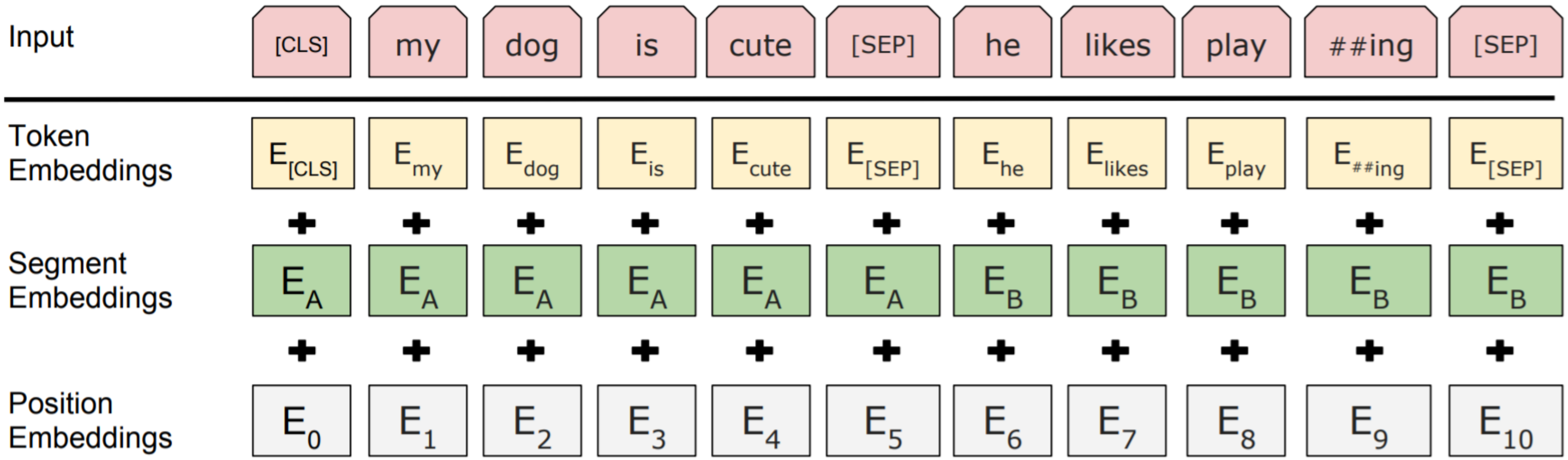
$$\sum_{i=1}^{K} m_i \log(P(t_i | t_1, ..., t_{i-1}, t_{i+1}, ..., t_K)$$

$m_i \in \{0, 1\}$ indicates whether t_i is masked or not.

BERT: Pre-training

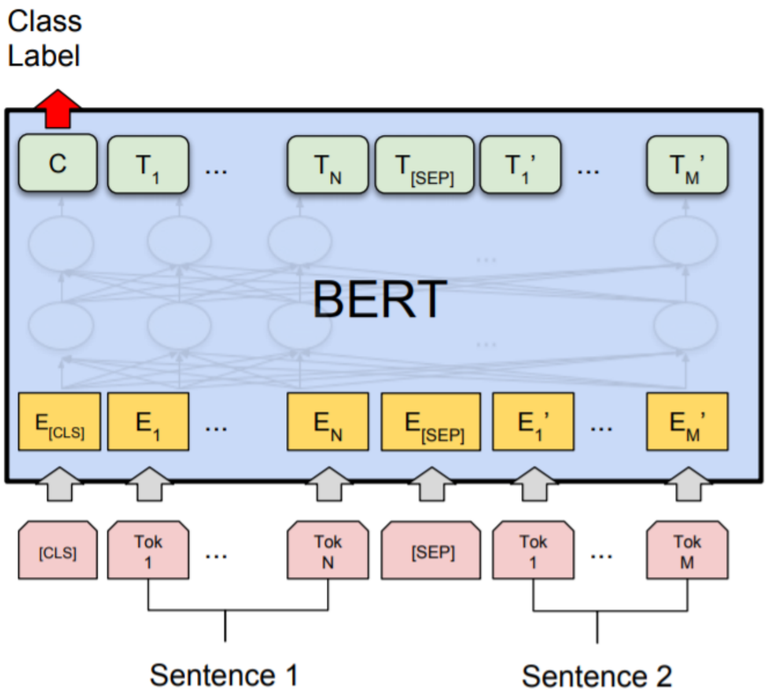
Next sentence prediction (binary classification) (done after the masked language model pre-training)

- 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)



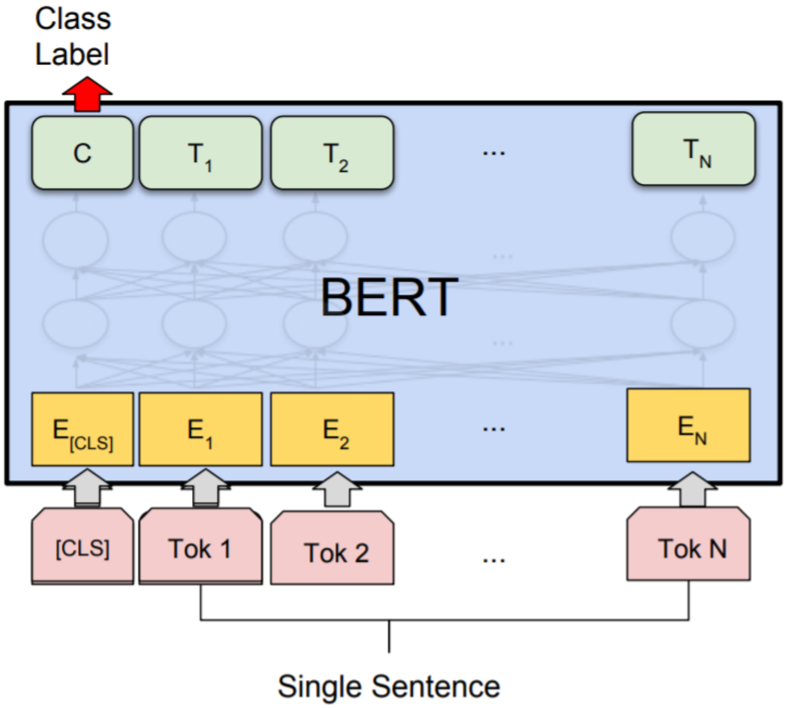
BERT: Fine-tuning For Sentence Pair Classification Tasks

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



$\Rightarrow$ 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, Ø> pair.
⇒ Use **[CLS]** for fine-tuning as usual.

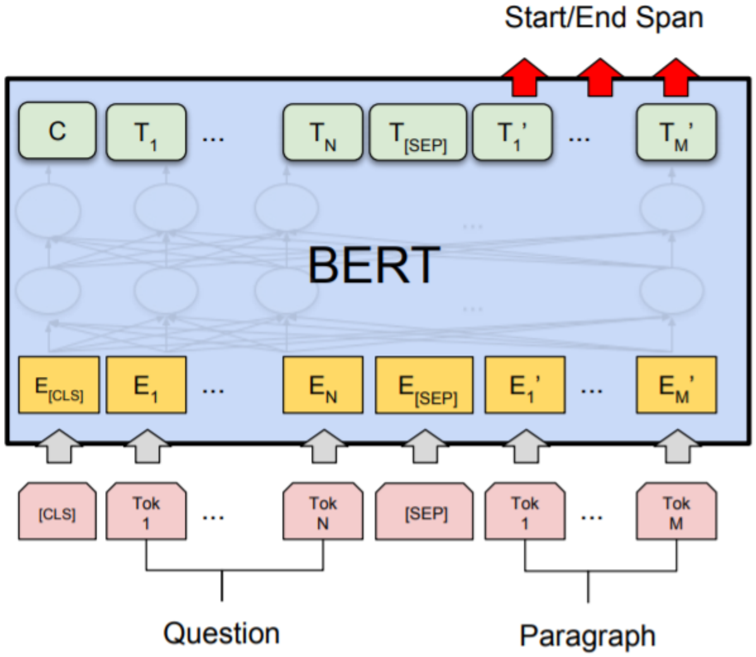
BERT: Evaluation

GLUE (General Language Understanding Evaluation) benchmark

System	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	**Average**
	392k	363k	108k	67k	8.5k	5.7k	3.5k	2.5k	-
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

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:

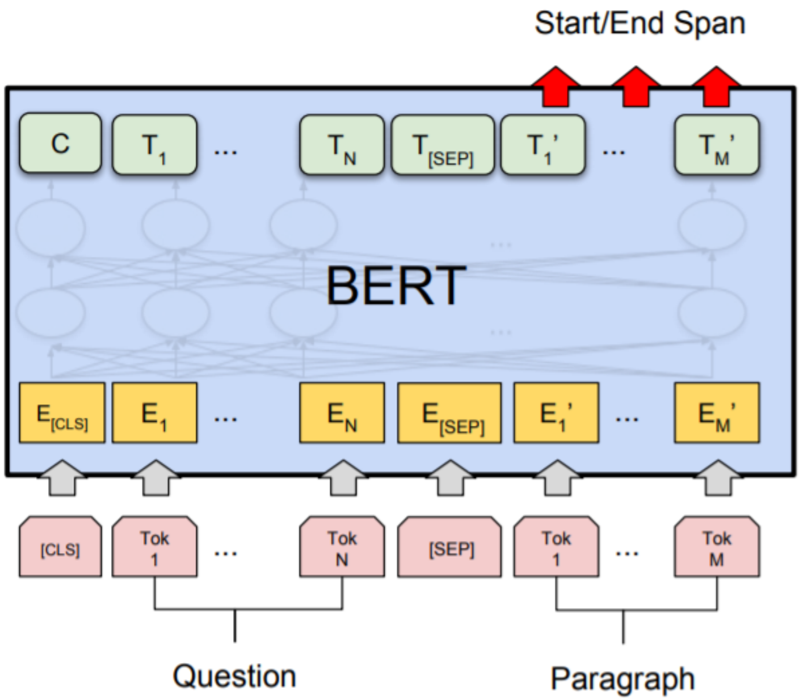
$$P(\text{end} = j) = \frac{e^{E \cdot T_j}}{\sum e^{E \cdot T_{j'}}}$$

- 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

System	Dev EM	Dev F1	Test EM	Test F1
Top Leaderboard Systems (Dec 10th, 2018)				
Human	-	-	82.3	91.2
#1 Ensemble - nlnet	-	-	86.0	91.7
#2 Ensemble - QANet	-	-	84.5	90.5
Published				
BiDAF+ELMo (Single)	-	85.6	-	85.8
R.M. Reader (Ensemble)	81.2	87.9	82.3	88.5
Ours				
$\text{BERT}_{\text{BASE}}$ (Single)	80.8	88.5	-	-
$\text{BERT}_{\text{LARGE}}$ (Single)	84.1	90.9	-	-
$\text{BERT}_{\text{LARGE}}$ (Ensemble)	85.8	91.8	-	-
$\text{BERT}_{\text{LARGE}}$ (Sgl.+TriviaQA)	**84.2**	**91.1**	**85.1**	**91.8**
$\text{BERT}_{\text{LARGE}}$ (Ens.+TriviaQA)	**86.2**	**92.2**	**87.4**	**93.2**

SQuAD v1.1.

System	Dev EM	Dev F1	Test EM	Test F1
Top Leaderboard Systems (Dec 10th, 2018)				
Human	86.3	89.0	86.9	89.5
#1 Single - MIR-MRC (F-Net)	-	-	74.8	78.0
#2 Single - nlnet	-	-	74.2	77.1
Published				
unet (Ensemble)	-	-	71.4	74.9
SLQA+ (Single)	-		71.4	74.4
Ours				
$\text{BERT}_{\text{LARGE}}$ (Single)	78.7	81.9	80.0	83.1

SQuAD v2.0.

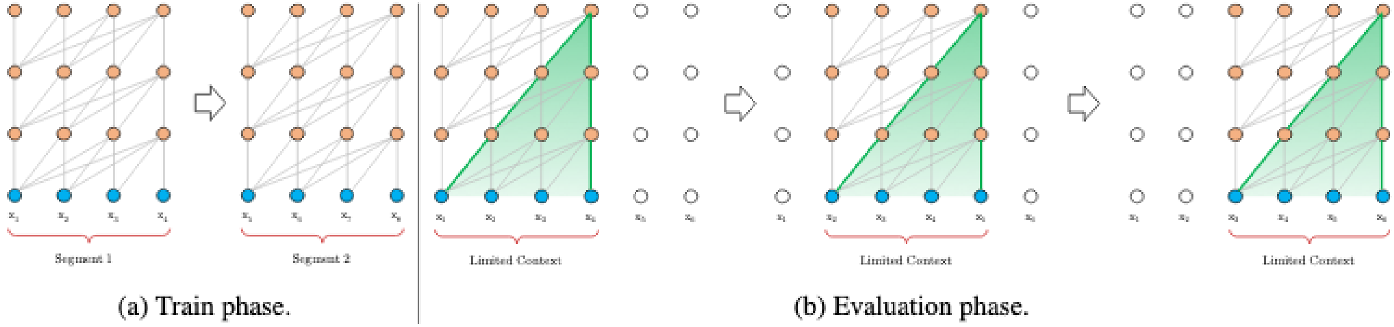
Problems with BERT

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).



(a) Train phase.

(b) Evaluation phase.

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: $\mathbf{s}_\tau = [x_{\tau,1}, \cdots, x_{\tau,L}]$ and $\mathbf{s}_{\tau+1} = [x_{\tau+1,1}, \cdots, x_{\tau+1,L}]$ be the two consecutive segments in training, and $\mathbf{h}_\tau^n \in \mathbb{R}^{L\times d}$ be the hidden state sequence of the n-layer of the model for the τ segment $\mathbf{s}_\tau$, then:

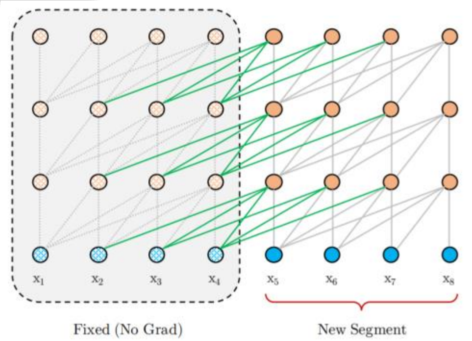
Any parameters inside SG will not be updated during backpropagation

$$\boxed{\begin{aligned} &\widetilde{\mathbf{h}}_{\tau+1}^{n-1} = \left[\mathrm{SG}(\mathbf{h}_\tau^{n-1}) \circ \mathbf{h}_{\tau+1}^{n-1}\right], \\ &\mathbf{q}_{\tau+1}^n, \mathbf{k}_{\tau+1}^n, \mathbf{v}_{\tau+1}^n = \mathbf{h}_{\tau+1}^{n-1}\mathbf{W}_q^\top, \widetilde{\mathbf{h}}_{\tau+1}^{n-1}\mathbf{W}_k^\top, \widetilde{\mathbf{h}}_{\tau+1}^{n-1}\mathbf{W}_v^\top, \\ &\mathbf{h}_{\tau+1}^n = \text{Transformer-Layer}\left(\mathbf{q}_{\tau+1}^n, \mathbf{k}_{\tau+1}^n, \mathbf{v}_{\tau+1}^n\right). \end{aligned}}$$

We only want to train this

where SG stands for stop-gradient

and $[\mathbf{h}_u \circ \mathbf{h}_v]$ is the concatenation operation.

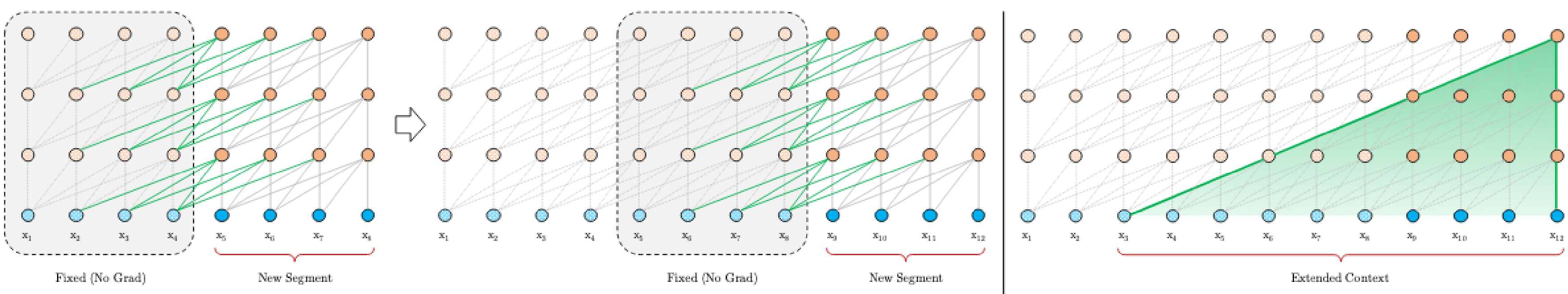


Transformer-XL: Segment-level Recurrence

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.



(a) Training phase.

(b) Evaluation phase.

Transformer-XL: Relative Position Encodings

With the standard absolute position encodings:
so no positional difference between $x_{\tau,j}$ and $x_{\tau+1,j}$

$$\boxed{\begin{aligned}\mathbf{h}_{\tau+1} &= f(\mathbf{h}_{\tau}, \mathbf{E}_{\mathbf{s}_{\tau+1}} + \mathbf{U}_{1:L}) \\ \mathbf{h}_{\tau} &= f(\mathbf{h}_{\tau-1}, \mathbf{E}_{\mathbf{s}_{\tau}} + \mathbf{U}_{1:L})\end{aligned}}$$

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 $q_{\tau,i}$).

Word representation = E ∘ U

$$\begin{aligned}\mathbf{A}_{i,j}^{\text{abs}} &= \underbrace{\mathbf{E}_{x_i}^{\top}\mathbf{W}_q^{\top}\mathbf{W}_k\mathbf{E}_{x_j}}_{(a)} + \underbrace{\mathbf{E}_{x_i}^{\top}\mathbf{W}_q^{\top}\mathbf{W}_k\mathbf{U}_j}_{(b)} \\ &+ \underbrace{\mathbf{U}_i^{\top}\mathbf{W}_q^{\top}\mathbf{W}_k\mathbf{E}_{x_j}}_{(c)} + \underbrace{\mathbf{U}_i^{\top}\mathbf{W}_q^{\top}\mathbf{W}_k\mathbf{U}_j}_{(d)}\end{aligned}$$

$$\begin{aligned}\mathbf{A}_{i,j}^{\text{rel}} &= \underbrace{\mathbf{E}_{x_i}^{\top}\mathbf{W}_q^{\top}\mathbf{W}_{k,E}\mathbf{E}_{x_j}}_{(a)} + \underbrace{\mathbf{E}_{x_i}^{\top}\mathbf{W}_q^{\top}\mathbf{W}_{k,R}\mathbf{R}_{i-j}}_{(b)} \\ &+ \underbrace{u^{\top}\mathbf{W}_{k,E}\mathbf{E}_{x_j}}_{(c)} + \underbrace{v^{\top}\mathbf{W}_{k,R}\mathbf{R}_{i-j}}_{(d)}.\end{aligned}$$

$\boldsymbol{E}_*$ is word embedding vector, $\boldsymbol{U}_*$ and $\boldsymbol{R}_*$ are absolution and relative position embeddings (respectively), and u and v are learnable vectors.

Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context. Dai et al. 2019.

Transformer-xl Evaluation On Language Modeling Datasets

Model	#Param	PPL
Grave et al. (2016b) - LSTM	-	48.7
Bai et al. (2018) - TCN	-	45.2
Dauphin et al. (2016) - GCNN-8	-	44.9
Grave et al. (2016b) - LSTM + Neural cache	-	40.8
Dauphin et al. (2016) - GCNN-14	-	37.2
Merity et al. (2018) - QRNN	151M	33.0
Rae et al. (2018) - Hebbian + Cache	-	29.9
Ours - Transformer-XL Standard	151M	**24.0**
Baevski and Auli (2018) - Adaptive Input$^\diamond$	247M	20.5
Ours - Transformer-XL Large	257M	**18.3**

Table 1: Comparison with state-of-the-art results on WikiText-103. $^\diamond$ indicates contemporary work.

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

Table 4: Comparison with state-of-the-art results on One Billion Word. $^\diamond$ 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:

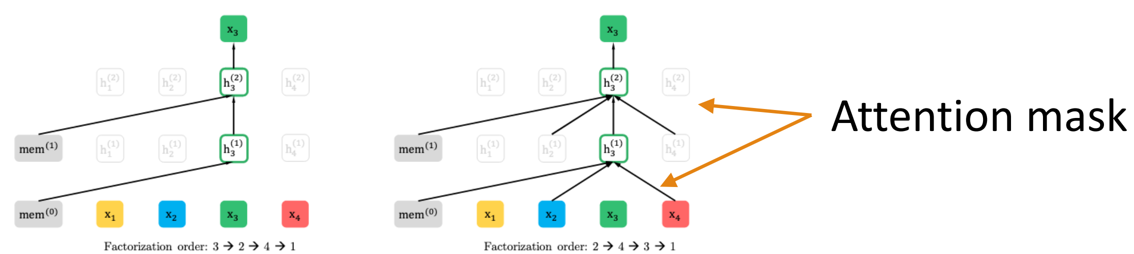
- tokens are indexed from 1 to T
- $\mathcal{Z}_T$ is the set of all possible permutation of $[1, 2, \ldots, T]$

$$\mathbf{z} = [z_1, z_2, \ldots, z_t, \ldots, z_T] \in \mathcal{Z}_T$$
$$\mathbf{z}_{<t} = [z_1, z_2, \ldots, z_{t-1}]$$

- The loss function:

$$\max_{\theta} \quad \mathbb{E}_{\mathbf{z} \sim \mathcal{Z}_T} \left[\sum_{t=1}^{T} \log p_{\theta}(x_{z_t} \mid \mathbf{x}_{\mathbf{z}_{<t}}) \right]$$

- 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.

$$\max_{\theta} \quad \mathbb{E}_{\mathbf{z}\sim\mathcal{Z}_T}\left[\log p_\theta(\mathbf{x}_{\mathbf{z}_{>c}} \mid \mathbf{x}_{\mathbf{z}_{\leq c}})\right] = \mathbb{E}_{\mathbf{z}\sim\mathcal{Z}_T}\left[\sum_{t=c+1}^{|\mathbf{z}|} \log p_\theta(x_{z_t} \mid \mathbf{x}_{\mathbf{z}_{<t}})\right]$$

* c is chosen such that $|\mathbf{z}| / (|\mathbf{z}| - c) \approx K$

XLNet: Two-Stream Self-Attention

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

$$p_\theta(X_{z_t} = x \mid \mathbf{x}_{z_{<t}}) = \frac{\exp\left(e(x)^\top h_\theta(\mathbf{x}_{\mathbf{z}_{<t}})\right)}{\sum_{x'} \exp\left(e(x')^\top h_\theta(\mathbf{x}_{\mathbf{z}_{<t}})\right)}$$

This does not depend on which position it will predict, i.e., z_t

- It can't see x_{z_t}, otherwise the objective is trivial (this is also why BERT needs masks)

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

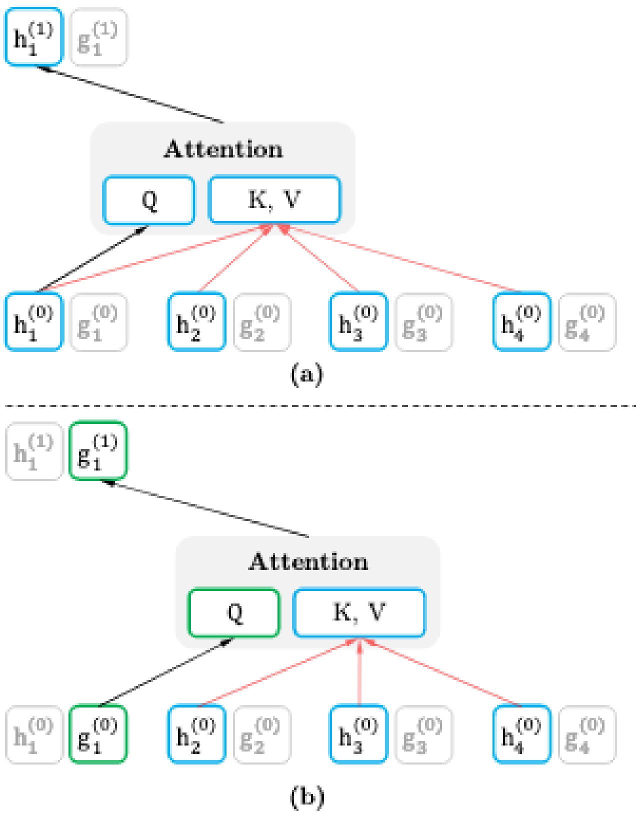
$$p_\theta(X_{z_t} = x \mid \mathbf{x}_{z_{<t}}) = \frac{\exp\left(e(x)^\top \boxed{g_\theta(\mathbf{x}_{\mathbf{z}_{<t}}, z_t)}\right)}{\sum_{x'} \exp\left(e(x')^\top g_\theta(\mathbf{x}_{\mathbf{z}_{<t}}, z_t)\right)}$$

Modeling this is non-trivial, i.e., need to handle the cases for $j = t$ and $j > t$ differently

Idea: using two sets of hidden representations:

- The context representations $h_\theta(\mathbf{x}_{\mathbf{z}_{\leq t}})$ to encode both the context and x_{z_t} itself.
- The query representations $g_\theta(\mathbf{x}_{\mathbf{z}_{<t}}, z_t)$ to only encode the contextual information $\mathbf{x}_{\mathbf{z}_{<t}}$ and the position z_t .

XLNet: Two-Stream Self-Attention



The content stream:

$$h_{z_t}^{(m)} \leftarrow \text{Attention}(\mathrm{Q} = h_{z_t}^{(m-1)}, \mathbf{KV} = \mathbf{h}_{\mathbf{z}_{\leq t}}^{(m-1)}; \theta)$$

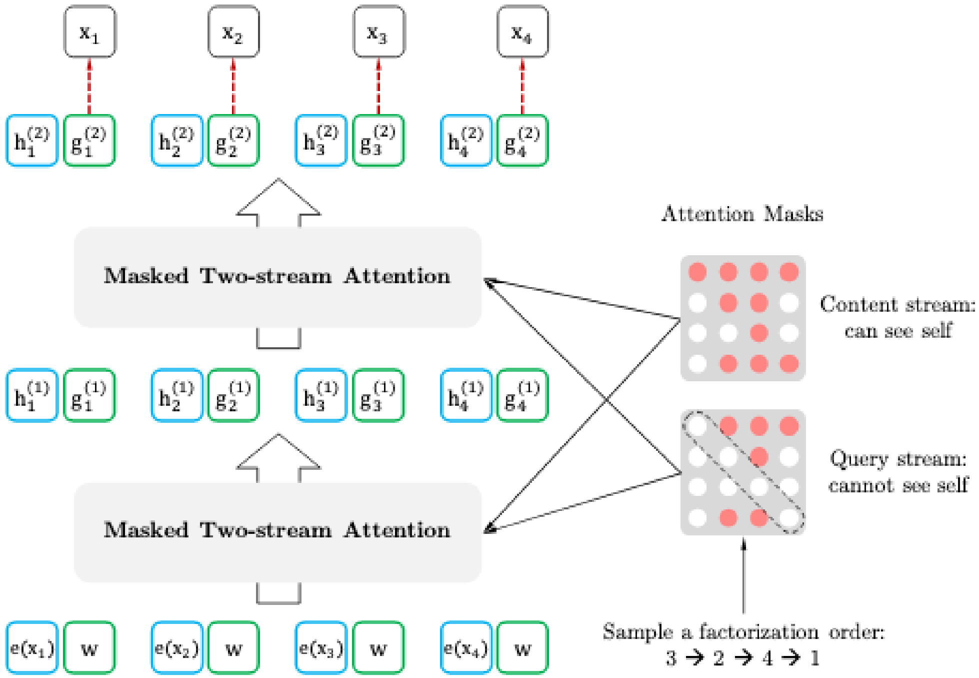
This is also integrated with the ideas of segment level recurrence and relative position encodings from Transformer-XL:

$$h_{z_t}^{(m)} \leftarrow \text{Attention}(\mathrm{Q} = h_{z_t}^{(m-1)}, \mathbf{KV} = \left[\tilde{\mathbf{h}}^{(m-1)}, \mathbf{h}_{\mathbf{z}_{\leq t}}^{(m-1)}\right]; \theta)$$

The query stream:

$$g_{z_t}^{(m)} \leftarrow \text{Attention}(\mathrm{Q} = g_{z_t}^{(m-1)}, \mathbf{KV} = \mathbf{h}_{\mathbf{z}_{<t}}^{(m-1)}; \theta)$$

XLNet: Two-Stream Self-Attention



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

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

$h_{z_t}^{(M)}$ is the contextualized embedding used for fine-tuning.

XLNet: Fine-tuning

Similar to BERT, 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:
  - $a_{ij} = (\mathbf{q}_i + \mathbf{b})^\top \mathbf{s}_{ij}$ added to:

$$\mathbf{A}^{\text{rel}}_{i,j} = \underbrace{\mathbf{E}_{x_i}^\top \mathbf{W}_q^\top \mathbf{W}_{k,E} \mathbf{E}_{x_j}}_{(a)} + \underbrace{\mathbf{E}_{x_i}^\top \mathbf{W}_q^\top \mathbf{W}_{k,R} \mathbf{R}_{i-j}}_{(b)} + \underbrace{u^\top \mathbf{W}_{k,E} \mathbf{E}_{x_j}}_{(c)} + \underbrace{v^\top \mathbf{W}_{k,R} \mathbf{R}_{i-j}}_{(d)}.$$

  - $\mathbf{s}_{ij} = \mathbf{s}_+$ if positions i and j belong to the same sentence; and $\mathbf{s}_{ij} = \mathbf{s}_-$ otherwise.

→ XLNet can directly model input with <u>more than two sentences</u>.

XLNet: Evaluation

The SQuAD question answering datasets

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

XLNet: Evaluation

The GLUE benchmark

Model	MNLI	QNLI	QQP	RTE	SST-2	MRPC	CoLA	STS-B	WNLI
Single-task single models on dev									
BERT [2]	86.6/-	92.3	91.3	70.4	93.2	88.0	60.6	90.0	-
XLNet	**89.8/-**	**93.9**	**91.8**	**83.8**	**95.6**	**89.2**	**63.6**	**91.8**	-
Single-task single models on test									
BERT [10]	86.7/85.9	91.1	89.3	70.1	94.9	89.3	60.5	87.6	65.1
Multi-task ensembles on test (from leaderboard as of June 19, 2019)									
Snorkel* [29]	87.6/87.2	93.9	89.9	80.9	96.2	91.5	63.8	90.1	65.1
ALICE*	88.2/87.9	95.7	**90.7**	83.5	95.2	92.6	**68.6**	91.1	80.8
MT-DNN* [18]	87.9/87.4	96.0	89.9	**86.3**	96.5	92.7	68.4	91.1	89.0
XLNet*	**90.2/89.7**†	**98.6**†	90.3†	**86.3**	**96.8**†	**93.0**	67.8	**91.6**	**90.4**

Datasets and Resources

Model	Pre-trained datasets
ELMo	-One Billion Word Benchmark
GPT	-BooksCorpus (800M words)
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!):
https://syncedreview.com/2019/06/27/the-staggering-cost-of-training-sota-ai-models/

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