# Question Generation

## Introduction

Now we want to make our app more real and enable interaction between humans and characters in the app by making characters ask the user question about the presentation topic and question related to his speech so in this chapter will introduce our question generation system.  
Texts with potential educational value are becoming available through the Internet (e.g., Wikipedia, news services). However, using these new texts in classrooms introduces many challenges, one of which is that they usually lack practice exercises and assessments. Here, we address part of this challenge by automating the creation of a specific type of assessment item. Specifically, we focus on automatically generating factual questions. Our goal is to create an automated system that can take as input a text and produce as output questions for assessing a reader’s knowledge of the information in the text.

## The dataset (SQuAD)

Large-scale manually annotated passage and question pairs play a crucial role in developing question generation systems. We propose to adapt the recently released Stanford Question Answering Dataset (SQuAD) (Rajpurkar et al., 2016) as  
the training and development datasets for the question generation task. In SQuAD, the answers are labeled as subsequences in the given sentences by crowed sourcing, and it contains more than 100K questions which makes it feasible to train our neural network models.

## Related work

Automatic question generation from natural language text aims to generate questions taking text as input, which has the potential value of education purpose (Heilman, 2011). As the reverse task of question answering, question generation also has the potential for providing a large scale corpus of question-answer pairs.  
Previous works for question generation mainly use rigid heuristic rules to transform a sentence into related questions (Heilman, 2011; Chali and  
Hasan, 2015). However, these methods heavily rely on human-designed transformation and generation rules, which cannot be easily adopted to  
other domains. Instead of generating questions from texts, Serban et al. (2016) proposed a neu- ral network method to generate factoid questions  
from structured data.

* RNN Seq2seq Models :

In this will introduce NG++ paper work , In this we conduct a preliminary study on question generation from text with neural networks, which idenoted as the Neural Question Generation (NQG) framework, to generate natural  
language questions from text without pre-defined rules.

The Neural Question Generation framework extends the sequence-to sequence models by enriching the encoder with answer and lexical features to generate answer focused questions. Concretely, the encoder reads not only the input sentence, but also the answer position indicator and  
lexical features. The answer position feature denotes the answer span in the input sentence, which is essential to generate answer relevant questions.  
The lexical features include part-of-speech (POS)  
and named entity (NER) tags to help produce better sentence encoding. Lastly, the decoder with attention mechanism (Bahdanau et al., 2015) generates an answer specific question of the sentence.

the NQG framework, which consists of a feature-rich encoder and an attention-based decoder. Figure 1 provides an overview of our NQG framework

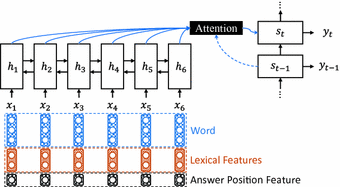


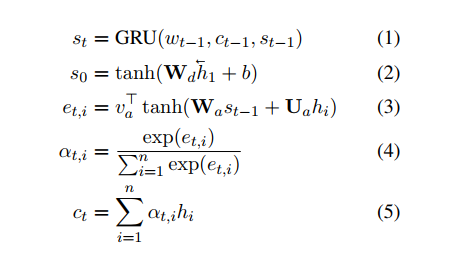
Figure 1: Overview of the Neural Question Generation (NQG) framework

**Feature-Rich Encoder**

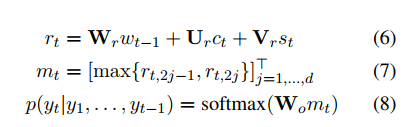
we use Gated Recurrent Unit (GRU) (Cho et al., 2014) to build the encoder. To capture more context information, we use bidirectional GRU (BiGRU) to read the inputs in both forward and backward orders. Inspired by Chen and Manning (2014); Nallapati et al. (2016), the Bi GRU encoder not only reads the sentence words, but also handcrafted features, to produce a sequence of word-and-feature vectors. We concatenate the word vector, lexical feature embedding vectors and answer position indicator embedding vector as the input of Bi GRU encoder. Concretely, the Bi GRU encoder reads the concatenated sentence word vector, lexical features, and answer position feature, x = (x1; x2; : : : ; xn), to produce two sequences of hidden vectors, i.e., the forward sequence (~h1;~h2; : : : ;~hn) and the backward sequence (h1~ ; h2~ ; : : : ; hn~ ). Lastly, the output sequence of the encoder is the concatenation  
of the two sequences, i.e., hi = [~hi; hi~ ].  
Answer Position Feature

To generate a question with respect to a specific answer in a sentence, we propose using answer position feature to locate the target answer. In this work, the BIO tagging scheme is used to label the position of a target answer. In this scheme, tag B denotes the start of an answer, tag I continues the answer and tag O marks words that do not form part of an answer. The BIO tags of answer position are embedded to real-valued vectors throu and fed to the feature rich encoder. With the BIO tagging feature, the answer position is encoded to the hidden vectors and used to generate answer focused questions.

**Lexical Features**

Besides the sentence words,we also feed other lexical features to the encoder.To encode more linguistic information, we selectword case, POS and NER tags as the lexical features. As an intermediate layer of full parsing,POS tag feature is important in many NLP tasks,such as information extraction and dependencyparsing (Manning et al., 1999). Considering thatSQuAD is constructed using Wikipedia articles,which contain lots of named entities, we add NERfeature to help detecting them. **Attention-Based Decoder**We employ an attention-based GRU decoder to decode the sentence and answer information to generate questions. At decoding time step t, the GRU  
decoder reads the previous word embedding wt-1 and context vector ct-1 to compute the new hidden state st. We use a linear layer with the last backward encoder hidden state h1~ to initialize the decoder GRU hidden state. The context vector ct for current time step t is computed through the concatenate attention mechanism (Luong et al., 2015), which matches the current decoder state st with each encoder hidden state hi to get an importance score. The importance scores are then normalized  
to get the current context vector by weighted sum:  
**

We then combine the previous word embedding wt-1, the current context vector ct, and the decoder state st to get the readout state rt. The readout state is passed through a max out hidden layer (Goodfellow et al., 2013) to predict the next word with a softmax layer over the decoder vocabulary:

  
where rt is a 2d-dimensional vector

**Copy Mechanism**To deal with the rare and unknown words problem, Gulcehre et al. (2016) propose using pointing mechanism to copy rare words from source sentence. We apply this pointing method in our NQG system. When decoding word t, the copy switch takes current decoder state st and context vector ctas input and generates the probability p of copying a word from source sentence:  
p = σ(Wst + Uct + b) (9)  
where σ is sigmoid function. We reuse the attention probability in equation 4 to decide which word to copy

## Attention Is All You Need

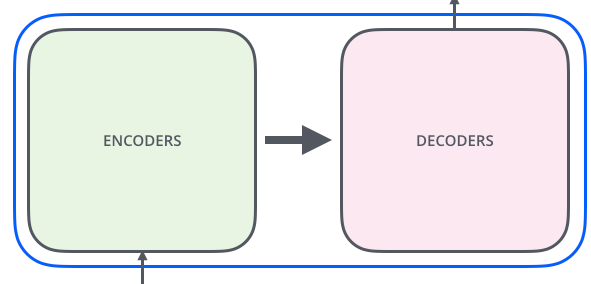
in the previous   [we looked at Attention](https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/) – a ubiquitous method in modern deep learning models. Attention is a concept that helped improve the performance of neural machine translation applications. Now, we will look at **The Transformer** – a model that uses attention to boost the speed with which these models can be trained. The Transformers outperforms the Google Neural Machine Translation model in specific tasks. The biggest benefit, however, comes from how The Transformer lends itself to parallelization. It is in fact Google Cloud’s recommendation to use The Transformer as a reference model to use their [Cloud TPU](https://cloud.google.com/tpu/) offering. So let’s try to break the model apart and look at how it functions.

The Transformer was proposed in the paper ([Attention is All You Need](https://arxiv.org/abs/1706.03762)).

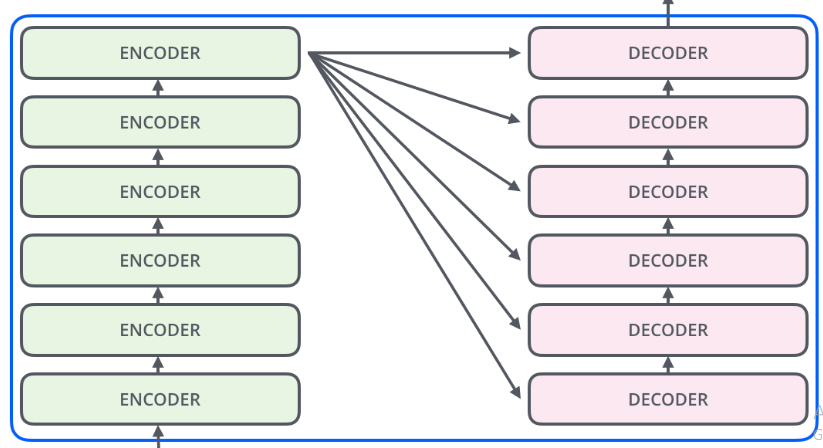
Let’s begin by looking at the model as a single black box.



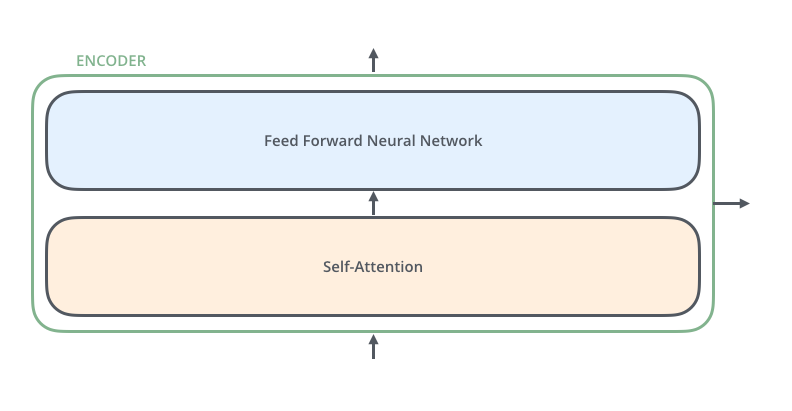
Popping open that Optimus Prime goodness, we see an encoding component, a decoding component, and connections between them.



The encoding component is a stack of encoders (the paper stacks six of them on top of each other – there’s nothing magical about the number six, one can definitely experiment with other arrangements). The decoding component is a stack of decoders of the same number.



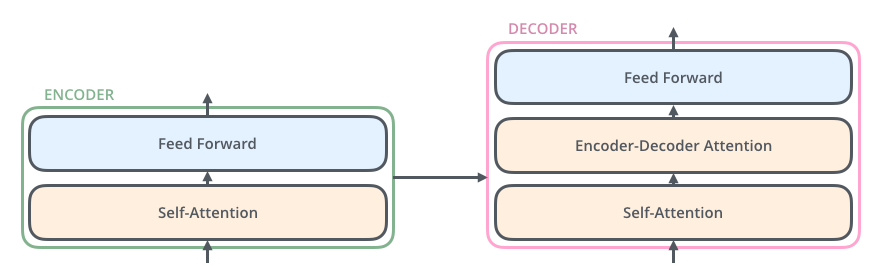
The encoders are all identical in structure (yet they do not share weights). Each one is broken down into two sub-layers:



The encoder’s inputs first flow through a self-attention layer – a layer that helps the encoder look at other words in the input sentence as it encodes a specific word .

The outputs of the self-attention layer are fed to a feed-forward neural network. The exact same feed-forward network is independently applied to each position.

The decoder has both those layers, but between them is an attention layer that helps the decoder focus on relevant parts of the input sentence (similar what attention does in [seq2seq models](https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/)).

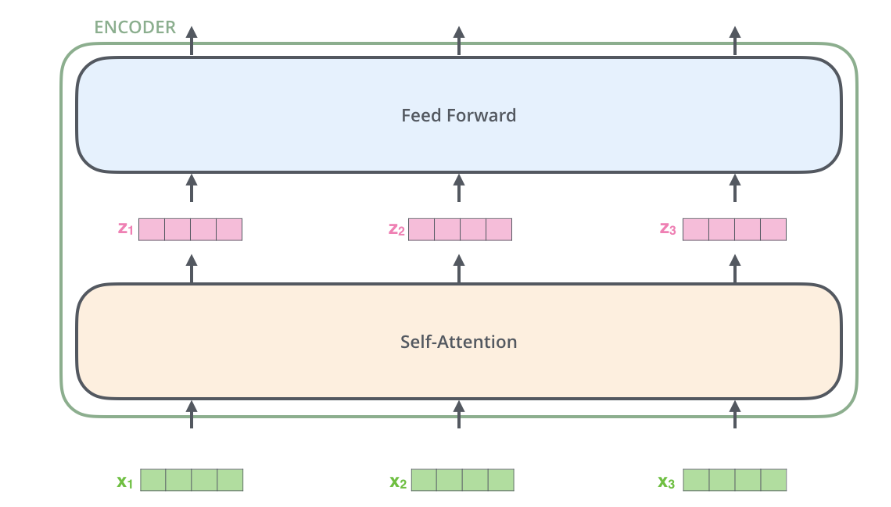


Now that we’ve seen the major components of the model, let’s start to look at the various vectors/tensors and how they flow between these components to turn the input of a trained model into an output.

As is the case in NLP applications in general, we begin by turning each input word into a vector using an [embedding algorithm](https://medium.com/deeper-learning/glossary-of-deep-learning-word-embedding-f90c3cec34ca).

The embedding only happens in the bottom-most encoder. The abstraction that is common to all the encoders is that they receive a list of vectors each of the size 512 – In the bottom encoder that would be the word embedding, but in other encoders, it would be the output of the encoder that’s directly below. The size of this list is hyper parameter we can set – basically it would be the length of the longest sentence in our training dataset.

After embedding the words in our input sequence, each of them flows through each of the two layers of the encoder.

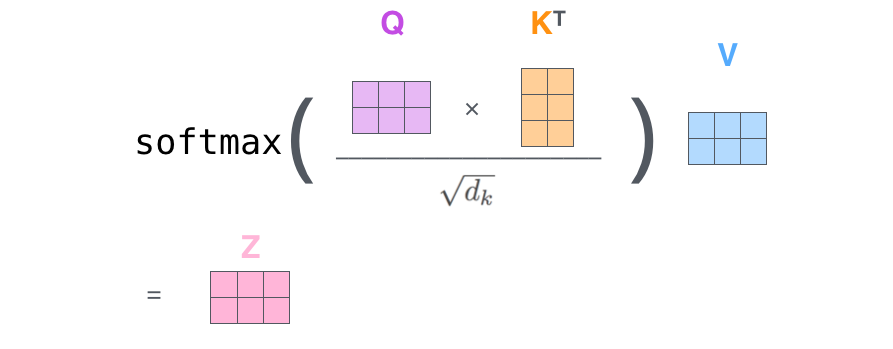


Here we begin to see one key property of the Transformer, which is that the word in each position flows through its own path in the encoder. There are dependencies between these paths in the self-attention layer. The feed-forward layer does not have those dependencies, however, and thus the various paths can be executed in parallel while flowing through the feed-forward layer.

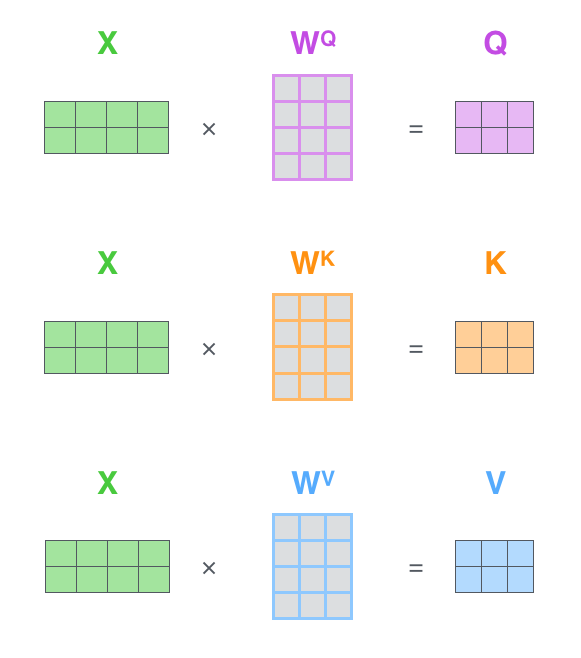
## **Matrix Calculation of Self-Attention**

**The first step** is to calculate the Query, Key, and Value matrices. We do that by packing our embedding into a matrix X, and multiplying it by the weight matrices we’ve trained (WQ, WK, WV).

**Finally**, since we’re dealing with matrices, we can condense steps two through six in one formula to calculate the outputs of the self-attention layer.

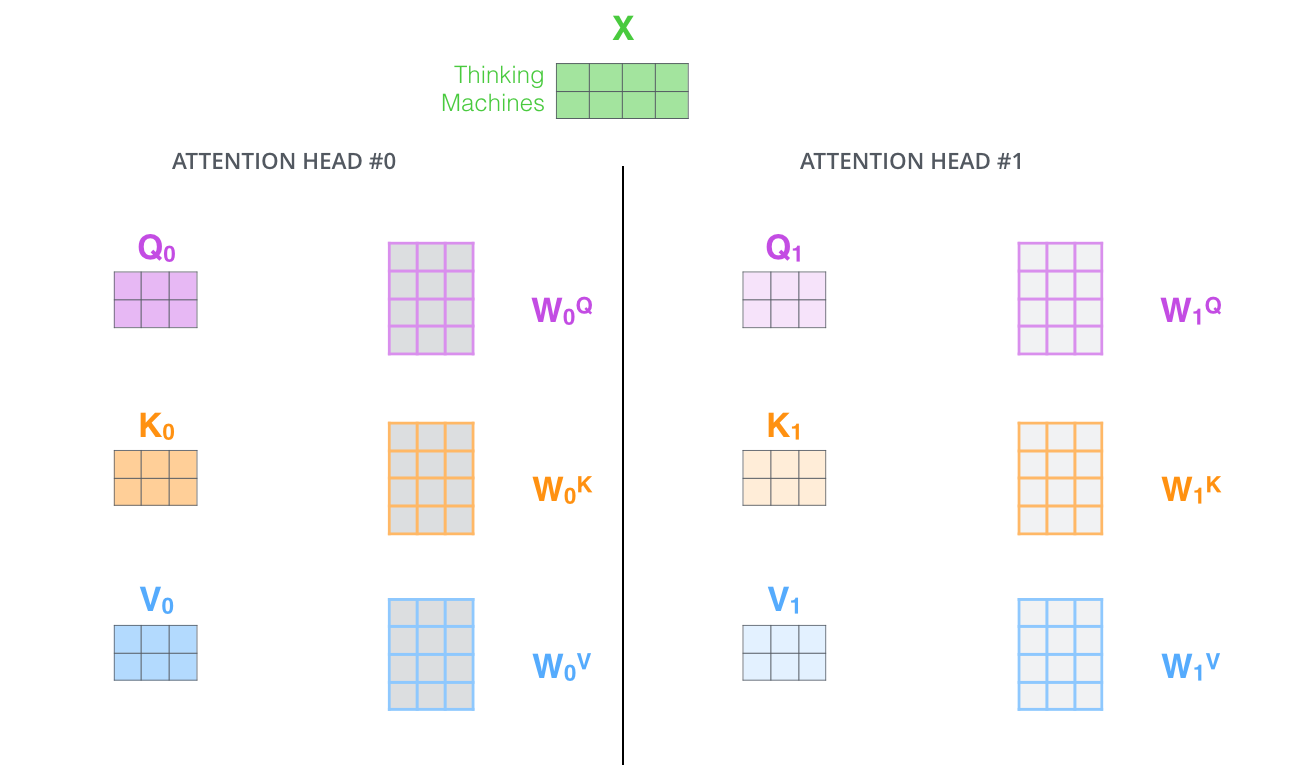


self-attention calculation in matrix form

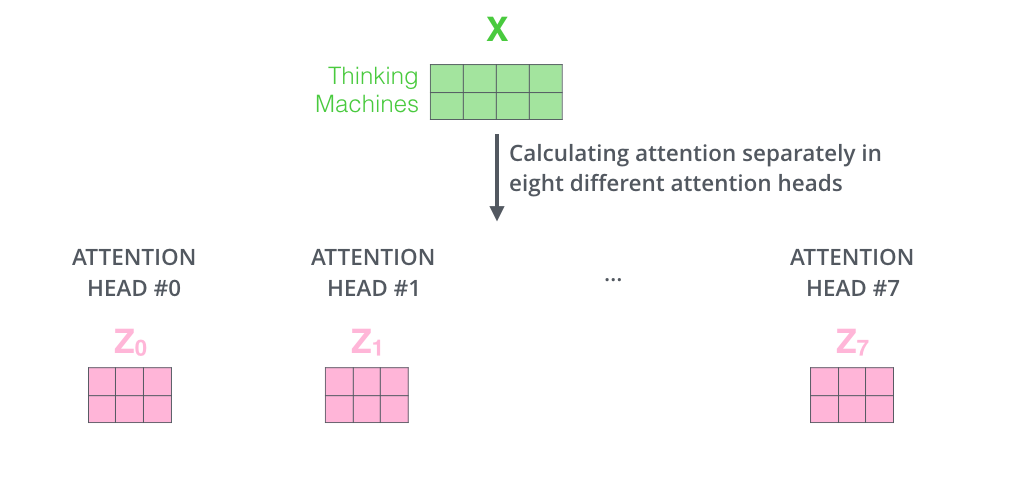
  
Every row in the X matrix corresponds to a word in the input sentence. We again see the difference in size of the embedding vector (512, or 4 boxes in the figure), and the q/k/v vectors (64, or 3 boxes in the figure)

The paper further refined the self-attention layer by adding a mechanism called “multi-headed” attention. This improves the performance of the attention layer in two ways:

1. It expands the model’s ability to focus on different positions. Yes, in the example above, z1 contains a little bit of every other encoding, but it could be dominated by the actual word itself.
2. It gives the attention layer multiple “representation subspaces”. As we’ll see next, with multi-headed attention we have not only one, but multiple sets of Query/Key/Value weight matrices (the Transformer uses eight attention heads, so we end up with eight sets for each encoder/decoder). Each of these sets is randomly initialized. Then, after training, each set is used to project the input embedding (or vectors from lower encoders/decoders) into a different representation subspace.

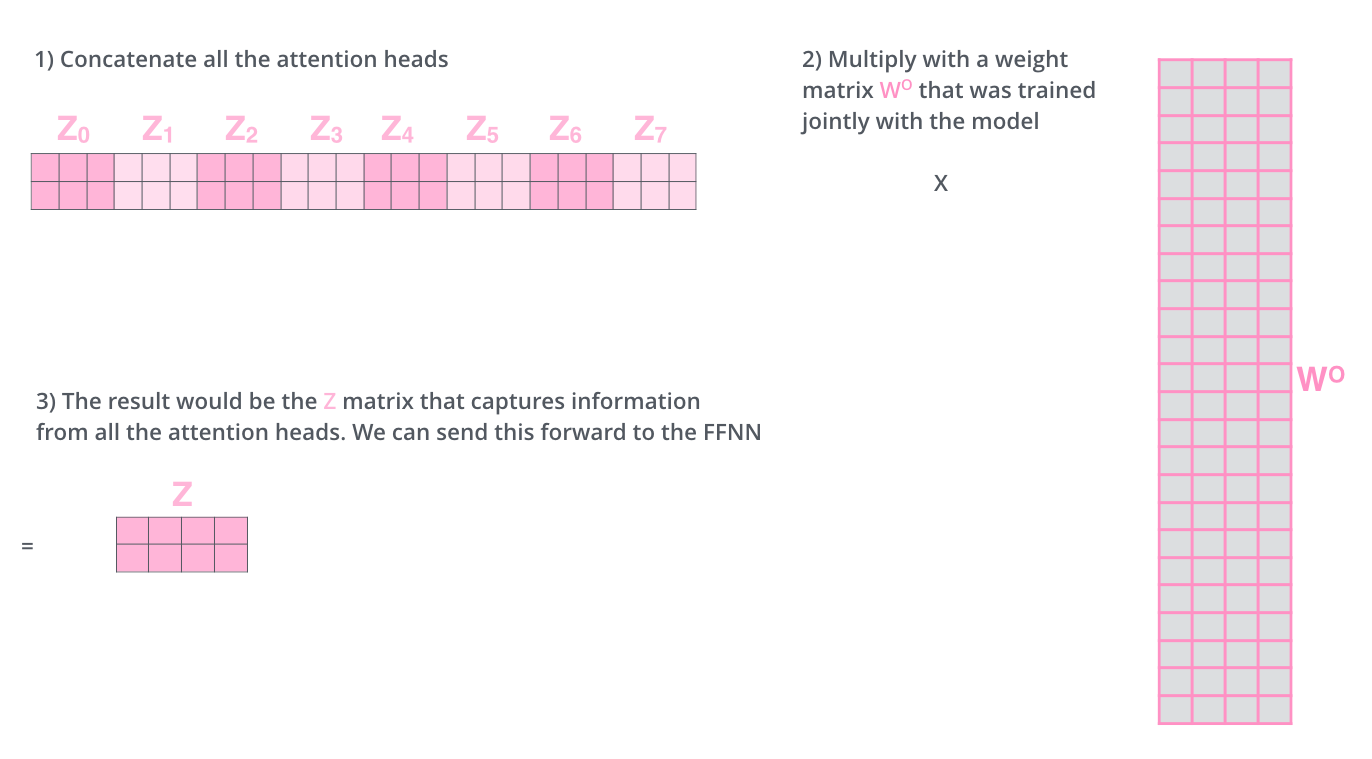
  
With multi-headed attention, we maintain separate Q/K/V weight matrices for each head resulting in different Q/K/V matrices. As we did before, we multiply X by the WQ/WK/WV matrices to produce Q/K/V matrices.

If we do the same self-attention calculation we outlined above, just eight different times with different weight matrices, we end up with eight different Z matrices

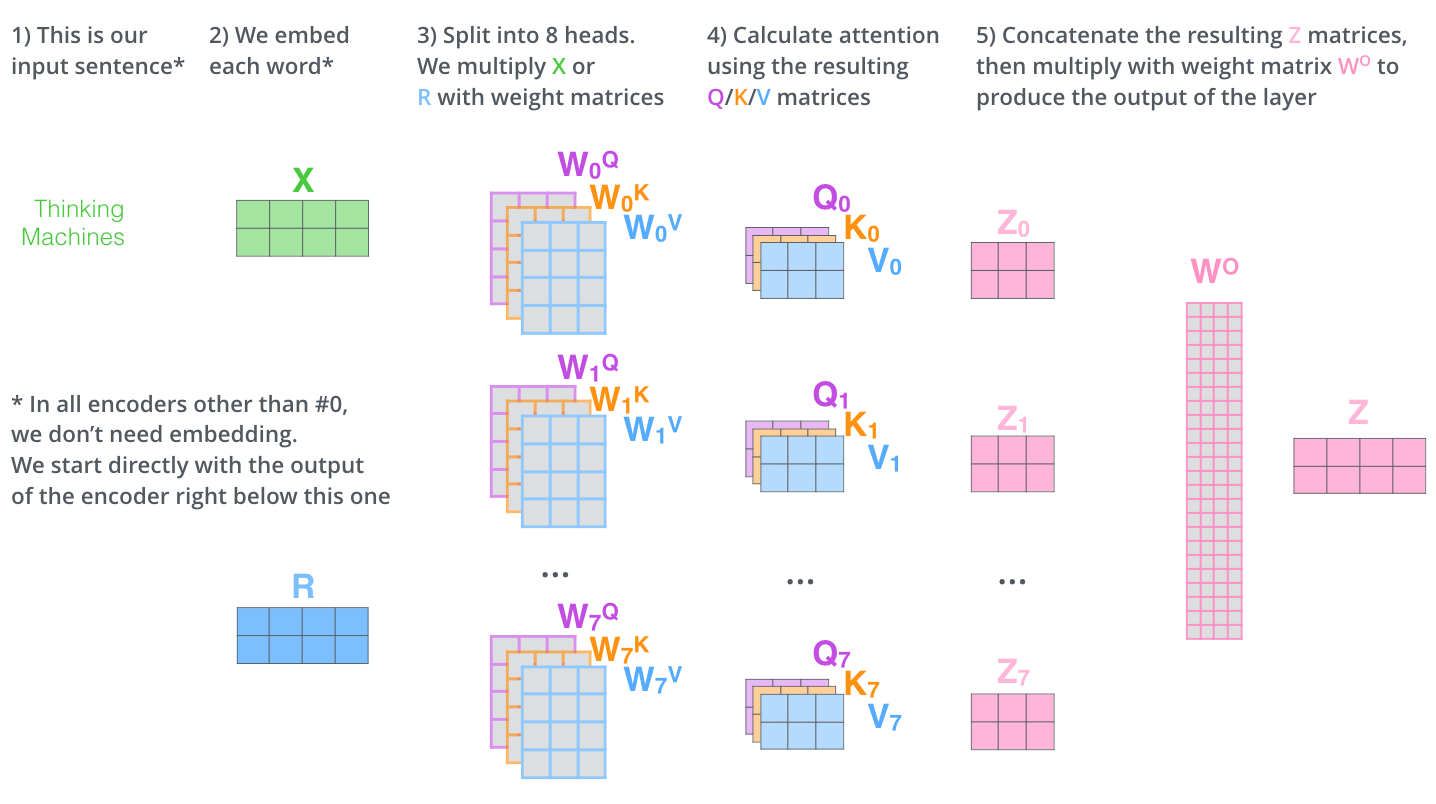


This leaves us with a bit of a challenge. The feed-forward layer is not expecting eight matrices – it’s expecting a single matrix (a vector for each word). So we need a way to condense these eight down into a single matrix.

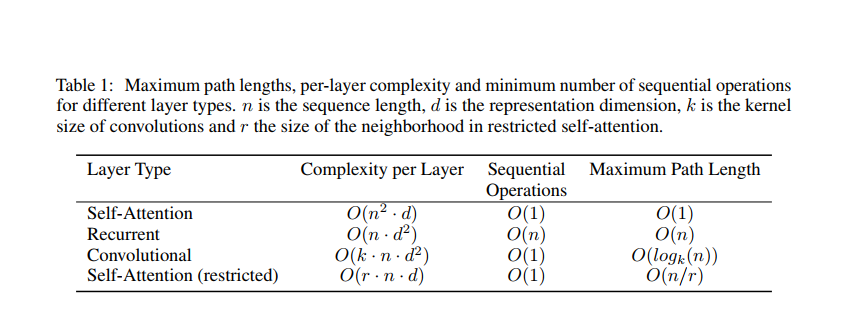
SO, We concatenate the matrices then multiple them by an additional weights matrix WO.



That’s pretty much all there is to multi-headed self-attention. It’s quite a handful of matrices, put them all in one visual so we can look at them in one place



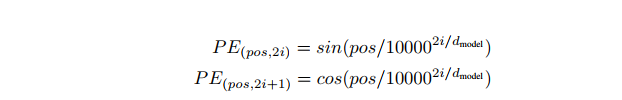
**Why Self-Attention**In this section we compare various aspects of self-attention layers to the recurrent and convolutional layers commonly used for mapping one variable-length sequence of symbol representations (x1; :::; xn) to another sequence of equal length (z1; :::; zn), with xi; zi 2 Rd, such as a hidden layer in a typical sequence transduction encoder or decoder. Motivating our use of self-attention we consider three desiderata. One is the total computational complexity per layer. Another is the amount of computation that can be parallelized, as measured by the minimum number of sequential operations required. The third is the path length between long-range dependencies in the network. Learning long-range dependencies is a key challenge in many sequence transduction tasks. One key factor affecting the ability to learn such dependencies is the length of the paths forward and backward signals have to traverse in the network. The shorter these paths between any combination of positions in the input and output sequences, the easier it is to learn long-range dependencies . Hence we also compare the maximum path length between any two input and output positions in networks composed of the  
different layer types.  
As noted in Table 1, a self-attention layer connects all positions with a constant number of sequentially executed operations, whereas a recurrent layer requires O(n) sequential operations. In terms of computational complexity, self-attention layers are faster than recurrent layers when the sequence  
length n is smaller than the representation dimensionality d, which is most often the case with sentence representations used by state-of-the-art models in machine translations, such as word-piece and byte-pair representations. To improve computational performance for tasks involving  
very long sequences, self-attention could be restricted to considering only a neighborhood of size r in 6 the input sequence centered around the respective output position. This would increase the maximum  
path length to O(n=r). We plan to investigate this approach further in future work.  
A single convolutional layer with kernel width k < n does not connect all pairs of input and output positions. Doing so requires a stack of O(n=k) convolutional layers in the case of contiguous kernels, or O(logk(n)) in the case of dilated convolutions [18], increasing the length of the longest paths  
between any two positions in the network. Convolutional layers are generally more expensive than recurrent layers, by a factor of k. Separable convolutions, however, decrease the complexity considerably, to O(k · n · d + n · d2). Even with k = n, however, the complexity of a separable convolution is equal to the combination of a self-attention layer and a point-wise feed-forward layer, the approach we take in our model.  
As side benefit, self-attention could yield more interpretable models. We inspect attention distributions  
from our models and present and discuss examples in the appendix. Not only do individual attention heads clearly learn to perform different tasks, many appear to exhibit behavior related to the syntactic and semantic structure of the sentences.



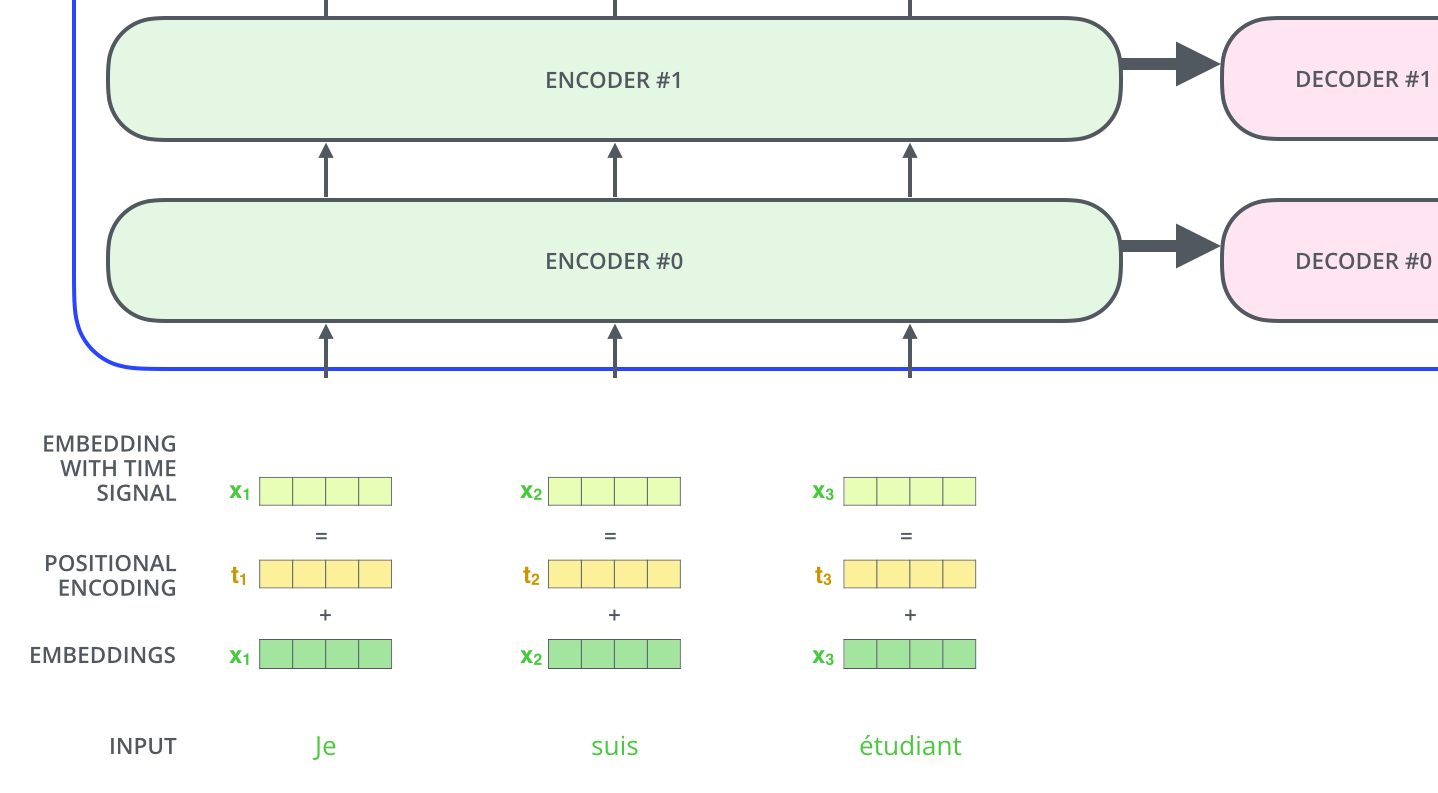
## **Representing The Order of The Sequence Using Positional Encoding**

One thing that’s missing from the model as we have described it so far is a way to account for the order of the words in the input sequence. To address this, the transformer adds a vector to each input embedding. These vectors follow a specific pattern that the model learns, which helps it determine the position of each word, or the distance between different words in the sequence. The intuition here is that adding these values to the embedding provides meaningful distances between the embedding vectors once they’re projected into Q/K/V vectors and during dot-product attention.

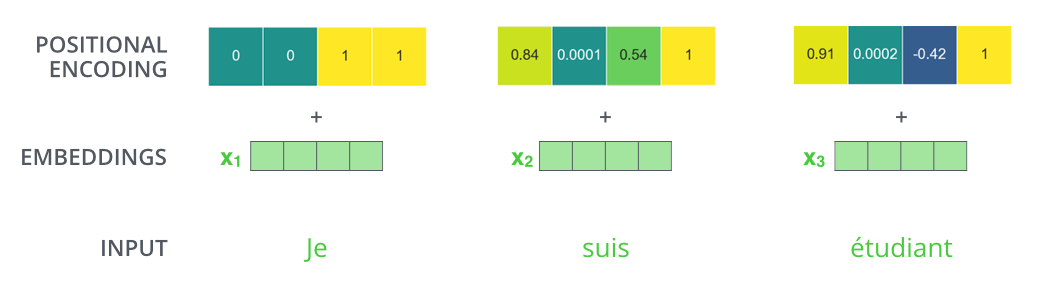
In this work, we use sine and cosine functions of different frequencies:



where pos is the position and i is the dimension. That is, each dimension of the positional encoding corresponds to a sinusoid. The wavelengths form a geometric progression from 2π to 10000 · 2π. We chose this function because we hypothesized it would allow the model to easily learn to attend by  
relative positions, since for any fixed offset k, PEpos+k can be represented as a linear function of PEpos.  
We also experimented with using learned positional embedding instead, and found that the two versions produced nearly identical results (. We chose the sinusoidal version because it may allow the model to extrapolate to sequence lengths longer than the ones encountered during training

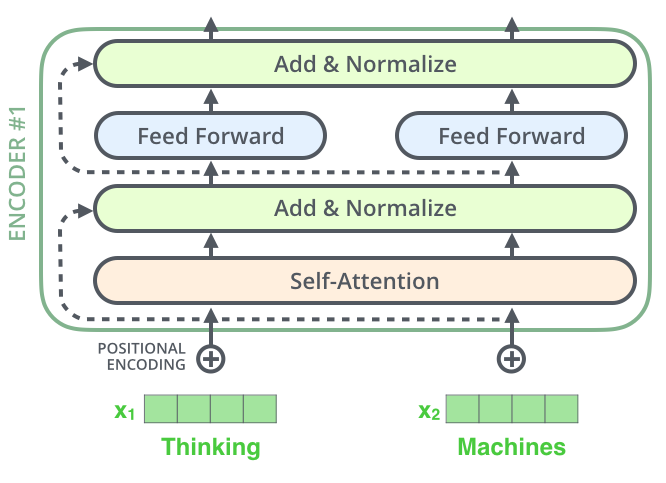
  
To give the model a sense of the order of the words, we add positional encoding vectors -- the values of which follow a specific pattern.

If we assumed the embedding has a dimensionality of 4, the actual positional encodings would look like this:

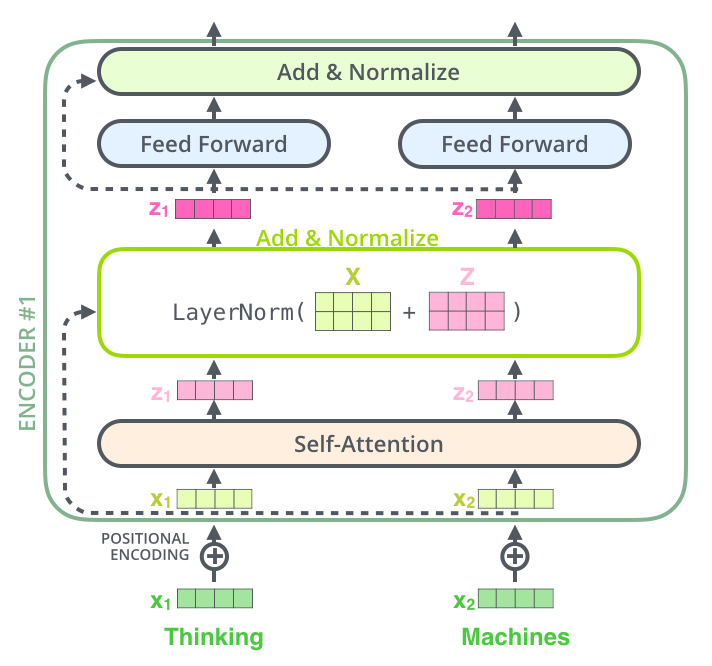
  
A real example of positional encoding with a toy embedding size of 4

## **The Residuals**

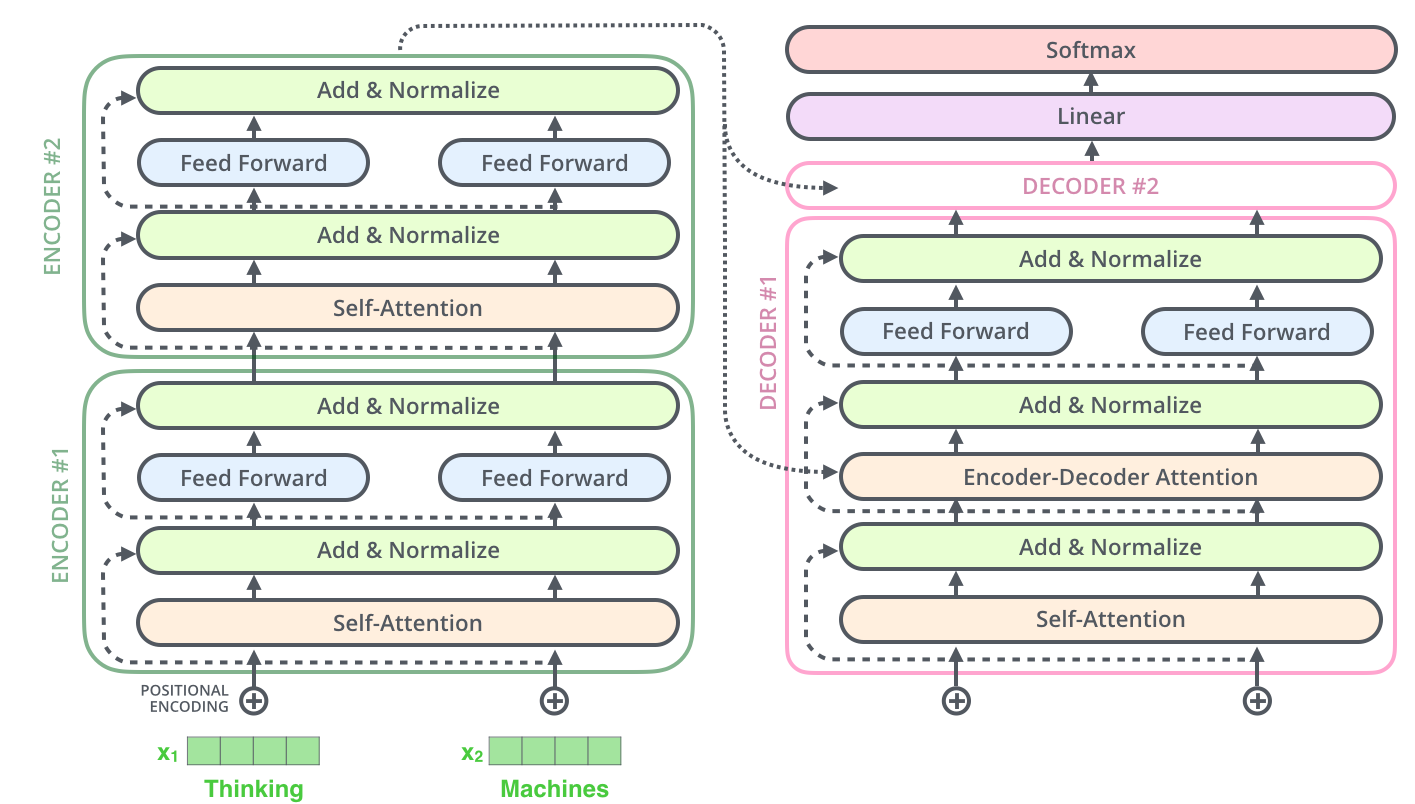
One detail in the architecture of the encoder that we need to mention before moving on, is that each sub-layer (self-attention, ffnn) in each encoder has a residual connection around it, and is followed by a [layer-normalization](https://arxiv.org/abs/1607.06450) step.



If we’re to visualize the vectors and the layer-norm operation associated with self attention, it would look like this:



This goes for the sub-layers of the decoder as well. If we’re to think of a Transformer of 2 stacked encoders and decoders, it would look something like this:



## **The Decoder Side**

Now that we’ve covered most of the concepts on the encoder side, we basically know how the components of decoders work as well. But let’s take a look at how they work together.

The encoder starts by processing the input sequence. The output of the top encoder is then transformed into a set of attention vectors K and V. These are to be used by each decoder in its “encoder-decoder attention” layer which helps the decoder focus on appropriate places in the input sequence: The following steps repeat the process until a special symbol is reached indicating the transformer decoder has completed its output. The output of each step is fed to the bottom decoder in the next time step, and the decoders bubble up their decoding results just like the encoders did. And just like we did with the encoder inputs, we embed and add positional encoding to those decoder inputs to indicate the position of each word.

The self-attention layers in the decoder operate in a slightly different way than the one in the encoder:

In the decoder, 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.

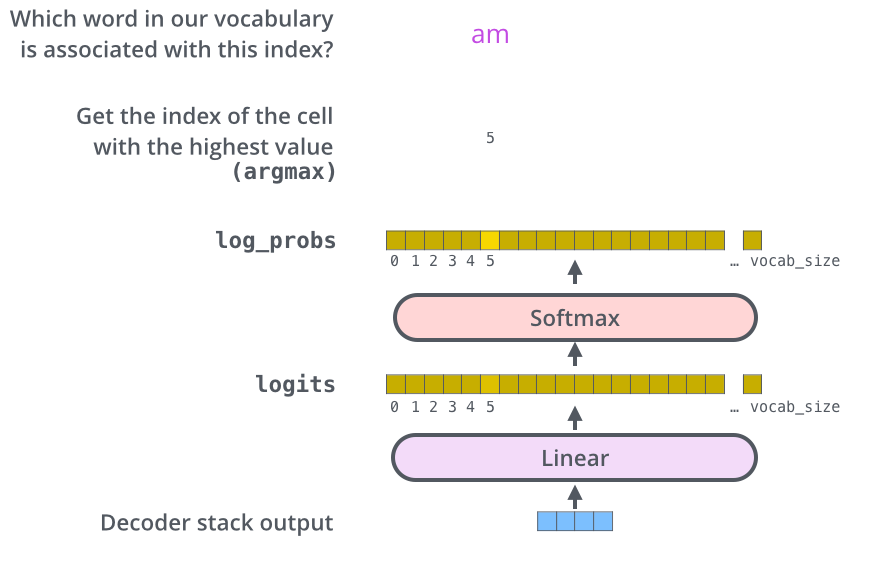
The “Encoder-Decoder Attention” layer works just like multiheaded self-attention, except it creates its Queries matrix from the layer below it, and takes the Keys and Values matrix from the output of the encoder stack.

## **The Final Linear and Softmax Layer**

The decoder stack outputs a vector of floats. How do we turn that into a word? That’s the job of the final Linear layer which is followed by a Softmax Layer.

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 those scores into probabilities (all positive, all add up to 1.0). The cell with the highest probability is chosen, and the word associated with it is produced as the output for this time step.

  
This figure starts from the bottom with the vector produced as the output of the decoder stack. It is then turned into an output word.

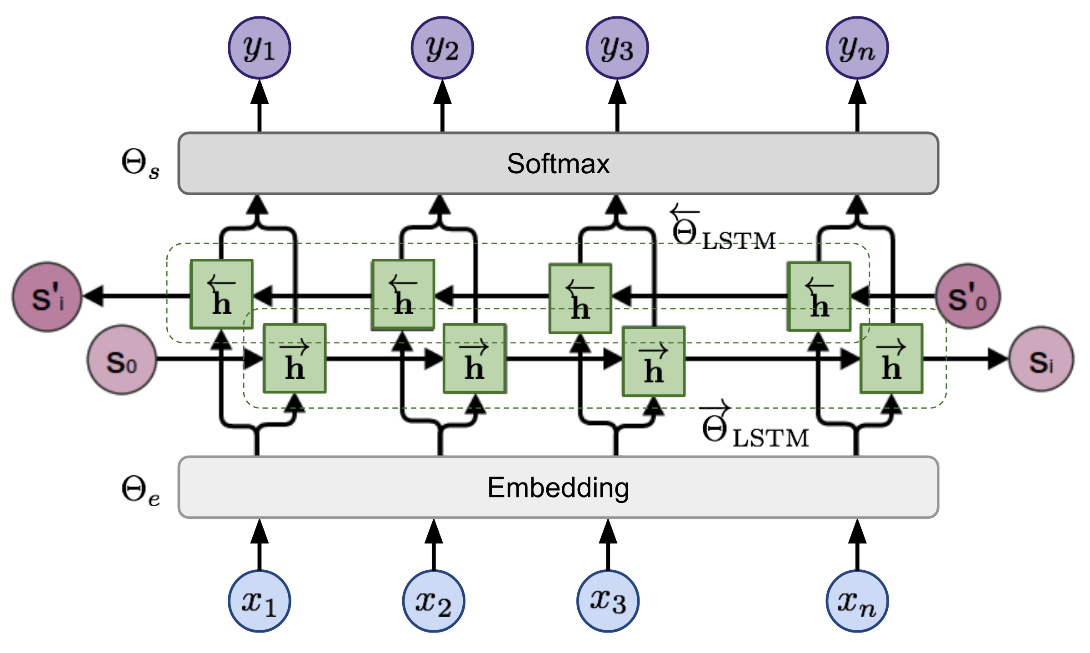
## Transfer Learning in nlp

The year 2018 has been an inflection point for machine learning models handling text (or more accurately, Natural Language Processing or NLP for short). Our conceptual understanding of how best to represent words and sentences in a way that best captures underlying meanings and relationships is rapidly evolving. Moreover, the NLP community has been putting forward incredibly powerful components that you can freely download and use in your own models and pipelines

# **ELMo**

[**ELMo**](https://arxiv.org/abs/1802.05365) stands for **Embedding from Language Model**,as the name suggests in this models the deeply contextualized word embedding are created from the Language Models (LM).

ELMo uses bidirectional language model (biLM) which is pre-trained on a large text corpus, to learn both words (e.g., syntax and semantics) and linguistic context (i.e., to model polysemy). BiLM capture **context-dependent**aspects of word meaning.



ELMo is applied on **semantic-intensive** and **syntax-intensive** tasks respectively using representations in different layers of biLM

* For a**semantic-intensive** task, the **top layer** is better than the first layer.
* And for a**syntax-intensive**task,**the first layer** is better than top layers.

# **OpenAI GPT-2**

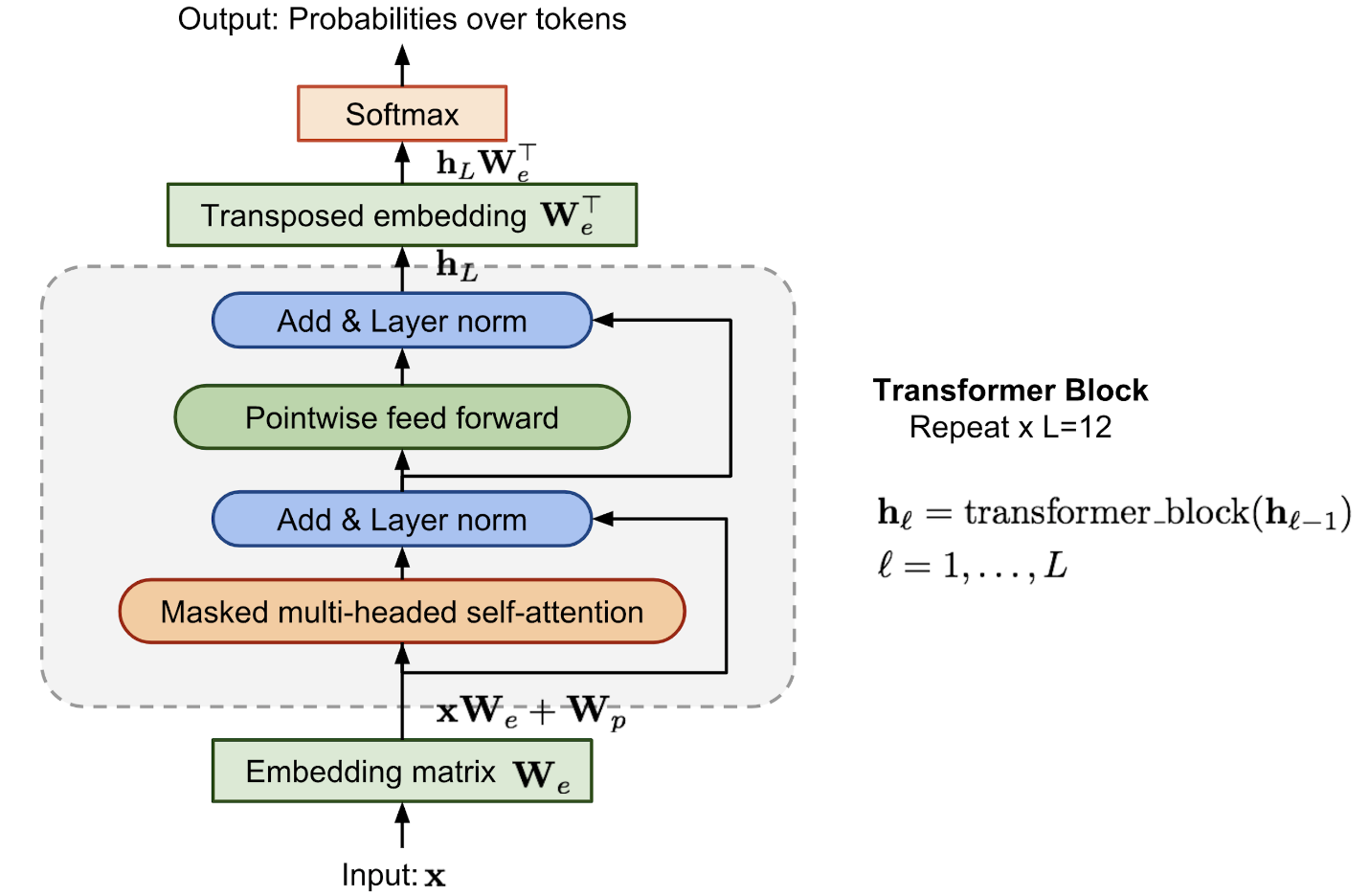
The [OpenAI GPT-2](https://openai.com/blog/better-language-models/" \t "_blank) is the successor of the [GPT](https://blog.openai.com/language-unsupervised/) model. GPT-2 is a large [transformer](https://arxiv.org/abs/1706.03762)-based language model, with **generative pre-training** of a language model on a diverse corpus of **unlabeled text**, followed by discriminative **fine-tuning** on each specific task.

GPT has two major differences from ELMo:

1. The model architecture: ELMo uses the concatenation of forward and backward LSTMs, but GPT uses **multi-layer transformers decoder.**
2. Contextualized embedding: ELMo uses unsupervised Feature-based approach, while GPT fine-tunes the same base model for all end tasks.

**Transformer Decoder as Language Model**Unlike [original transformer](https://arxiv.org/abs/1706.03762) architecture, the [transformer decoder](https://arxiv.org/abs/1801.10198) model discards the encoder part, so there is only one single input sentence rather than two separate source and target sequences.

Transformer block contains a masked multi-headed self-attentionfollowed by pointwise feed-forward**layer** and normalization layers in between. The final output produces a distribution over target tokens after softmax.

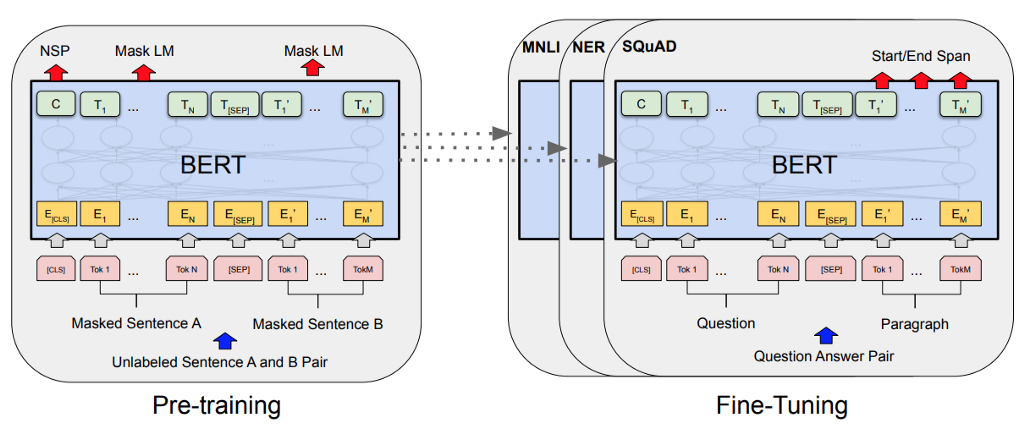


**BERT**

[BERT](https://arxiv.org/pdf/1810.04805.pdf) stands for **B**idirectional **E**ncoder **R**epresentations from **T**ransformers, as the name suggests this model is based on bidirectional representations from the **unlabeled text** by jointly conditioning on both **left and right context** in all layers. As a result, BERT is one of the most breakthroughs ideas in the last few years.

Compared to GPT, the largest difference and improvement of BERT is to make training **bi-directional**. The paper claim that:

*“bidirectional nature of our model is the single most important new contribution”*



**Pre-Training BERT**Pre-training BERT uses two unsupervised tasks, that are **Masked LM**and **Next Sentence Prediction (NSP)**to train.

Task 1:**Masked Language Model** (MLM)  
Learning the context **around a word** rather than learning just after the word makes it able to better capture its meaning, both syntactically and semantically.

The training data generator chooses **15% of the token**positions at**random** for **prediction**. If the iᵗʰ token is chosen, we replace the iᵗʰ token with

1. The **[MASK]** token **80%** of the time  
2. A **random** token **10%** of the time  
3. The **unchanged** iᵗʰ token **10%** of the time  
Ti will be used to predict the original token with **cross-entropy loss**

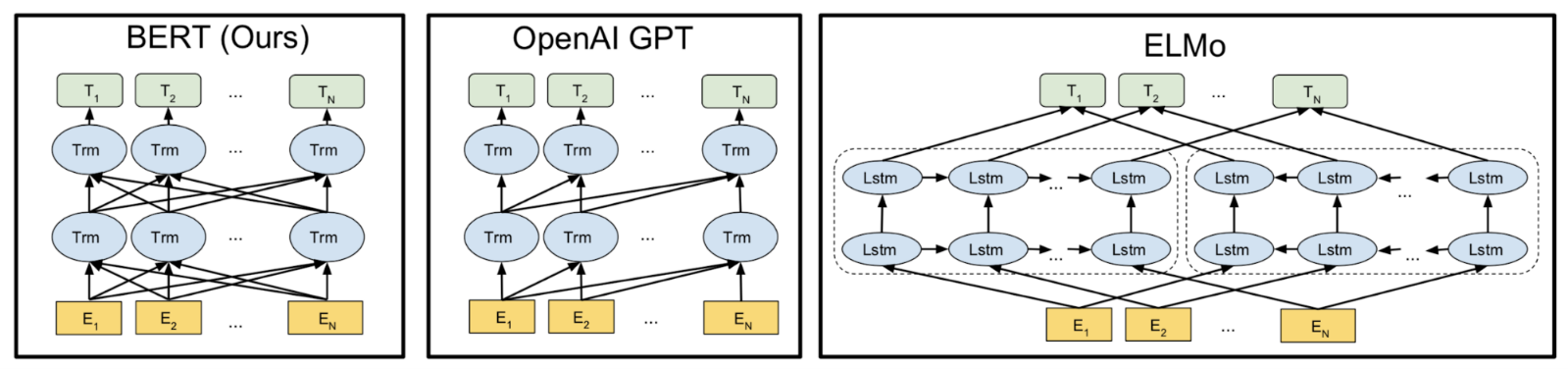
Task 2: **Next Sentence Prediction**(NSP)  
Many important downstream tasks such as Question Answering (QA) are based on the relationship between **two sentences**, which is not directly captured by language modeling.

BERT performs **state-of-the-art results**in many **NLP task** such as :

* Multi-Genre Natural Language Inference (MNLI)
* Quora Question Pairs (QQP)
* Question Natural Language Inference (QNLI)
* The Stanford Sentiment Treebank (SST-2)
* The Corpus of Linguistic Acceptability (CoLA)
* The Semantic Textual Similarity Benchmark (STS-B)
* Microsoft Research Paraphrase Corpus (MRPC)
* Recognizing Textual Entailment (RTE) etc.

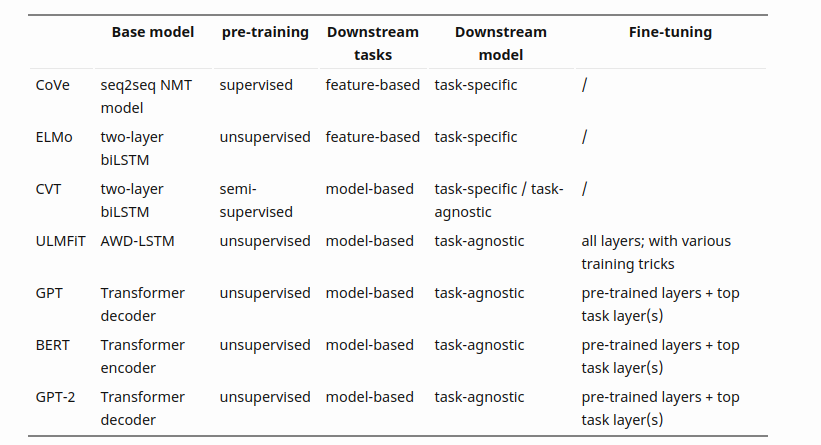
# **Comparison of BERT, GPT-2 and ELMo**

The comparisons between the model architectures are shown visually below. Note that in addition to the architecture differences, **BERT and OpenAI GPT**are **finetuning approaches**, while**ELMo** is a **feature-based approach**.



Comparison of BERT, OpenAI GPT and ELMo, (Image source: BERT [original paper](https://arxiv.org/abs/1810.04805))

* BERT and GPT are transformer-based architecture while ELMo is Bi-LSTM Language model.
* BERT is purely **Bi-directional**, GPT is **unidirectional**and ELMo is semi-bidirectional.
* GPT is trained on the **BooksCorpus (800M words);** BERT is trained on the **BooksCorpus (800M words)** and **Wikipedia (2,500M words)**.
* GPT uses a sentence separator ([SEP]) and classifier token ([CLS]) which are only introduced at fine-tuning time; BERT learns [SEP], [CLS] and sentence A/B embeddings during pre-training.
* GPT was trained for 1M steps with a batch size of 32,000 words; BERT was trained for 1M steps with a batch size of 128,000 words.
* GPT used the same learning rate of 5e-5 for all fine-tuning experiments; BERT chooses a task-specific fine-tuning learning rate which performs the best on the development set.



Summary of various models, Figure: From [Lil’Log](https://lilianweng.github.io/lil-log/2019/01/31/generalized-language-models.html" \t "_blank) by [Lilian](https://lilianweng.github.io/lil-log/contact.html).

## Unified Language Model Pre-training for Natural Language Understanding and Generation

## We propose a model called UNIfied pre-trained Language Model (UNILM).Fine-tune model that can do both NLU and NLG Pretraining with 3 LM tasks (unidirectional,bidirectional ,sequence-to-sequence prediction)

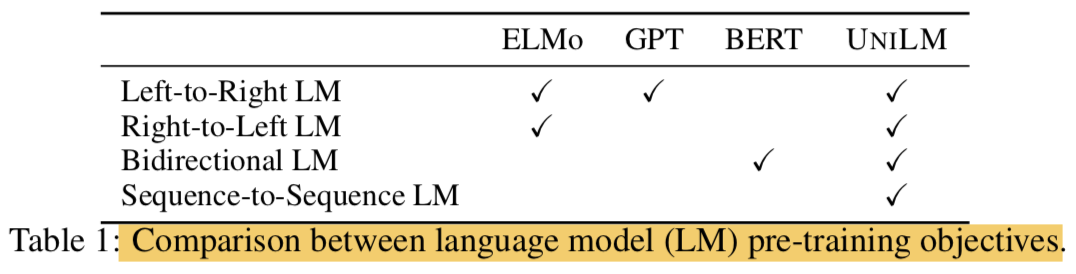
LM pre-training enables SOTA imaging in a variety of NLP tasks (substantially advanced)

Pre-trained LMs learn contextualized text representations by predicting words using context around words and use large amounts of text data.

Pre-trained LMs can be fine-tuned for downstream tasks Various prediction tasks and training objectives have been used depending on the type of pre-training LMs.

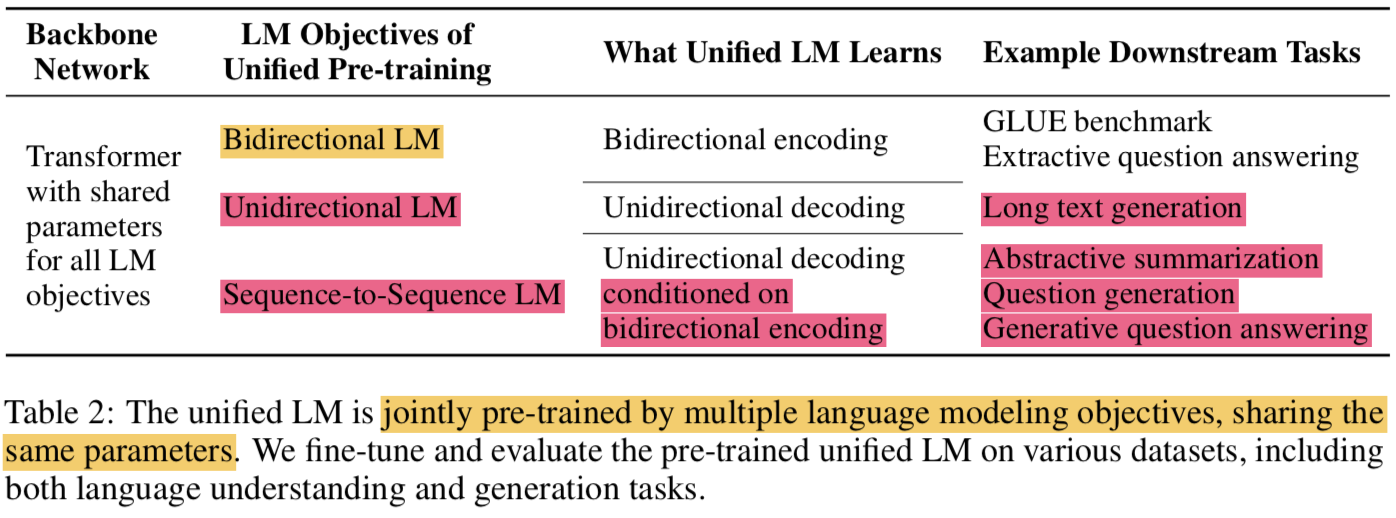
ELMo uses two unidirectional LMs. Because it learns left-to-right and right-to-left In the case of GPT, it is left-to-right

BERT is bidrectional LM



Although BERT is a very good model, it is difficult to apply to NLG task due to its characteristics.In this study, we propose a UNIfied pre-trained Language Model (UNILM) and apply the model to both NLU and NLG tasks.

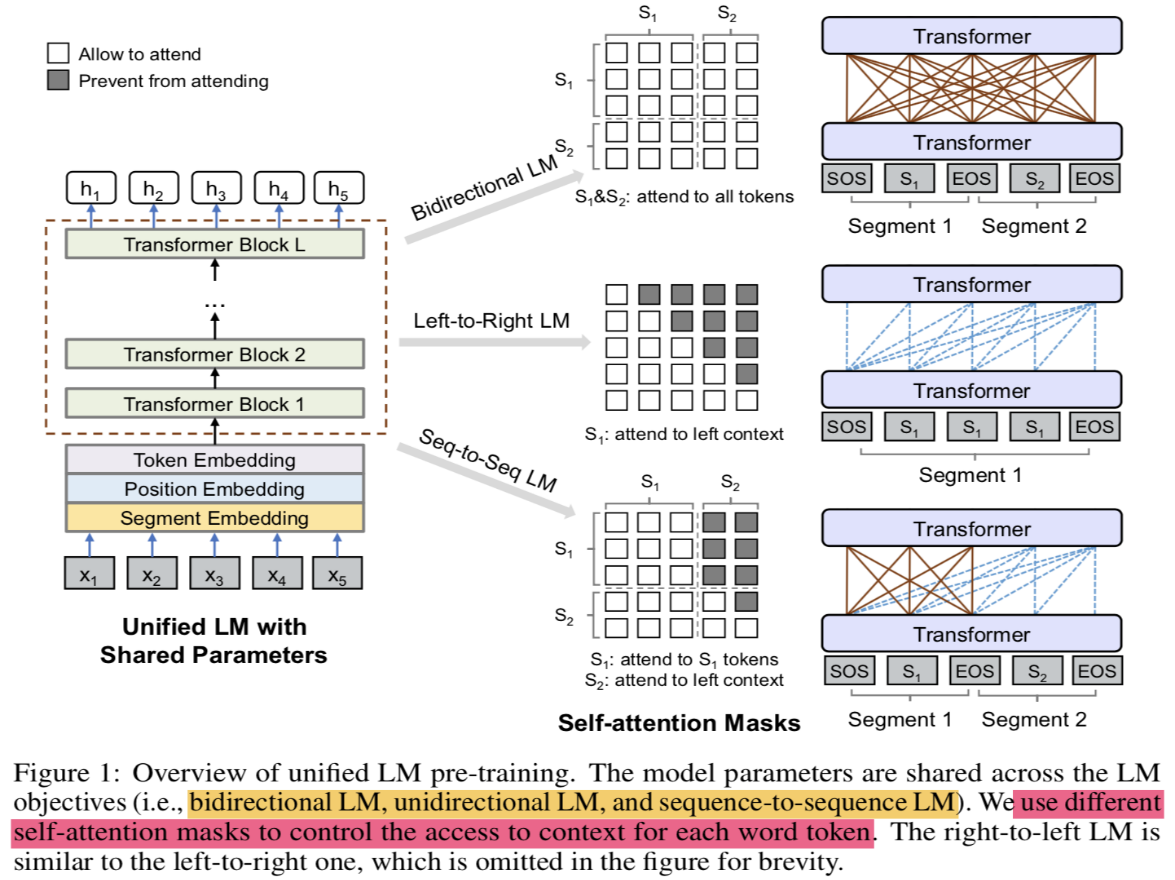
UNILM is a multi-layer transformer network and pre-trains and learns three types of unsupervised language modeling objectives at the same time.



* We have specifically designed some cloze tasks (filling in the blanks) and the context we see there is:
  + unidirectional LM
    - left-to-right unidirectional LM
      * context becomes all words on the left
    - right-to-left unidirectional LM
      * Conversely, all words on the right
  + bidirectional LM
    - context is all words around a word that includes both left and right directions
  + sequence-to-sequence LM
    - The context is the information of the encoder and all words before the word to be predicted in the target sequence.
* Similar to BERT, pre-trained UNILM can be fine-tuned ( with additional task-specific layers if necessary), but unlike BERT whose main NLU task is, UNILM is designed to use different self-attention masks to combine contexts of different types of LMs. And both NLG tasks
* The proposed UNILM has three advantages
  + The unified pre-training procedure allows a single Transformer LM to share model parameters and architecture for various types of LMs ( alleviating the need of separately training and hosting multiple LMs)
  + Learning different LM objectives that capture context differently prevents overfitting that can occur in any sing LM task, so this parameter sharing makes learned text representations more general.
  + UNILM uses sequence-to-sequence LM, which is a natural choice for NLG ( such as abstractive summarization and question generation)
  + According to the experimental results, the proposed model using the bidirectional encoder is comparable to the BERT in GLUE, and also gives good results in two extractive QA tasks (both NLU and NLG are good).

#### . **Unified Language Model Pre-training**

* Given input sequence x=x1⋅⋅⋅xnx=x1⋅⋅⋅xnFor, UNILM obtains contextualized vector representation for each token.
* In the pre-training phase, the shared Transformer network unidirectional LM, bidirectional LM, and sequence-to-sequence LMis learned as LM objectives.
* To this end, different masks were introduced for self-attention ( use masking to control how much context the token should attend)
* After pre-training, you can use fine-tuning with task-specific data for downstream tasks.



##### **Input Representation**

* Special token added
  + [SOS]: start-of-sequence
  + [EOS]: end-of-sequence
* input representation follows BERT format
* Tokenized with WordPiece
* Segment varies depending on the type of LM (see Figure 1)

##### Backbone Network: Multi-Layer Transformer

Using transformer that we explain before

##### **Pre-training Objectives**

* The parameters of UNILM are learned to minimize the cross-entropy loss computed using the predicted tokens and the original tokens
* LM type
  + Unidirectional LM:
    - use both left-to-right and right-to-left LM objectives
    - For instance, to predict the masked token of “x1x2x1x2 [MASK] x4x4”, only tokens x1,x2x1,x2 and itself can be used. This is done by using a triangular matrix for the self-attention mask MM
  + LM Bidirectional:
    - the self-attention mask MM is a zero matrix, so that every token is allowed to attend across all positions in the input sequence.
  + Sequence-to-Sequence LM:
    - the tokens in the first (source) segment can attend to each other from both directions within the segment, while the tokens of the second (target) segment can only attend to the leftward context in the target segment and itself, as well as all the tokens in the source segment
    - “[SOS] t1t2t1t2 [They] t3t4t5t3t4t5 [EOS]” into the model. While both t1 and t2 have access to the first four tokens, including [SOS] and [EOS], t4 can only attend to the first six tokens
    - In the case of sequence-to-sequence LM, it can be considered as learning bidirectional encoder and unidirectional decoder
* Next Sentence Prediction:
  + NSP is applied to Bidirectional LM

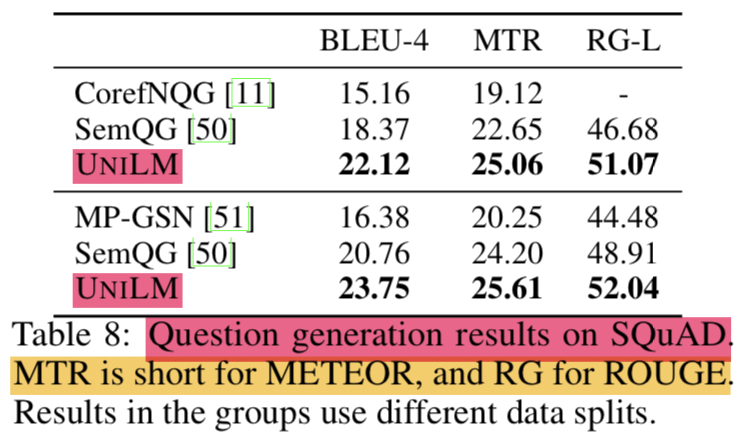
##### **Pre-training Setup**

* Per one training batch, 1/3 is bidrectional LM objective, 1/3 is seq2seq LM objective, and the other 1/3 is unidirectional LM objective (left-to-right, right-to-left)
* The structure of the model
  + gelu activation
  + 24-layer transformer (340M params)
    - with 1,024 hidden size
    - 16 attention heads
  + weight matrix of the softmax classifier is tied wtih token embeddings
  + BISRTLARGISBISRTLARGISInitialized with the weight of
* Corpus uses English Wikipedia and BookCorpus
* Vocab size: 28,996
* Maximum lengths of input seq: 512
* Masking Prob: 15%
  + 80%: [MASK]
  + 10%: random token
  + 10%: original token
* When masking, 80% is masked with one token and the remaining 20% ​​is masked with bigram or trigram.
* Optimizer:
  + Adam: b1=0.9,b2=0.999b1=0.9,b2=0.999
  + lr: 3e-5
  + warm up: first 40,000 steps (and linear decay)
  + weight decay: 0.01
* Dropout rate: 0.1
* Batch size: 330 ()
* pre-training procedure runs: 770,000 steps
* time: 7 hours for 10,000 steps
* GPUs: 8 Nvidia Telsa V100 32GB

##### **Fine-tuning on Downstream NLU and NLG Tasks**

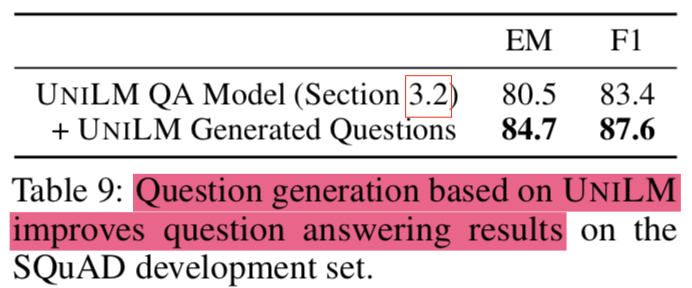
* For NLU task, fine-tuning like BERT
  + [SOS] vector for tokens hL1h1LAttach a randomly initialized softmax classifier to
* Similar to seq2seq task for NLG tasks
  + Notation
    - S1: source sequence
    - S2: target sequence
  + Pack into one
    - "[SOS] S1 [them], S2 [them]"
  + fine-tuning method:
    - Learn to match the tokens in the target sequence after randomly masking them at a specific rate ( masking some percentage of tokens in the target sequence at random, and learning to recover the masked words.)
    - The training objective is to maximize the likelihood of masked tokens given context
    - Masking is also good for [EOS], which is also used as a means to end generation, because the model can learn when to end the generation process ( It is worth noting that [EOS], which marks the end of the target sequence, can also be masked during fine-tuning, thus when this happens, the model learns when to emit [EOS] to terminate the generation process of the target sequence)
    - Use content selector to select salient phrases
    - 3 beam size)

##### **Question Generation**



* Generating a question when passage and answer are given
* Solved as seq2seq problem
  + 1st seg: input passage + answer
  + 2nd seg: generated question
* SQuAD 1.1 dataset is used as evaluation set
* As in the previous study, the original training set is divided into training and test sets, and the original dev set is left as it is.
* hyper params:
  + epoch: 10
  + batch size: 32
  + mask prob: 0.7
  + lr: 2e-5
  + label smoothing: 0.1

###### **GENERATED QUESTIONS IMPROVE QA**



* Creating a question with the question generation model (data augmentation) and learning again improves the performance of the existing question answering model.

Reference

* Iulian Vlad Serban, Alberto Garc´ıa-Duran, Caglar ´Gulcehre, Sungjin Ahn, Sarath Chandar, Aaron Courville, and Yoshua Bengio. 2016. Generating  
  factoid questions with recurrent neural networks: The 30m factoid question-answer corpus. In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: LongPapers). Association for Computational Linguistics, Berlin, Germany, pages 588–598.
* Michael Heilman. 2011. Automatic factual questiongeneration from text. Ph.D. thesis, Carnegie Mellon University.
* Yllias Chali and Sadid A. Hasan. 2015. Towards topicto-question generation. Comput. Linguist. 41(1):1–20
* Diederik Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In Proceedings
* Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In Proceedings of 3rdInternational Conference for Learning Representations. San Diego
* [1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. arXiv preprint
* arXiv:1607.06450, 2016.
* [2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly
* learning to align and translate. CoRR, abs/1409.0473, 2014.
* [3] Denny Britz, Anna Goldie, Minh-Thang Luong, and Quoc V. Le. Massive exploration of neural
* machine translation architectures. CoRR, abs/1703.03906, 2017.
* [4] Jianpeng Cheng, Li Dong, and Mirella Lapata. Long short-term memory-networks for machine
* reading. arXiv preprint arXiv:1601.06733, 2016.
* [5] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Fethi Bougares, Holger Schwenk,
* and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical
* machine translation. CoRR, abs/1406.1078, 2014.
* [6] Francois Chollet. Xception: Deep learning with depthwise separable convolutions. arXiv
* preprint arXiv:1610.02357, 2016.
* 10[7] Junyoung Chung, Çaglar Gülçehre, Kyunghyun Cho, and Yoshua Bengio. Empirical evaluation
* of gated recurrent neural networks on sequence modeling. CoRR, abs/1412.3555, 2014.
* [8] Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. Recurrent neural
* network grammars. In Proc. of NAACL, 2016.
* [9] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. Convolutional sequence to sequence learning. arXiv preprint arXiv:1705.03122v2, 2017.
* [10] Alex Graves. Generating sequences with recurrent neural networks. arXiv preprint
* arXiv:1308.0850, 2013.
* [11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern
* Recognition, pages 770–778, 2016.
* [12] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, and Jürgen Schmidhuber. Gradient flow in
* recurrent nets: the difficulty of learning long-term dependencies, 2001.
* [13] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. Neural computation,
* 9(8):1735–1780, 1997.
* [14] Zhongqiang Huang and Mary Harper. Self-training PCFG grammars with latent annotations
* across languages. In Proceedings of the 2009 Conference on Empirical Methods in Natural
* Language Processing, pages 832–841. ACL, August 2009.
* [15] Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. Exploring
* the limits of language modeling. arXiv preprint arXiv:1602.02410, 2016.
* [16] Łukasz Kaiser and Samy Bengio. Can active memory replace attention? In Advances in Neural
* Information Processing Systems, (NIPS), 2016.
* [17] Łukasz Kaiser and Ilya Sutskever. Neural GPUs learn algorithms. In International Conference
* on Learning Representations (ICLR), 2016.
* [18] Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, and Koray Kavukcuoglu. Neural machine translation in linear time. arXiv preprint arXiv:1610.10099v2,
* 2017.
* [19] Yoon Kim, Carl Denton, Luong Hoang, and Alexander M. Rush. Structured attention networks.
* In International Conference on Learning Representations, 2017.
* [20] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In ICLR, 2015.
* [21] Oleksii Kuchaiev and Boris Ginsburg. Factorization tricks for LSTM networks. arXiv preprint
* arXiv:1703.10722, 2017.
* [22] Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen
* Zhou, and Yoshua Bengio. A structured self-attentive sentence embedding. arXiv preprint
* arXiv:1703.03130, 2017.
* [23] Minh-Thang Luong, Quoc V. Le, Ilya Sutskever, Oriol Vinyals, and Lukasz Kaiser. Multi-task
* sequence to sequence learning. arXiv preprint arXiv:1511.06114, 2015.
* [24] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attentionbased neural machine translation. arXiv preprint arXiv:1508.04025, 2015.
* [25] Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated
* corpus of english: The penn treebank. Computational linguistics, 19(2):313–330, 1993.
* [26] David McClosky, Eugene Charniak, and Mark Johnson. Effective self-training for parsing. In
* Proceedings of the Human Language Technology Conference of the NAACL, Main Conference,
* pages 152–159. ACL, June 2006.
* 11[27] Ankur Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. A decomposable attention
* model. In Empirical Methods in Natural Language Processing, 2016.
* [28] Romain Paulus, Caiming Xiong, and Richard Socher. A deep reinforced model for abstractive
* summarization. arXiv preprint arXiv:1705.04304, 2017.
* [29] Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. Learning accurate, compact,
* and interpretable tree annotation. In Proceedings of the 21st International Conference on
* Computational Linguistics and 44th Annual Meeting of the ACL, pages 433–440. ACL, July
* 2006.
* [30] Ofir Press and Lior Wolf. Using the output embedding to improve language models. arXiv
* preprint arXiv:1608.05859, 2016.
* [31] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words
* with subword units. arXiv preprint arXiv:1508.07909, 2015.
* [32] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton,
* and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts
* layer. arXiv preprint arXiv:1701.06538, 2017.
* [33] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. Journal of Machine
* Learning Research, 15(1):1929–1958, 2014.
* [34] Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. End-to-end memory
* networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors,
* Advances in Neural Information Processing Systems 28, pages 2440–2448. Curran Associates,
* Inc., 2015.
* [35] Ilya Sutskever, Oriol Vinyals, and Quoc VV Le. Sequence to sequence learning with neural
* networks. In Advances in Neural Information Processing Systems, pages 3104–3112, 2014.
* [36] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna.
* Rethinking the inception architecture for computer vision. CoRR, abs/1512.00567, 2015.
* [37] Vinyals & Kaiser, Koo, Petrov, Sutskever, and Hinton. Grammar as a foreign language. In
* Advances in Neural Information Processing Systems, 2015.
* [38] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang
* Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google’s neural machine
* translation system: Bridging the gap between human and machine translation. arXiv preprint
* arXiv:1609.08144, 2016.
* [39] Jie Zhou, Ying Cao, Xuguang Wang, Peng Li, and Wei Xu. Deep recurrent models with
* fast-forward connections for neural machine translation. CoRR, abs/1606.04199, 2016.
* [40] Muhua Zhu, Yue Zhang, Wenliang Chen, Min Zhang, and Jingbo Zhu. Fast and accurate
* shift-reduce constituent parsing. In Proceedings of the 51st Annual Meeting of the ACL (Volume
* 1: Long Papers), pages 434–443. ACL, August 2013.
* [1] Alexei Baevski, Sergey Edunov, Yinhan Liu, Luke Zettlemoyer, and Michael Auli. Cloze-driven
* pretraining of self-attention networks. arXiv preprint arXiv:1903.07785, 2019.
* [2] Roy Bar-Haim, Ido Dagan, Bill Dolan, Lisa Ferro, and Danilo Giampiccolo. The second
* PASCAL recognising textual entailment challenge. In Proceedings of the Second PASCAL
* Challenges Workshop on Recognising Textual Entailment, 01 2006.
* [3] Luisa Bentivogli, Ido Dagan, Hoa Trang Dang, Danilo Giampiccolo, and Bernardo Magnini.
* The fifth pascal recognizing textual entailment challenge. In In Proc Text Analysis Conference
* (TAC-09), 2009.
* [4] Ziqiang Cao, Wenjie Li, Sujian Li, and Furu Wei. Retrieve, rerank and rewrite: Soft template
* based neural summarization. In Proceedings of the 56th Annual Meeting of the Association for
* Computational Linguistics, pages 152–161, Melbourne, Australia, July 2018. Association for
* Computational Linguistics.
* [5] Daniel Cer, Mona Diab, Eneko Agirre, Inigo Lopez-Gazpio, and Lucia Specia. Semeval-2017
* task 1: Semantic textual similarity-multilingual and cross-lingual focused evaluation. arXiv
* preprint arXiv:1708.00055, 2017.
* [6] Zewen Chi, Li Dong, Furu Wei, Wenhui Wang, Xian-Ling Mao, and Heyan Huang. Crosslingual natural language generation via pre-training. ArXiv, abs/1909.10481, 2019.
* [7] Ido Dagan, Oren Glickman, and Bernardo Magnini. The pascal recognising textual entailment challenge. In Proceedings of the First International Conference on Machine Learning
* Challenges: Evaluating Predictive Uncertainty Visual Object Classification, and Recognizing
* Textual Entailment, MLCW’05, pages 177–190, Berlin, Heidelberg, 2006. Springer-Verlag.
* [8] Andrew M Dai and Quoc V Le. Semi-supervised sequence learning. In Advances in Neural
* Information Processing Systems 28, pages 3079–3087. Curran Associates, Inc., 2015.
* [9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of
* deep bidirectional transformers for language understanding. CoRR, abs/1810.04805, 2018.
* [10] William B Dolan and Chris Brockett. Automatically constructing a corpus of sentential paraphrases. In Proceedings of the Third International Workshop on Paraphrasing (IWP2005),
* 2005.
* [11] Xinya Du and Claire Cardie. Harvesting paragraph-level question-answer pairs from Wikipedia.
* In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics,
* pages 1907–1917, Melbourne, Australia, July 2018. Association for Computational Linguistics.
* [12] Xinya Du, Junru Shao, and Claire Cardie. Learning to ask: Neural question generation for
* reading comprehension. In Proceedings of the 55th Annual Meeting of the Association for
* Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long
* Papers, pages 1342–1352, 2017.
* [13] Sergey Edunov, Alexei Baevski, and Michael Auli. Pre-trained language model representations
* for language generation. CoRR, abs/1903.09722, 2019.
* [14] Michel Galley, Chris Brockett, Xiang Gao, Jianfeng Gao, and Bill Dolan. Grounded response
* generation task at dstc7. In AAAI Dialog System Technology Challenges Workshop, 2019.
* [15] Jianfeng Gao, Michel Galley, Lihong Li, et al. Neural approaches to conversational ai. Foundations and Trends in Information Retrieval, 13(2-3):127–298, 2019.
* [16] Sebastian Gehrmann, Yuntian Deng, and Alexander Rush. Bottom-up abstractive summarization.
* In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing,
* pages 4098–4109, Brussels, Belgium, October-November 2018. Association for Computational
* Linguistics.
* 10[17] Danilo Giampiccolo, Bernardo Magnini, Ido Dagan, and Bill Dolan. The third PASCAL
* recognizing textual entailment challenge. In Proceedings of the ACL-PASCAL Workshop
* on Textual Entailment and Paraphrasing, pages 1–9, Prague, June 2007. Association for
* Computational Linguistics.
* [18] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (GELUs). arXiv preprint
* arXiv:1606.08415, 2016.
* [19] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. In Proceedings of the 56th Annual Meeting of the Association for Computational
* Linguistics, pages 328–339, Melbourne, Australia, July 2018. Association for Computational
* Linguistics.
* [20] Minghao Hu, Furu Wei, Yuxing Peng, Zhen Huang, Nan Yang, and Ming Zhou. Read + verify:
* Machine reading comprehension with unanswerable questions. CoRR, abs/1808.05759, 2018.
* [21] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. arXiv preprint
* arXiv:1412.6980, 2014.
* [22] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In 3rd
* International Conference on Learning Representations, San Diego, CA, 2015.
* [23] Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander Rush. OpenNMT:
* Open-source toolkit for neural machine translation. In Proceedings of ACL 2017, System
* Demonstrations, pages 67–72, Vancouver, Canada, July 2017. Association for Computational
* Linguistics.
* [24] Hector Levesque, Ernest Davis, and Leora Morgenstern. The winograd schema challenge.
* In Thirteenth International Conference on the Principles of Knowledge Representation and
* Reasoning, 2012.
* [25] Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In Text Summarization Branches Out: Proceedings of the ACL-04 Workshop, pages 74–81, Barcelona, Spain, July
* 2004. Association for Computational Linguistics.
* [26] Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. Multi-task deep neural networks
* for natural language understanding. CoRR, abs/1901.11504, 2019.
* [27] Yang Liu. Fine-tune BERT for extractive summarization. CoRR, abs/1903.10318, 2019.
* [28] Romain Paulus, Caiming Xiong, and Richard Socher. A deep reinforced model for abstractive
* summarization. CoRR, abs/1705.04304, 2018.
* [29] Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton
* Lee, and Luke Zettlemoyer. Deep contextualized word representations. In Proceedings of
* the 2018 Conference of the North American Chapter of the Association for Computational
* Linguistics: Human Language Technologies, pages 2227–2237, New Orleans, Louisiana, June
* 2018. Association for Computational Linguistics.
* [30] Lianhui Qin, Michel Galley, Chris Brockett, Xiaodong Liu, Xiang Gao, Bill Dolan, Yejin
* Choi, and Jianfeng Gao. Conversing by reading: Contentful neural conversation with ondemand machine reading. In Proceedings of the 57th Annual Meeting of the Association
* for Computational Linguistics, pages 5427–5436, Florence, Italy, July 2019. Association for
* Computational Linguistics.
* [31] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language
* understanding by generative pre-training. 2018.
* [32] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language
* models are unsupervised multitask learners. 2019.
* [33] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. In Proceedings of the 2016 Conference on Empirical
* Methods in Natural Language Processing, pages 2383–2392, Austin, Texas, November 2016.
* Association for Computational Linguistics.
* 11[34] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don’t know: Unanswerable
* questions for SQuAD. In Proceedings of the 56th Annual Meeting of the Association for
* Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 2: Short
* Papers, pages 784–789, 2018.
* [35] Siva Reddy, Danqi Chen, and Christopher D. Manning. CoQA: A conversational question
* answering challenge. Transactions of the Association for Computational Linguistics, 7:249–266,
* March 2019.
* [36] Alexander M. Rush, Sumit Chopra, and Jason Weston. A neural attention model for abstractive
* sentence summarization. In Proceedings of the 2015 Conference on Empirical Methods in
* Natural Language Processing, pages 379–389, Lisbon, Portugal, September 2015. Association
* for Computational Linguistics.
* [37] Abigail See, Peter J. Liu, and Christopher D. Manning. Get to the point: Summarization with
* pointer-generator networks. In Proceedings of the 55th Annual Meeting of the Association for
* Computational Linguistics, pages 1073–1083, Vancouver, Canada, July 2017. Association for
* Computational Linguistics.
* [38] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng,
* and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment
* treebank. In Proceedings of the 2013 conference on empirical methods in natural language
* processing, pages 1631–1642, 2013.
* [39] Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. Mass: Masked sequence to
* sequence pre-training for language generation. arXiv preprint arXiv:1905.02450, 2019.
* [40] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In Proceedings of the IEEE Conference
* on Computer Vision and Pattern Recognition, pages 2818–2826, 2016.
* [41] Y Tam, Jiachen Ding, Cheng Niu, and Jie Zhou. Cluster-based beam search for pointer-generator
* chatbot grounded by knowledge. In AAAI Dialog System Technology Challenges Workshop,
* 2019.
* [42] Wilson L Taylor. Cloze procedure: A new tool for measuring readability. Journalism Bulletin,
* 30(4):415–433, 1953.
* [43] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez,
* Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Advances in Neural Information Processing Systems 30, pages 5998–6008. Curran Associates, Inc., 2017.
* [44] Alex Wang and Kyunghyun Cho. BERT has a mouth, and it must speak: BERT as a markov
* random field language model. CoRR, abs/1902.04094, 2019.
* [45] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman.
* GLUE: A multi-task benchmark and analysis platform for natural language understanding. In
* International Conference on Learning Representations, 2019.
* [46] Alex Warstadt, Amanpreet Singh, and Samuel R Bowman. Neural network acceptability
* judgments. arXiv preprint arXiv:1805.12471, 2018.
* [47] Adina Williams, Nikita Nangia, and Samuel Bowman. A broad-coverage challenge corpus for
* sentence understanding through inference. In Proceedings of the 2018 Conference of the North
* American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 1112–1122, New Orleans, Louisiana, June 2018. Association for Computational
* Linguistics.
* [48] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang
* Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah,
* Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo,
* Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason
* Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey
* Dean. Google’s neural machine translation system: Bridging the gap between human and
* machine translation. CoRR, abs/1609.08144, 2016.
* 12[49] Dani Yogatama, Cyprien de Masson d’Autume, Jerome Connor, Tomas Kocisky, Mike
* Chrzanowski, Lingpeng Kong, Angeliki Lazaridou, Wang Ling, Lei Yu, Chris Dyer, and
* Phil Blunsom. Learning and evaluating general linguistic intelligence. arXiv preprint
* arXiv:1901.11373, 2019.
* [50] Shiyue Zhang and Mohit Bansal. Addressing semantic drift in question generation for semisupervised question answering. CoRR, abs/1909.06356, 2019.
* [51] Yao Zhao, Xiaochuan Ni, Yuanyuan Ding, and Qifa Ke. Paragraph-level neural question
* generation with maxout pointer and gated self-attention networks. In Proceedings of the 2018
* Conference on Empirical Methods in Natural Language Processing, pages 3901–3910, Brussels,
* Belgium, October-November 2018. Association for Computational Linguistics.
* [52] Qingyu Zhou, Nan Yang, Furu Wei, Chuanqi Tan, Hangbo Bao, and Ming Zhou. Neural question
* generation from text: A preliminary study. In Xuanjing Huang, Jing Jiang, Dongyan Zhao,
* Yansong Feng, and Yu Hong, editors, Natural Language Processing and Chinese Computing,
* pages 662–671. Springer International Publishing, 2018.
* [53] Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba,
* and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by
* watching movies and reading books. In Proceedings of the IEEE International Conference on
* Computer Vision, pages 19–27, 2015.