UNIVERSITY OF ZAGREB
**FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING**

**SEMINAR**

# Advancements in Language Model Pretraining Based on Transformers Architecture: A Comparative Overview

*Jerko Šegvić*
Mentor: *prof. dr. sc. Jan Šnajder, mag. ing. David Dukić*

Zagreb, February 2024.

# CONTENTS

# 1. Introduction

Transformer models are the representatives of the family of deep learning models based on the self-attention mechanism first proposed in Vaswani et al. (2017). After their introduction, they became highly popular in many fields of Artificial Intelligence including Natural Language Processing (in which they were first applied). Today all state-of-the-art models in NLP are based on transformer architecture. Those models are most often trained on large amounts of data with a lot of computational resources. However, it turned out that the pre-trained transformer-based language models can be easily fine-tuned for any downstream task accompanied with annotated data, which makes them feasible for adapting on consumer-grade hardware. The models described in this work are the first transformer proposed in Vaswani et al. (2017), the GPT model proposed in Radford et al. (2018), BERT proposed in Devlin et al. (2019), XLNet introduced in Yang et al. (2019), and ELECTRA introduced in Clark et al. (2020). For each, we state their main contributions and the results that they achieved.

# 2. Attention is all you need (Vaswani et al. (2017))

The Transformer was proposed in Vaswani et al. (2017) for the problem of machine language translation. The models for such a problem are usually trained on datasets consisting of pairs of sentences written in different languages (in the paper, English-German and English-French datasets). The results showed that the model from the paper outperformed all other state-of-the-art models on those datasets, often by a large margin, and that training was much cheaper. The paper's main contribution was the proposition of using the self-attention mechanism in favor of convolutional or recurrent layers. Self-attention turned out to be much more suitable for feature extraction from text than other mechanisms and also more parallelizable.

## 2.1. High-level architecture of Transformer

In Figure 2.1 is ilustrated high-level arhitecture of Transformer and self-attention mechanism. The main components of the model are the encoder and decoder. The idea is to encode the input text to some abstract representation with an encoder and then generate the text in the target language with the decoder. Both encoder and decoder are made of multiple identical layers (in the paper, six encoder and six decoder layers), often called encoder and decoder blocks, respectively.

The input text is represented using word embedding and positional encoding. The word embedding is a standard procedure of representing a token as a fixed-sized vector (it was already used in older models at the time). Positional encoding is a function that gives each position in the text a unique vector of the dimension, the same as the word embedding. The input fed into the model, the positional embedded token, is defined as a sum of its positional encoding and word embedding. In that way, embedding has both information about the word and the position of the word in the sequence. The positional encoding function can be either learned during the training or fixed. In the paper, the authors pointed out that they got similar results using fixed trigonometric functions and learned functions, so they used fixed ones for

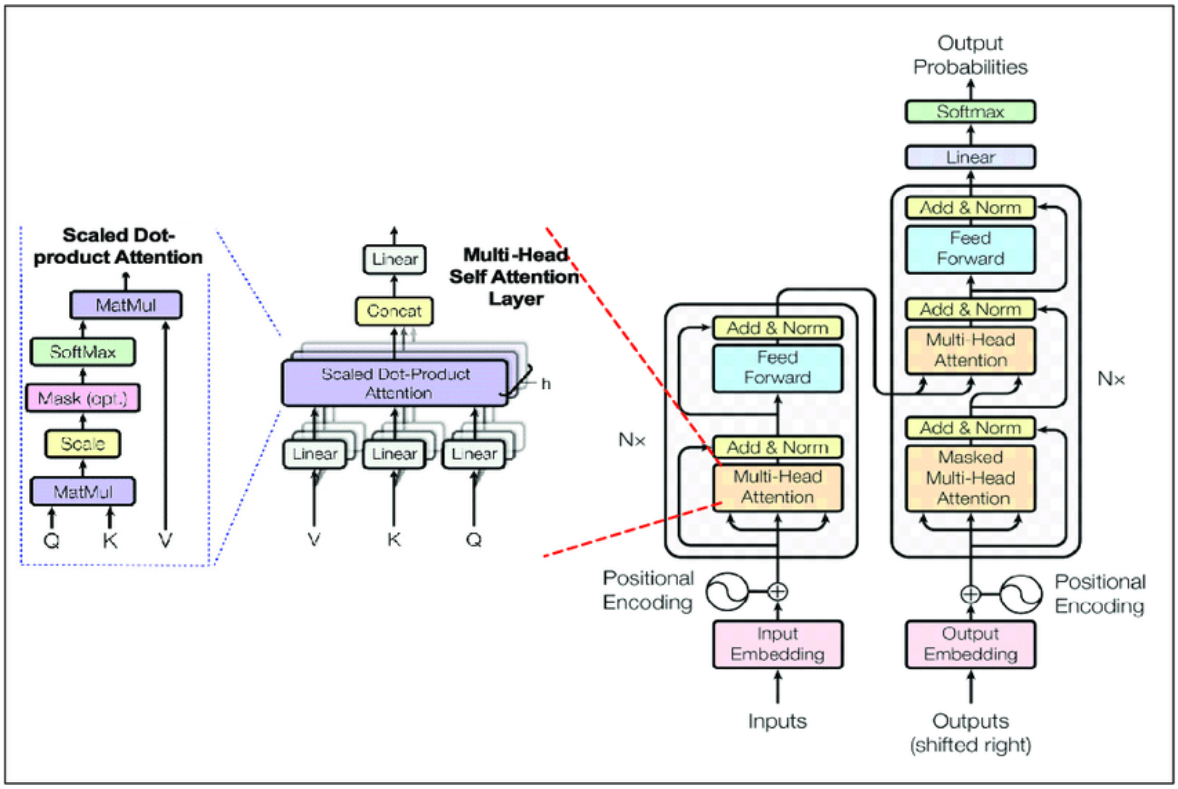


**Figure 2.1:** Architecture of Transformer

ResearchGate

positional encoding embeddings.

Each encoder block has two main parts: a multi-head self-attention mechanism and a fully connected feed-forward network. Each of those is followed by a residual connection with its input and normalization layer so that its output is equal to $LayerNorm(x + Sublayer(x))$, where $Sublayer$ is either a self-attention mechanism or feed-forward network and $x$ is an input.

Each decoder block is similar to encoder blocks. In addition to the parts previously described, it also has a masked multi-head self-attention mechanism. Masked multi-head self-attention is used to prevent current positions from attending subsequent positions during the training procedure. The multi-head self-attention in the decoder attends to the last hidden states of the last encoder block. The last hidden state of the last decoder block is fed into the linear layer and then into the softmax layer, the output of which presents probabilities of generating the next token.

## 2.2. Self-attention mechanism

The self-attention mechanism is the most important part of the Transformer. It enables the model to weigh the importance of different words in a sequence concerning a specific token

in that sequence. It also enables better parallelization than older models with recurrent neural networks (RNNs). Unlike the models with RNNs, which process tokens one after the other, self-attention, together with positional embedding, enables simultaneous processing of all tokens in a sequence, resulting in faster training.

The self-attention mechanism assumes that every token has three vector representations: query, key, and value. Those representations are projections of the embedding vectors into different subspaces. The projection matrices are learned during the training. When calculating the importance of one token to another, we will compare (in the literature model for that comparison is often called *aligment model*) the query representation of the first token to the key representation of the second token (and get some scalar value of *importance*), and then multiply the result with the value representation of the second vector. When it's done on all tokens in the sequence, we will get a weighted sum of value representation vectors as a result. In the paper, authors used Scaled dot-product attention, which means that they measured the importance of one token to another as a dot product of their query and key representations (as it is shown in Figure 2.1).

To boost the power of the self-attention mechanism, authors created a multi-head self-attention layer. This layer is a combination of multiple self-attention layers (in the paper, authors used eight heads), every with different matrices for query, key, and value representations. Authors suggest that multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions.

## 2.3. Results

The model was trained on the WMT 2014 English-to-German translation task and the WMT 2014 English-to-French translation task. On the English-to-German task, the Transformer (big) model set a new state-of-the-art BLEU score of $28.4$, surpassing all previous models by over $2.0$ BLEU. The base model also outperformed all prior models with a much cheaper training cost. Similarly, on the English-to-French task, the big model achieved a BLEU score of $41.0$, surpassing all previous models with less than $\frac{1}{4}$ of the training cost.

# 3. Improving language understanding by generative pretraining (Radford et al. (2018))

The GPT (Generative pretrained transformer) is a transformer decoder-only model proposed by OpenAI in Radford et al. (2018). The model was pretrained on a language modeling task. The main difference between GPT and Transformer is that GPT has only the decoder part of Transformer because GPT's pretraining objective isn't language translation but auto-regressive language modeling (so there is no need to encode the input text with a different sub-model). The main contribution of the paper was the idea that after pretraining on a large amount of data generatively for text generation, the model can be easily fine-tuned on any discriminative downstream task and achieve bettter performance than fine-tuning from random intialization.

## 3.1. High-level architecture of GPT

In figure 3.1 is ilustrated high level arhitecture of GPT model. The model is made of multiple decoder blocks, and each decoder block is similar to the ones proposed in Vaswani et al. (2017) (in the paper, 12 decoder blocks). The mechanism of token embedding is the same as in the original Transformer model. The positional embedding mechanism is also the same as in the original Transformer model.

Each decoder block has two main parts: masked multi-head self-attention and a fully connected feed-forward network (note that it has just one layer of attention, unlike the decoder blocks in the Transformer). The masked multi-head self-attention is the same as in the original Transformer. In the feed-forward network, the authors used the GeLu activation function, unlike the authors of Transformer, who used ReLu.

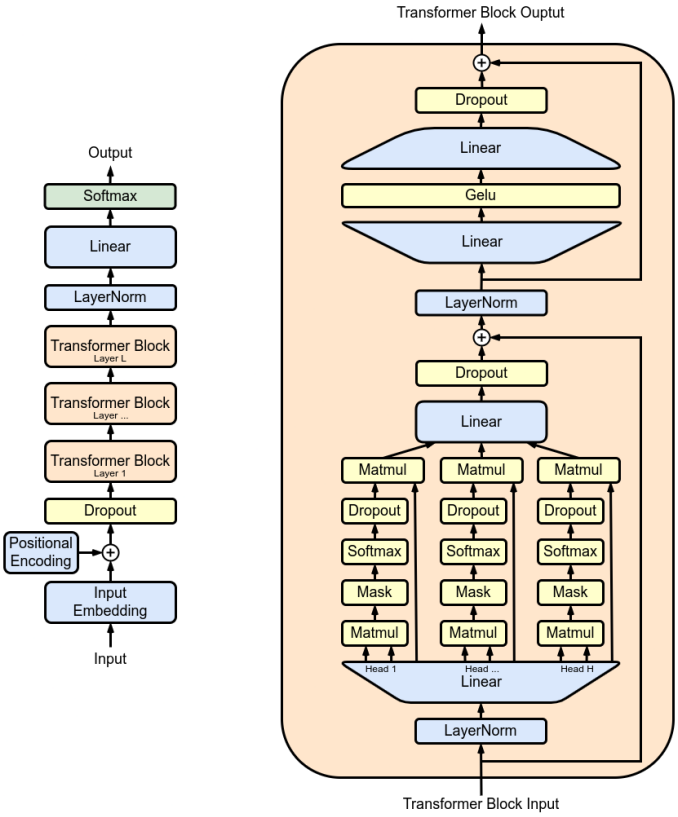


**Figure 3.1:** Architecture of GPT

Wikipedia

## 3.2. Fine tuning on downstream tasks

Figure 3.2 illustrates how GPT is adapted to different training objectives for any downstream task. The GPT model is used as a feature extractor, and linear and softmax layers are added at the top. The last hidden state of the last token of the input sequence is the input to the added layers. That way, the only new parameters that need to be learned from scratch during fine-tuning are those in the added linear layer, and training is much cheaper for specific downstream tasks.

During the pretraining, the loss function used was cross-entropy loss. For a sequence of tokens $U = u_1, u_2, ..., u_n$, the objective was maximizing the log-likelihood

$$L_1(U) = \sum_i \log P(u_i|u_{i-k}, ..., u_{i-1}; \theta)$$

where k is the size of the context window (the maximum number of tokens model can work with) and $\theta$ are model parameters. Maximizing that likelihood is equal to minimizing cross-entropy loss.

During the fine-tuning, it is assumed that the target dataset is made of sequence $X =$

$x_1, ..., x_m$ and label $y$ pairs. That task gives us the objective of maximizing the likelihood

$$L_2(C) = \sum_{(X,y)} \log P(y|x_1, ..., x_m; \theta)$$

where C is the target dataset. The maximization of that likelihood is equal to the minimization cross-entropy loss. The authors suggested that defining the objective during the fine-tuning as the sum of the objective of the pretraining task and the objective of the target task resulted in better generalization and faster training. With that in mind, the final objective for fine-tuning is equal to the likelihood

$$L(C) = L_2(C) + \lambda * L_1(c)$$

where $C$ is the target dataset, $L_1$ is the likelihood of the pretraining task, $L_2$ is the likelihood of the dataset and $\lambda$ is hyperparameter which determines the impact of pretraining likelihood.

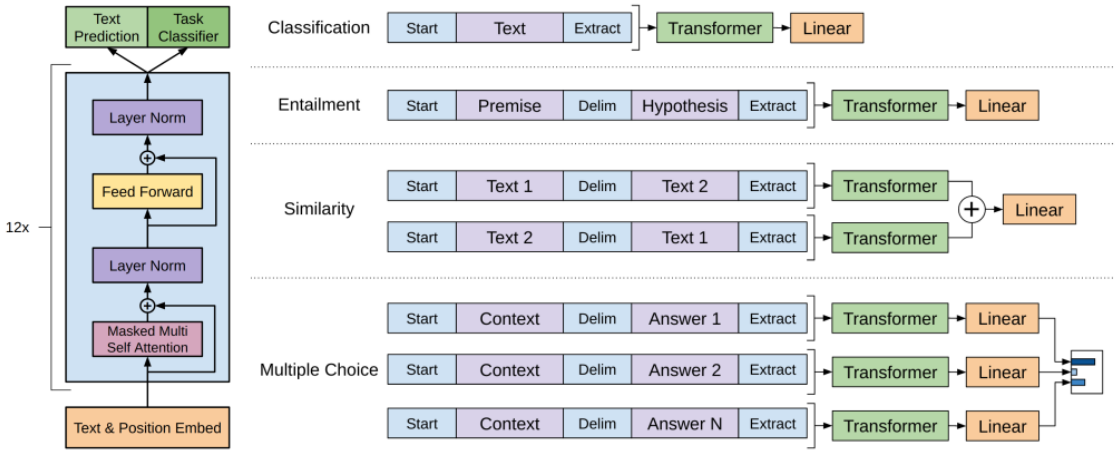


**Figure 3.2:** GPT fine-tuning

## 3.3. Results

The model was pretrained on BooksCorpus dataset. It is made of over 7000 unique unpublished books from various genres. The model was fine-tuned on twelve different datasets with different target tasks, including classification, sentence similarity, question answering, and natural language interference. The GPT set new state-of-the-art results on four out of five natural language interference datasets, all four question-answering datasets, two out of three semantic similarity datasets, and one out of two classification datasets.

# 4. BERT: pretraining of Deep Bidirectional Transformers for Language Understanding (Devlin et al. (2019))

The BERT (Bidirectional Encoder Representations from Transformers) is a transformer encoder-only model proposed in Devlin et al. (2019). The authors introduced a new pretraining task called masked language modeling, which turned out to be more effective than auto-regressive language modeling in GPT for natural language understanding and the models of similar size. The main contribution of the paper is the bidirectional encoding of the input text. Authors argue that because other language models are limited to unidirectional encoding, they can't extract as good features as BERT.

## 4.1. High-level architecture of BERT

Figure 4.1 illustrates the high-level architecture of BERT. The model is made of multiple encoder blocks, and each encoder block is identical to the ones proposed in Vaswani et al. (2017) (the authors created two versions of the model, one with 12 encoder blocks and one with 24 encoder blocks). In Figure 4.1, every box inside the *BERT box* represents one hidden state of one token. The last hidden state of each token is labeled as $T_i$. The model is fed with a pair of sequences separated by a special $[SEP]$ token, and the beginning of the first sentence is marked with a special $[CLS]$ token.

## 4.2. pretraining tasks

Maksed language modeling is the pretraining task which can be described as guessing the missing words in text (also known as the cloze task). The pretraining involves replacing some amount of tokens with a special $[MASK]$ token, and the goal is to predict what is the replaced token. The last hidden state of every position represents the probability distribution of the token on that position (after it is fed into the softmax layer). If the token is not

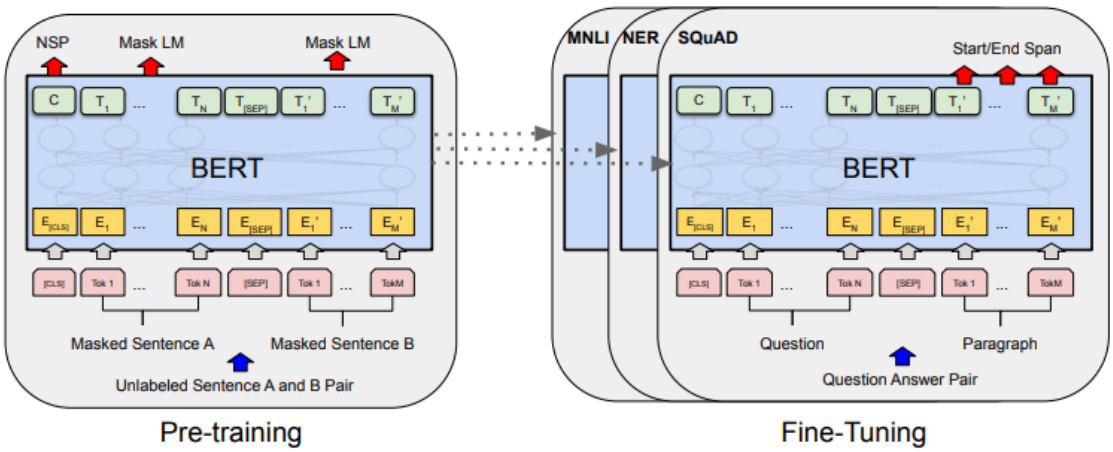


**Figure 4.1:** Architecture of BERT

replaced, the problem is trivial, so the loss is calculated only on masked tokens. The authors masked $15\%$ of tokens in all experiments. Although, in this way, the model was allowed to obtain a bidirectional context, the authors pointed out that adding $[MASK]$ tokens created a mismatch between pretraining and fine-tuning (since those tokens don't appear during fine-tuning). To overcome this issue, the authors proposed a different masking strategy. They again chose $15\%$ of tokens but replaced them with the $[MASK]$ token only $80\%$ of time, replaced them with a random token $10\%$ of the time, and left them unchanged $10\%$ of the time.

Next sentence prediction (NSP) is the pretraining task in which the model has to decide whether the two sentences are coming one after another in the corpus. Because of that, the input in BERT is divided with $[SEP]$ token. The last hidden state of the $[CLS]$ token is used to predict if the two sentences are coming one after another or not. During the pretraining, the authors created a dataset so that $50\%$ of time the second sentence is coming after the first sentence in the corpus and $50\%$ of time is not.

## 4.3. Fine-tuning on downstream tasks

Figure 4.2 illustrates how BERT is fine-tuned on downstream tasks. For classification tasks (involving one sentence or two sentences), the last hidden state of the $[CLS]$ token is used for classifying the output. For tasks of question answering (such as SQuAD, which contains a question and a paragraph, and the model needs to determine where in the paragraph is the answer to the question), the inputs of the model are question and paragraph, separated by $[SEP]$ token. The probabilities that $i$-th token in a paragraph is the start or the end of the

answer to the question are equal to

$$P(start = i) = \frac{\exp ST_i}{\sum_j \exp ST_j}$$

and

$$P(end = i) = \frac{\exp ET_i}{\sum_j \exp ET_j}$$

where $S$ and $E$ are vectors for start and end which are learned during the fine-tuning. In other words, all the last hidden states of tokens in the paragraph are multiplied by the vectors $S$ and $E$, and those values are fed into the softmax layer to get probabilities that the $i$-th token is the start or the end of the answer.

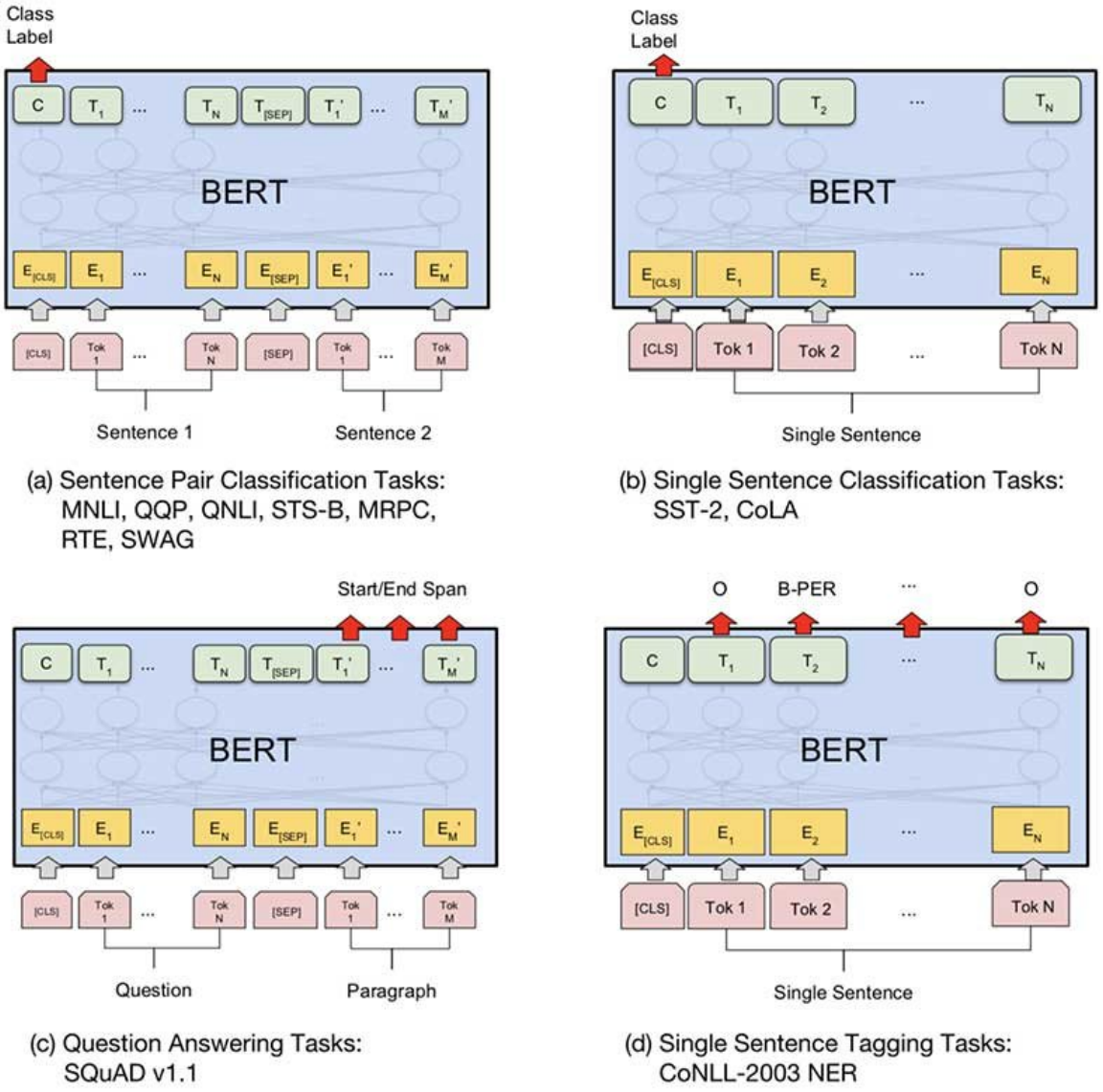


**Figure 4.2:** BERT fine-tuning

## 4.4. Results

The model was pretrained on BooksCorpus and Wikipedia datasets. The Wikipedia dataset is made of articles from Wikipedia and contains around $2500$M words. The BookCorpus contains around $800$M words. The model was fine-tuned on the GLUE (General Language Understanding Evaluation) benchmark datasets (eight of them), SQuAD v1.1 and SQuAD v2.0 question answering datasets, and SWAG (Situations With Adversarial Generations) dataset. BERT set the new state-of-the-art result on all of them, often by a large margin.

# 5. XLNet: Generalized Autoregressive Pretraining for Language Understanding (Yang et al. (2019))

XLNet is a transformer encoder-only model proposed in Yang et al. (2019). The model creation was inspired by BERT and GPT. The authors generalized BERT's and GPT's pretraining objective (masked language modeling and auto-regressive language modeling) and called it permutation Language Modeling (PLM). The main contribution of the paper is the new pretraining objective. The authors pointed out that the model outperformed BERT on twenty downstream tasks, often by a large margin.

## 5.1. pretraining task

Auto-Regressive Language Modeling (AR LM) is a pretraining task used in generative models like GPT. Given a text sequence $X = x_1, x_2, ..., x_T$ the objective of AR language modeling is maximizing the likelihood

$$\max_{\theta} \log p(X|\theta) = \sum_{t=1}^{T} \log p(x_t|X_{<t}, \theta) = \sum_{t=1}^{T} \log \frac{\exp\left(h_\theta(X_{1:t-1})^T e(x_t)\right)}{\sum_{x'} \exp\left(h_\theta(X_{1:t-1})^T e(x')\right)}$$

where $h_\theta(X_{1:t-1})$ is a context representation produced by the model (the last hidden state) with parameters $\theta$ and $e(x)$ is embedding of $x$ (*one hot* encoding).

Masked Language Modeling (MLM) is a pretraining task introduced in BERT. When training BERT on a text sequence $X$, $X$ is firstly corrupted so that some amount (in paper $15\%$) of tokens in $X$ are replaced with special $[MASK]$ token. The training objective of MLM is maximizing the likelihood

$$\max_{\theta} \log p(\bar{X}|\hat{X}, \theta) \approx \sum_{t=1}^{T} m_t \log p(x_t|\hat{X}, \theta) = \sum_{t=1}^{T} m_t \log \frac{\exp\left(H_\theta(\hat{X})_t^T e(x_t)\right)}{\sum_{x'} \exp\left(H_\theta(\hat{X})_t^T e(x')\right)}$$

where $\bar{X}$ is set of symbols that are masked, $\hat{X}$ is masked version of $X$, $m_t$ is a indicator variable if $x_t$ is masked (it's equal to $1$ if it is and $0$ otherwise), $H_\theta$ model that maps token

sequence of length $T$ into the sequence of vectors (i.e. sequence of the last hidden states) of length $T$, and $(H_\theta(X))_t$ is the last hidden state of the $t$-th token.

The authors compared those two objectives with the reference to the independence assumption, the presence of noise, and the context dependency. The MLM's weakness and ARLM's advantage is the independence assumption in the likelihood factorization. The factorization of $p(\bar{x}|\hat{x})$ in MLM's objective is based on the assumption that all masked tokens are separately reconstructed, while ARLM's objective is factorized using the product rule. The MLM's other weakness is the presence of noise during the pretraining. The special $[MASK]$ tokens, present only during the pretraining, create a discrepancy between pretraining and fine-tuning tasks, while ARLM doesn't suffer from this issue (because the ARLM is not adding any special tokens to the text during the pretraining). The MLM's advantage over ARLM is the ability to capture bidirectional context. While ARLM's last hidden states are created based only on right-to-left context, MLM captures context from both right to left and left to right (because of that, MLM transformer models use *ordinary* self-attention and ARLM transformer models use masked self-attention mechanism).

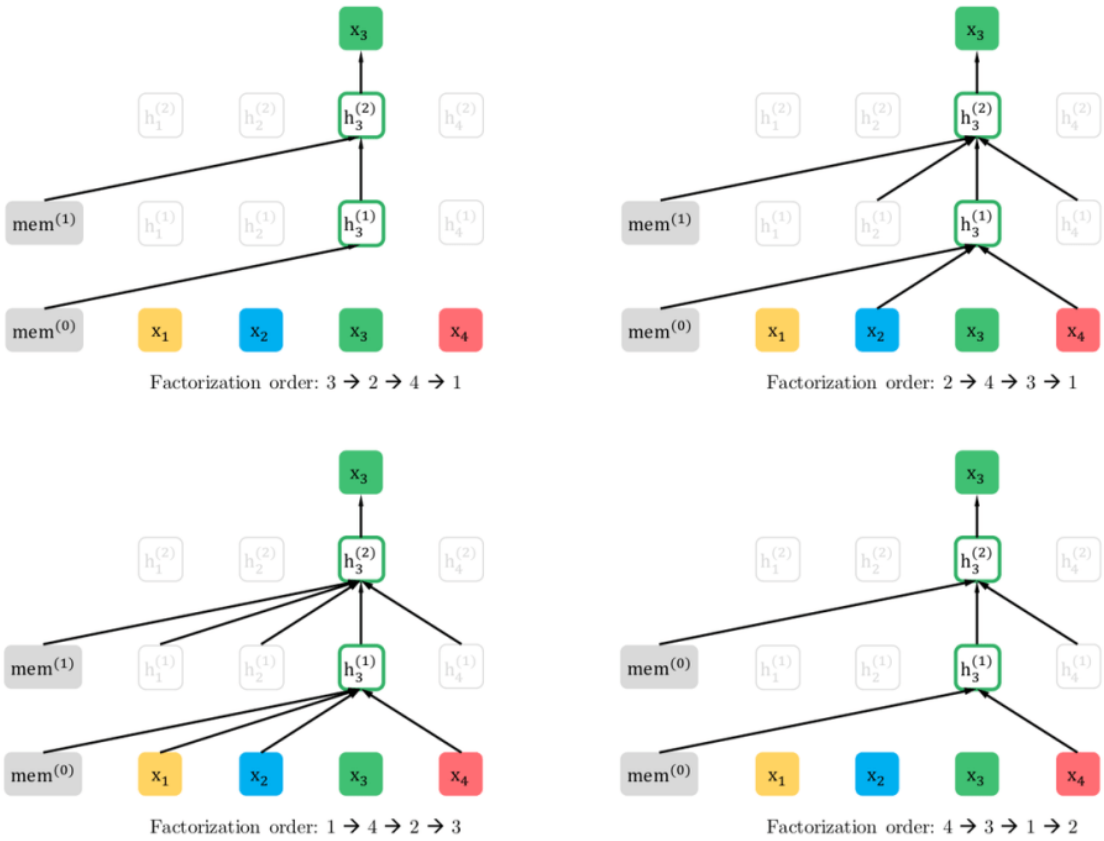


Figure 1: Illustration of the permutation language modeling objective for predicting $x_3$ given the same input sequence $\mathbf{x}$ but with different factorization orders.

**Figure 5.1:** Permutation language modeling

To overcome those disadvantages, the authors proposed a permutation language modeling task. The objective is to maximize the expectation

$$\max_{\theta} \mathbb{E}_{z \sim Z_T} \left[ \sum_{t=1}^{T} \log p(x_{z_t} | X_{z_{<t}}, \theta) \right]$$

where $Z_T$ is the set of all possible permutations of the sequence $[1, 2, ..., T]$, $z$ is some element of $Z_T$, $z_{<t}$ is the sequence of the first $t-1$ elements of the $z$, and $X_{z_{<t}}$ are all symbols in sequence $X$ whose indices are in $z_{<t}$. Figure 5.1 illustrates which tokens are accessed when pretraining the symbol $x_3$ with different factorization orders. With this objective, the model will maximize the likelihood of every token conditioned on all combinations of other tokens. In that way, the model is able to capture bidirectional context without creating a pretrain-finetune discrepancy and without any independence assumptions.

## 5.2. High-level architecture of XLNet

The model is an encoder-only transformer with multiple encoder blocks. The authors created two versions of the model: one with 12 and one with 24 encoder blocks (same as BERT-base and BERT-large, so that they can make a fair comparison between the two).

The novelty is a two-stream self-attention mechanism created for the PLM objective. The encoder blocks with two streams of self-attention have two hidden states instead of one: content representation and query representation. Figure 5.2 illustrates how are those two hidden states calculated using the standard masked-self attention (parts *(a)* and *(b)*) and the high-level architecture (part *(c)*)). The query representations at the input are set to trainable vector $w$, and the content representations are set to corresponding word embeddings. The next layer representations are calculated as

$$g_{z_t}^{(m)} \leftarrow Attention(Q = g_{z_t}^{(m-1)}, KV = h_{z_{<t}}^{(m-1)}; \theta)$$

$$h_{z_t}^{(m)} \leftarrow Attention(Q = h_{z_t}^{(m-1)}, KV = h_{z_{\leq t}}^{(m-1)}; \theta)$$

where $g_i^{(m)}$ is query representation of the $i$-th token in the $m$-th layer, $h_i^{(m)}$ is content representation of the $i$-th token in the $m$-th layer, and $h_{z_{<t}}^{(m-1)}$ is set of all tokens whose index is less than $z$. In that way, the content representation in the next layer for a token on position $i$ is calculated based on all hidden states whose index is less than or equal to $i$ and it is queried by the content representation. On the other hand, query representation in the next layer on position $i$ is calculated based on all hidden states whose index is less than $i$ (excluding the $i$-th), and it is queried by the query representation. Finally, predictions are made based on query representation.

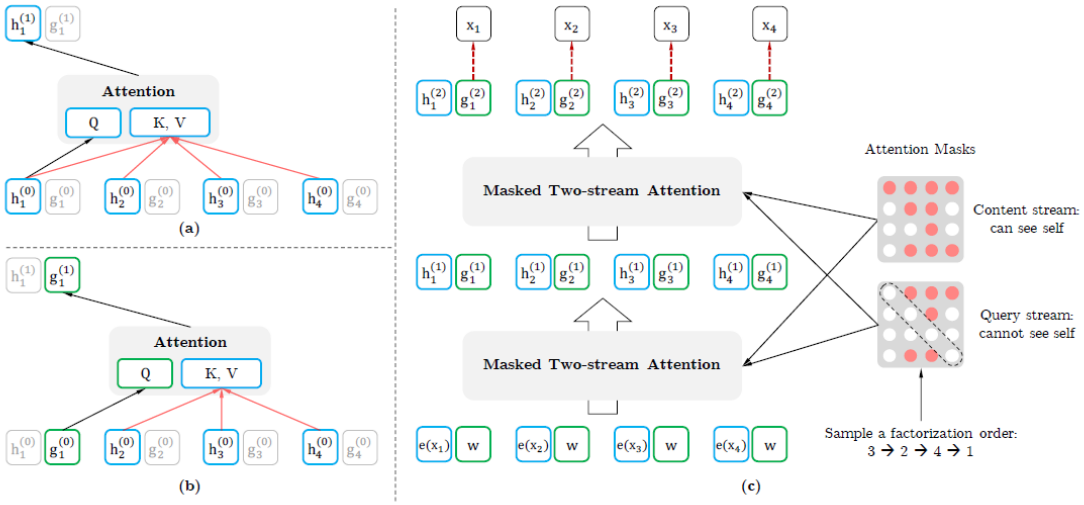


Figure 1: (a): Content stream attention, which is the same as the standard self-attention. (b): Query stream attention, which does not have access information about the content $x_{z_t}$. (c): Overview of the permutation language modeling training with two-stream attention.

**Figure 5.2:** Arhitecture of XLNet

The motivation for two-stream self-attention is the need to embed the positional information of the token that is being predicted. To explain this issue, suppose we have two permutations $z^{(1)}$ and $z^{(2)}$ such that

$$z^{(1)}_{<t} = z^{(2)}_{<t} = z_{<t}$$

$$z^{(1)}_t = i \neq j = z^{(2)}_t.$$

The probabilities that the tokens on $i$-th and $j$-th position are equal to $x$ with standard language modeling head are

$$p(x_i = x|x_{z<t}, \theta) = p(x_j = x|x_{z<t}, \theta) = \frac{\exp\left(h_\theta(x_{z<t})^T e(x)\right)}{\sum_{x'} \exp\left(h_\theta(x_{z<t})^T e(x')\right)}.$$

In other words, the probability distribution of two different target positions conditioned on the same set of tokens is exactly the same (and they should most certainly be different). With two-stream self-attention, those distributions are equal to

$$p(x_i = x|x_{z<t}, \theta) = \frac{\exp\left(g_\theta(x_{z<t}, i)^T e(x)\right)}{\sum_{x'} \exp\left(g_\theta(x_{z<t}, i)^T e(x')\right)}$$

$$p(x_j = x|x_{z<t}, \theta) = \frac{\exp\left(g_\theta(x_{z<t}, j)^T e(x)\right)}{\sum_{x'} \exp\left(g_\theta(x_{z<t}, j)^T e(x')\right)}.$$

Note that now last hidden state is also indexed by $i$ and $j$, which are exactly the positions of tokens that are being predicted.

## 5.3. Results

The model was pretrained for comparison with BERT on BookCorpus and English Wikipedia datasets. Additionally, it was pretrained on Giga5, ClueWeb, and Common Crawl datasets (for the comparison with RoBERTa, Liu et al. (2019)). The version that was trained for BERT comparison outperformed BERT on all considered datasets. Similarly, the version pretrained on the same datasets as RoBERTa outperformed RoBERTa on all considered datasets on test sets when ensembles were used. It also outperformed RoBERTa on six out of eight considered datasets on dev sets when single models are used. It also set new state-of-the-art results on the GLUE benchmark, SQuAD dataset, and a few classification datasets.

# 6. ELECTRA: pretraining Text Encoders as Discriminators Rather Than Generators (Clark et al. (2020))

ELECTRA is a transformer encoder-only model proposed in Clark et al. (2020). The authors wanted to create a more efficient pretraining task. They argue that today's pretraining tasks like MLM don't use the training data (the text corpus) efficiently enough; for example, MLM uses only $15\%$ of the tokens (the ones that are masked out). The proposed model performs comparably to RoBERTa and XLNet when using $\frac{1}{4}$ of their compute and outperforms BERT and GPT (GPT is trained using $30$ times more compute).

## 6.1. pretraining task

The new pretraining task is called replaced token detection, and it is performed using a model called discriminator. The text corpus is corrupted similarly to MLM, but tokens are replaced with the random token (sampled by another model called generator) instead of the $[MASK]$ token. The objective of the Electra model is to detect which tokens are replaced. More formally, for the token sequence $x$, the sequence $x^{masked}$ is the masked version of it, and $x^{corrupted}$ is the corrupted version of it. Those two sequences are constructed as

$$x^{masked} = replace(x, m, [MASK])$$

$$x^{corrupted} = replace(x, M, \hat{x})$$

where $m$ is sequence of indices which were chosen for masking and $\hat{x}$ is sampled as

$$\hat{x_i} \sim p_G(x_i|x^{masked})$$

and $p_G$ is the probability distribution of the model which generates the corrupted text. The loss of the discriminator is calculated as a binary cross-entropy loss:

$$L_{disc}(x, \theta) = \mathbb{E}\left[\sum_{t=1}^{n} -\mathbb{1}(x_t^{corrupt} = x_t)\log D(x^{corruppt}, t) - \mathbb{1}(x_t^{corrupt} \neq x_t)\log(1 - D(x^{corruppt}, t))\right].$$

## 6.2. High-level arhitecture of ELECTRA

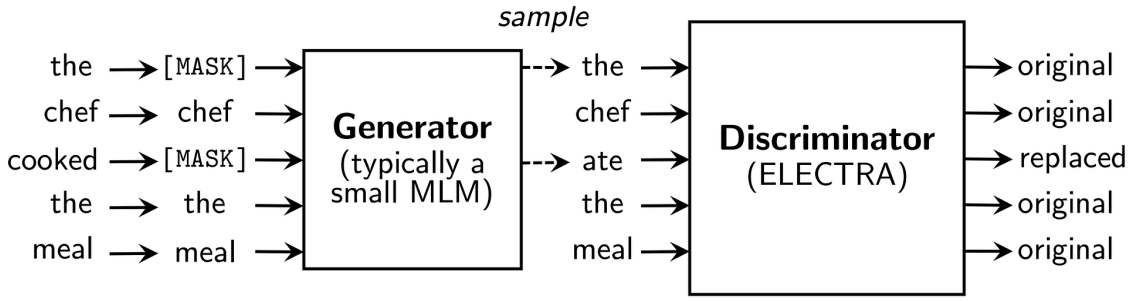


**Figure 6.1:** Arhitecture of ELECTRA

Figure 6.1 illustrates the high-level architecture of the ELECTRA. The two main components of the ELECTRA model are the generator and the discriminator. The generator is a model which is used only for the purpose of pretraining the discriminator. The discriminator is the model we "take" after the pretraining and fine-tune it for downstream tasks. Both generator and discriminator are transformer encoder-only models, but they are not necessarily the same size and they are pretrained differently.

As said before, the discriminator is pretrained on the replaced token detection task, while the generator is pretrained on MLM. The generator generates tokens on masked positions and the discriminator detects if the generated tokens are those that were initially replaced. The generator and discriminator can be trained separately or at the same time. If trained together, the objective is to minimize both the generator's and discriminator's loss:

$$\min_{\theta_G,\theta_D} \sum_{x \in X} L_{MLM}(x, \theta_G) + \lambda L_{disc}(x, \theta_D),$$

where $X$ is a corpus of text. It is worth noting that the discriminator's predictions do not influence the generator's loss $L_{MLM}$; rather, it only depends on the generator's predictions. On the other hand, the discriminator's loss $L_{disc}$ is calculated based on both the generator's and discriminator's predictions, but only the discriminator's parameters are optimized based on it. The authors pointed out that it is challenging to optimize the generator's parameters based on the discriminator's output because it's not possible to back-propagate the loss through the generator's sampling layer. If the generator and discriminator are trained separately, the generator is trained first (with MLM). After the training of the generator is over, the parameters in the generator are frozen, and the generator is used only for generating training examples for the discriminator. The discriminator is then trained with replaced token detection, and the optimized loss is equal to binary cross entropy.

The authors also concluded that the size of the generator is a very important hyperparameter. If the generator is too large (for example, same sized as the discriminator), the

generator will be "too powerful" and it will learn to predict masked tokens too well. In that case, the discriminator cannot learn which are the replaced tokens (because there will be too few of them). On the other hand, if the generator is too small, it can't properly learn token distribution, and the problem becomes too trivial for the discriminator.

## 6.3. Results

The model was pretrained on the same data as XLNet and RoBERTa. The model has three versions: small, base, and large. The small version has the same number of parameters as BERT small (the ELECTRA's authors implementation of BERT with same hyperparameters as ELECTRA small). The base model has the same number of parameters as the BERT base, while the large version has around the same number of parameters as BERT large and XLNet large. The ELECTRA small model outperforms the BERT small and GPT models on the GLUE benchmark by a large margin (the GPT model is sized as ELECTRA base and trained using $30$ times more compute than). The ELECTRA large has similar results as the other state-of-the-art models on the SQuAD dataset but often outperforms them on GLUE's datasets. The main advantage of the ELECTRA is its effective and cheap pretraining, which can be seen in the results of the smaller models.

# 7. Conclusion

The transformer architecture showed as the best choice for achieveing state-of-the-art results in all NLP tasks. Their ability to grasp a basic language understanding from the pretraining tasks and after that being able to adapt to all kinds of different downstream tasks with relatively cheap fine-tuning made the transformer models the best choice for both budget-friendly and high-end projects. After the first transformer in late 2017, many research teams proposed different approaches on how to utilize and improve them.

The first approaches (besides GPT and Transformer-XL (Dai et al. (2019))) were concentrated on the research of encoder-only transformers (BERT, RoBERTa, ALBERT (Lan et al. (2020)), XLNet, ELECTRA, etc.), with a goal of building a general-purpose model that can be fine-tuned for any downstream NLP task. The decoder-only models couldn't capture bidirectional context (because their original purpose was to be the auto-regressive text generation models), and the public opinion was that they could not outperform encoder-only models on natural language understanding tasks. The decoder-only models are also usually trained as generative models, which at the time wasn't very popular (the people didn't see market potential for products using those models). However, after the OpenAI scaled up and presented much larger GPT-2 and GPT-3 decoder models ($1.5$ and $175$ billion parameters, while at the day, state-of-the-art models had a few hundred million parameters), the decoder models slowly started taking the scene. With the introduction of GPT-3 powered ChatGPT API, the generative models got much more public attention because, until then, the generative models weren't so easily available to the public.

Today's new models are mostly decoder-only transformers and much larger than those described in this work (For example, Google's PaLM (Chowdhery et al. (2022))). They are also often multimodal, meaning they can embed and process different kinds of input (for example, audio, images, and videos). It will most certainly be interesting to see in which direction the development of those new LLMs will go. It is not excluded that encoder models will again become popular. Maybe scaling up encoder models or new pretraining tasks put them again in the game.

# 8. Bibliography

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, i Noah Fiedel. Palm: Scaling language modeling with pathways, 2022.

Kevin Clark, Minh-Thang Luong, Quoc V. Le, i Christopher D. Manning. Electra: Pre-training text encoders as discriminators rather than generators. U *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=r1xMH1BtvB`.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V. Le, i Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context, 2019.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, i Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, i Radu Soricut. Albert: A lite bert for self-supervised learning of language representations, 2020.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, i Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, i Illia Polosukhin. Attention is all you need, 2017.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, i Quoc V. Le. Xlnet: Generalized autoregressive pretraining for language understanding. *CoRR*, abs/1906.08237, 2019. URL `http://arxiv.org/abs/1906.08237`.