Sequence Models

2000 – 2020

Zachary R. McCaw

November 2025

2016 and earlier

1.1 (03/2003) Neural Probabilistic Language Models

Bengio *et al* [1] develop a likelihood-based approach to language modeling. The likelihood of a sequence of words $x = (x_1, \ldots, x_T)$ can be decomposed as:

$$\mathbb{P}(x) = \mathbb{P}(x_1) \prod_{t=2}^{T} \mathbb{P}(x_t | x_1, \ldots, x_{t-1}).$$

A common simplification is to assume the conditional distribution of the present token x_t only depends on the previous k tokens, where k is the length of the context window:

$$\mathbb{P}(x) = \mathbb{P}(x_1) \prod_{t=2}^{T} \mathbb{P}(x_t | x_{t-1}, \ldots, x_{t-k+1})$$

If a standard start token is prepended to all sequences, then $\mathbb{P}(x_1) = 1$, and the likelihood simplifies:

$$\mathbb{P}(x) = \prod_{t=2}^{T} \mathbb{P}(x_t | x_{t-1}, \ldots, x_{t-k+1})$$

The proposed framework has 3 key ideas:

1. Associate with each discrete word (or token) a continuous, distributed representation (or embedding).
2. Express the likelihood as a function of the distributed representations.
3. Simultaneously learn the parameters of the representation and probability models.

Let $C : V \to \mathbb{R}^d$ denote a mapping from the discrete vocabulary V to vectors in real d-dimensional space. The conditional probability $\mathbb{P}(x_t | x_{t-1}, \ldots, x_{t-k+1})$, which defines a distribution over the vocabulary V, is modeled as:

$$\mathbb{P}(x_t | x_{t-1}, \ldots, x_{t-k+1}) = g_\theta\{C(x_{t-1}), \ldots, C(x_{t-k+1})\}.$$

The function g_θ can be implemented with a feed-forward or recurrent neural network. Training proceeds by maximizing the penalized log-likelihood:

$$\ell(\theta, C) = \sum_{t=2}^{T} \ln g_\theta\{C(x_{t-1}), \ldots, C(x_{t-k+1})\} + R(\theta, C).$$

Cases where a token outside of the sequence is required (i.e. x_t for $t \leq 0$) can be addressed by inserting a special padding token.

1.2 (01/2013) Word2Vec

In the Word2Vec paper, Mikolov *et al* [2] introduced two techniques for learning distributed representations (*embeddings*) of words: the continuous bag-of-words model and the continuous skip-gram model. In the continuous bag-of-words model, a set of surrounding context words is used to predict the focus word. In the continuous skip-gram model, the focus word is used to predict the set of context words. Mikolov *et al* also showcase that vector arithmetic in the learned vector space can recapitulate the logical relationships among words. For example, the word whose embedding was closest to *Paris - France + Italy* was *Rome*.

1.3 (09/2014) Neural Machine Translation*

Neural machine translation attempts to build a single, jointly learned neural network that takes in a sentence in one language and outputs the translation into another [3]. A common architecture consists of an encoder and a decoder, where the encoder compresses a source sentence into a fixed-length embedding, then the decoder converts the encoder into a sequence. However, the performance of such a system deteriorates as the length of the input sentence increases. Bahdanau *et al* [3] propose an architecture (RNNsearch) where the source sentence is encoded as a sequence of vectors, which the decoder can adaptively attend to while translating. The proposed architecture increasingly outperforms using a fixed-length encoding as the length of the input sentence increases.

Methods

The problem of machine translation can be cast as finding the target sentence Y^* that maximized the conditional probability of Y given source sentence $X = x$. That is:

$$Y^* = \arg\max_{Y \in \mathcal{Y}} \mathbb{P}(Y|X = x).$$

Given an input sequence $x = (x_1, \ldots, x_{T_X})$, an RNN produces a sequence of hidden states based on the current input token x_t and the previous hidden state h_{t-1}:

$$h_t = g(x_t, h_{t-1}).$$

In a fixed-length encoding scheme, the *context vector* for x can be represented as:

$$c_x = g(h_1, \ldots, h_{T_X}).$$

The decoder often operates by defining a conditional probability distribution over the next token y_t given the context vector c_x and the previous tokens:

$$\mathbb{P}(y_t|c_x, y_1, \ldots, y_{t-1}).$$

If the decoder is another RNN with hidden states h_t^* at time step t, the conditional probability may be parameterized as:

$$\mathbb{P}(y_t|c_x, y_1, \ldots, y_{t-1}) = g(c_x, h_t^*, y_{t-1}).$$

Bahdanau *et al* [3] propose a model where the context vector c_t is a time-dependent. Specifically, the context vector at decoding time-step t is:

$$c_t = \sum_{j=1}^{T_X} w_{tj} h_j, \qquad w_{tj} = \frac{\exp(a_{tj})}{\sum_{k=1}^{T_X} \exp(a_{tk})}$$

Here a_{tj} is an *alignment* or *attention* score that quantifies the relevance of input position j to output position t. The proposed attention score is parameterized as:

$$a_{tj} = a(h_{t-1}^*, h_j).$$

The attention of each output position to each input position can be visualized as a 2D heat map. To allow the entire source sentence to inform the hidden state at input position j, h_j is in fact the concatenation of the hidden states from a forward and a reverse RNN.

1.4 (09/2014) Sequence-to-Sequence Learning

Sequences pose a challenge to feed-forward neural networks because such networks typically require that the sequence length is known and fixed. Sutskever *et al* demonstrate that the LSTM architecture can be used to solve the general sequence-to-sequence translation problem [5]. The approach uses one LSTM to map the input sequence to a fixed-length embedding, then a second LSTM to decode the embedding vector to the output sequence. In detail, the encoder maps the input sequence $x = (x_1, \ldots, x_T)$ to an embedding c_x. The decoder produces a probability distribution, over the vocabulary, for the present output token y_t conditional on the input embedding and the preceding output tokens:

$$p(y_t|c_x, y_{t-1}, \ldots, y_1) = g(c_x, y_{t-1}, \ldots, y_1).$$

Sutskever *et al* report that the performance of the system is significantly improved by reversing the order of the input sequence only, such that $(x_T, x_{T-1}, \ldots, x_1)$ is mapped to $(y_1, y_2, \ldots, y_{T'})$. Decoding is performed via beam search, in which B candidates are maintained in parallel. Each candidate is extended with all possible words in the vocabulary, then the set of candidates is pruned back such that only the top B candidates remain, as scored by the model's log likelihood. Generation of a candidate terminates once the end-of-sequence `[EOS]` token is appended.

1.5 (10/2014) GloVe

In Global Vectors for Word Representation, Pennington *et al* [4] introduce a bilinear model for learning word embeddings based on the global co-occurrence statistics for a corpus. Specifically, let X_{ij} denote the number of times word j appears in the context of word i. The size of the context window is an adjustable hyperparameter, and the context window may not be symmetric. The objective function is the weighted least squares criterion:

$$L = \sum_{i=1}^{V} \sum_{j=1}^{V} g(X_{ij}) \big(w_i^\top \tilde{w}_j + b_i + \tilde{b}_j - \ln X_{ij}\big)^2.$$

Here, the learnable parameters are $(w_i, \tilde{w}_j, b_i, \tilde{b}_j)$. w_i is the *word vector* for word i, $\tilde{w}_j$ is a distinct *context vector* for word j, b_i and $\tilde{b}_j$ are intercepts. The proposed weight function is:

$$g(x) = \begin{cases} (x/x_{\max})^\alpha & \text{if } x < x_{\max}, \\ 1 & \text{otherwise.} \end{cases}$$

This weight function gives co-occurrences up to $x_{\max}$ a weight increasing in the number of co-occurrences, then provides constant weight beyond $x_{\max}$ so as to not over-emphasize commonly occurring word pairs. Empirically, $x_{\max} = 100$ and $\alpha = 3/4$ were suggested. The final representation e_i for word i is the sum of its word and context vectors: $e_i = w_i + \tilde{w}_i$.

1.6 (07/2015) Deep Averaging Networks

Vector space models like Word2Vec provide a mechanism of embedding individual words. To embed collections of words, such as sentences or documents, a *composition function* must be selected for combining the word-level embeddings [8]. Composition functions may be unordered or syntactic. Unordered functions regard the word embeddings as a set, whereas syntactic functions take word order and sentence structure into account. The proposed deep averaging network (DAN) takes the (unordered) vector average of the embeddings associated with the input sequence, then passes this average through a feedforward network. To regularize the DAN, *word dropout* is proposed, where the embeddings corresponding to randomly selected words are omitted from the average. The DAN with word dropout achieved performance comparable to state-of-the-art RNNs at sentiment analysis and question-answering, while requiring only a fraction of the training time.

In the context of sentiment analysis, the authors perform an interesting *perturbation* study. A template sentence is constructed, then a word pivotal to determining the sentiment is increasingly perturbed (e.g. "awesome" is changed to "underwhelming"). For each layer in the network, the effect of the change is measured by the distance of the resulting activations from those of the original sentence.

1.7 (08/2015) BPE

Neural machine translation (NMT) systems equipped with a fixed vocabulary struggle on out-of-vocabulary words, yet human translators are able to translate novel words, such as named entities, by translating subunits (morphemes or phonemes). Earlier approaches such as falling back to dictionary lookup were imperfect insofar as rare words do not always have a 1-to-1 translation between the source and target languages. Byte pair encoding (BPE) is a data compression technique that iteratively replaces the most common pair of bytes in a sequence with a single available byte. Sennrich *et al* [9] apply this technique to create a vocabulary from sequences of characters, starting with the individual characters and a special end-of-word token. Frequent words are assigned individual tokens, but rare and unseen words are always expressible because the vocabulary includes the individual characters. In this way, BPE enables NMT systems to translate words that do not appear in the training data. Two methods of applying BPE are evaluated: marginal and joint. The marginal method segments the source and target vocabularies separately while the joint method operates on the union of the two vocabularies. Marginal encoding results in a smaller vocabulary while joint encoding improves consistency between the two segmentations and generally results in improved performance.

1.8 (06/2016) Gaussian Error Linear Units

Hendrycks and Gimpel [11] propose a form of regularization, similar to dropout, where each input x to a neuron is multiplied by a mask $m(x) \sim \text{Bernoulli}\{F(x)\}$, with $F(x)$ denoting a cumulative distribution function. Such masking makes retention of inputs (i.e. $m = 1$) more likely for inputs with larger values. To convert this masking strategy into a deterministic activation function, the expected value of $x \cdot m(x)$ is taken, yielding $h(x) = xF(x)$. In particular, when F is the CDF of the standard normal distribution, the activation is called the Gaussian Error Linear Unit (GELU). The GELU activation is compared with RELU and ELU:

$$\text{GELU}(x) = x\Phi(x), \qquad \text{ReLU}(x) = x\mathbb{I}(x > 0), \qquad \text{ELU} = \begin{cases} x & x > 0, \\ \alpha\{\exp(x) - 1\} & x \leq 0. \end{cases}$$

Note α is an optional hyperparameter often fixed to $\alpha = 1$. Across a variety of tasks, including digit classification with MNIST and part of speech tagging with Twitter, GELU provides improved classification performance. Compared with ReLU and ELU, distinguishing features of GELU are that it is 1. non-convex, 2. non-monotonic, and 3. non-linear in the position domain.

1.9 (07/2016) Layer Normalization

Batch normalization [7] was introduced to facilitate the training of deep neural networks. In batch normalization, the input to internal layers is standardized, for each dimension, to have mean zero and unit variance, where the mean and variance calculated across the batch. Learnable shift and scale parameters may be included to allow the network to modify the standardization. Batch normalization has several drawbacks, including sensitivity to the batch size and different behavior during training and test time. Layer normalization [10] was introduced to address these limitations. In layer normalization, the mean and variance are calculated for each example across the dimensions in a layer. Each dimension in a layer applies the same shift and scale, but each example will receive a different shift and scale.

1.10 (09/2016) WordPiece

The WordPiece Model (WPM) was developed in the context of neural machine translation [12], and guarantees a deterministic segmentation of any possible sequence of characters. The tokens in the WPM vocabulary

may represent words or subunits of words. The WPM is initialized with the individual characters of a language. Through a greedy, iterative procedure, frequent combinations of characters are combined to form new tokens, and the process continues until the target vocabulary size or performance threshold is achieved. The WPM improves the handling of out of vocabulary words, as these can often be at least partially broken into known sub-units. In the context of translation, rare tokens should often be copied from the source to the target language. To facilitate this behavior, the WPM is trained jointly on the source and target languages.

2017

2.1 (06/2017) Transformers

The **transformer** was introduced by Vaswani *et al* [13] as an alternative to RNNs for sequence modeling, and in particular for machine translation. The model uses an encoder-decoder architecture, with 6 layers in each stack. Each layer includes 2 sub-layers: multi-headed self-attention followed by a feed-forward network. A residual connection followed by layer normalization routes around each sub-layer. For a sequence of T hidden states $h = (h_1, \ldots, h_T)$, a Transformer block can be represented as:

$$h \leftarrow \text{LayerNorm}\{h + \text{Attention}(Q = h, K = h, V = h)\},$$
$$h_t \leftarrow \text{LayerNorm}\{h_t + \text{FeedForward}(h_t)\} \text{ for } t = 1, \ldots, T.$$

Attention layers in the decoder also employ masking to ensure the output at a given position only depends on outputs at previous positions. The output of an attention layer is a weighted average of the *values*, where the weights depend on the dot products between the *queries* and *keys*. In the absence of masking, self-attention allows the representation of each token to interact with that of every other token. In multi-headed attention, multiple sets of queries, keys, and values are created, then the outputs are concatenated and linearly projected to the target dimension. Since the transformer does not employ recursion, positional encodings are added to the input embeddings to keep track of positional information. The absence of recursion makes transformers more parallelizable than RNNs, however the required volume of training data may be higher.

2018

3.1 (01/2018) ULMFiT

Universal Language Model Fine-Tuning [16] was developed to address the failure of fine-tuning to improve language model performance (at the time of writing), as compared with computer vision models. The framework is universal in the senses that it: 1. works across tasks varying in document size, number, and label type; 2. uses a single architecture and training process; 3. requires no custom feature engineering; and 4. does not require additional in-domain documents or labels [16]. ULMFiT has 3 main steps (**Figure 1**). The first is language modeling (LM) on a general corpus using standard optimization. The second is fine-tuning via LM in the target domain, with two variations on the standard optimization procedure: *discriminative fine-tuning* and a *slanted triangular* learning rate schedule. In discriminative fine-tuning, each layer is updated with a different learning rate, with learning rates defined recursively as $\eta_{l-1} = \eta_l / k$ with $k > 1$. Layers closer to the inputs, which contain more-generalizable information, are updated more slowly. In the slanted triangular learning rate schedule, the learning rate is ramped up relatively quickly, then wound down relatively slowly, with linear ramp up and wind down functions. The third step is target task fine-tuning. New linear layers are added specific to the target task (e.g. classification). In addition to discriminative fine-tuning and a slanted triangular learning rate schedule, target task fine-tuning uses *gradual unfreezing* to avoid catastrophic forgetting. In gradual unfreezing, the last layer is trained on the first epoch, with the preceding layers frozen; the last two layers are trained on the second epoch; and so on until all layers are unfrozen and trained. ULMFiT has high label efficiency as compared with purely supervised training.

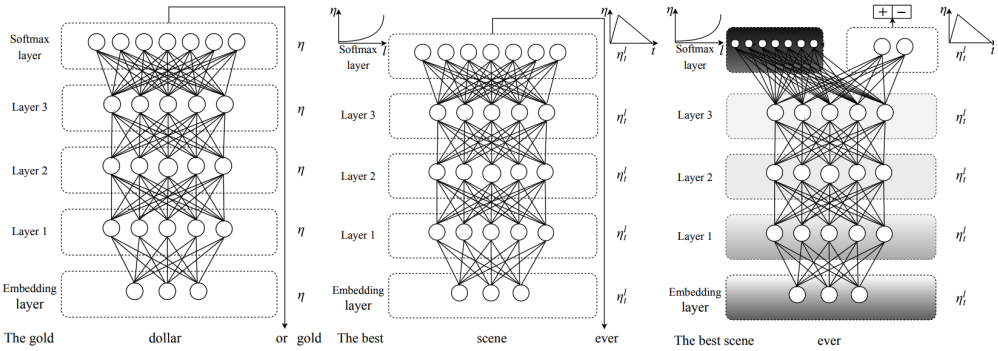


Figure 1: ULMFiT consists of three stages: a) The LM is trained on a general-domain corpus to capture general features of the language in different layers. b) The full LM is fine-tuned on target task data using discriminative fine-tuning ('*Discr*') and slanted triangular learning rates (STLR) to learn task-specific features. c) The classifier is fine-tuned on the target task using gradual unfreezing, '*Discr*', and STLR to preserve low-level representations and adapt high-level ones (shaded: unfreezing stages; black: frozen).

Figure 1. **Source:** Universal Language Model Fine-tuning for Text Classification [16]

3.2 (01/2018) Generating Wikipedia

Liu *et al* [17] consider the task of generating the first section of a Wikipedia article conditional on text from the citations and search results. Due to the volume of text, the task is divided into two steps: extraction and abstraction. During extraction, the paragraphs in the context documents (citations and search results) are scored for relevance, and the top L retained. The input to the abstractive stage is the concatenation of the title and the L most-relevant paragraphs. Several abstractive models are considered, including an LSTM and a transformer decoder. For the transformer decoder, the input and output sequences are concatenated, separated by a special token. The model's task is to predict the next token, conditional on the previous ones. Two approaches are evaluated to reduce the memory requirements of self-attention: local attention within blocks of tokens, and memory-compressed attention, in which the numbers of keys/values are reduced via strided convolution.

3.3 (02/2018) ELMo

The Embeddings from Language Models (ELMo) framework [18] developed word representations from the internal states of a bidirectional language model, allowing the representations to incorporate information at various levels of abstraction and from surrounding words in both directions. The bidirectional language model is the combination of a forward and a reverse LSTM, where the parameters for representing tokens and for the terminal softmax layer are shared between models. The final ELMo embedding is a learned linear combination of the input embedding and the L hidden states from each of the forward and the reverse models ($2L + 1$ representations in total).

3.4 (03/2018) Relative Position Encoding*

The original transformer model encoded position with sinusoids of varying frequency that were added to the input embeddings prior to the first layer. Shaw *et al* [20] propose an alternative, relative position encoding scheme. Let $x = (x_1, \ldots, x_n)$ denote the sequence input to a self-attention layer. The output $z = (z_1, \ldots, z_n)$ is a sequence of the same length, where each output element is a weighted average of the (linearly transformed) input elements:

$$z_i = \sum_{j=1}^{n} w_{ij}(W_V x_j), \qquad w_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{n} \exp(e_{ij})}, \qquad e_{ij} = \frac{(W_Q x_i)^\top (W_K x_j)}{\sqrt{\dim(z)}}.$$

Here, the weights w_{ij} are formed via a softmax over $(e_{i1}, \ldots, e_{1n})$, where e_{ij} is a measure of the attention between tokens i and j. Shaw *et al* [20] introduce two learnable vectors to encoder relative position, a_{ij}^V and a_{ij}^K, which are incorporated into the attention layer as follows:

$$z_i = \sum_{j=1}^{n} w_{ij}(W_V x_j + a_{ij}^V), \qquad w_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{n} \exp(e_{ij})}, \qquad e_{ij} = \frac{(W_Q x_i)^\top (W_K x_j + a_{ij}^K)}{\sqrt{\dim(z)}}.$$

Under the hypothesis that precise relative position cases to convey additional information beyond a certain distance k, the relative position vectors are parameterized via $2k+1$ learnable weight vectors:

$$a_{ij}^K = w_{\text{clip}(j-i,k)}^K, \qquad a_{ij}^V = w_{\text{clip}(j-i,k)}^V.$$

An additional advantage of parameterizing the position vectors in this way is the ability to generalize to sequence lengths longer than those seen during training.

3.5 (03/2018) Universal Sentence Encoder

Cer *et al* [14] develop two models for embedding sentences (as opposed to individual words). The first uses the Transformer Encoder to generate embeddings for each word, then takes the element-wise sum to form the sentence-level representation. The second is a deep averaging network (DAN), whereby input embeddings for words and bi-grams are first averaged, then passed through a feedforward network. Transformer-based sentence embeddings generally provide the best performance, but at the cost of greater compute requirements. Due to self-attention, the computational complexity of the Transformer scales quadratically in the input sequence length, whereas that of the DAN scales linearly. Across multiple downstream tasks, models provided with sentence-level embeddings outperform those provided with word-level embeddings. However, models provided with both word- and sentence-level embeddings achieve the best performance overall.

3.6 (06/2018) Generative Pre-Training

Methods

Radford *et al* [19] describe generative pre-training (GPT) on a large corpus of unlabeled text followed by discriminative fine-tuning on specific tasks. Unsupervised pre-training seeks to maximize the *language modeling* objective:

$$L_1 = \sum_i \ln \mathbb{P}(u_i | u_{i-k}, \ldots, u_{i-1}; \theta)$$

where $u_i \in \{u_1, \ldots, u_n\}$ is a corpus of tokens, k is the size of the *context window*, and the conditional probability is modeled with a neural network parameterized by θ. The language model is a transformer decoder [13, 17] (**Figure 2**) with the following structure:

$$\begin{aligned} h_0 &= UW_e + W_p, \\ h_l &= \text{Transformer}(h_{l-1}), \quad l \in \{1, \ldots, b\} \\ h_{\text{out}} &= \text{Softmax}(h_b W_e'). \end{aligned}$$

Here $U = (u_{-k}, \ldots, u_{-1})$ is the *context vector*, W_e is an embedding weight, and W_p is a position weight. The Transformer layer uses multi-headed self-attention and masking. The output is a probability distribution over the space of possible tokens. To adapt the pre-trained model for downstream tasks, the Softmax layer is removed and replaced with an output layer specific to the task. For a categorical output, the final layer would be replaced by $\text{Softmax}(h_m W_y) = \mathbb{P}(y|x)$. The supervised objective is:

$$L_2 = \sum_{(x,y)} \ln \mathbb{P}(y|x).$$

Incorporating the language modeling objective into the supervised objective, as in $L_3 = L_2 + \lambda L_1$, was found to improve generalization and accelerate convergence. To address different styles of input (e.g. entailment, question answering), various syntactic tokens are introduced, such as start, end, and delimiter.

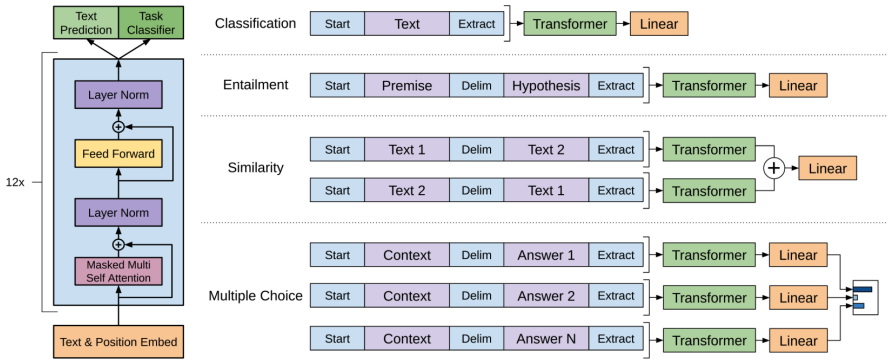


Figure 1: **(left)** Transformer architecture and training objectives used in this work. **(right)** Input transformations for fine-tuning on different tasks. We convert all structured inputs into token sequences to be processed by our pre-trained model, followed by a linear+softmax layer.

Figure 2. **Source:** Improving Language Understanding by Generative Pre-Training [19].

Evaluations

GPT is evaluated on several tasks. Natural language inference or *textual entailment* involves reading a pair of sentences, and determining whether the relationship between them is entailment, contradiction, or neutral. *Question answering* consists of reading multiple sentences then answering associated questions. *Semantic similarity* requires determine whether two sentences have the same meaning. *Classification* tasks include determining whether or not a sentence is grammatical, and whether the sentiment is positive or negative.

3.7 (11/2018) BERT

Methods

The Bidirectional Encoder Representations from Transformers (BERT) framework involves self-supervised pretraining of a base model, followed by fine-tuning of the base model for specific tasks [15]. Input text is tokenized with WordPiece. The first token of every sequence is `[CLS]`, which also serves as the aggregate sequence representation for classification tasks. Pairs of sentences are separated by a `[SEP]` token. During pretraining, 15% of tokens are selected for potential masking. Of these, 80% are replaced by a learnable `[MASK]` token, 10% are replaced by a random token, and 10% are left unchanged. The main task, referred to as **masked language modeling** (MLM), is to predict the original identities of the selected tokens. A secondary task is next-sentence prediction, where the model is presented with a pair of sentences, A and B, and must predict whether B follows A. Pretraining was performed on the BooksCorpus and English Wikipedia. A new model is fine-tuned for each application. The weights are initialized to their pretrained values, and a new output layer is added appropriate to the application.

Evaluations

BERT was evaluated on the multi-part General Language Understanding Evaluation (GLUE); the Stanford Question Answering Dataset (SQuAD), which involves extracting information from a passage of text; and the Situations With Adversarial Generations (SWAG) test, where the task is to select the most plausible continuation from among 4 possibilities.

Notes

The input tensor to the BERT model will typically be structured as (B, S), where B is the batch size and S is the maximum sequence length. Shorter sequences are padded to the maximum length, while longer sequences are truncated. The initial embedding layer will map each token to a learnable embedding vector

of dimension E, resulting in a tensor of shape (B, S, E). The output will also have shape (B, S, E). The embedding corresponding to the initial `[CLS]` token is often used to represent the entire sequence.

2019

4.1 (01/2019) Transformer-XL

A simple approach for applying attention over long sequences is to divide the sequence into non-overlapping segments, then apply attention within each segment. With this approach, the longest possible dependence is limited by the segment length, and no information flows across the often artificial segment boundaries. In Transformer-XL [23], the hidden state of a segment is stored, then used to provide extended context when processing downstream segments. Within the attention mechanism, queries depend on the current segment only, while keys and values contain the concatenated hidden states from the current and previous segments. Absolute positional encodings become problematic because positions no longer reflect the correct distances between tokens when past and current segments are concatenated. To avoid this, the attention calculation is modified to incorporate relative positional information. The pre-softmax attention between query q_i and key k_j is generalized from $e_{ij} = q_i^\top k_j$ to:

$$e_{ij} = q_i^\top k_j + q_i^\top r_{i-j} + u^\top k_j + v^\top r_{i-j},$$

where r_{i-j} is a learned *relative position* embedding, and u, v are global trainable parameters.

4.2 (01/2019) BioBERT

BioBERT is a domain-specific language embedding model pretrained specifically on biomedical text [26]. BioBERT utilizes the same WordPiece tokenization adopted by BERT, allowing the weights for BioBERT to be initialized with those from BERT. BioBERT is then pretrained on PubMed and PMC full-text articles, and fine-tuned for particular tasks, such as biomedical named entity recognition, relation extraction, and question answering.

4.3 (02/2019) GPT-2

Language modeling is often framed as unsupervised distribution learning. The goal is to estimate the probability of a sequence $x = (x_1, \ldots, x_T)$, which is typically factorized as:

$$\mathbb{P}(x) = \prod_{t=1}^{T} \mathbb{P}(x_t | x_{t-1}, \ldots, x_0).$$

Here x_0 is a start of sequence token. Radford *et al* argue that sufficient proficiency in unsupervised language modeling will enable a model to effectively generalize to diverse downstream tasks without the need for supervised fine-tuning [30]. They demonstrate that the 1.5B parameter GPT-2 model achieves state of the art performance on multiple NLP tasks when evaluated in a zero-shot manner, meaning that neither the architecture nor the model parameters are specifically optimized for the evaluation task. GPT-2 is trained on WebText, a curated subset of 8M documents linked to from Reddit. To ensure a rigorous evaluation, all Wikipedia articles were removed from WebText, as Wikipedia is a common source of text used in evaluation tasks.

4.4 (03/2019) SciBERT

SciBERT [22] is a domain-specific BERT trained on a corpus of full-text articles from Semantic Scholar, covering both computational and biomedical science. A vocabulary specific to the corpus was generated using WordPiece. The overlap between the resulting `SciVocab` and that used by base BERT was only 42%, underscoring the distinctive vocabulary of the scientific literature. SciBERT outperforms base BERT on a suite of scientific NLP tasks, whether used as a fixed featurizer or fully fine-tuned. Using domain-specific vocabulary generally improves performance, but prevents initialization from models trained with a different tokenizer (e.g. base BERT).

4.5 (04/2019) Clinical BERT

Alsentzer *et al* [21] train BERT models specialized to generic clinical text, and to discharge summaries in particular. On the MedNLI natural language inference task, they find that a Clinical BERT initialized from the BioBERT model [26] and fine-tuned on clinical notes from MIMIC-III outperforms both base BERT and BioBERT. The authors note that the practice of redacting protected health information with a `[PHI]` token (as in MIMIC-III) introduces a discrepancy between training and deployment. An alternative approach is to replace PHI information with synthetic but realistic values (e.g. randomly generated names).

4.6 (04/2019) ERNIE

Chen *et al* [33] note that masking individual words can disrupt semantic units, such as multi-word phrases (e.g. scientific literature) or proper nouns (e.g. Albert Einstein). They propose ERNIE, a multi-stage learning strategy in which individual words are masked in the first stage, phrases in the second stage, and entities in the third stage. They demonstrate improved performance on downstream tasks as compared with masking at the word level only.

4.7 (05/2019) SuperGLUE Benchmark

The General Language Understanding Evaluation (GLUE) was a collection of 9 language understanding tasks that provided a single-number target metric [34]. SuperGLUE is a more difficult benchmark that was introduced in response to LLMs surpassing human performance on GLUE. The tasks included in SuperGLUE include Boolean question answering, causal reasoning, reading comprehension, entailment, word sense disambiguation, and pronoun referent resolution.

4.8 (06/2019) XLNet

Yang *et al* [35] classify unsupervised representation learning for language into two main paradigms: auto-regressive and auto-encoding. **Auto-regressive** models, like GPT, factorize the likelihood into a forward $\mathbb{P}(x) = \prod_{t=1}^{T} \mathbb{P}(x_t|x_{s<t})$ or reverse $\mathbb{P}(x) = \prod_{t=1}^{T} \mathbb{P}(x_t|x_{s>t})$ product, and learn to predict the distribution of the present token conditional on the preceding tokens. In terms of the context h_θ and embedding e_θ models, the forward AR likelihood takes the form:

$$\ln p_\theta(x) = \sum_{t=1}^{T} \ln \frac{\exp\{h_\theta(x_{<t})^\top e_\vartheta(x_t)\}}{\sum_{x \in V} \exp\{h_\theta(x_{<t})^\top e_\vartheta(x)\}}.$$

Auto-encoding models, like BERT, corrupt the sequence by introducing a special `[MASK]` token, then learn to reconstruct the original sequence. Auto-encoding models have the advantage of using bidirectional context, but the disadvantage of not providing a generative model, and of introducing a train-test discrepancy. The AE objective takes the form:

$$\ln p_\theta(x_M|x_C) \approx \sum_{t=1}^{T} m_t \ln \frac{\exp\{h_\theta(x_C)^\top e_\vartheta(x_t)\}}{\sum_{x \in V} \exp\{h_\theta(x_C)^\top e_\vartheta(x)\}}$$

where x_M denotes the original values of the masked tokens, x_C denotes the corrupted version of X, and $m_t = 1$ if x_t is among the masked tokens. The equality is approximate in the AE objective because the RHS assumes independence across the masked tokens.

Methods

In XLNet, Yang *et al* [35] proposed the objective:

$$\ell_\theta(x) = \mathbb{E}_{z \sim \Pi(1,\ldots,T)} \left\{ \sum_{t=1}^{T} \ln p_\theta(x_{z_t}|x_{z<t}) \right\}.$$

Here $z \sim \Pi(1, \ldots, T)$ represents a random permutation of the sequence $(1, \ldots, T)$. Note that only the order of the factorization, not the order of the sequence, changes. For example, in the case $T = 3$, the standard forward factorization is:

$$\mathbb{P}(x_1, x_2, x_3) = \mathbb{P}(x_1)\mathbb{P}(x_2|x_1)\mathbb{P}(x_3|x_2, x_1).$$

Another factorization, corresponding to the permutation $(2, 1, 3)$, is:

$$\mathbb{P}(x_1, x_2, x_3) = \mathbb{P}(x_2)\mathbb{P}(x_1|x_2)\mathbb{P}(x_3|x_1, x_2).$$

Both factorizations are valid expressions for the probability of the original sequence (x_1, x_2, x_3). Like AE models, the permutation objective conditions of information on both sides of the present position (in expectation), yet unlike the AE model, the permutation objective introduces no discrepancy between pretraining and fine-tuning. Random factorization introduces an identifiability problem insofar as the index of the present position is no longer clear from the positions in the conditioning set. This is addressed by conditioning on the index z_t, but not the content x_{z_t}, of the present position.

4.9 (07/2019) SpanBERT

In SpanBERT, Joshi *et al* [24] propose a self-supervised pretraining procedure that builds on BERT in 3 ways. First, rather than masking individual tokens, SpanBERT masks contiguous spans of complete words. The length L, in number of words, of the span is drawn from a geometric distribution ($p = 0.2$, $L_{\max} = 10$), then the starting point of the span is selected uniformly at random. Similar to BERT, for 80% of spans, all tokens are replaced by `[MASK]`, for 10%, with random tokens, and for 10% with the original tokens. Second, SpanBERT introduces the **Span Boundary Objective** (SBO), in which the *tokens* at each masked position within the span must be predicted from the representations of the observed tokens at the boundaries of the span, plus the positional embedding for the target token. In particular, let $(x_1, \ldots, x_n)$ denote the tokens in an input sentence, and let $(x_s, \ldots, x_e)$ denote the tokens in the span, where $s, e \in \{1, \ldots, n\}$ are integers identifying the start and end positions of the span. For each $i \in \{s, \ldots, e\}$, the task is to predict $x_i \in V$ given (r_{s-1}, r_{e+1}, p_i) where r denotes the representation, and p the positional embedding. Third, in contrast to BERT, whose input includes two subsequences that may or may not be contiguous, the input to SpanBERT is a single contiguous sequence. Consequently, SpanBERT discards the next sentence prediction (NSP) task employed by BERT.

SpanBERT outperforms BERT on multiple tasks, including question-answering, coreference resolution, relation extraction, and GLUE. Interestingly, although generally outperformed by SpanBERT, BERT trained on single contiguous sequences and without NSP significantly outperformed the original BERT. Joshi *et al* [24] speculate that the reason NSP proved useful to the original BERT was because the model input still consisted of two potentially unrelated subsequences during the ablation analysis where NSP was removed. For SpanBERT, masking random spans typically outperforms masking linguistic units, such as named entities or noun phrases. The SBO is also shown to improve performance.

4.10 (07/2019) RoBERTa

Liu *et al* [28] find that the original BERT was under-trained, and introduce RoBERTa (Robustly optimized BERT approach), a model with the same architecture as BERT but optimized hyperparameter settings. RoBERTa is trained with dynamic masking on longer sequences and more data, with larger batch sizes and without the next-sentence prediction loss. The original BERT employed a *static masking* setup, where the data were masked only once during preprocessing (although multiple copies of each sentences with different masks were included in the training data). By contrast, RoBERTa employs *dynamic masking*, where a new mask is generated each time a training sentence is loaded. Performance was improved by packing sequences with full contiguous sentences. Although restricting to sentences that did not cross document boundaries led to the best performance, crossover between documents was allowed in order to keep the batch size constant. The batch size was significantly increased, from 256 to 8K, via *gradient accumulation*, in which multiple mini-batches are processed before taking each optimization step.

4.11 (09/2019) Language models as knowledge bases

Language model analysis (LAMA) is an evaluation intended to test the relational knowledge content of language models [29]. The evaluation is based on a corpus of facts encoded as (subject, relation, object) triples and (question, answer) pairs. Each fact is converted into a cloze statement, which the language model is tasked with completing. Evaluation is based on the mean precision@k, the probability that the correct completion occurs within the top k results. BERT, whose training is not tailored to relation extraction, is competitive with, and in some cases outperforms, a specialized relation extraction knowledge base. In fact, the knowledge base is competitive with BERT only when operating in oracle mode, which forces an extracted relation to attach the correct entities. The performance of BERT is best for 1-to-1 relations, and declines for many-to-1 and many-to-many; interestingly, the knowledge base exhibits the opposite behavior. BERT under-performs the baseline on question-answering, but the gap diminishes when the metric is relaxed from P@1 to P@10. The authors note that performance is sensitive to the framing of the cloze statement. In this way, performance based on a fixed template provides only a lower bound of what the model knows.

4.12 (09/2019) ALBERT

ALBERT (A Lite BERT) [25] employs a similar architecture to BERT but implements two parameter-reduction techniques to reduce memory requirements and accelerate training: embedding factorization and cross-layer parameter sharing. In BERT, the vocabulary embedding dimension E is tied to the hidden layer size H (i.e. $E = H$). ALBERT decouples these, resulting in significant parameter savings when $E \ll H$. Specifically, in ALBERT, each token in the vocabulary is initially represented by an embedding of dimension E (e.g. 128), which is subsequently projected into the hidden state dimension H (e.g. 768). By default, ALBERT also implements full parameter sharing across layers, including in the attention and feed-forward modules. Although parameter sharing objectively hurts performance when the model architecture is fixed, the reduction in the total number of free parameters may facilitate training an ALBERT model with more layers or a larger hidden state (which will likely still have fewer parameters than a BERT model of smaller size). For example, a BERT large model with $L = 24$ layers and $H = 1024$ has 334M parameters. An ALBERT of the same size has only 18M parameters, and even an ALBERT XXL model with $L = 24$ layers and $H = 4096$ has only 235M parameters.

Beyond parameter efficiencies, ALBERT introduces the sentence order prediction (SOP) task as a replacement for the next sentence prediction (NSP) task from BERT. In NSP, the negative examples (i.e. cases where the second sentence did not follow the first) were selected from different documents, often resulting in a candidate sentence from different topics. By contrast, in SOP the negative examples are formed by reversing the order of two contiguous sentences, reducing the risk of a change in topic. SOP is a more difficult, and thereby more instructive, task than NSP. This is demonstrated by showing that a model trained on SOP learns to perform NSP, but a model trained on NSP cannot effectively perform SOP.

4.13 (10/2019) DistilBERT

Knowledge distillation is a technique where a compact model is trained to emulate the behavior of a larger teacher model, which may be an ensemble. In DistilBERT, Sanh *et al* [32] train a model that is 40% smaller and 60% faster than BERT, while retaining similar performance across the GLUE benchmark. The loss has 3 components. One component is the same masked language modeling (MLM) loss employed in BERT. The remaining components are a student-teacher cross-entropy loss and a student-teacher cosine embedding loss. The cross-entropy loss takes the form $-\sum_{i \in V} t_i(x) \ln s_i(x)$, where $t_i(x)$ and $s_i(x)$ respectively denote the probabilities assigned to vocabulary token i by the student and the teacher, given input x. Following Hinton *et al* [6], the probabilities are calculated via a softmax with temperature. Training takes place at an elevated temperature $T > 1$, resulting in a higher-entropy ("softer") distribution over the vocabulary. While the cross-entropy loss aligns the model outputs, the cosine similarity loss aligns the hidden states. Among the 3 components of the loss, the alignment objectives contributed significantly more to the overall performance on the GLUE benchmark than the MLM objective. Initializing weights based on those in the teacher network was also essential, and only possible because the student network had a similar architecture to the original BERT (though with half as many layers).

4.14 (10/2019) T5*

Raffel *et al* train a text-to-text transfer transformer (T5) model that regards all NLP problems as text-to-text mapping [31]. To place all tasks within a common text-to-text paradigm, inputs are prefixed with key words specifying the task to be performed, such as `translate English to German` or `summarize`. This approach enables many tasks to share a common loss function and decoding procedure. The baseline model has an architecture similar to BERT, although with both an encoder and a decoder stack, as in the original transformer paper [13]. Sharing parameters between the encoder and decoder led to only a minor drop in performance. Although doing so reduces the parameter count by a factor of 2, it does not reduce the computational complexity.

A language model (LM) is defined as a model that simply predicts the next token in an auto-regressive manner. In a standard LM, such as the transformer decoder, the model can only attend to positions $i < j$ when generating the output token at position j. A variation is the LM with a prefix, in which the model has full visibility for tokens in the prefix and causal attention for tokens produced thereafter. The **language modeling** objective, as used by GPT, is to predict the next token given those preceding. The **denoising** objective, as used by BERT, is to predict the original values of tokens that have been masked or otherwise corrupted. The denoising objective is found to produce the best model performance, with variations on the denoising objective (e.g. different masking rates, or masking spans versus individual tokens) making only minor differences. Given this similar performance, Raffel *et al* suggest selecting the objective to optimize computational cost.

T5 is trained on the Colossal Cleaned Crawled Corpus, a curated subset of the Common Crawl. A variety of heuristics, such as removing documents that contained placeholder expressions or code, were applied to clean the data set. Although substantially larger, training on the uncleaned version of the data uniformly degraded performance. By training models on data sets of equal size but differing levels of repetition, the authors show that having more unique text on which to train results in better models, although some degree of repetition is admissible.

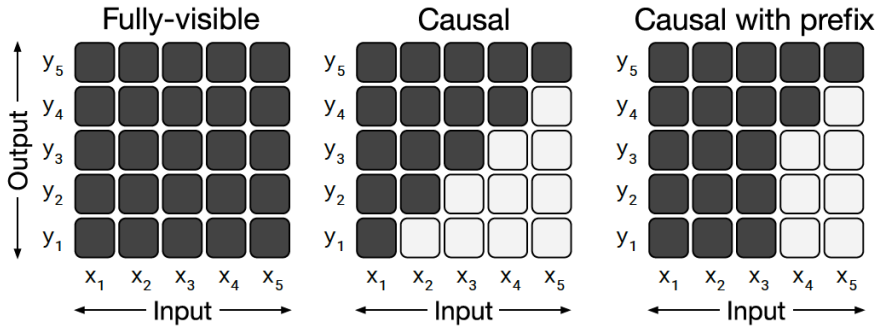
Several approaches to fine-tuning are explored. The *adapter layer* method inserts additional dense layers after the feedforward layers in existing transformer blocks and fine-tunes only these parameters. The *gradual unfreezing* method starts by only fine-tuning parameters of the last layer, then the last two layers, and so on until all layers are fine-tuned. Ultimately, simple fine-tuning of all parameters in the original network, without adapters or gradual fine-tuning, led to the best performance on downstream benchmarks. The baseline strategy of pretraining on the denoising objective followed by task-specific fine-tuning outperformed a multi-task training strategy, where the model was trained on multiple tasks at the same time. Fine-tuning after multi-task pretraining achieved performance comparable to the fine-tuning after unsupervised pretraining.

In scaling experiments, both training larger models and training longer improved performance. Of the two, training a larger model for fewer steps often outperformed training a smaller model for more steps. In practice, the benefits of training a larger model should be weighed against the added cost of more-expensive inference. In addition, the cost of training longer can be amortized if the base model is fine-tuned for multiple downstream tasks. Ensembling was also found to improve performance, even when the ensemble consisted of a single base-model fine-tuned in different ways (training multiple separate models for the ensemble performs better still). The final T5-11B model sets a new state of the art on multiple tasks, including GLUE and SuperGLUE.

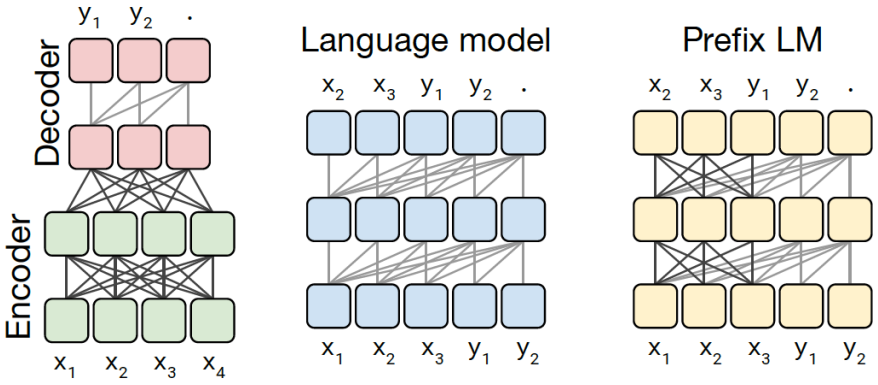
4.15 (10/2019) BART

The Bidirectional and Auto-Regressive Transformers (BART) model combines features of both BERT and GPT [27]. BART is a denoising sequence-to-sequence auto-encoder that applies bidirectional encoding to the corrupted input text, and auto-regressively generates output text. BART combines various methods of introducing noise, including masking individual tokens, spans of tokens, or simply inserting mask tokens. When masking spans, a single `[MASK]` replaces the entire span, forcing the model to learn how long the in-filling text should be. Additional noising procedures, not based on masking, include token deletion,

(a) **Attention masks**



(b) **Model architectures**



(c) **Unsupervised objectives**

Objective	Inputs	Targets
Prefix language modeling	Thank you for inviting	me to your party last week .
BERT-style Devlin et al. (2018)	Thank you <M> <M> me to your party apple week .	*(original text)*
Deshuffling	party me for your to . last fun you inviting week Thank	*(original text)*
MASS-style Song et al. (2019)	Thank you <M> <M> me to your party <M> week .	*(original text)*
I.i.d. noise, replace spans	Thank you <X> me to your party <Y> week .	<X> for inviting <Y> last <Z>
I.i.d. noise, drop tokens	Thank you me to your party week .	for inviting last
Random spans	Thank you <X> to <Y> week .	<X> for inviting me <Y> your party last <Z>

Figure 3. **Source:** Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer [31].

sentence permutation, and document rotation. BART employs the cross entropy loss with label smoothing. For discriminative tasks, the input to the classifier is the final state of the *last* token from the decoder. For translation tasks, a small adapter specific to the source-language is added upstream of the pre-trained encoder. The adapter translates the input into noisy English, which the remainder of the model denoises. BART achieves performance similar to RoBERTa on discriminative tasks, and sets new state of the art results of several generative tasks.

2020

5.1 (01/2020) Reformer

The Reformer architecture [44] aims to provide similar performance to the Transformer while improving computational efficiency. Standard dot-product attention has quadratic complexity in sequence length $\mathcal{O}(L^2)$. Reformer uses locality sensitive hashing (LSH), a technique for quickly finding approximate nearest neighbors in high-dimensional space, to group similar tokens into buckets. The hash function is chosen such that the buckets are similar size with high probability. Attention is only calculated among tokens belonging

to the same bucket. Throughput is further improved by breaking buckets into smaller chunks that are processed in parallel, and only allowing attention among tokens belonging to the same bucket in consecutive chunks. LSH reduces the computational cost of long-range approximate attention to $\mathcal{O}(L \ln L)$. In addition, Reformer employs reversible layers. Standard Transformers require storing the activations of each layer for use in calculating gradients during the backward pass. Reversible layers enable the activations to be easily recovered during the backward pass, eliminating the need to store them.

5.2 (01/2020) Scaling Laws

Kaplan *et al* [42] investigate how the performance of the transformer decoder model, as quantified by the cross entropy loss, scales with the number of model parameters N, the dataset size D, and the amount of compute C used for training. Performance follows smooth power-law scaling. Within reasonable limits, performance does not depend strongly on model architecture. For optimal performance, N should scale as $D^{3/4}$. Larger models are more sample efficient, achieving the same level of performance with less training time or fewer training examples. Training to convergence is inefficient: for a fixed compute budget, training a larger model for fewer steps is preferable to training a smaller model to convergence. The loss attainable upon further training can be predictably extrapolated from the curve of log loss versus the number of training steps.

5.3 (02/2020) REALM

The Retrieval Augmented Language Model (REALM) supplements masked language modeling (MLM) with a learned textual knowledge retriever [41]. REALM takes an input x and learns a distribution y over possible sequences $p(y|x)$. For the target task of open question-answering, x is the query, and y is the answer. REALM decomposes $p(y|x)$ into sampling documents z relevant to x, then generating y conditional on (x, z):

$$p(y|x) = \sum_z p(y|x, z)p(z|x). \tag{1}$$

The **retriever** is defined through an inner product model:

$$p(z|x) \propto \exp\{E_X(x)^\top E_Z(z)\}$$

where E_X is an input embedding model and E_Z is a document embedding model. Operationally, the embedding models return the `[CLS]` token from a BERT model. The **generator** (which the authors call an encoder) returns a distribution over y. Information on (x, z) is provided to the generator in the form of an embedding generated from the concatenation of x and z. The marginalization in (1) over all documents in the corpus is not tractable. Instead, the top K documents are retrieved using Maximum Inner Product Search (MIPS), which is sub-linear in the number of documents. In principle, embeddings for all documents in the corpus should be regenerated each time $p(z|x)$ is updated. In practice, re-embedding and re-indexing are performed asynchronously every several hundred training steps.

The model is pre-trained by MLM. Specifically, y consists of the identities of the masked tokens in x. The model is then fine-tuned towards open question-answering. The authors observe that, early in training when E_X and E_Z are poor, the retrieved documents are likely to be irrelevant, which could encourage the generator to ignore z. To fix this *cold-start problem*, E_X and E_Z are themselves pre-trained on an *inverse Cloze task* where, given a sentence x, the model must find the document z from which it came.

5.4 (03/2020) ELECTRA

Clark *et al* propose **replaced token detection** as a pretext task [39]. A random subset of tokens in the input are corrupted based on proposals from a small generative model (**Figure 4**), then the discriminator must decide for each token whether it is original or replaced. Replacement rather than masking resolves a training-evaluation discrepancy present in BERT where `[MASK]` tokens are only encountered during training. Adjudicating the originality of all tokens allows the model to learn from all tokens, not only those that have

been corrupted. Although the generator-discriminator architecture is similar to a GAN, the generator is trained by maximum likelihood rather than to fool the discriminator. The overall loss is:

$$\mathcal{L} = \mathcal{L}_{\text{MLM}} + \lambda \mathcal{L}_{\text{D}},$$

$$\mathcal{L}_{\text{MLM}} = -\mathbb{E} \sum_{i \in \text{masked}} \ln p_G(x_i | x_{\text{masked}}),$$

$$\mathcal{L}_{\text{D}} = -\mathbb{E} \sum_{t \in \text{seq}} \mathbb{I}(x_{\text{corrupt},t} = x_t) \ln p_D(t | x_{\text{corrupt}}) + \mathbb{I}(x_{\text{corrupt},t} \neq x_t) \ln\{1 - p_D(t | x_{\text{corrupt}})\}.$$

Here $p_G(x_i | x_{\text{masked}})$ is a probability distribution over the vocabulary for the ith masked token given the masked sequence, $\mathbb{I}(x_{\text{corrupt},t} = x_t)$ evaluates whether the token at position t in the corrupted sequence equals its original value, and similarly for $\mathbb{I}(x_{\text{corrupt},t} \neq x_t)$, and $p_D(t | x_{\text{corrupt}})$ is the probability that the token at position t is real given the corrupted sequence.

Benchmark performance was improved by having a lower capacity generator than discriminator and by training the generator with maximum likelihood rather than adversarially. In comparison with existing models, ELECTRA (Efficiently Learning an Encoder that Classifies Token Replacements Accurately) was more parameter and compute efficient.

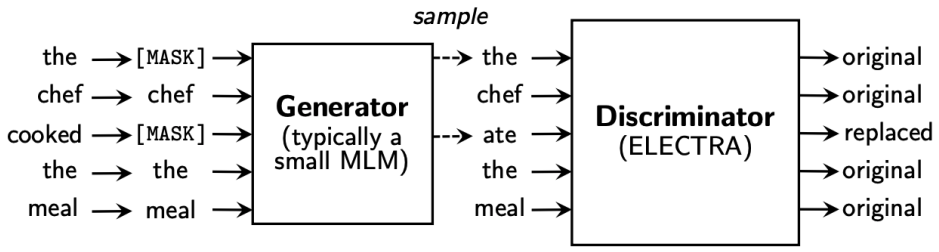


Figure 2: An overview of replaced token detection. The generator can be any model that produces an output distribution over tokens, but we usually use a small masked language model that is trained jointly with the discriminator. Although the models are structured like in a GAN, we train the generator with maximum likelihood rather than adversarially due to the difficulty of applying GANs to text. After pre-training, we throw out the generator and only fine-tune the discriminator (the ELECTRA model) on downstream tasks.

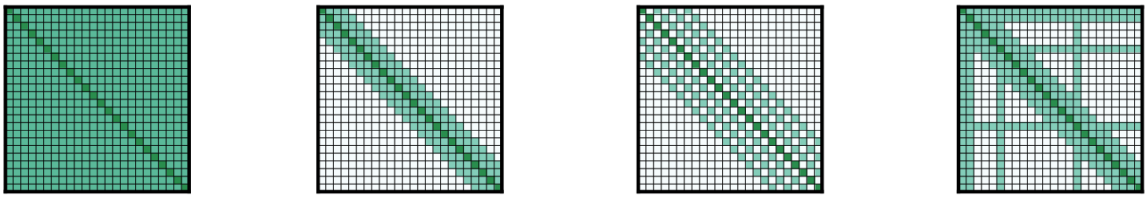
Figure 4. **Source:** ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators [39].

5.5 (04/2020) Longformer

Longformer addresses the quadratic time and memory complexity of full self-attention for long input sequences [36], reducing the computational complexity from $\mathcal{O}(n^2)$ to $\mathcal{O}(n)$, where n is sequence length. Longformer combines sliding-window local attention with global attention at specific positions, such as for the `[CLS]` token (**Figure 5**). The local context window can also be expanded via dilated attention, where gaps are placed between the attended tokens. For multi-headed dilated attention, it is beneficial to have different dilation patterns for different heads. Model weights and positional encodings were initialized based on RoBERTa [28]. Longformer extends the context window by a factor of 8 relative to BERT, from 512 to 4,096.

5.6 (04/2020) Dense Passage Retrieval

Given a query q and a corpus $\mathcal{C}$, a **retriever** is a function R that maps to a smaller, *filtered* set of documents $\mathcal{C}_k$ of size $|\mathcal{C}_k| = k \ll |\mathcal{C}|$. The goal of the Dense Passage Retriever (DPR) is to index all passages to a continuous, low-dimensional representation that allows for efficient retrieval of the top k passages [43]. A query encoder E_Q maps q to the embedding space, and a separate passage encoder E_P does likewise for each



(a) Full n^2 attention (b) Sliding window attention (c) Dilated sliding window (d) Global+sliding window

Figure 2: Comparing the full self-attention pattern and the configuration of attention patterns in our Longformer.

Figure 5. **Source:** Longformer: The Long-Document Transformer [36]

.

passage p. A BERT model is selected for each encoder, and the embedding is the final state of the `[CLS]` token. Similarity between a query and a passage is determined by the inner product:

$$\text{sim}(q, p) = E_Q(q)^\top E_P(p).$$

Given training data composed of queries q_i, positive text passages p_i^+, and 1 or more negative text passages $p_{i,1}^-, \ldots, p_{i,n}^-$, the training objective is to minimize the InfoNCE loss:

$$\mathcal{L} = -\sum_{i=1}^{N} \ln \left[\frac{\exp\{\text{sim}(q_i, p_i^+)\}}{\exp\{\text{sim}(q_i, p_i^+)\} + \sum_{j=1}^{n} \exp\{\text{sim}(q_i, p_{i,j}^-)\}} \right].$$

Negatives can be chosen as the positive pairs for other queries in the mini-batch.

5.7 (05/2020) Retrieval-Augmented Generation

Retrieval-augmented generation (RAG) [45] couples a parametric sequence-to-sequence transformer model with a "non-parametric" memory store, namely a dense vector index of Wikipedia. A **retriever** $p_\eta(z|x)$ returns a distribution over text passages z given the input x. A **generator** $p_\theta(y_i|x, z, y_{1:(i-1)})$ auto-regressively produces the output sequence conditional on the input, the retrieved document z, and the preceding output tokens $y_{1:(i-1)}$. The retriever is based on dense passage retrieval [43]:

$$p_\eta(z|x) \propto \exp\{d(z)^\top q(x)\}$$

where $q(x)$ is a *query* embedding and $d(z)$ is a *document* embedding, both produced with a BERT-base document encoder. The generator $p_\theta(y_i|x, z, y_{1:(i-1)})$ is a BART model [27]. Two styles of decoder are considered. The **RAG-seq** model retrieves the top K documents, produces an output sequence for each, then marginalizes over documents:

$$p(y|x) \approx \sum_{z \in \text{Top K}} p_\eta(z|x) \prod_{i=1}^{N} p_\theta(y_i|x, z, y_{1:(i-1)}).$$

The **RAG-token** model instead marginalizes over documents at the token level:

$$p(y|x) \approx \prod_{i=1}^{N} \sum_{z \in \text{Top K}} p_\eta(z|x) p_\theta(y_i|x, z, y_{1:(i-1)})$$

RAG-seq set the state of the art on several question-answering tasks, and RAG-token achieved good performance on text-generation tasks, outperforming existing models in terms of output diversity, specificity, and factuality. RAG borrows many ideas from REALM [41].

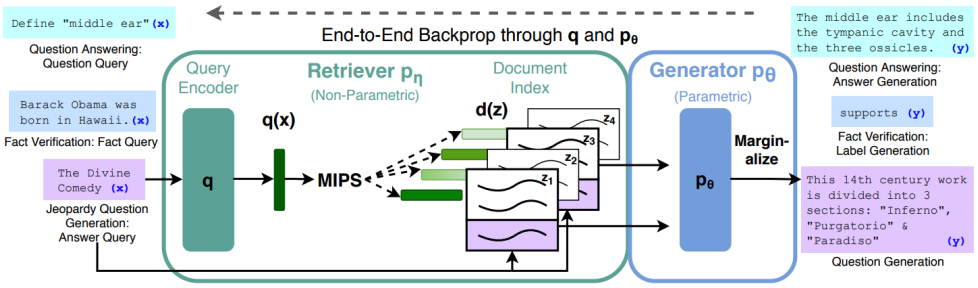


Figure 1: Overview of our approach. We combine a pre-trained retriever (*Query Encoder + Document Index*) with a pre-trained seq2seq model (*Generator*) and fine-tune end-to-end. For query x, we use Maximum Inner Product Search (MIPS) to find the top-K documents z_i. For final prediction y, we treat z as a latent variable and marginalize over seq2seq predictions given different documents.

Figure 6. **Source:** Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks [45].

5.8 (05/2020) GPT-3

GPT-3 is a 175B parameter language model trained in an autoregressive manner on a massive and diverse corpus of text [37]. GPT-3 exhibits **meta-learning**, or the ability to adapt to a wide variety of tasks based on the language understanding acquired through self-supervised pre-training. Given only a text description plus zero or more examples as *conditioning*, GPT-3 achieves state of the art performance on many tasks. Importantly, in contrast to traditional fine-tuning, this **in-context learning** involves no gradient-based parameter updates. Moreover, whereas fine-tuning generally requires a moderately-sized labeled data set, a small number of examples is typically sufficient for in-context learning. Three modes of in-context learning are distinguished based on the number of examples provided: zero-shot (no examples), one-shot (a single example), and few-shot (typically 10 to 100 examples; as many as fit within the context window). Larger models exhibit stronger in-context learning abilities, suggesting that a better baseline understanding of language improves downstream task performance. This supports the hypothesis that scale enhances meta-learning.

5.9 (06/2020) Funnel Transformer

Dai *et al* [40] observe that for many downstream tasks, a vectorial representation is desired for the entire sequence rather than each individual token. As such, producing encodings for the full sequence is unnecessary and computationally wasteful. Funnel-Transformer employs a series of transformer blocks with progressively shorter sequence lengths. Down-sampling is achieved by passing the sequence of hidden states through a strided mean-pooling layer. The shortened sequence is then passed to an attention layer as the query, with the original sequence supplied for generating keys and values. For pre-training tasks that require token-level representations, such as masked-language modeling, a decoder is employed to up-sample from the final sequence length to the original sequence length. For multiple components of the GLUE benchmark, Funnel-Transformer outperforms the standard transformer while requiring fewer floating point operations.

5.10 (06/2020) Linformer

The main efficiency bottleneck in transformers is calculation of the token-by-token self-attention, which has complexity quadratic in the sequence length n. Common approaches to accelerate this calculation include using mixed-precision or sparse attention. In the latter case, inner products are only calculated between a subset of tokens, for example based on distance or similarity. Linformer [46] reduces the complexity of self-attention to $\mathcal{O}(nk)$ by adopting a low-rank approximation. Recall that the output of an attention layer is expressible as:

$$\text{Attn}(Q, K, V) = \text{Softmax}(d^{-1/2}QK^\top)V,$$

where Q, K, and V are each typically $n \times d$ matrices. Linformer introduces two additional, learnable projections $E, F \in \mathbb{R}^{n \times k}$ where $k \ll n$, and calculates attention as:

$$\text{Attn}(Q, EK, FV) = \text{Softmax}\big\{d^{-1/2}Q(EK)^\top\big\}FV.$$

In essence, Linformer projects the sequence length down to dimension $k \ll n$, then calculates full attention on the reduced sequence. The paper experiments with several methods of parameter sharing, including using a common (E, F) across attention heads, using a single $E = F$ for each layer, and most restrictively, using a single $E = F$ for all layers. The last approach achieved the best performance in a fine-tuning experiment.

5.11 (07/2020) BigBird

The BigBird paper [47] proposes a sparse attention mechanism that reduces time and memory complexity from quadratic to linear, while retaining the theoretical guarantees of the full Transformer, such as being Turing complete and a universal sequence-to-sequence approximator. The attention mechanism is viewed as a directed graph over a vertex set of length n, equal to the input sequence length $x = (x_1, \ldots, x_n)$. Connections in this graph are described by an adjacency matrix A, where $A_{ij} = 1$ if query i attends to key j, and $A_{ij} = 0$ otherwise. The original Transformer is characterized by a fully connected graph, with $A_{ij} = 1$ for $\forall(i, j)$. Reducing the complexity of self-attention can thus be viewed as a graph sparsification problem. For the BigBird model, there are three types of connectivity. All tokens exhibit local connections, attending to and being attended to by tokens within a sliding window. A subset of tokens exhibit global connections, both attending to and being attended to by all other tokens. Lastly, each query also attends to a random subset of r keys, in a manner that may not be symmetrical. BigBird achieved state-of-the-art performance on several question answering and document summarization tasks. It was also applied to biological tasks such as predicting promoter regions and chromatin markers directly from DNA sequences.

5.12 (09/2020) Performers

The Performer model [38] approximates the performance of regular full-rank softmax attention with provable accuracy using only linear (as opposed to quadratic) time and space complexity. The approximation employs Fast Attention Via positive Orthogonal Random features (FAVOR). For a sequence of length L, bidirectional scaled dot-product attention is expressible as:

$$\text{Attention}(Q, K, V) = D^{-1}AV, \qquad A = \exp(d^{-1/2}QK^\top), \qquad D = \text{diag}(A1_L).$$

Here $Q, K, V \in \mathbb{R}^{L \times d}$, d is the embedding dimension, and $D \in \mathbb{R}^{L \times L}$ is a diagonal normalization matrix with $D_{ii} = \sum_{l=1}^{L} \exp(d^{-1/2}q_i^\top k_l)$. Left-multiplication by D^{-1} performs row-wise normalization, such that:

$$D^{-1}A = \text{Softmax}(d^{-1/2}QK^\top).$$

FAVOR observes that $A_{ij} = \exp(d^{-1/2}q_i^\top k_j)$ is the exponential of an inner product, which can be approximated by:

$$A_{ij} \approx \phi(q_i)^\top \phi(k_j),$$

where $\phi(\cdot)$ is a random feature map. In particular, ϕ is selected as:

$$\phi(x) = \frac{h(x)}{\sqrt{m}}\left[f_1(\omega_1^\top x), \ldots, f_1(\omega_m^\top x), \ldots, f_l(\omega_1^\top x), \ldots, f_l(\omega_m^\top x)\right].$$

Here the ω_i are random vectors, the f_l are known functions, and $m \leq d$ is the number of random projections. For example, $h(x) = \exp(-||x||^2/2)$ coupled with $l = 2$, $f_1(u) = \exp(u)$ and $f_2(u) = \exp(-u)$ is considered. Ultimately, bidirectional dot-product attention is approximated by:

$$\text{Attention}(Q, K, V) \approx \tilde{D}^{-1}Q'(K')^\top V, \qquad \tilde{D} = \text{diag}\{Q'(K')^\top 1_L\},$$

where $Q' = \phi(Q)$ and $K' = \phi(K)$ are the matrices formed by applying ϕ row-wise to the queries and keys for each token. Observe that application of ϕ will *increase* the rows of Q and K from dimension d to dimension $r = m \times l > d$. The computation savings derives from never having to explicitly calculate $A \in \mathbb{R}^{L \times L}$. Lastly performance is improved by orthogonalizing the m random projection vectors $(\omega_1, \ldots, \omega_m)$ via the Gram-Schmidt procedure.

References

[1] Y Bengio et al. "A neural probabilistic language model". In: *JMLR* (Mar. 2003). https://dl.acm.org/doi/10.5555/944919.944966.

[2] T Mikolov et al. "Efficient Estimation of Word Representations in Vector Space". In: *arXiv* (Jan. 2013). https://arxiv.org/abs/1301.3781.

[3] D Bahdanau, K Cho, and Y Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate". In: *arXiv* (Sept. 2014). https://arxiv.org/abs/1409.0473.

[4] J Pennington, R Socher, and C Manning. "GloVe: Global Vectors for Word Representation". In: *ACL Anthology* (Oct. 2014). https://aclanthology.org/D14-1162.

[5] I Sutskever, O Vinyals, and QV Le. "Sequence to Sequence Learning with Neural Networks". In: *arXiv* (Sept. 2014). https://arxiv.org/abs/1409.3215.

[6] G Hinton, O Vinyals, and J Dean. "Distilling the Knowledge in a Neural Network". In: *arXiv* (Mar. 2015). https://arxiv.org/abs/1503.02531.

[7] S Ioffe and C Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *arXiv* (Feb. 2015). https://arxiv.org/abs/1502.03167.

[8] M Iyyer et al. "Deep Unordered Composition Rivals Syntactic Methods for Text Classification". In: *ACL Anthology* (July 2015). https://aclanthology.org/P15-1162/.

[9] R Sennrich, B Haddow, and A Birch. "Neural Machine Translation of Rare Words with Subword Units". In: *arXiv* (Aug. 2015). https://arxiv.org/abs/1508.07909.

[10] JL Ba, JR Kiros, and GJ Hinton. "Layer Normalization". In: *arXiv* (July 2016). https://arxiv.org/abs/1607.06450.

[11] D Hendrycks and K Gimpel. "Gaussian Error Linear Units (GELUs)". In: *arXiv* (June 2016). https://arxiv.org/abs/1606.08415.

[12] Y Wu, M Schuster, Z Chen, et al. "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation". In: *arXiv* (Sept. 2016). https://arxiv.org/abs/1609.08144.

[13] A Vaswani, N Shazeer, N Parmar, et al. "Attention Is All You Need". In: *arXiv* (June 2017). https://arxiv.org/abs/1706.03762.

[14] D Cer, Y Yang, S Kong, et al. "Universal Sentence Encoder". In: *arXiv* (Mar. 2018). https://arxiv.org/abs/1803.11175.

[15] J Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *arXiv* (Oct. 2018). https://arxiv.org/abs/1810.04805.

[16] J Howard and S Ruder. "Universal Language Model Fine-tuning for Text Classification". In: *arXiv* (Jan. 2018). https://arxiv.org/abs/1801.06146.

[17] PJ Liu, M Saleh, E Pot, et al. "Generating Wikipedia by Summarizing Long Sequences". In: *arXiv* (Jan. 2018). https://arxiv.org/abs/1801.10198.

[18] ME Peters, M Neumann, M Iyyer, et al. "Deep contextualized word representations". In: *arXiv* (Feb. 2018). https://arxiv.org/abs/1802.05365.

[19] A Radford et al. "Improving Language Understanding by Generative Pre-Training". In: (June 2018). https://paperswithcode.com/paper/improving-language-understanding-by.

[20] P Shaw, J Uszkoreit, and A Vaswani. "Self-Attention with Relative Position Representations". In: *arXiv* (Mar. 2018). https://arxiv.org/abs/1803.02155.

[21] E Alsentzer, JR Murphy, W Boag, et al. "Publicly Available Clinical BERT Embeddings". In: *arXiv* (Apr. 2019). https://arxiv.org/abs/1904.03323.

[22] Iz Beltagy, K Lo, and A Cohan. "SciBERT: A Pretrained Language Model for Scientific Text". In: *arXiv* (Mar. 2019). https://arxiv.org/abs/1903.10676.

[23] Z Dai, Z Yang, Y Yang, et al. "Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context". In: *arXiv* (Jan. 2019). https://arxiv.org/abs/1901.02860.

[24] M Joshi, D Chen, Y Liu, et al. "SpanBERT: Improving Pre-training by Representing and Predicting Spans". In: *arXiv* (July 2019). https://arxiv.org/abs/1907.10529.

[25] Z Lan, M Chen, S Goodman, et al. "ALBERT: A Lite BERT for Self-supervised Learning of Language Representations". In: *arXiv* (Sept. 2019). https://arxiv.org/abs/1909.11942.

[26] J Lee, W Yoon, S Kim, et al. "BioBERT: a pre-trained biomedical language representation model for biomedical text mining". In: *arXiv* (Jan. 2019). https://arxiv.org/abs/1901.08746.

[27] M Lewis, Y Liu, N Goyal, et al. "BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension". In: *arXiv* (Oct. 2019). https://arxiv.org/abs/1910.13461.

[28] Y Liu, M Ott, N Goyal, et al. "RoBERTa: A Robustly Optimized BERT Pretraining Approach". In: *arXiv* (July 2019). https://arxiv.org/abs/1907.11692.

[29] F Petroni, T Rocktaschel, P Lewis, et al. "Language Models as Knowledge Bases?" In: *arXiv* (Sept. 2019). https://arxiv.org/abs/1909.01066.

[30] A Radford, J Wu, R Child, et al. "Language Models are Unsupervised Multitask Learners". In: *OpenAI* (Feb. 2019). https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.

[31] C Raffel, N Shazeer, A Roberts, et al. "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer". In: *arXiv* (Oct. 2019). https://arxiv.org/abs/1910.10683.

[32] V Sanh et al. "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter". In: *arXiv* (Oct. 2019). https://arxiv.org/abs/1910.01108.

[33] Y Sun, S Wang, Y Li, et al. "ERNIE: Enhanced Representation through Knowledge Integration". In: *arXiv* (Apr. 2019). https://arxiv.org/abs/1904.09223.

[34] A Wang, Y Pruksachatkun, N Nangia, et al. "SuperGLUE: A Stickier Benchmark for General-Purpose Language Understanding Systems". In: *arXiv* (May 2019). https://arxiv.org/abs/1905.00537.

[35] Z Yang, Z Dai, Y Yang, et al. "XLNet: Generalized Autoregressive Pretraining for Language Understanding". In: *arXiv* (June 2019). https://arxiv.org/abs/1906.08237.

[36] I Beltagy, ME Peters, and A Cohan. "Longformer: The Long-Document Transformer". In: *arXiv* (Apr. 2020). https://arxiv.org/abs/2004.05150.

[37] TB Brown, B Mann, N Ryder, et al. "Language Models are Few-Shot Learners". In: *arXiv* (May 2020). https://arxiv.org/abs/2005.14165.

[38] K Choromanski, V Likhosherstov, D Dohan, et al. "Rethinking Attention with Performers". In: *arXiv* (Sept. 2020). https://arxiv.org/abs/2009.14794.

[39] K Clark et al. "ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators". In: *arXiv* (Mar. 2020). https://arxiv.org/abs/2003.10555.

[40] Z Dai et al. "Funnel-Transformer: Filtering out Sequential Redundancy for Efficient Language Processing". In: *arXiv* (June 2020). https://arxiv.org/abs/2006.03236.

[41] K Guu et al. "REALM: Retrieval-Augmented Language Model Pre-Training". In: *arXiv* (Feb. 2020). https://arxiv.org/abs/2002.08909.

[42] J Kaplan, S McCandlish, T Henighan, et al. "Scaling Laws for Neural Language Models". In: *arXiv* (Jan. 2020). https://arxiv.org/abs/2001.08361.

[43] V Karpukhin, B Oguz, S Min, et al. "Dense Passage Retrieval for Open-Domain Question Answering". In: *arXiv* (Apr. 2020). https://arxiv.org/abs/2004.04906.

[44] N Kitaev, L Kaiser, and A Levskaya. "Reformer: The Efficient Transformer". In: *arXiv* (Jan. 2020). https://arxiv.org/abs/2001.04451.

[45] P Lewis, E Perez, A Piktus, et al. "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks". In: *arXiv* (May 2020). https://arxiv.org/abs/2005.11401.

[46] S Wang et al. "Linformer: Self-Attention with Linear Complexity". In: *arXiv* (June 2020). https://arxiv.org/abs/2006.04768.

[47] M Zaheer, G Guruganesh, A Dubey, et al. "Big Bird: Transformers for Longer Sequences". In: *arXiv* (July 2020). https://arxiv.org/abs/2007.14062.