Pretraining and Finetuning

He He

NEW YORK UNIVERSITY

October 9, 2023

Table of Contents

Representation learning

What are good representations?

- Enable a notion of distance over text (word embeddings)
- Contains good features for downstream tasks

Representation learning

What are good representations?

- Enable a notion of distance over text (word embeddings)
- Contains good features for downstream tasks

Examples: negative the food is good but doesn't worth an hour wait

- Simple features (e.g. BoW) require complex models.
- Good features only need simple models (e.g. linear classifier) .

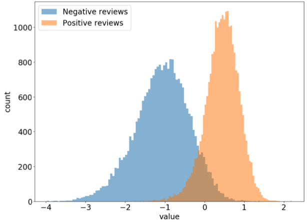


Figure: Sentiment neuron [Radford et al., 2017]

Representation learning

What can we do with good representations:

- Learning with small data: fine-tuning learned representations
- Transfer learning: one model/representation for many tasks
- Metric learning: get a similarity metric for free

Representation learning

What can we do with good representations:

- Learning with small data: fine-tuning learned representations
- Transfer learning: one model/representation for many tasks
- Metric learning: get a similarity metric for free

How to obtain such a representation:

- Training a neural network on any task gives us a representation good for *that task*.
- But on which task can we learn good *general* representations?

What can we learn from word guessing?

- The cats that are raised by my sister ________ sleeping.

What can we learn from word guessing?

- The cats that are raised by my sister ________ sleeping. *syntax*
- Jane is happy that John invited ________ friends to his birthday party.

What can we learn from word guessing?

- The cats that are raised by my sister ________ sleeping. *syntax*
- Jane is happy that John invited ________ friends to his birthday party. *coreference*
- ________ is the capital of Tanzania.

What can we learn from word guessing?

- The cats that are raised by my sister ______ sleeping. *syntax*
- Jane is happy that John invited ______ friends to his birthday party. *coreference*
- ______ is the capital of Tanzania. *knowledge*
- The boy is ______ because he lost his keys.

What can we learn from word guessing?

- The cats that are raised by my sister ________ sleeping. *syntax*
- Jane is happy that John invited ________ friends to his birthday party. *coreference*
- ________ is the capital of Tanzania. *knowledge*
- The boy is ________ because he lost his keys. *commonsense*
- John took 100 bucks to Vegas. He won 50 and then lost 100. Now he only has ________ to go home.

What can we learn from word guessing?

- The cats that are raised by my sister ________ sleeping. *syntax*
- Jane is happy that John invited ________ friends to his birthday party. *coreference*
- ________ is the capital of Tanzania. *knowledge*
- The boy is ________ because he lost his keys. *commonsense*
- John took 100 bucks to Vegas. He won 50 and then lost 100. Now he only has ________ to go home. *numerical reasoning*

What can we learn from word guessing?

- The cats that are raised by my sister ________ sleeping. *syntax*
- Jane is happy that John invited ________ friends to his birthday party. *coreference*
- ________ is the capital of Tanzania. *knowledge*
- The boy is ________ because he lost his keys. *commonsense*
- John took 100 bucks to Vegas. He won 50 and then lost 100. Now he only has ________ to go home. *numerical reasoning*

Word guessing entails many tasks related to language understanding!

Self-supervised learning

Key idea: predict parts of the input from the rest

- No supervision is needed—both input and output are from the raw data.
- Easy to scale—only need unlabeled data.
- Learned representation is general—useful for many tasks.

Self-supervised learning

Key idea: predict parts of the input from the rest

- No supervision is needed—both input and output are from the raw data.
- Easy to scale—only need unlabeled data.
- Learned representation is general—useful for many tasks.

Approach:

- **Pretrain**: train a model using self-supervised learning objectives on large data.
- **Finetune**: update part or all of the parameters of the pretrained model (which provides an initialization) on labeled data of a downstream task.

A bit of history

- Pretrain an RNN model on unlabeled data and finetune on supervised tasks [Dai et al., 2015] [ULMFiT; Howard et al., 2018]
  - Promising results on a small scale

A bit of history

- Pretrain an RNN model on unlabeled data and finetune on supervised tasks [Dai et al., 2015] [ULMFiT; Howard et al., 2018]
  - Promising results on a small scale
- ELMo: replace static word embedding by contextual word embeddings from pretrained bi-LSTM [Peters et al., 2018]
  - First impactful result in NLP

A bit of history

- Pretrain an RNN model on unlabeled data and finetune on supervised tasks [Dai et al., 2015] [ULMFiT; Howard et al., 2018]
  - Promising results on a small scale
- ELMo: replace static word embedding by contextual word embeddings from pretrained bi-LSTM [Peters et al., 2018]
  - First impactful result in NLP
- Pretrain a Transformer model and finetune on supervised tasks
  - GPT [Radford et al., 2018], BERT [Devlin et al., 2018]

A bit of history

- Pretrain an RNN model on unlabeled data and finetune on supervised tasks [Dai et al., 2015] [ULMFiT; Howard et al., 2018]
  - Promising results on a small scale
- ELMo: replace static word embedding by contextual word embeddings from pretrained bi-LSTM [Peters et al., 2018]
  - First impactful result in NLP
- Pretrain a Transformer model and finetune on supervised tasks
  - GPT [Radford et al., 2018], BERT [Devlin et al., 2018]
- Scale the pretrained model to larger sizes
  - GPT-2 (1.5B), T5 (11B), GPT-3 (175B), PaLM (540B)
  - We will talk about 100B+ models in the third module

Table of Contents

Types of pretrained models

- **Encoder models**, e.g., BERT
  - Encode text into vector representations that can be used for downstream classification tasks

Types of pretrained models

- **Encoder models**, e.g., BERT
  - Encode text into vector representations that can be used for downstream classification tasks
- **Encoder-decoder models**, e.g., T5
  - Encode input text into vector representations and generate text conditioned on the input

Types of pretrained models

- **Encoder models**, e.g., BERT
  - Encode text into vector representations that can be used for downstream classification tasks
- **Encoder-decoder models**, e.g., T5
  - Encode input text into vector representations and generate text conditioned on the input
- **Decoder models**, e.g., GPT-2
  - Read in text (prefix) and continue to generate text

Types of pretrained models

- **Encoder models**, e.g., BERT
  - Encode text into vector representations that can be used for downstream classification tasks
- **Encoder-decoder models**, e.g., T5
  - Encode input text into vector representations and generate text conditioned on the input
- **Decoder models**, e.g., GPT-2
  - Read in text (prefix) and continue to generate text

Types of pretrained models

- **Encoder models**, e.g., BERT
  - Encode text into vector representations that can be used for downstream classification tasks
- **Encoder-decoder models**, e.g., T5
  - Encode input text into vector representations and generate text conditioned on the input
- **Decoder models**, e.g., GPT-2
  - Read in text (prefix) and continue to generate text

Current pretrained models are all transformer based.

Encoder models

An encoder takes a sequence of tokens and output their *contextualized* representations:

$$h_1, \ldots, h_n = \mathrm{Encoder}(x_1, \ldots, x_n)$$

We can then use $h_1, \ldots, h_n$ for other tasks.

Encoder models

An encoder takes a sequence of tokens and output their *contextualized* representations:

$$h_1, \ldots, h_n = \mathrm{Encoder}(x_1, \ldots, x_n)$$

We can then use $h_1, \ldots, h_n$ for other tasks.

How do we train an $\mathrm{Encoder}$?

- Use any supervised task: $y = f(h_1, \ldots, h_n)$
- Use self-supervised learning: predict a word from its context

Masked language modeling

? language processing is ?

Masked language modeling

? language processing is ?

Learning objective (MLE):

$$\max \sum_{x \in \mathcal{D}, i \sim p_{\text{mask}}} \log p(x_i \mid x_{-i}; \theta)$$

- x: a sequence of tokens sampled from a corpus $\mathcal{D}$
  natural language processing is fun
- p_{mask}: mask generator
  Sample two positions uniformly at random, e.g., 1 and 5
- x_{-i}: noisy version fo x where x_i is corrupted
  [MASK] language processing is [MASK]

BERT: objective

- **Masked language modeling**:
  - Randomly sample 15% tokens as prediction targets
  - Replace the target tokens by `[MASK]` or a random token, or leave it unchanged

    cats are cute $\rightarrow$ cats `[MASK]`/is/are cute
  - Later work has shown that just use `[MASK]` is sufficient

BERT: objective

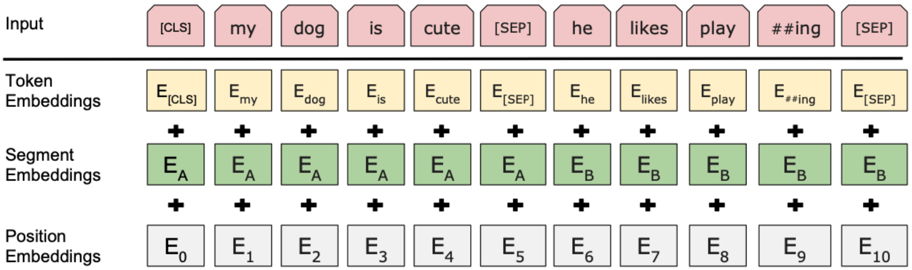
- **Masked language modeling**:
  - Randomly sample 15% tokens as prediction targets
  - Replace the target tokens by `[MASK]` or a random token, or leave it unchanged

    cats are cute $\rightarrow$ cats `[MASK]`/is/are cute
  - Later work has shown that just use `[MASK]` is sufficient
- **Next sentence prediction**: predict whether a pair of sentences are consecutive

$$\max \sum_{x \sim \mathcal{D}, x_n \sim p_{\text{next}}} \log p(y \mid x, x_n; \theta)$$

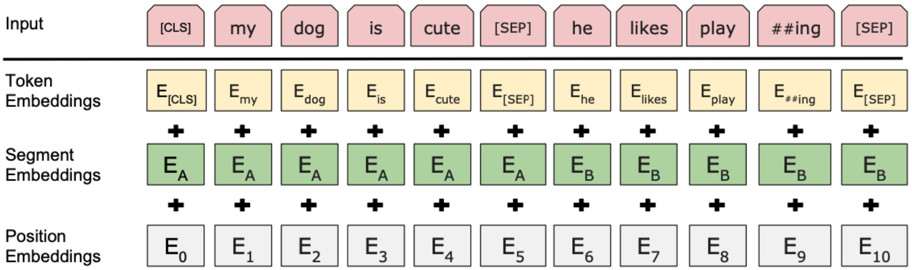
  - x_n: either the sentence following x or a randomly sampled sentence
  - y: binary label of whether x_n follows x
  - Later work has shown that this objective is not necessary

BERT: architecture



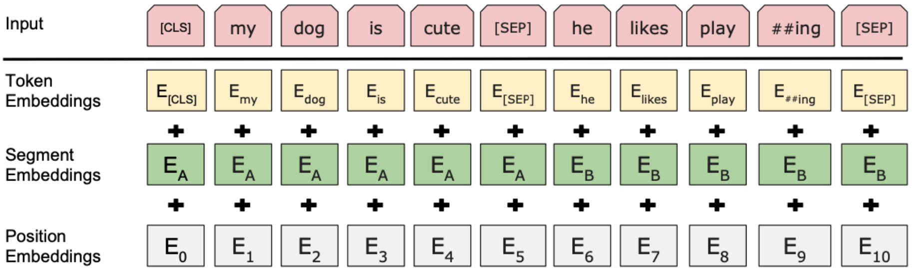
- Tokenization: wordpiece (similar to byte pair encoding) (see details)

BERT: architecture



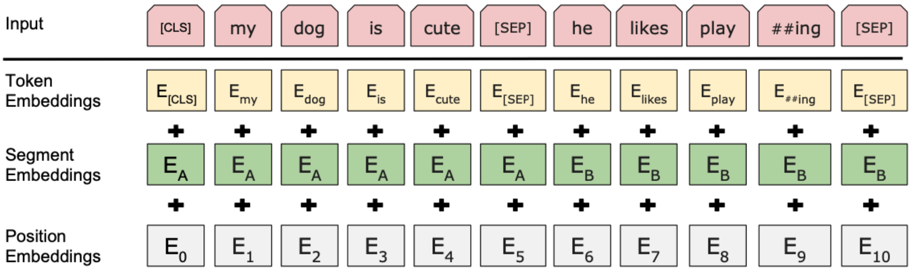
- Tokenization: wordpiece (similar to byte pair encoding) (see details)
- `[CLS]`: first token of all sequences; used for next sentence prediction

BERT: architecture



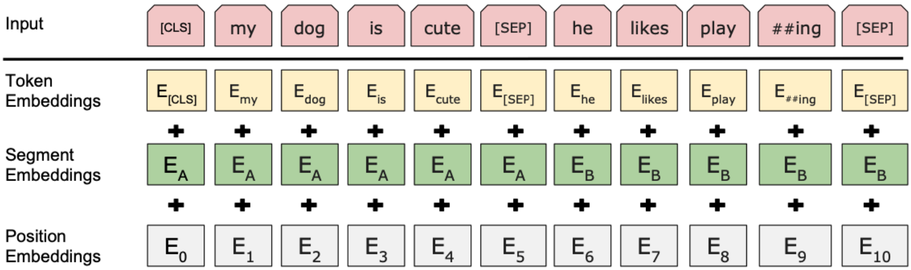
- Tokenization: wordpiece (similar to byte pair encoding) (see details)
- `[CLS]`: first token of all sequences; used for next sentence prediction
- Distinguish two sentences in a pair: `[SEP]` and segment embedding

BERT: architecture



- Tokenization: wordpiece (similar to byte pair encoding) (see details)
- `[CLS]`: first token of all sequences; used for next sentence prediction
- Distinguish two sentences in a pair: `[SEP]` and segment embedding
- Learned position embedding

BERT: architecture

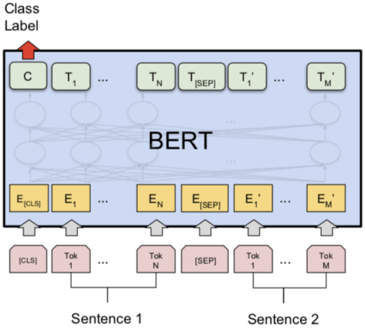


- Tokenization: wordpiece (similar to byte pair encoding) (see details)
- `[CLS]`: first token of all sequences; used for next sentence prediction
- Distinguish two sentences in a pair: `[SEP]` and segment embedding
- Learned position embedding
- 12 (base; 110M params) or 24 (large; 340M params) layer Transformer

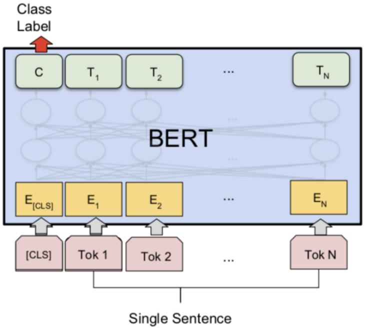
Finetuning BERT

Classification tasks: Add a linear layer (randomly initialized) on top of the [CLS] embedding

$$p(y \mid x) = \mathrm{softmax}(W h_{[\mathrm{CLS}]} + b)$$



(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG

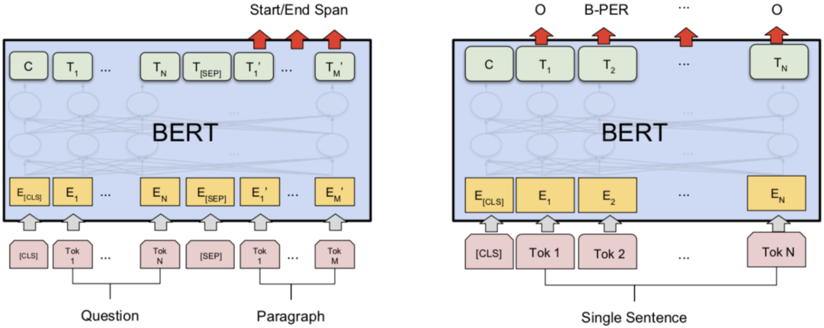


(b) Single Sentence Classification Tasks: SST-2, CoLA

Finetuning BERT

Sequence labeling tasks: Add linear layers (randomly initialized) on top of every token

$$p(y_i \mid x) = \mathrm{softmax}(Wh_i + b)$$



(c) Question Answering Tasks: SQuAD v1.1

(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

Finetuning BERT

- Finetune all parameters (both the newly added layer and the pretrained weights)
- Use a small learning rate (e.g., 1e-5)
- Train for a small number of epochs (e.g, 3 epochs)
- Led to SOTA results on many NLU tasks

How to generate text from BERT?

Encoder-decoder models

An encoder-decoder model encodes input text to a sequence of contextualized representations, and decodes a sequence of tokens autoregressively.

$$
\begin{aligned}
h_1, \ldots, h_n &= \mathrm{Encoder}(x_1, \ldots, x_n) \\
s_1, \ldots, s_m &= \mathrm{Decoder}(y_0, \ldots, y_{m-1}, h_1, \ldots, h_n) \\
p(y_i \mid x, y_{<i}) &= \mathrm{softmax}(W s_i + b)
\end{aligned}
$$

Encoder-decoder models

An encoder-decoder model encodes input text to a sequence of contextualized representations, and decodes a sequence of tokens autoregressively.

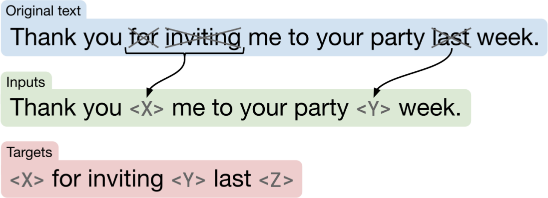
$$
\begin{aligned}
h_1, \ldots, h_n &= \mathrm{Encoder}(x_1, \ldots, x_n) \\
s_1, \ldots, s_m &= \mathrm{Decoder}(y_0, \ldots, y_{m-1}, h_1, \ldots, h_n) \\
p(y_i \mid x, y_{<i}) &= \mathrm{softmax}(W s_i + b)
\end{aligned}
$$

How do we train the encoder-decoder?

- Use any supervised task, e.g., machine translation
- Use self-supervised learning: predict text spans from their context

Masked language modeling using an encoder-decoder

Input: text with corrupted spans
Output: recovered spans

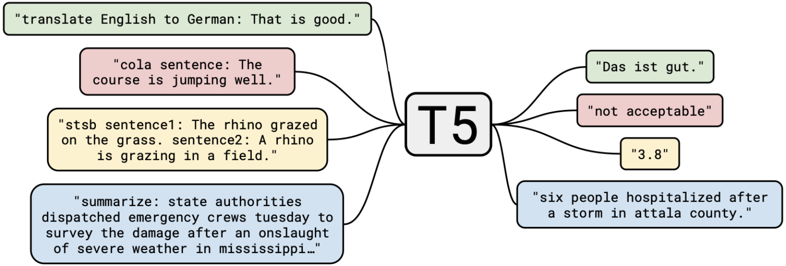


Compare with encoder-only models:

- Encoder: predict single tokens based on encoder representation
- Encoder-decoder: predict a sequence of tokens (flexibility in objective design)

T5: objective

- First train on unlabele data by **masked language modeling**
  - Predict corrupted spans as a sequence
- Then continue training by **supervised multitask learning**
  - Formulate tasks as text-to-text format using a prefix to denote the task
  - Mixing examples from different datasets when constructing batches



- Jointly training with the two objectives works slightly worse

T5: finetune

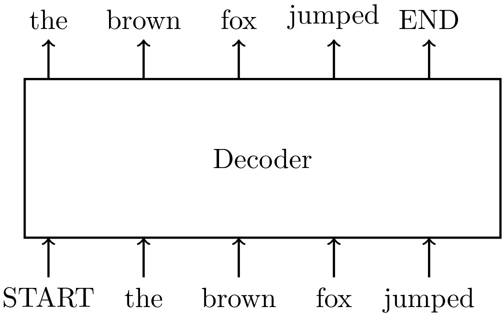
- Formulate the task in text-to-text format
- Fine-tune all parameters (similar to BERT fine-tuning)
- Advantages over encoder models: unified modeling of many different tasks including text generation

Decoder-only models

A decoder-only model predicts the next token given the prefix autoregressively.

$$s_1, \ldots, s_m = \mathrm{Decoder}(y_0, \ldots, y_{m-1}, h_1, \ldots, h_n)$$
$$p(y_i \mid y_{<i}) = \mathrm{softmax}(W s_i + b)$$

(A prefix of y can be the input.)



(more on language models later)

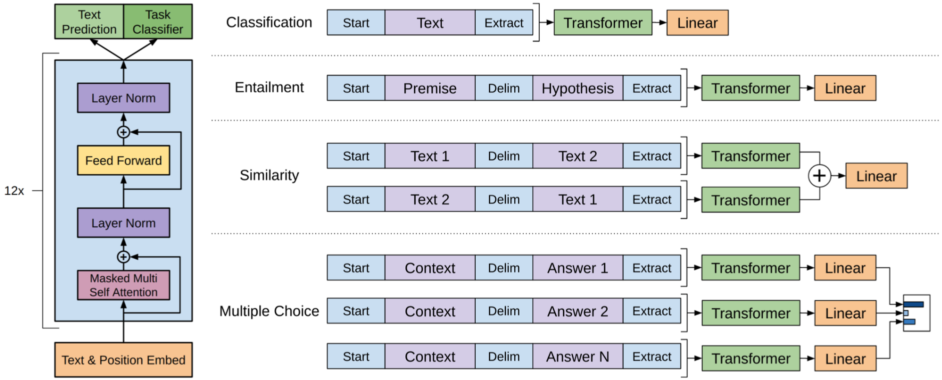
Generative Pretraining (GPT)

- **Model**: 12 layer decoder-only transformer
- **Objective**: next word prediction

$$\max \sum_{y \in \mathcal{D}} \sum_{i} \log p(y_i \mid y_{<i})$$

- **Finetuning**: auxiliary LM objective $L_{\text{task}} + \lambda L_{\text{LM}}$ (next word prediction on labeled task data)

Generative Pretraining (GPT): task-specific finetuning



- Single input: linear on top of `extract`
- Multiple input: process each input separately then aggregate

Ablation studies of GPT

Architecture, pretraining, finetuning: which is critical?

Method	Avg. Score	CoLA (mc)	SST2 (acc)	MRPC (F1)	STSB (pc)	QQP (F1)	MNLI (acc)	QNLI (acc)	RTE (acc)
Transformer w/ aux LM (full)	74.7	45.4	91.3	82.3	82.0	**70.3**	**81.8**	**88.1**	**56.0**
Transformer w/o pre-training	59.9	18.9	84.0	79.4	30.9	65.5	75.7	71.2	53.8
Transformer w/o aux LM	**75.0**	**47.9**	**92.0**	**84.9**	**83.2**	69.8	81.1	86.9	54.4
LSTM w/ aux LM	69.1	30.3	90.5	83.2	71.8	68.1	73.7	81.1	54.6

- Auxiliary objective only helps on larger datasets (MNLI, QQP)
- Pretrained transformer $>$ pretrained LSTM (single layer) $>$ non-pretrained transformer

Compare with BERT

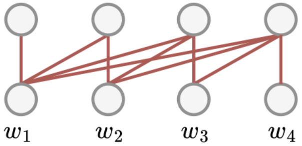
System	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	**Average**
	392k	363k	108k	67k	8.5k	5.7k	3.5k	2.5k	-
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1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

Table 1: GLUE Test results, scored by the evaluation server (https://gluebenchmark.com/leaderboard). The number below each task denotes the number of training examples. The "Average" column is slightly different than the official GLUE score, since we exclude the problematic WNLI set.[8] BERT and OpenAI GPT are single-model, single task. F1 scores are reported for QQP and MRPC, Spearman correlations are reported for STS-B, and accuracy scores are reported for the other tasks. We exclude entries that use BERT as one of their components.

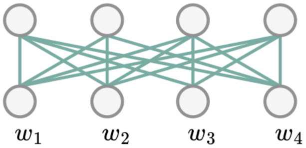
Medium-sized encoder models tend to work better than decoder-only models when finetuned

Encoder-only vs decoder-only models: attention

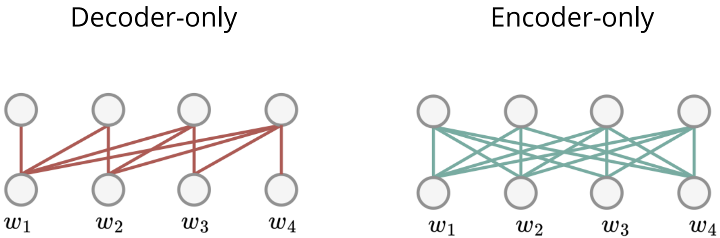
Decoder-only



Encoder-only



Encoder-only vs decoder-only models: attention



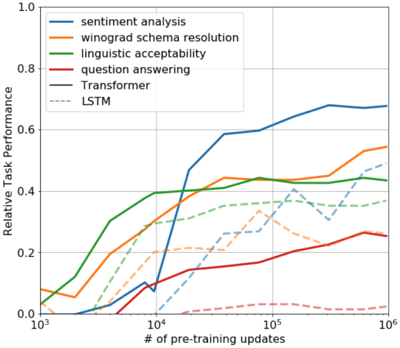
Encoder-only models provides better embeddings due to bidirectional attention.

Encoder-only vs decoder-only models: generation

Decoder-only models can make predictions through generation *without finetuning*

Encoder-only vs decoder-only models: generation

Decoder-only models can make predictions through generation *without finetuning*



Heuristics for zero-shot prediction:

- Sentiment classification: [example] + very + {positive, negative} *prompting*
- Linguistic acceptability: thresholding on log probabilities
- Multiple choice: predicting the answer with the highest log probabilities

Scaling trend: zero-shot performance increases during pretraining

Encoder-only vs decoder-only models: training efficiency

On each sequence:

- Encoder-only models are trained on 15% (mask rate) of the tokens
- Decoder-only models are trained on all tokens

Encoder-only vs decoder-only models: training efficiency

On each sequence:

- Encoder-only models are trained on 15% (mask rate) of the tokens
- Decoder-only models are trained on all tokens

What about encoder-decoder models?

- Better for sequence-to-sequence tasks
- Need to maintain two separate architectures, additional cross attention
- Overall limited advantage over decoder-only models

What are these models trained on?

Both quantity and quality are important

- Wikipedia: encyclopedia articles (clean, single domain)
- Toronto Books Corpus: e-books (diverse domain)
- WebText (40GB): content submitted to Reddit with a vote ≥ 3 (diverse, bias)
- CommonCrawl (20TB): scraped HTML with markers removed (diverse, large, noisy, bias)
  - A cleaned version: C4 (750GB)

Active research area: What data is good for pretraining?

Table of Contents

Overview

Approaches to speed up pretraining

Overview

Approaches to speed up pretraining

- Reduce model size
- Design more sample-efficient learning objectives
- Improve efficiency of self-attention
- Improve system-level efficiency

Approach 1: Reduce model size

Idea 1: reduce the number of parameters

ALBERT (a lite BERT) [Lan et al., 2020]

- **Factorization**:
  - Recall that in Transformer, we first need to map the one-hot encoding (of size V) of a token to Q, K, V embeddings (of size H)
  - The number of parameters is $V \times H$
  - We can instead first map it to a lower-dim space (of size E) so that the number of params is $V \times E + E \times H$

Approach 1: Reduce model size

Idea 1: reduce the number of parameters

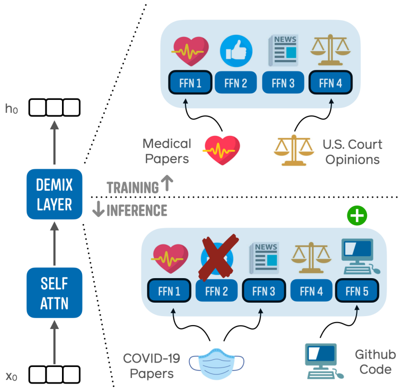
ALBERT (a lite BERT) [Lan et al., 2020]

- **Parameter sharing**:
  - Share feedforward network weights across layers
  - Share self-attention weights across layers
  - ALBERT: share all params across layers

Approach 1: Reduce model size

Idea 2: reduce interaction among parameters (sparse/modular architectures)

DEMix [Gururangan et al., 2022]



- Replace the FFN layer with an ensemble of n experts
- Route examples to experts corresponding to its domain determinstically

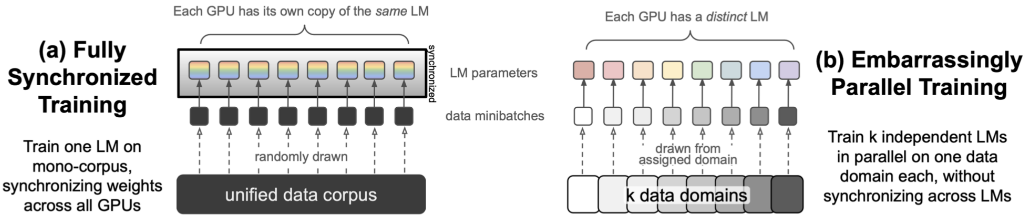
$$\mathrm{FFN}(h) = \sum_{i=1}^{n} \mathbb{I}[x \in \text{domain } i]\mathrm{FFN}_i(x)$$

- Only a subset of params are active for each example/batch

Approach 1: Reduce model size

Idea 2: reduce interaction among parameters (sparse/modular architectures)

Branch-Train-Merge [Li et al., 2022]



- Train domain experts in parallel and ensemble them (or take weighted average of their parameters)
- Reduce synchronization among GPUs at the cost of increased model size
- Easy to expand/remove domain experts

Approach 2: design sample-efficient learning objectives

ALBERT: Inter-sentence coherence loss

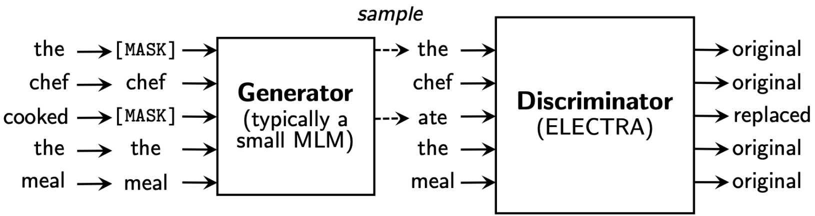
- Motivation: the next sentence prediction task is too easy
- Design hard negative examples
- Input: take two consecutive sentences, swap their order randomly
- Output: predict if they are in natural order

  I went home. SEP *I slept.* +1

  I slept. SEP *I went home.* -1
- Model needs to learn temporal order of events (commonsense, causality etc.)

Approach 2: design sample-efficient learning objectives

ELECTRA [Clark et al., 2020]: discriminate from true vs guessed tokens



- First train the generator for n steps using the MLM objective.
- Freeze generator weights. Train the discriminator using the sequence classification objective. Keep discriminator for finetuning.
- Comparison with MLM: predict at every position; hard negative examples.

Approach 2: design sample-efficient learning objectives

ELECTRA result:

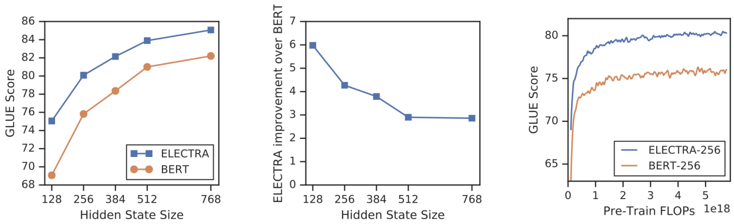


Figure: Finetuning result on the GLUE benchmark

- Larger improvement at smaller model sizes
- Faster training
- An effective approach if you don’t have large compute for pretraining

Approach 3: alternatives to self-attention

Transformer recap

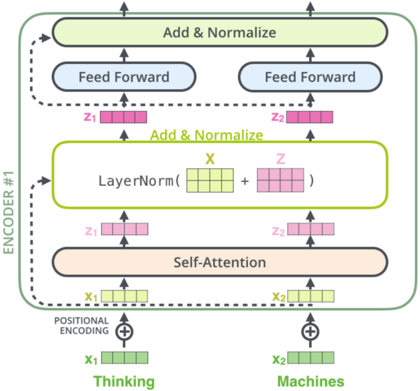


Figure: From The Illustrated Transformer

Which components require matrix multiplication?

Approach 3: alternatives to self-attention

Transformer recap

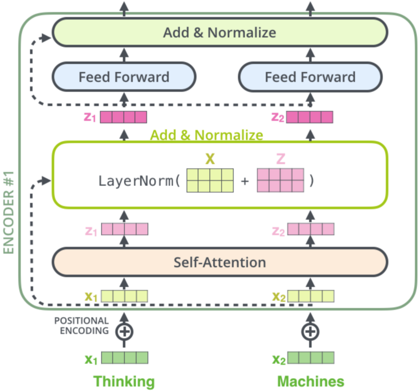


Figure: From The Illustrated Transformer

Which components require matrix multiplication?

- Self-attention
  - Q,K,V projection
  - Scaled dot-product attention
- Feed-forward layer

Compute cost of transformers

Q, K, V projection:

$$n \times d_e \xrightarrow{\text{linear}} n \times d$$

Scaled dot-product attention:

$$n \times d \quad d \times n \xrightarrow{\text{matmul}} n \times n$$

Compute cost of transformers

Q, K, V projection:

$$\boxed{n \times d_e} \xrightarrow{\text{linear}} \boxed{n \times d} \qquad O(n \times d_e \times d)$$

Scaled dot-product attention:

$$\boxed{n \times d} \quad \boxed{d \times n} \xrightarrow{\text{matmul}} \boxed{n \times n} \qquad O(d \times n^2)$$

Compute cost of transformers

Feed-forward layer (GPT-2):

$$\boxed{n \times d} \xrightarrow{\text{linear+ReLU}} \boxed{n \times d_h}$$

$$O(n \times d \times d_h)$$

- Two-layer FFN
- $d_h = 4d$ ($d > 1K$) by default in GPT-2
- Approximately half of the compute time

Compute cost of transformers

Feed-forward layer (GPT-2):

$$\boxed{n \times d} \xrightarrow{\text{linear+ReLU}} \boxed{n \times d_h} \xrightarrow{\text{linear+ReLU}} \boxed{n \times d}$$

$$O(n \times d \times d_h)$$

- Two-layer FFN
- $d_h = 4d$ $(d > 1K)$ by default in GPT-2
- Approximately half of the compute time

Improve efficiency of self-attention (for long sequences)

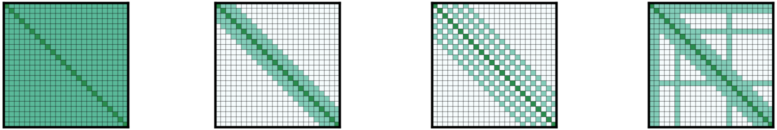
Improve efficiency of self-attention (for long sequences)

Key idea: reduce the $O(n^2)$ time and memory cost

- Sparsify the attention matrix
  - Deterministic mask
  - Data-dependent mask (Reformer [Kitaev et al., 2020])
- Compress the key-value memory
  - Low-rank projection
  - Attention-based projection

Sparse attention

Longformer [Beltagy et al., 2020]: attention within a local window

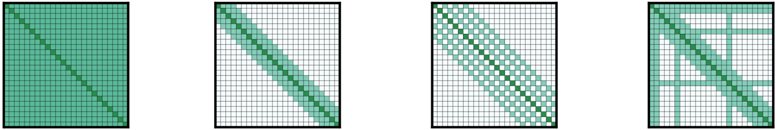


(a) Full n^2 attention (b) Sliding window attention (c) Dilated sliding window (d) Global+sliding window

- Sliding window: attending to a *local* window of size w around each token $O(n \times w)$
- Dilated sliding window: reaching *longer range* with a larger window size with gaps
- Global window: *full attention* on specific tokens, e.g., `[CLS]` in BERT

Sparse attention

Longformer [Beltagy et al., 2020]: attention within a local window

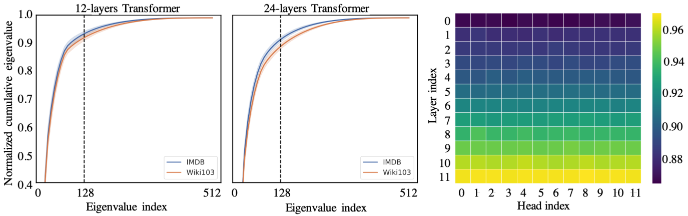


(a) Full n^2 attention (b) Sliding window attention (c) Dilated sliding window (d) Global+sliding window

- Sliding window: attending to a *local* window of size w around each token
  $O(n \times w)$
- Dilated sliding window: reaching *longer range* with a larger window size with gaps
- Global window: *full attention* on specific tokens, e.g., `[CLS]` in BERT
- Details: balancing efficiency and performance
  - Adding dilation on some heads
  - Using small window size on lower layers and larger ones on higher layers

Compresse the KV memory

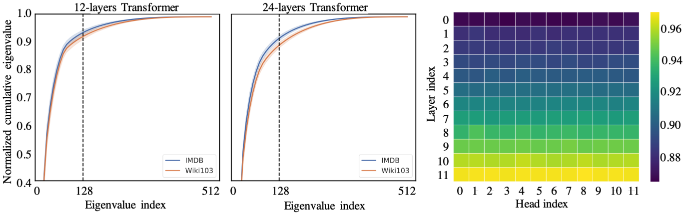
Self-attention is low rank [Wang et al., 2020]



- Left: cumulative eigenvalues of pretrained transformer with $n = 512$

Compresse the KV memory

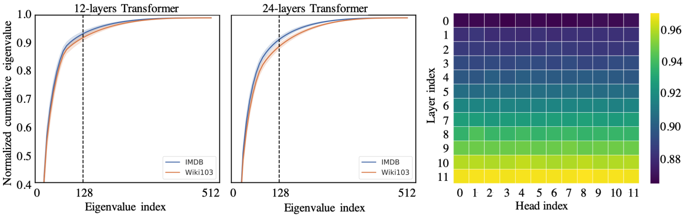
Self-attention is low rank [Wang et al., 2020]



- Left: cumulative eigenvalues of pretrained transformer with $n = 512$
  - Most information in the attention matrix can be recovered by the top 128 eigenvectors

Compresse the KV memory

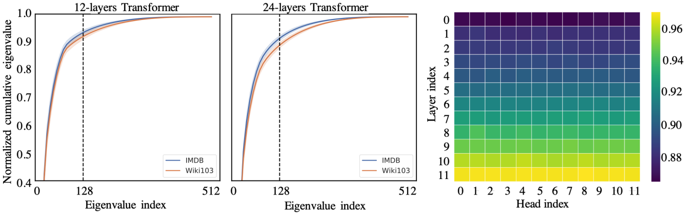
Self-attention is low rank [Wang et al., 2020]



- Left: cumulative eigenvalues of pretrained transformer with $n = 512$
  - Most information in the attention matrix can be recovered by the top 128 eigenvectors
- Right: cumulative eigenvalues of the top 128 eigenvalues across layers

Compresse the KV memory

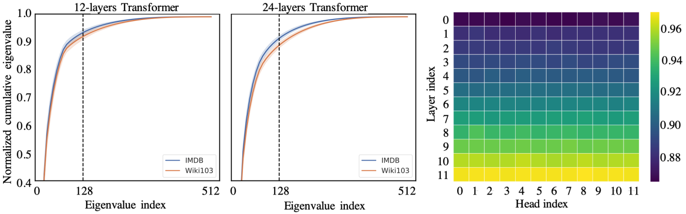
Self-attention is low rank [Wang et al., 2020]



- Left: cumulative eigenvalues of pretrained transformer with $n = 512$
  - Most information in the attention matrix can be recovered by the top 128 eigenvectors
- Right: cumulative eigenvalues of the top 128 eigenvalues across layers
  - Higher layers are more low-rank

Compresse the KV memory

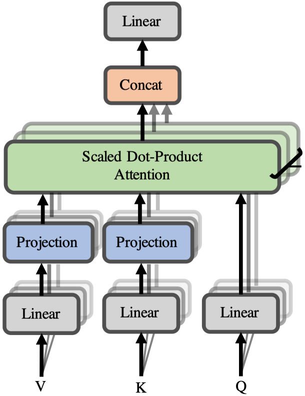
Self-attention is low rank [Wang et al., 2020]



- Left: cumulative eigenvalues of pretrained transformer with $n = 512$
  - Most information in the attention matrix can be recovered by the top 128 eigenvectors
- Right: cumulative eigenvalues of the top 128 eigenvalues across layers
  - Higher layers are more low-rank
- **Idea**: instead of attending to n tokens, attend to k principal components

Summarize the KV memory

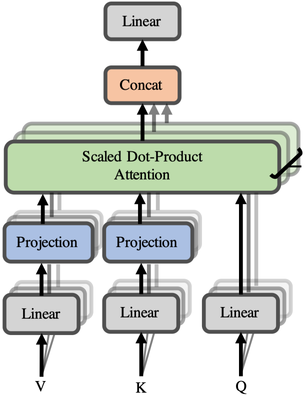
Linformer [Wang et al., 2020]: compute self-attention in a lower dimension



- Reduce dimensionality of the "memory": Map K, V from $n \times d$ to $k \times d$

Summarize the KV memory

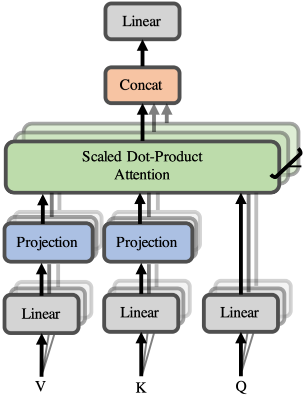
Linformer [Wang et al., 2020]: compute self-attention in a lower dimension



- Reduce dimensionality of the "memory": Map K, V from $n \times d$ to $k \times d$
- Attend to the lower-dimensional memory: $\mathrm{softmax}\left(Q_{n\times d} K^T_{k\times d} / \sqrt{d}\right)$

Summarize the KV memory

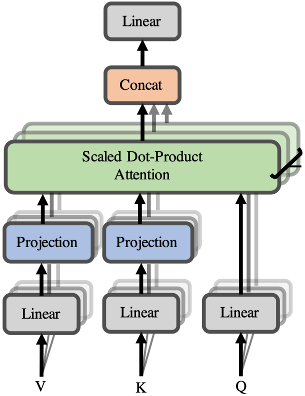
Linformer [Wang et al., 2020]: compute self-attention in a lower dimension



- Reduce dimensionality of the “memory”: Map K, V from $n \times d$ to $k \times d$
- Attend to the lower-dimensional memory: $\mathrm{softmax}\left(Q_{n\times d} K^T_{k\times d} / \sqrt{d}\right)$
  - What’s the dimension of the attention matrix?

Summarize the KV memory

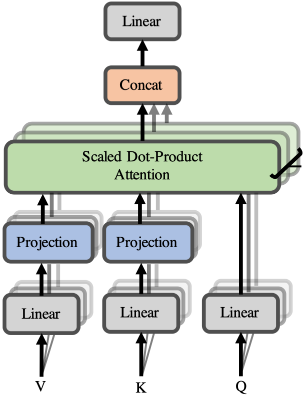
Linformer [Wang et al., 2020]: compute self-attention in a lower dimension



- Reduce dimensionality of the "memory": Map K, V from $n \times d$ to $k \times d$
- Attend to the lower-dimensional memory: $\mathrm{softmax}\left(Q_{n\times d} K^T_{k\times d} / \sqrt{d}\right)$
  - What's the dimension of the attention matrix?
  - What's the dimension of the self-attention output?

Summarize the KV memory

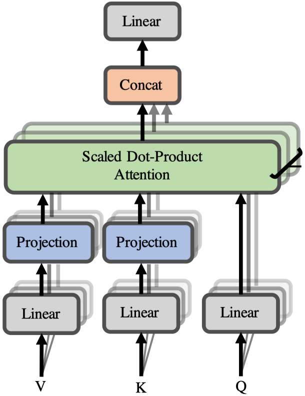
Linformer [Wang et al., 2020]: compute self-attention in a lower dimension



- Reduce dimensionality of the "memory": Map K, V from $n \times d$ to $k \times d$
- Attend to the lower-dimensional memory: $\mathrm{softmax}\left(Q_{n\times d} K^T_{k\times d} / \sqrt{d}\right)$
    - What's the dimension of the attention matrix?
    - What's the dimension of the self-attention output?
- Computation cost: $O(nk)$ (linear in n)

Summarize the KV memory

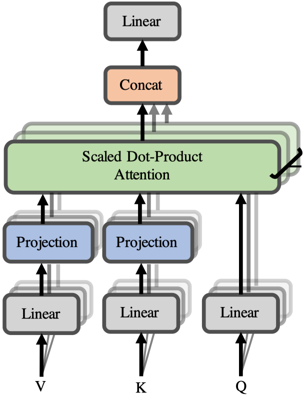
Linformer [Wang et al., 2020]: compute self-attention in a lower dimension



- Reduce dimensionality of the "memory": Map K, V from $n \times d$ to $k \times d$
- Attend to the lower-dimensional memory: $\mathrm{softmax}\left(Q_{n\times d} K^T_{k\times d} / \sqrt{d}\right)$
  - What's the dimension of the attention matrix?
  - What's the dimension of the self-attention output?
- Computation cost: $O(nk)$ (linear in n)
- Downside of uisng Linformer as a decoder?

Summarize the KV memory

Linformer [Wang et al., 2020]: compute self-attention in a lower dimension



- Reduce dimensionality of the "memory": Map K, V from $n \times d$ to $k \times d$
- Attend to the lower-dimensional memory: $\mathrm{softmax}\left(Q_{n\times d}K^T_{k\times d}/\sqrt{d}\right)$
  - What's the dimension of the attention matrix?
  - What's the dimension of the self-attention output?
- Computation cost: $O(nk)$ (linear in n)
- Downside of uisng Linformer as a decoder?
  - Unclear how to mask: past and future are mixed

Summary on efficient self-attention

Improve the quadratic time and space complexity of self-attention

- Sparsify the attention matrix
- Compress the KV memory

Summary on efficient self-attention

Improve the quadratic time and space complexity of self-attention

- Sparsify the attention matrix
- Compress the KV memory

Bad news: Most techniques are not widely used in large pretrained models now. Why?

- Improvement in time/space complexity doesn't always translate to real time/space savings
- These techniques often breaks structure and sacrifice the batching ability on GPUs
- Only see improvement on very long sequences

Approach 4: system-level approaches

- Operates at a lower abstraction level
- Often brings more direct impact on efficiency
- Example:
  - Gradient accumulation
  - Model and data parallelism (e.g., deepspeed)
  - Flash attention: exploit GPU memory asymmetry