Pretraining and Finetuning

He He

NEW YORK UNIVERSITY

February 19, 2025

Table of Contents

Last week

- **Encoders**: tokens to vectors
- **Decoders**: vectors to tokens
- **Key difference**: (autoregressive) decoders cannot look at the future
  - Need causal masking
  - Sequential output

Last week

- **Encoders**: tokens to vectors
- **Decoders**: vectors to tokens
- **Key difference**: (autoregressive) decoders cannot look at the future
  - Need causal masking
  - Sequential output
- Can you use an encoder to generate tokens?

Last week

- **Encoders**: tokens to vectors
- **Decoders**: vectors to tokens
- **Key difference**: (autoregressive) decoders cannot look at the future
    - Need causal masking
    - Sequential output
- Can you use an encoder to generate tokens?
- Can you use a decoder to encode text?

Table of Contents

Representation learning

Recap: What are good representations?

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

Representation learning

Recap: 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. unigram BoW) require complex models.

Representation learning

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

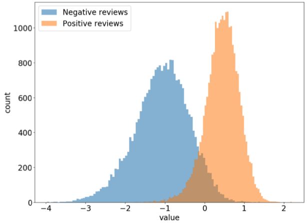


Figure: Sentiment neuron [Radford et al., 2017]

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

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

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 prediction given context?

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

What can we learn from word prediction given context?

- 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 prediction given context?

- 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 prediction given context?

- 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 prediction given context?

- 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 prediction given context?

- 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 prediction given context?

- 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*

Text contains a surprisingly large number of tasks!

Self-supervised learning for representation learning

Key idea: predict parts of the input from the rest

- No additional supervision is needed—both input and output are from the raw text data.
- Easy to scale—massive amount of text on the Internet.
- Learned representation is general—useful for any tasks that can be performed in textual mode.

Self-supervised learning for representation learning

Key idea: predict parts of the input from the rest

- No additional supervision is needed—both input and output are from the raw text data.
- Easy to scale—massive amount of text on the Internet.
- Learned representation is general—useful for any tasks that can be performed in textual mode.

How is this different from skip-gram / CBOW?

Self-supervised learning for representation learning

Key idea: predict parts of the input from the rest

- No additional supervision is needed—both input and output are from the raw text data.
- Easy to scale—massive amount of text on the Internet.
- Learned representation is general—useful for any tasks that can be performed in textual mode.

How is this different from skip-gram / CBOW?

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 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 pretrained transformer to larger data and compute
  - Can directly answer user questions and solve many tasks, e.g., ChatGPT, Claude, Deepseek-chat

Table of Contents

Challenges in tokenization

We want to train models on all text on the internet. What are the challenges in tokenization?

Challenges in tokenization

We want to train models on all text on the internet. What are the challenges in tokenization?

- A long tail of rare words
  Neologism, terminologies, misspelling, informal text, etc.
- A mixture of multiple natural languages, programming languages, special symbols
  Low-resource language, math equations, code-switching, emoji, etc.
- Efficiency
  Trade-off between vocab size and sequence length, latency

Subword tokenization

The most widely adopted solution: decomposing words into subword units

- A long tail of rare words
  `bioorthogonal` $\rightarrow$ `bio ##ortho ##gonal`

Subword tokenization

The most widely adopted solution: decomposing words into subword units

- A long tail of rare words
  `bioorthogonal` → `bio ##ortho ##gonal`
- A mixture of multiple natural languages, programming languages, special symbols
  `Donaudampfschifffahrtsgesellschaft` → `Donaudampf ##schiff ##fahrts ##gesellschaft` (German compound noun meaning ”Danube steamship company”)

Subword tokenization

The most widely adopted solution: decomposing words into subword units

- A long tail of rare words
  `bioorthogonal` → `bio ##ortho ##gonal`
- A mixture of multiple natural languages, programming languages, special symbols
  `Donaudampfschifffahrtsgesellschaft` → `Donaudampf ##schiff ##fahrts ##gesellschaft` (German compound noun meaning "Danube steamship company")
- Efficiency
  Balancing granularity and efficiency: reducing token count without losing meaning

Byte pair encoding (BPE)

What is a "token"?

- A sequence of characters that carries some meaning and re-occurs in a corpora
- Can we find these character units based on their frequency?

Byte pair encoding (BPE)

What is a "token"?

- A sequence of characters that carries some meaning and re-occurs in a corpora
- Can we find these character units based on their frequency?

BPE:

- Origin: a compression algorithm that iteratively replace the most common character sequences by a single symbol, e.g., $\texttt{un} \rightarrow \texttt{A}$
- Start with individual characters as tokens
- Merge the most frequent pair of tokens and treat them as a single token
- Update the input with the new token and repeat the process
- Output: tokenized text and a set of merge rules

BPE Example (Step-by-Step)

Initial Sequence:

- Words: `banana`, `band`, `ban`
- Initial tokenization (by character):
  - `b a n a n a`
  - `b a n d`
  - `b a n`

BPE Example (Step-by-Step)

Initial Sequence:

- Words: `banana`, `band`, `ban`
- Initial tokenization (by character):
  - `b a n a n a`
  - `b a n d`
  - `b a n`

Step 1: Count Pairs

- What is the most frequent pair:

BPE Example (Step-by-Step)

Initial Sequence:

- Words: `banana`, `band`, `ban`
- Initial tokenization (by character):
  - `b a n a n a`
  - `b a n d`
  - `b a n`

Step 1: Count Pairs

- What is the most frequent pair: `a n`

BPE Example (Step-by-Step)

Initial Sequence:

- Words: `banana`, `band`, `ban`
- Initial tokenization (by character):
  - `b a n a n a`
  - `b a n d`
  - `b a n`

Step 1: Count Pairs

- What is the most frequent pair: `a n`

Step 2: Merge

- New merge rule: `a`, `n` $\rightarrow$ `an`
- Updated tokenization:
  - `b a n a n a` $\rightarrow$
  - `b a n d` $\rightarrow$
  - `b a n` $\rightarrow$

BPE Example (Step-by-Step)

Step 3: Count Pairs Again

- Updated tokenization:
  - `b an an a`
  - `b an d`
  - `b an`
- Most frequent pair: `b an`

Step 4: Merge

- New merge rule: `b, an` $\rightarrow$ `ban`
- Updated tokenization:
  - `ban an a`
  - `ban d`
  - `ban`

BPE: practicalities

- Repeat the process until the desired number of merges or vocabulary size is reached (a hyperparameter to decide). Typically vocabulary sizes are 32-64K.
- Break ties deterministically, e.g., lexicographical order, occurrence in the corpus etc.
- Use bytes as the initial tokens (adopted by GPT-2)
- Variants: instead of merging the pair with the largest frequency, **WordPiece** merges the pair that maximizes the log likelihood of the training data, i.e. Merge `a, b` if

$$\log p(a, b) - \log p(a)p(b)$$

  is the largest among all pairs

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

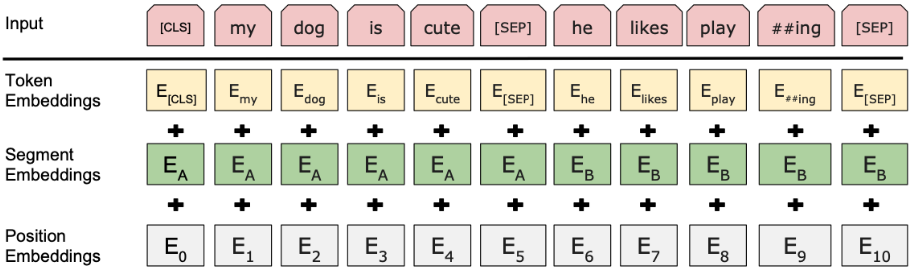
- **Masked language modeling**:
  - Randomly sample 15% tokens as prediction targets
  - Replace the target tokens by `[MASK]` or a random token, or leave it unchanged

    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

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

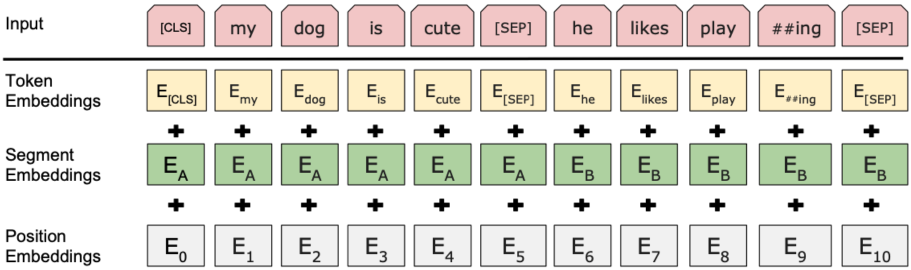
  - x_n: either the sentence following x or a randomly sampled sentence
  - y: binary label of whether x_n follows x
  - Later work has shown that this objective is not necessary

BERT: architecture



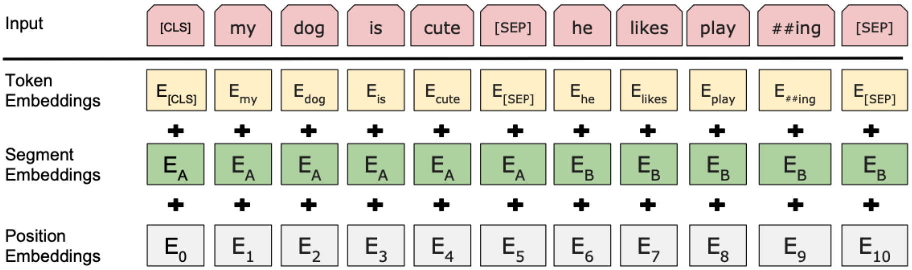
- Tokenization: wordpiece (similar to byte pair encoding) (see details)

BERT: architecture



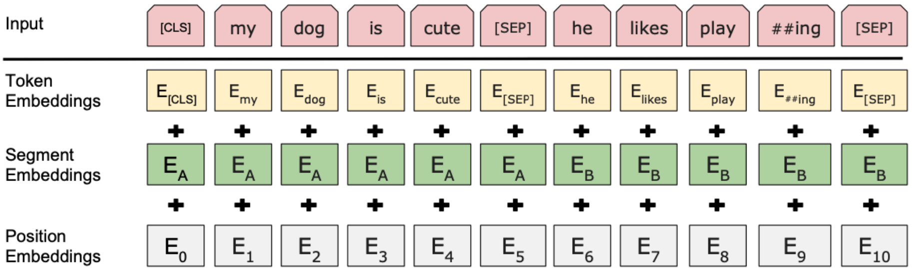
- Tokenization: wordpiece (similar to byte pair encoding) (see details)
- `[CLS]`: first token of all sequences; used for next sentence prediction

BERT: architecture



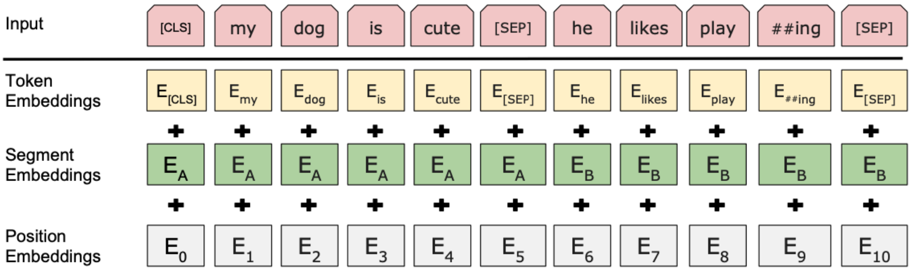
- Tokenization: wordpiece (similar to byte pair encoding) (see details)
- `[CLS]`: first token of all sequences; used for next sentence prediction
- Distinguish two sentences in a pair: `[SEP]` and segment embedding

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

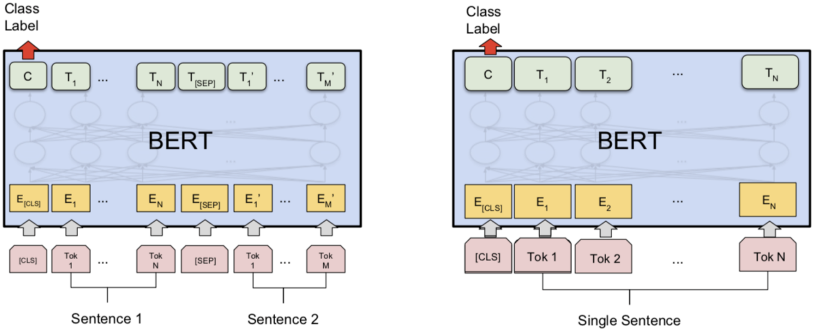


- Tokenization: wordpiece (similar to byte pair encoding) (see details)
- `[CLS]`: first token of all sequences; used for next sentence prediction
- Distinguish two sentences in a pair: `[SEP]` and segment embedding
- Learned position embedding
- 12 (base; 110M params) or 24 (large; 340M params) layer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

Finetuning BERT

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

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



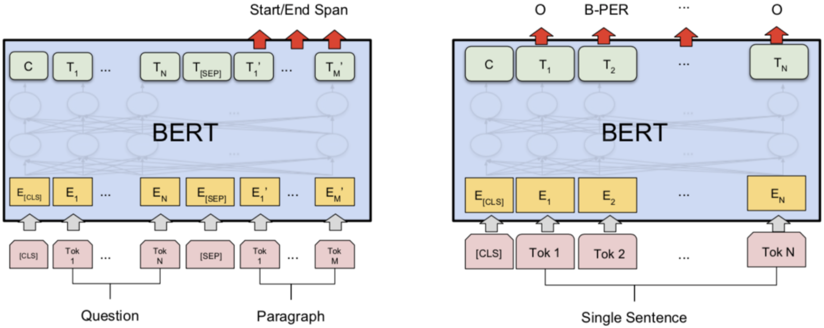
(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG

(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.

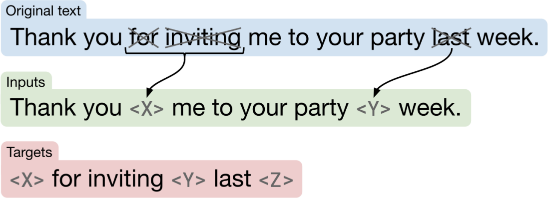
$$
\begin{aligned}
h_1, \ldots, h_n &= \mathrm{Encoder}(x_1, \ldots, x_n) \\
s_1, \ldots, s_m &= \mathrm{Decoder}(y_0, \ldots, y_{m-1}, h_1, \ldots, h_n) \\
p(y_i \mid x, y_{<i}) &= \mathrm{softmax}(W s_i + b)
\end{aligned}
$$

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

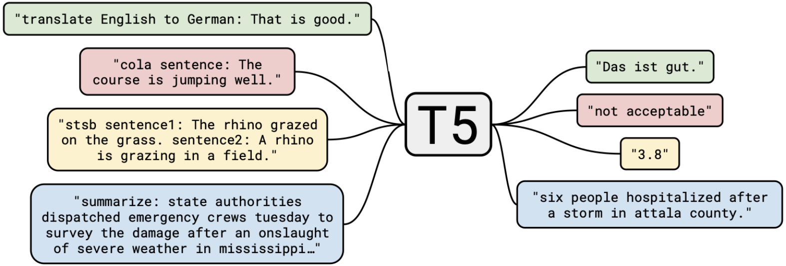


Compare with encoder-only models:

- 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

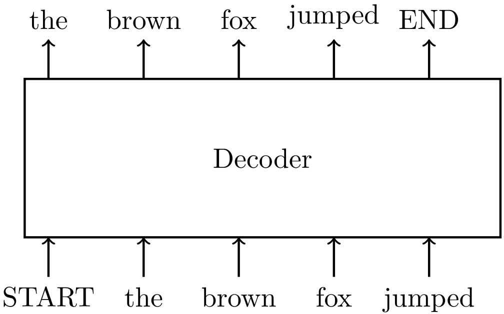
- Formulate the task in text-to-text format
- Fine-tune all parameters (similar to BERT fine-tuning)
- Advantages over encoder models: unified modeling of many different tasks including text generation

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)

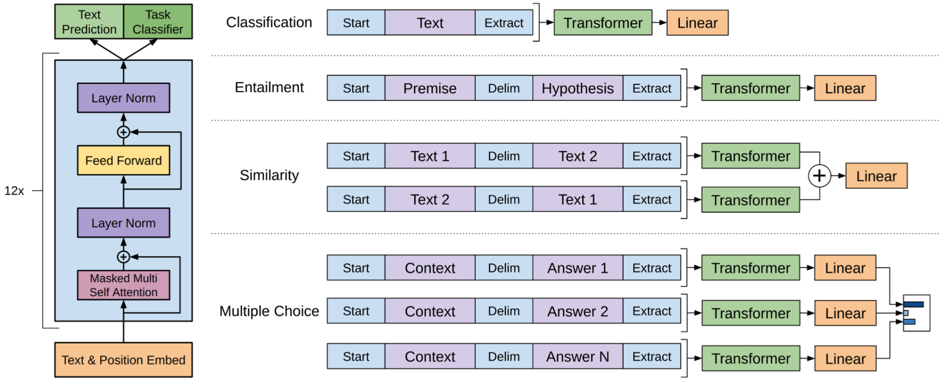
Generative Pretraining (GPT)

- **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

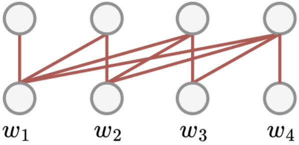
System	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	**Average**
	392k	363k	108k	67k	8.5k	5.7k	3.5k	2.5k	-
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$\text{BERT}_{\text{BASE}}$	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.

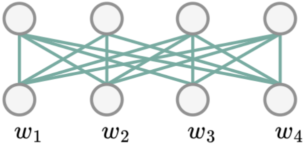
Medium-sized encoder models tend to work better than decoder-only models when finetuned

Encoder-only vs decoder-only models: attention

Decoder-only

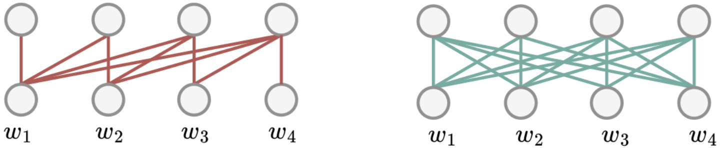


Encoder-only



Encoder-only vs decoder-only models: attention

Decoder-only Encoder-only



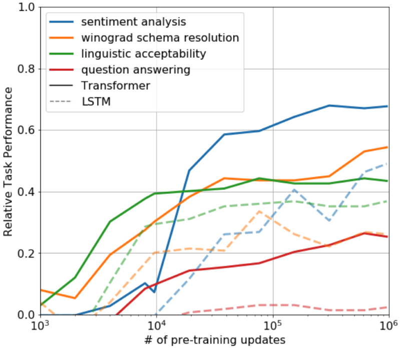
Encoder-only models provides better embeddings due to bidirectional attention.

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?

- Flexibility on encoder design, e.g., full attention on input context
- Limited advantage on long-form generation tasks over decoder-only model
- More resource available for decoder-only models

Table of Contents

Optimization for pretraining

What are challenges of optimization in pretraining?

- Numerical stability (no NaN and diverging losses)
  initialization, normalization, learning rate schedule
- Memory efficiency (work with billions of parameters)
  mixed precision, gradient accumulation

Adam optimizer

Key ideas:

- Momentum
  - Motivation: having an inertia of moving in the same direction $\rightarrow$ reduce oscillation
  - How: maintain a ”memory” (moving average) of past updates

Adam optimizer

Key ideas:

- Momentum
  - Motivation: having an inertia of moving in the same direction $\rightarrow$ reduce oscillation
  - How: maintain a ”memory” (moving average) of past updates
- Adaptive step size for each parameter
  - Intuition:
    flat regions (gradient changing slowly) $\rightarrow$ take larger steps
    steep regions (gradient changing fast) $\rightarrow$ take smaller steps
  - Handle sparse features well (e.g., rare words embeddings aren’t updated often)

Adam optimizer

- Compute gradient g_t

Adam optimizer

- Compute gradient g_t
- Update first order moments (mean)

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{1}$$

Adam optimizer

- Compute gradient g_t
- Update first order moments (mean)

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{1}$$

- Update second order moments (variance)

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{2}$$

Adam optimizer

- Compute gradient g_t
- Update first order moments (mean)

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{1}$$

- Update second order moments (variance)

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{2}$$

- Correct bias in moment estimates

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{3}$$

Adam optimizer

- Compute gradient g_t
- Update first order moments (mean)

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{1}$$

- Update second order moments (variance)

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{2}$$

- Correct bias in moment estimates

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{3}$$

- Update parameters

$$\theta_t = \theta_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{4}$$

Adam optimizer

- Compute gradient g_t
- Update first order moments (mean)

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{1}$$

- Update second order moments (variance)

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{2}$$

- Correct bias in moment estimates

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{3}$$

- Update parameters

$$\theta_t = \theta_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{4}$$

Adam optimizer

- Compute gradient g_t
- Update first order moments (mean)

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{1}$$

- Update second order moments (variance)

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{2}$$

- Correct bias in moment estimates

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{3}$$

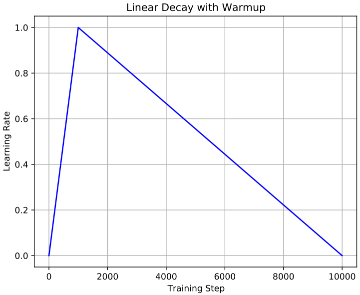
- Update parameters

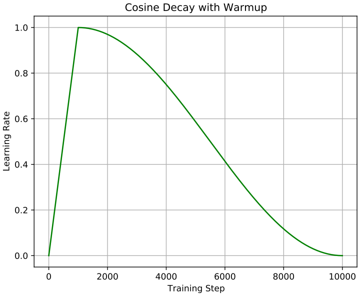
$$\theta_t = \theta_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{4}$$

What is the memory cost of SGD vs Adam?

Learning rate schedule

- **Warmup**: don't want to start with large learning rate for stability
- **Decay**: reducing learning rate as model converges
  linear (used by BERT), cosine, exponential





Gradient accumulation

Simulate large batch training with limited GPU memory

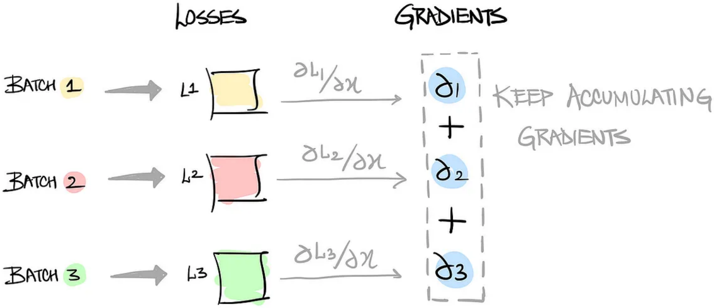


Figure: From Harshit Sharma