Natural Language Processing (NLP) and Large Language Models (LLMs)
Lecture 8-1: Pretraining

Chendi Wang (王晨笛)
chendi.wang@xmu.edu.cn

WISE @ XMU

2025 年 4 月 22 日

Adapted in part from CS 224n, Dive into deep learning, and On the Opportunities and Risks of Foundation Models

1. Section 1: An overview of pretraining and finetuning
2. Section 2: Details of pretraining
3. Section 3: Details of GPT and BERT

The age of pretraining

- BERT: Devlin et al., 2018
- GPT-2: Radford et al., 2019
- GPT-3: Brown et al., 2020
- Vision transformer
- GPT-4, deepseek, Llama, qwen, ...

Pretraining

- Pretraining generally refers to training a very large model on a very large dataset.
- Empirical scaling laws demonstrate the relationship between model size, dataset size, and performance: Kaplan et al., 2020

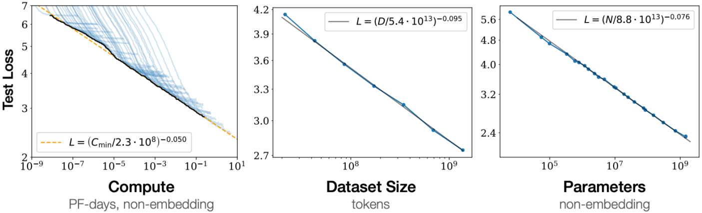


Figure is from Kaplan et al., 2020

Pretrained Models Achieve Strong Performance

Benchmarks

Add a Result

These leaderboards are used to track progress in Language Modelling

Trend	Dataset	Best Model	Paper	Code	Compare
	WikiText-103	RETRO (7.5B)			See all
	Penn Treebank (Word Level)	GPT-3 (Zero-Shot)			See all
	enwik8	GPT-2 (48 layers, h=1600)			See all
	The Pile	Test-Time Fine-Tuning with SIFT + Llama-3.2 (3B)			See all
	WikiText-2	SparseGPT (175B, 50% Sparsity)			See all
	LAMBADA	PaLM-540B (Few-Shot)			See all
	One Billion Word	MDLM (AR baseline)			See all
	Text8	GPT-2			See all

Foundation Models

- Models such as GPT-3, GPT-2, and BERT are referred to as foundation models due to their critically central yet inherently incomplete nature (Bommasani et al., 2021).
- A foundation model is any model pretrained on broad data—typically using large-scale self-supervision—that can be adapted (e.g., via fine-tuning) to a wide range of downstream tasks.

Pretrained foundation models

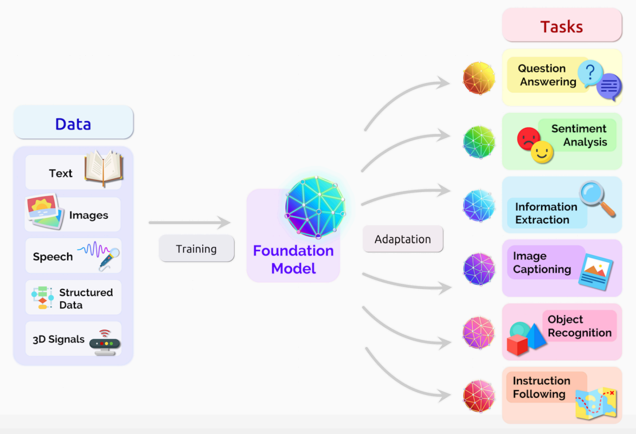
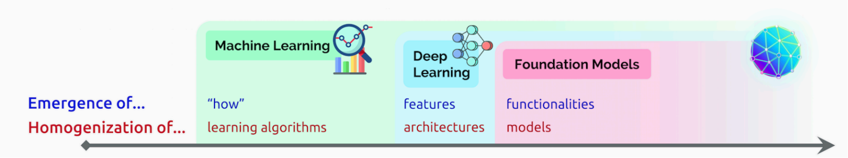
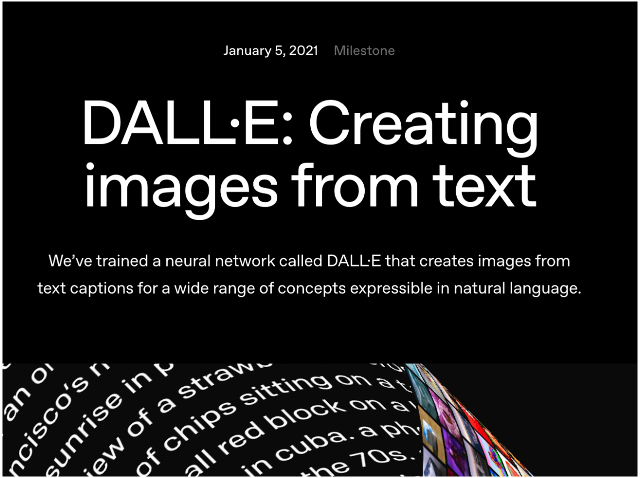


Figure is from Bommasani et al., 2021

Foundation models are not new to us.



Foundation models can be multimodal



What can pretrained LLMs do?

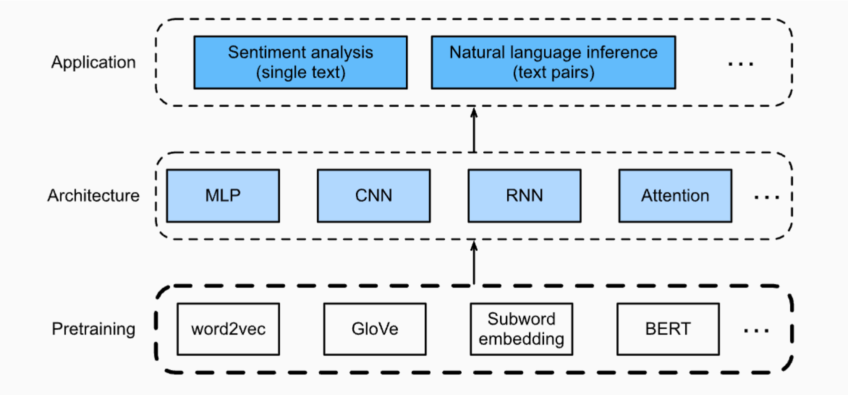


Figure is from d2l

What we have done so far? Pretrained word embeddings

- The word embeddings can be pretrained by Skip-gram models or Glove models.

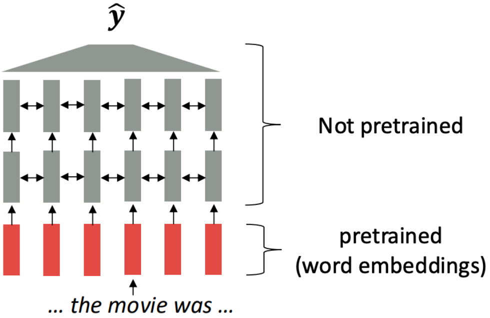
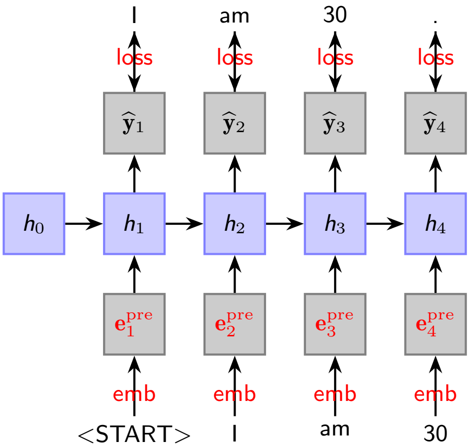


Figure is from Stanford CS 224n

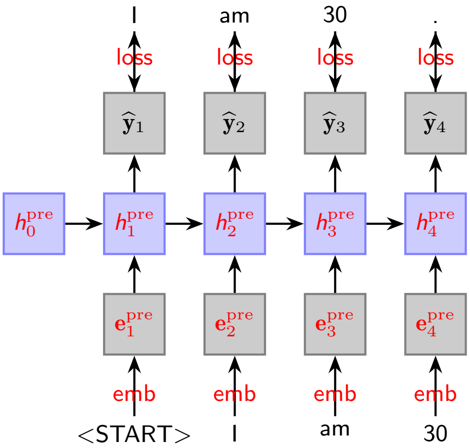
What Have We Done So Far? Pretrained Word Embeddings (Assignment 3)

- Begin with pretrained word embeddings.
- Each word is assigned the same embedding, regardless of the sentence in which it appears.
- Use RNNs or Transformers to learn from context, updating embeddings based on the downstream task.
- The training data for downstream tasks (e.g., question answering, named entity recognition) must be sufficient to capture all relevant contextual information.

Pretrained embeddings with glove



Pretraining whole models are more popular in modern machine learning



Pretraining and Fine-Tuning

- Let D_{pretrain} denote the dataset used for pretraining.
- Define the corresponding loss function $\mathcal{L}_{\text{pretrain}}(\theta)$ based on D_{pretrain} and a deep neural network f_θ.
- Learn the pretrained parameters $\widehat{\theta}_{\text{pretrain}}$ by minimizing the loss:

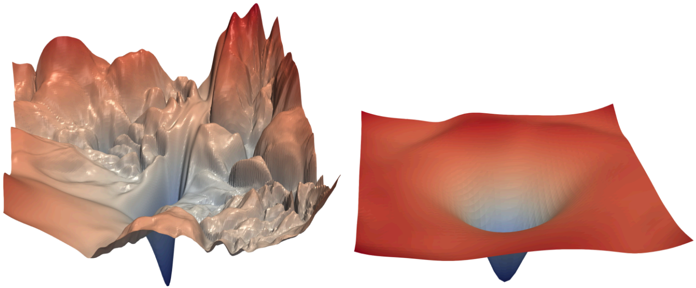
$$\min_\theta \mathcal{L}_{\text{pretrain}}(\theta).$$

- Obtain a foundation model $f_{\widehat{\theta}}$.

Pretraining and Fine-Tuning

- Let $D_{\text{finetuning}}$ denote the downstream dataset, which can be relatively small.
- Define the corresponding loss function $\mathcal{L}_{\text{finetuning}}(\theta)$ for a deep neural network f_θ.
- Minimize $\mathcal{L}_{\text{finetuning}}(\theta)$ using SGD or its variants, initialized with the pretrained parameters $\widehat{\theta}_{\text{pretrain}}$.

Loss landscape of deep neural networks



(a) without skip connections (b) with skip connections

Figure 1: The loss surfaces of ResNet-56 with/without skip connections. The proposed filter normalization scheme is used to enable comparisons of sharpness/flatness between the two figures.

Figure is from https://arxiv.org/abs/1712.09913

Why Pretraining Matters: Possible Explanations

- Certain local minima near $\widehat{\theta}_{\mathrm{pretrain}}$ may exhibit better generalization performance.
- The gradients of the fine-tuning loss near $\widehat{\theta}_{\mathrm{pretrain}}$ may propagate more effectively, facilitating optimization.

Sometimes We May Not Need to Fine-Tune All Parameters

- In some cases, we may freeze a subset of the pretrained parameters and only fine-tune the remaining ones.
- Mathematically, we partition the model parameters as $\theta = (\theta^{(1)}, \theta^{(2)})$.
- The pretrained parameters can be expressed as $\widehat{\theta}_{\text{pretrain}} = (\widehat{\theta}^{(1)}_{\text{pretrain}}, \widehat{\theta}^{(2)}_{\text{pretrain}})$.
- To fine-tune only $\theta^{(1)}$ while freezing $\widehat{\theta}^{(2)}_{\text{pretrain}}$, we define the modified loss:

$$\mathcal{L}'_{\text{finetune}}(\theta^{(1)}) = \mathcal{L}_{\text{finetune}}(\theta^{(1)}, \widehat{\theta}^{(2)}_{\text{pretrain}}).$$

- We then minimize this loss w.r.t. $\theta^{(1)}$ using SGD or its variants, initializing at $\widehat{\theta}^{(1)}_{\text{pretrain}}$.

Linear Probing

- In fine-tuning, when only the last layer of a neural network is trainable while all preceding layers are frozen, the procedure is referred to as linear probing (since the model is linear).
- Linear probing is often sufficient when the downstream dataset closely resembles a subset of the data used during pretraining.
- This can be interpreted from the perspective of representation learning. Recommended reading:
  - Kumar et al. (2022). Fine-Tuning Can Distort Pretrained Features and Underperform Out-of-Distribution.
  - Wang et al. (2024). Neural Collapse Meets Differential Privacy: Curious Behaviors of NoisyGD with Near-Perfect Representation Learning.

Linear Probing

- Recall the structure of a (fully connected) deep neural network:

$$f_\theta(\mathbf{x}) = \mathrm{SoftMax}\left(\mathbf{W}^{[L]}\sigma(\mathbf{W}^{[L-1]}\sigma(\cdots\sigma(\mathbf{W}^{[1]}\mathbf{x}+\mathbf{b}^{[1]}))+\mathbf{b}^{[L-1]})+\mathbf{b}^{[L]}\right).$$

- Let $\widehat{\theta}_{\mathrm{pretrain}} = \left(\widehat{\mathbf{W}}^{[L]}, \widehat{\mathbf{b}}^{[L]}, \cdots, \widehat{\mathbf{W}}^{[1]}, \widehat{\mathbf{b}}^{[1]}\right)$ denote the parameters obtained from pretraining.
- Define $\widehat{\mathbf{h}}(\mathbf{x})$ as the output of the penultimate layer under the pretrained network, and consider the model:

$$f_{\mathbf{W}^{[L]},\mathbf{b}^{[L]}}(\mathbf{x}) = \mathrm{SoftMax}\left(\mathbf{W}^{[L]}\widehat{\mathbf{h}}(\mathbf{x})+\mathbf{b}^{[L]}\right).$$

- In linear probing, only $\mathbf{W}^{[L]}$ and $\mathbf{b}^{[L]}$ are updated by minimizing the fine-tuning loss, while $\widehat{\mathbf{h}}(\mathbf{x})$ remains fixed.

Adding New Layers During Fine-tuning

- Recall that $f_{\widehat{\theta}}(\mathbf{x})$ denotes a pretrained neural network.
- To enhance the model's capacity or adapt it to a new task, we can append additional layers:

$$f_{\mathbf{W}^{\text{new}},\mathbf{b}^{\text{new}}}(\mathbf{x}) = \mathbf{W}^{\text{new}} f_{\widehat{\theta}}(\mathbf{x}) + \mathbf{b}^{\text{new}}.$$

- The loss function can then be minimized with respect to:
  - only the new parameters $\mathbf{W}^{\text{new}}, \mathbf{b}^{\text{new}}$, or
  - both the new parameters and the original pretrained parameters θ.

A Simple Fine-tuning Demo (Related to Final Project)

- A demonstration notebook is provided: `FineTuneBERT_mirror.ipynb`.

Key Points to Keep in Mind When Fine-tuning

- Use the Hugging Face mirror. A tutorial is available here.
- Use GPU acceleration—use our server for training.
- This is a warm-up for your final project:
  - Basic implementation similar to the demo (score range: 60–70).
  - Enhancements such as hyperparameter tuning, using larger or additional datasets, incorporating alternative models beyond BERT, or tackling more complex tasks with online datasets (score range: 70–80).
  - All the above, plus a clearly written and well-organized report (score range: 80–90).
  - All the above, with in-depth comparisons of models or datasets, the use of self-collected downstream datasets (including collection methodology), or the integration of advanced techniques (e.g., privacy-preserving methods, complex task settings such as multi-modal) (score range: 90–100).

Key to Pretraining

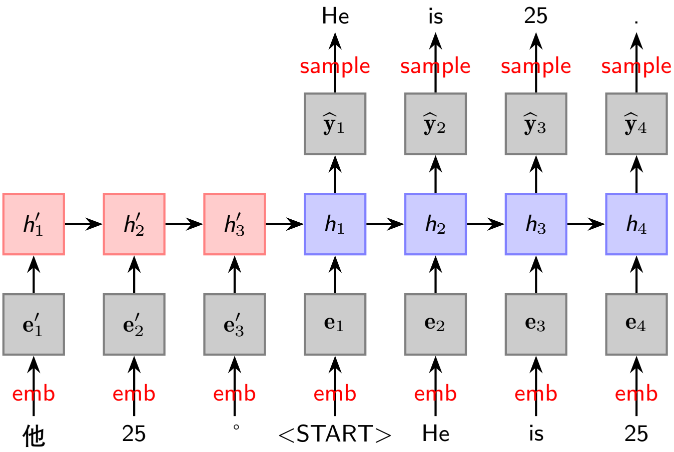
- The model should be sufficiently large to process large-scale, diverse datasets.
- Most of the data used in pretraining are unlabeled.
- The performance of pretrained models often follows a *scaling law*, where increased data, model size, and compute lead to improved results.

Why Unlabeled Data?

- The vast majority of data available is unlabeled.

Dataset	Number of Tokens
SQuAD 2.0 (labeled)	< 50 million
DCLM-pool	240 trillion
Pretraining Data	510T (indexed), 3100T (total)

Recap: The Encoder–Decoder Architecture



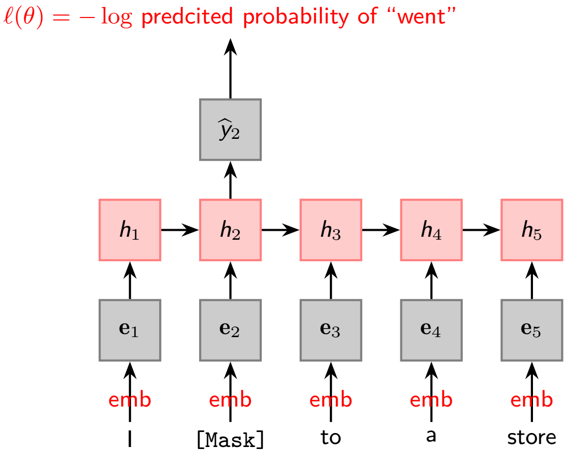
Three Approaches to Pretraining

- **Encoder-only:** Trained with bidirectional context—can "see" both past and future tokens (e.g., BERT).
- **Decoder-only:** Autoregressive language models that predict the next token based only on past context (e.g., GPT).
- **Encoder-decoder:** Combines both components—encoder processes the input, decoder generates the output (e.g., T5).

Pretraining Encoders

- Encoders are trained to capture bidirectional context—but how?
- Mask a subset of tokens using a special `[MASK]` token during training.
- For example: "I `[MASK]` to a store."
- This model is called a masked language model

Pretraining Encoders



Pretraining Encoders: Masked Language Modeling

- Given a sequence $\mathbf{x} = (x_1, \cdots, x_L)$, let $h_t = \mathrm{Encoder}(x_t)$ denote the contextual embedding at position t.
- Randomly replace a subset of tokens $\{x_{t_i}\}_{i=1}^{L'}$ with the special token `[MASK]`.
- Predict the original tokens using:

$$\widehat{y}_{t_i} = \mathrm{SoftMax}\left(\mathbf{W}_y h_{t_i}(\,\texttt{[MASK]}\,) + \mathbf{b}_y\right).$$

- Compute the cross-entropy loss using the ground truth label, e.g., $y_2 =$ "went":

$$\mathrm{CE}(\widehat{y}_{t_i}, y_{t_i}).$$

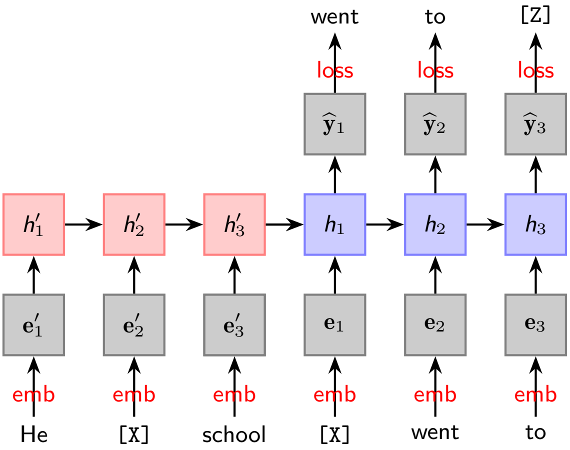
- The final loss is the average over all masked positions:

$$\frac{1}{L'} \sum_{i=1}^{L'} \mathrm{CE}(\widehat{y}_{t_i}, y_{t_i}).$$

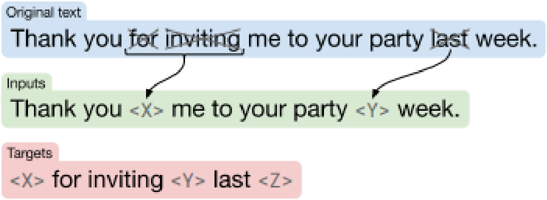
Pretraining Encoder-Decoder Models

- We illustrate the use of the "Text-to-Text Transfer Transformer" (T5), a pretrained encoder-decoder model proposed by Raffel et al., 2018.
- **Encoder training:** The input sequence is partially masked by replacing variable-length spans with unique sentinel tokens.
- **Decoder training:** The target output is formed by concatenating the dropped spans in order, each delimited by the corresponding sentinel token, ending with a final sentinel token `[Z]`.

Pretraining encoder-decoder models



Pretraining encoder-decoder models



An example from Raffel et al., 2018

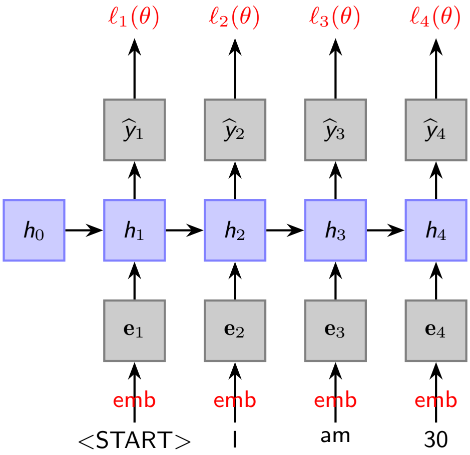
Pretraining encoder-decoder models

- The encoder-decoder model may have better performance in some tasks like denoising.

Architecture	Objective	Params	Cost	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Encoder-decoder	Denoising	$2P$	M	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	**39.82**	**27.65**
Enc-dec, shared	Denoising	P	M	82.81	18.78	**80.63**	**70.73**	26.72	39.03	**27.46**
Enc-dec, 6 layers	Denoising	P	$M/2$	80.88	18.97	77.59	68.42	26.38	38.40	26.95
Language model	Denoising	P	M	74.70	17.93	61.14	55.02	25.09	35.28	25.86
Prefix LM	Denoising	P	M	81.82	18.61	78.94	68.11	26.43	37.98	27.39
Encoder-decoder	LM	$2P$	M	79.56	18.59	76.02	64.29	26.27	39.17	26.86
Enc-dec, shared	LM	P	M	79.60	18.13	76.35	63.50	26.62	39.17	27.05
Enc-dec, 6 layers	LM	P	$M/2$	78.67	18.26	75.32	64.06	26.13	38.42	26.89
Language model	LM	P	M	73.78	17.54	53.81	56.51	25.23	34.31	25.38
Prefix LM	LM	P	M	79.68	17.84	76.87	64.86	26.28	37.51	26.76

An example from Raffel et al., 2018

Pretraining decoders



Pretraining decoders

- Given a sequence $\mathbf{x} = (x_1, \cdots, x_L)$, let $h_t = \mathrm{Encoder}(x_t)$ denote the contextual embedding at position t.
- Predict the $t+1$-th token with probability:

$$\widehat{y}_t = \mathrm{SoftMax}\left(\mathbf{W}_y h_t(x_t) + \mathbf{b}_y\right).$$

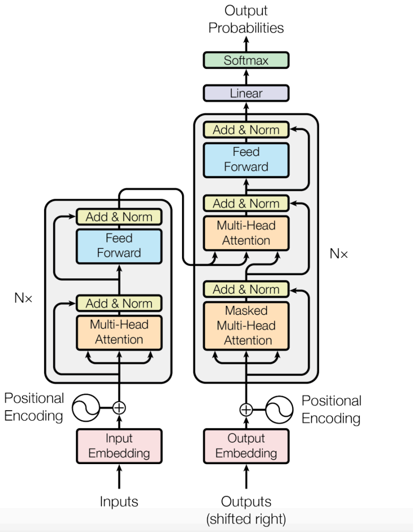
- Compute the cross-entropy loss using the ground truth label

$$\mathrm{CE}(\widehat{y}_t, y_t).$$

- The final loss is the average over all masked positions:

$$\frac{1}{L}\sum_{t=1}^{L}\mathrm{CE}(\widehat{y}_t, y_t).$$

Recap:transformer encoder decoder



Bidirectional Encoder Representations from Transformers (BERT)

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Jacob Devlin Ming-Wei Chang Kenton Lee Kristina Toutanova
Google AI Language
{jacobdevlin,mingweichang,kentonl,kristout}@google.com

Input Sentence Structure

- Each input sentence in BERT's training data is wrapped with two special tokens: `[CLS]` at the beginning and `[SEP]` at the end (or between two segments).
- For example:

```
[CLS] my dog is cute [SEP] he likes play ##ing [SEP]
```

- The `[CLS]` token is used for classification tasks, while `[SEP]` marks the end of a sentence or separates two segments.

Input Embeddings

- Token Embeddings: Pretrained WordPiece embeddings (Wu et al., 2016) with a vocabulary of 30,000 tokens.
- Positional Embeddings: As introduced in the Transformer section, these embeddings encode the position of tokens in a sequence.
- Segment Embeddings: These embeddings differentiate between sentences.
- Input Embeddings: $\mathbf{e}_t =$ Token Embeddings $+$ Positional Embeddings $+$ Segment Embeddings

Input Embeddings

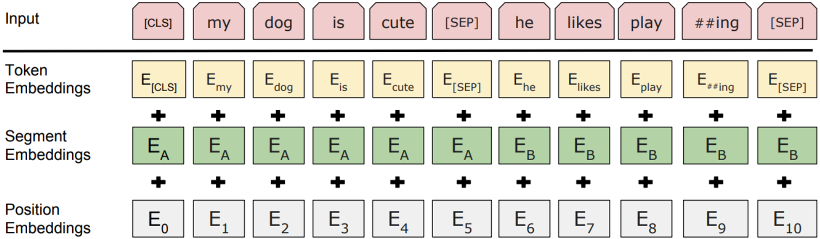


Figure is from Devlin et al., 2018

Masked Language Modeling for BERT

- In this pretraining task, 15% of tokens are randomly selected to be masked for prediction.
- However, the special token [Mask] is never used during fine-tuning.
- To prevent a mismatch between pretraining and fine-tuning, if a token is masked for prediction, it is replaced with other tokens randomly as follows:
  - A special "<mask>" token is used 80% of the time (e.g., "this movie is great" becomes "this movie is <mask>").
  - The original token is used 10% of the time (e.g., "this movie is great" remains "this movie is great").
  - A random token is used 10% of the time (e.g., "this movie is great" becomes "this movie is drink").

Next Sentence Prediction

- While masked language modeling captures bidirectional context to represent words, it does not explicitly model the logical relationship between text pairs.
- To address this, BERT incorporates a binary classification task called next sentence prediction during its pretraining.
- During pretraining, half of the sentence pairs are consecutive sentences labeled as "True."
- For the other half, the second sentence is randomly sampled from the corpus and labeled as "False."
- This task is facilitated by the `[CLS]` token. What is the embedding of the `[CLS]` token?

Next Sentence Prediction

- The [CLS] token has only two possible values (True or False), and its embedding $\mathbf{e}_{\texttt{[CLS]}}$ lies in $\mathbb{R}^2$.

```python
class NextSentencePred(nn.Module):
    """The next sentence prediction task of BERT."""
    def __init__(self, **kwargs):
        super(NextSentencePred, self).__init__(**kwargs)
        self.output = nn.LazyLinear(2)

    def forward(self, X):
        # `X` shape: (batch size, `num_hiddens`)
        return self.output(X)
```

Code is from dive into machine learning

Model Structure and Details of BERT-Base

- We begin by introducing a smaller model—BERT-base.
- Each encoder transformer block (ignoring the "Add & Norm" layer) has the following structure:

$$\text{FF}(\text{Multi-head attention}(\text{Input}))$$

- Output dimension of the multi-head attention layer: 768-dimensional hidden states.
- Output dimension of the feed-forward layer: 4×768.
- Number of attention heads: 12.
- Depth: 12 encoder transformer blocks.

Model Structure and Details of BERT-Large

- BERT-Large is a larger model than BERT-base.
- Output dimension of the multi-head attention layer: 1024-dimensional hidden states.
- Output dimension of the feed-forward layer: 4×1024.
- Number of attention heads: 16 (The dimension of each head is the same as BERT-base).
- Depth: 16 encoder transformer blocks.

Training data for BERT

- BooksCorpus (800 million words)
- English Wikipedia (2,500 million words)

Pretraining is time-consuming

- BERT is pretrained on TPUs, specialized hardware designed to accelerate tensor operations.
- The pretraining of BERT was conducted using 64 TPU chips over a span of 4 days.
- Finetuning is practical and common on a single GPU!!!

Fine-Tuned BERT is Powerful on Many Diverse Datasets

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**

Figure adapted from Devlin et al., 2018

The Larger, the Better

System	Dev EM	Dev F1	Test EM	Test F1
Top Leaderboard Systems (Dec 10th, 2018)				
Human	-	-	82.3	91.2
#1 Ensemble - nlnet	-	-	86.0	91.7
#2 Ensemble - QANet	-	-	84.5	90.5
Published				
BiDAF+ELMo (Single)	-	85.6	-	85.8
R.M. Reader (Ensemble)	81.2	87.9	82.3	88.5
Ours				
$\text{BERT}_{\text{BASE}}$ (Single)	80.8	88.5	-	-
$\text{BERT}_{\text{LARGE}}$ (Single)	84.1	90.9	-	-
$\text{BERT}_{\text{LARGE}}$ (Ensemble)	85.8	91.8	-	-
$\text{BERT}_{\text{LARGE}}$ (Sgl.+TriviaQA)	**84.2**	**91.1**	**85.1**	**91.8**
$\text{BERT}_{\text{LARGE}}$ (Ens.+TriviaQA)	**86.2**	**92.2**	**87.4**	**93.2**

Table 2: SQuAD 1.1 results. The BERT ensemble is 7x systems which use different pre-training checkpoints and fine-tuning seeds.

Figure adapted from Devlin et al., 2018

Limitations of BERT

- So, why don’ t we use BERT for everything?
- BERT, like other pretrained encoders, is not naturally suited for autoregressive sequence generation tasks.
- If your task involves sequence generation, consider using a pretrained decoder.

Generative Pretrained Transformer (GPT)

- GPT-2 (Radford et al., 2018) was a significant success in pretraining a decoder!
- Transformer decoder with 12 layers and 117M parameters.
- 768-dimensional hidden states and 3072-dimensional feed-forward hidden layers.
- Tokens and Embeddings: Pretrained Byte-Pair Encoding (BPE) with 40,000 merges (tokens).
- Trained on BooksCorpus: over 7,000 unique books, containing long spans of contiguous text to learn long-distance dependencies.