6. Pretrain then Finetune

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—related to 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 using supervised data for a task.

Types of pretrained models (all are transformer based)

- 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

Encoder models

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

$h_1, ..., h_n = \text{Encoder}(x_1, ..., x_n)$

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

How do we train Encoder?

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

Masked language modeling

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

x_{-i}: noisy version of x where x_i is corrupted

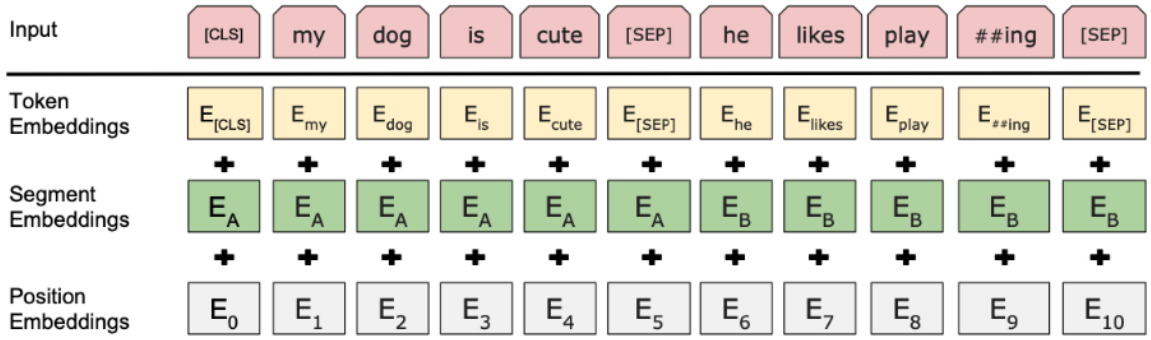
p_{mask}: mask generator

BERT objective

- Masked language modeling: Randomly sample 15% tokens as prediction targets

- Next Sentence Prediction: predict whether a pair of sentences are consecutive (later work has shown that this objective is not necessary)

BERT architecture



Finetuning BERT

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

$p(y|x) = \text{softmax}(Wh_{[CLS]})$

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

$p(y_i|x) = \text{softmax}(Wh_i)$

- 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
- Not straightforward to use on text generation tasks

Encoder-decoder models

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

$h_1, ..., h_n = \text{Encoder}(x_1, ..., x_n)$

$s_1, ..., s_m = \text{Decoder}(y_0, ..., y_{m-1}, h_1, ..., h_n)$

$p(y_i|x, y_{<i}) = \text{softmax}(W_{s_i})$

How do we train the encoder-decoder?

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

Masked language modeling using an encoder-decoder

Input: text with corrupted spans

Output: recovered spans

T5

- 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
  - Use a prefix to denote the task
  - Mixing examples from different datasets when constructing batches
- Jointly training with the two objectives works slightly worse

Finetuing T5

- 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

Efficient pretraining

Idea 1: reducing the number of parameters smartly

- Example: ALBERT [Lan et al., 2020] — share all parameters across layers

Idea 2: design harder learning objectives

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