CS188 Discussion W5

Mingyu Derek Ma

Email: ma@cs.ucla.edu

Reminder

- HW1 solution released on BruinLearn
  - Coding part solution is at PlusLabNLP/cs188-w22-hw1 (github.com)
  - Solution will also automatically show up on Gradescope after the grades are released next week
- Midterm next Monday
  - Multi-selection questions + T/F questions + short answer questions
  - Cover content before syntax

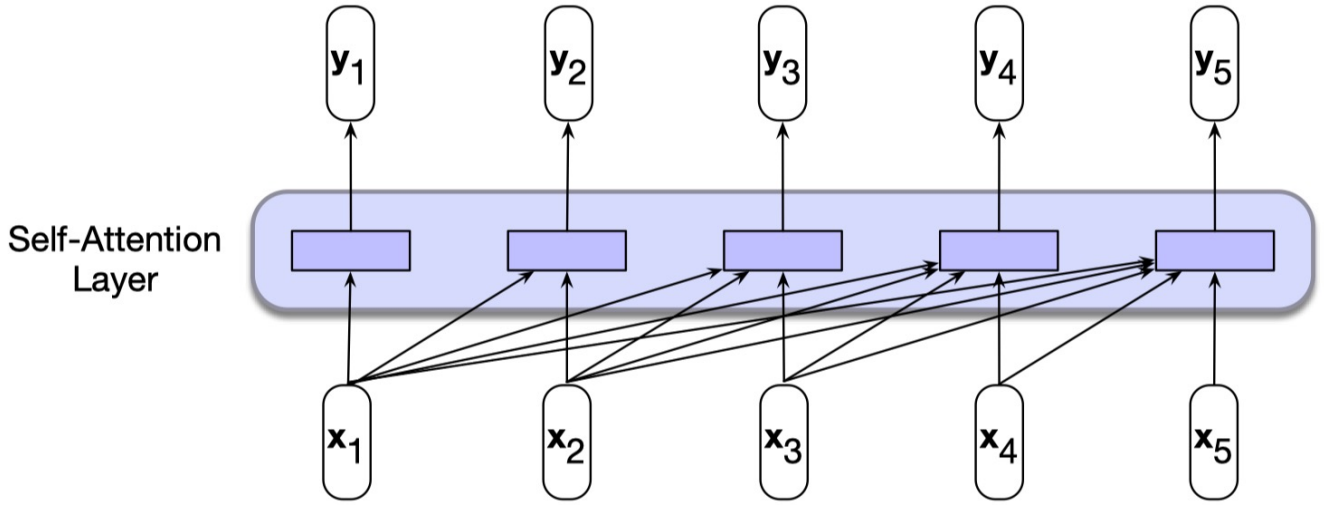
Today: Training Transformers

- Two types of transformers
  - Causal / left-to-right transformer
  - Bidirectional transformer encoder
- Three ways to train transformers
  - Next word prediction
  - Masked Language Modeling
    - Predicting words from surrounding contexts with the goal of producing effective word-level representations
  - Next Sentence Prediction
    - Determining the relationship between pairs of sentences

Approach 1: Next Word Prediction

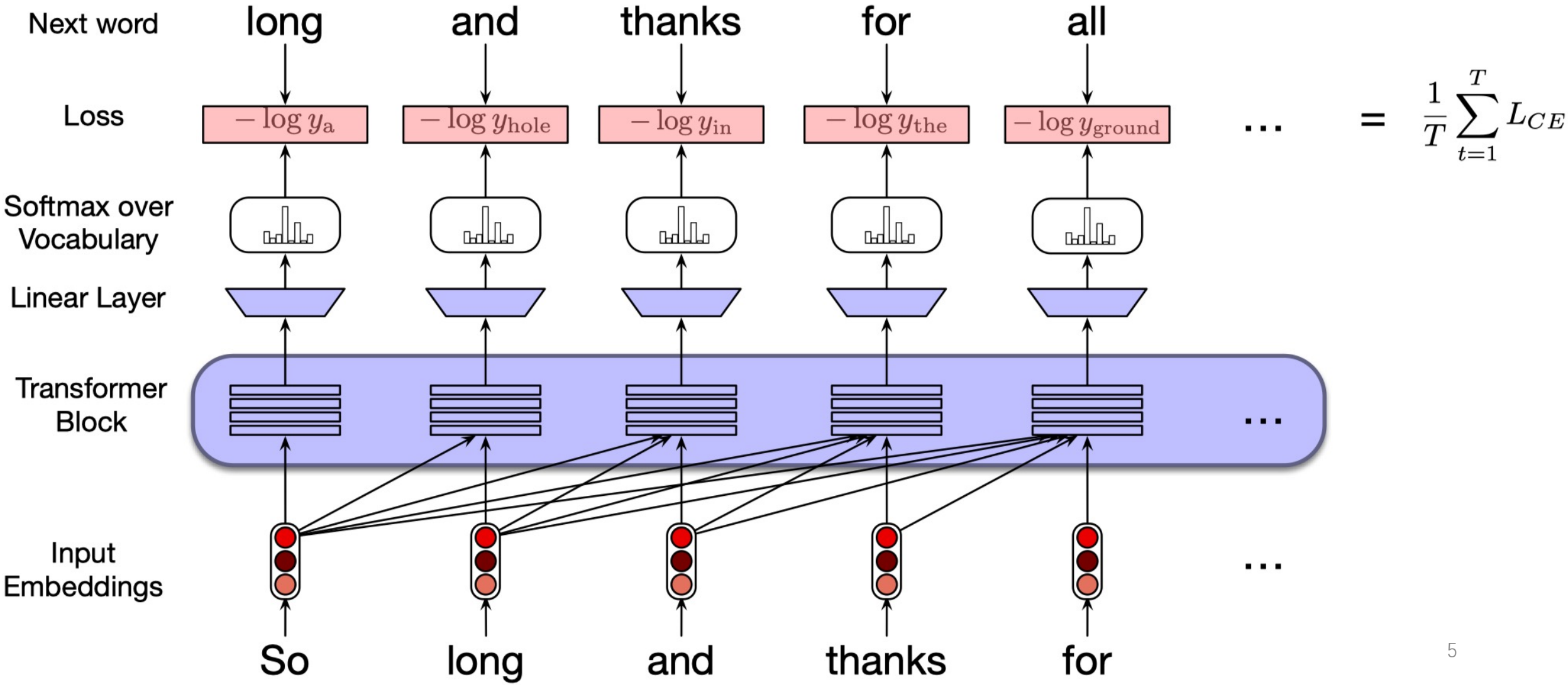
Causal/left-to-right transformers

- Can apply to autoregressive generation problems such as contextual generation, summarization and machine translation
- But it cannot see the future part of the sentence, short for sequence classification and labeling problems



Training causal transformers with NWP

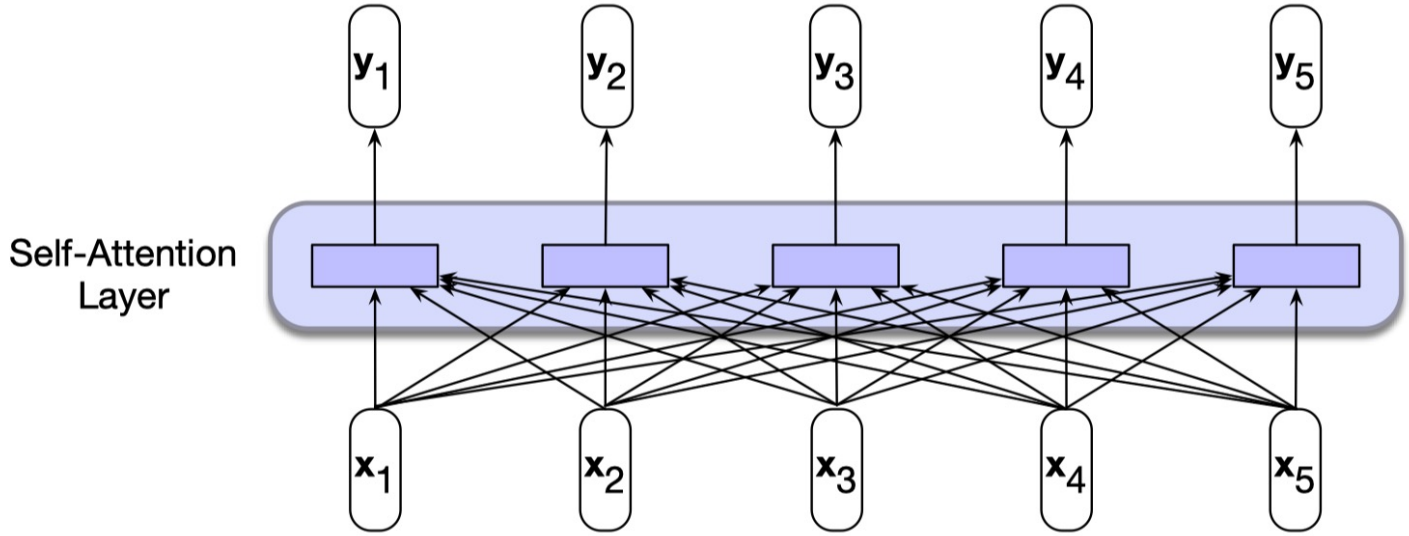
The loss should be -log y_{long} etc



Approach 2: Masked Language Modeling

Bidirectional encoders

- Focus on computing contextualized representations of the tokens in an input sequence that are generally useful across a range of downstream applications.
- The model attends to all inputs, both before and after the current one



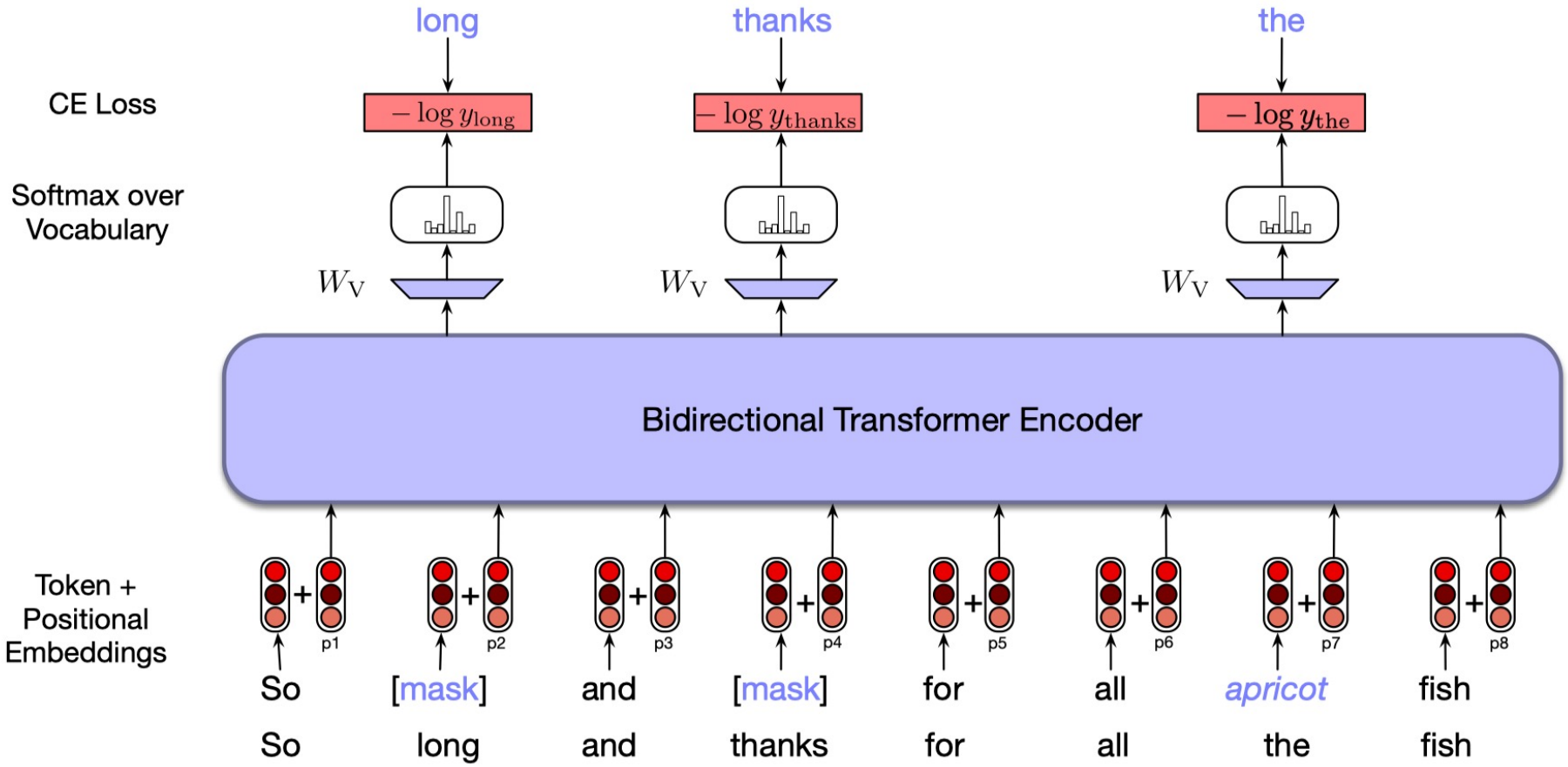
Training bidirectional encoders with MLM

- Cloze task, instead of next word prediction

Please turn your homework _____ . → Please turn _____ homework in.

- Masked Language Modeling (MLM)
  - Use unannotated text from a large corpus to train the model
  - Random sample of tokens from each training sequence is selected (15% for BERT)
  - Then for the chosen tokens, do either
    - Replaced with [MASK] (80% for BERT)
    - Replaced with another token from the vocabulary (10% for BERT)
    - Unchanged (10% for BERT)

Training bidirectional encoders with MLM



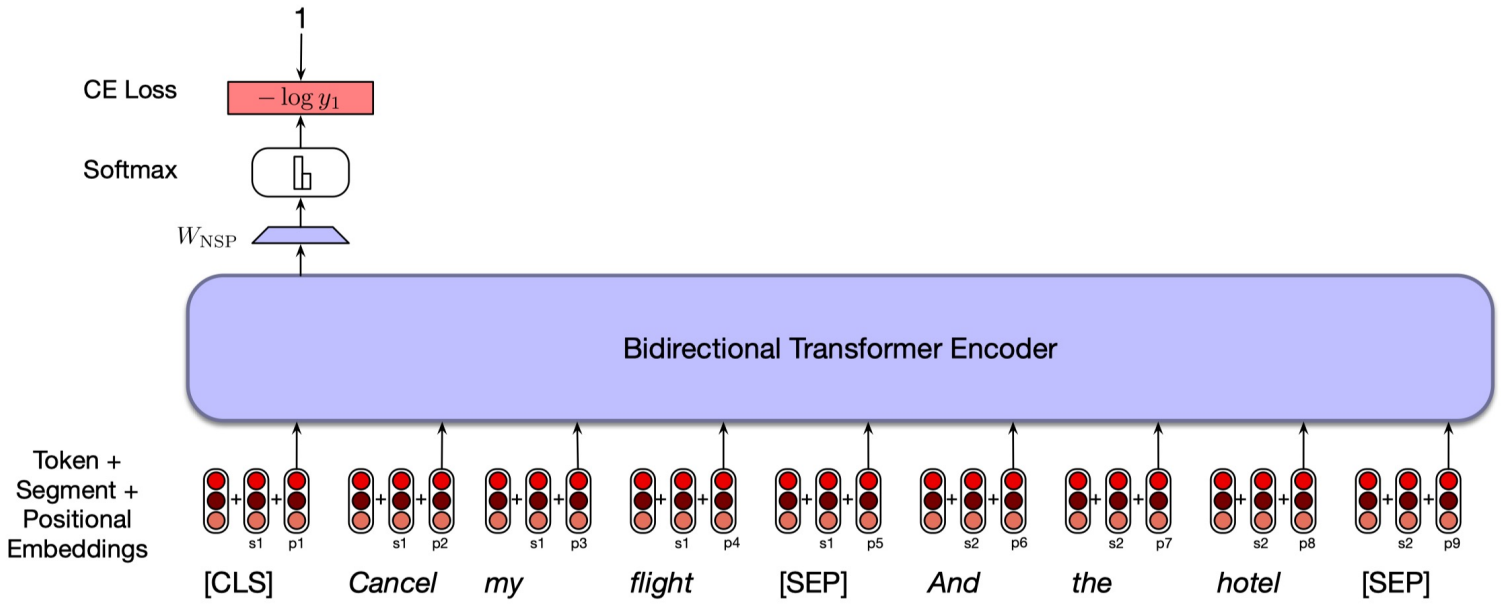
- Get word embeddings
- Combined with positional embeddings
- Pass through a stack of bidirectional transformer blocks
- Output vector from the final transformer layer multiply with weights
- Then obtain probability with a softmax function
- Finally the loss is cross-entropy for each masked item

Approach 3: Next Sentence Prediction

Next Sentence Prediction

- The model is presented with pairs of sentences and is asked to predict whether each pair consists of an actual pair of adjacent sentences from the training corpus or a pair of unrelated sentences.
- In BERT
  - 50% of training pairs: use adjacent sentences from corpus -> positive pairs
  - 50% of training pairs: use random sampled two sentences -> negative pairs
- The model need to distinguish adjacent vs random pairs

Training bidirectional encoders with NSP



- There is a [CLS] token prepended to the input sentence pair
- [SEP] is placed to separate two sentences
- Final layer output vector associated with [CLS] token is used to perform classification
- We still use softmax function to get two-class prediction
- Loss: traditional classification loss, cross entropy loss

Commonly used pretrained language models

- Three models mentioned in the class project description are all bidirectional encoders
- **BERT**
  - Trained with an 800 million word corpus of book texts called BooksCorpus + 2.5 billion word corpus from English Wikipedia
  - Combined loss from the MLM and NSP objectives
  - 100M parameters (BERT-base), using WordPiece (based on subword tokens rather than words)
  - Fixed input size of 512 subword tokens
- **RoBERTa**
  - Trained on much more data: BERT training data + CommonCrawl News + more
  - Only MLM objective
  - Same parameter counts as BERT
- **DeBERTa**
  - Imporve upon RoBERTa

Contextual vs static embeddings

- Contextual embeddings are vectors representing some aspect of the meaning of a token in context
- Static embeddings (such as word2vec, GloVe) represent the meaning of word types (vocabulary entries)
- Contextual embeddings represent the meaning of word tokens (instances of a particular word type in a particular context)

How to obtain contextual embeddings

- Given a pretrained language model, we can input a novel input sentence $x_1, \ldots, x_n$
- Contextual embedding of the i-th token x_i can be the output vector y_i from the final layer of the transformer
- You can also use other way to extract contextual embeddings from the trained model parameters

Pretraining vs fine-tuning

- **Pretraining**: the process of learning some sort of representation of meaning for words or sentences by processing very large amounts of text
  - E.g. BERT is a pretrained language model, trained on large corpus
- **Fine-tuning**: take the presentations from these pretrained models and further training the model, to perform some downstream task
  - Adapt to the downstream task (e.g. commonsense statement classification)
- Pretraining-finetune paradigm is an instance of transfer learning
  - Acquire knowledge from one task/domain, apply it to solve a new task.

MLM in class project

- You will need to implement MLM in the class project
- Then you can run pretraining script to perform the MLM task

3.3 Masked Language Modeling (MLM)

You have learned the basic concepts and algorithm of the masked language modeling widely adopted in modern NLP language models, now it is time for you to actually implement it (as learning by doing is always the best)! Although we do not really request you to perform a large scale pretraining for this part of the basic tasks, you may find the MLM pretraining phase handy when you are working on the open-ended parts of this project.

NOTE: We will introduce a few potentially useful pretraining datasets later on, stay tuned!

Please finish the `TODO` blocks in the function `mask_tokens` in `trainers/train_utils.py` following the guided comments. You can test your implemented functionalities by the following command:

```
python3 -m trainers.train_utils
```

For this part, we only check the specific functionality of this function, and you can execute the script `scripts/run_pretraining.sh` after your full training and evaluation loop in Section 3.4 is done to see how it works in practice.