Language Models

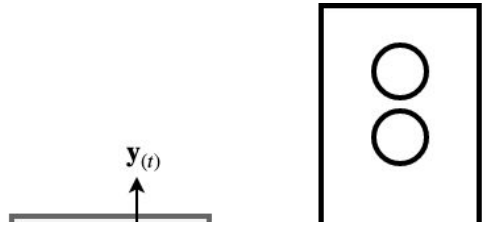
A *Language Model* is an instance of the "predict the next" paradigm where

- given a sequence of words
- we try to predict the next word

Recall the architecture to solve "predict the next word" and data preparation

Language Modeling task

Architecture	Data preparation



$\mathbf{y} = \mathbf{y}_{(1)}, \ldots, \mathbf{y}_{(T)}$

The raw data

- e.g., the sequence of words $\mathbf{s} = \mathbf{s}_{(1)}, \ldots \mathbf{s}_{(\bar{T})}$

is not naturally labeled.

We need a Data Preparation step to create labeled example i

$$\begin{aligned} \mathbf{x}^{(\mathbf{i})} &= \mathbf{s}_{(1)}, \ldots \mathbf{s}_{(i)} \\ \mathbf{y}^{(\mathbf{i})} &= \mathbf{s}_{(i+1)} \end{aligned}$$

We have called this method of turning unlabeled data into labeled examples: *Semi-Supervised* Learning.

In the NLP literature, it is called *Unsupervised Learning*.

There are abundant sources of raw text data

- news, books, blogs, Wikipedia
- not all of the same quality

The large number of examples that can be generated facilitates the training of models with very large number of weights.

This is extremely expensive but, fortunately, the results can be re-used.

- Someone with abundant resources trains a Language Model on a broad domain
- Publishes the architecture and weights
- Others re-use

Predict the next ? Really: predict the *distribution* of next

We have casually defined the Language Modeling objective as predicting the next token.

As you can see: the head layer is a Classifier

- produces a probability for *each token* in the vocabulary as being the next
- We choose one token by sampling from this probability distribution
    - Greedy sampling: always chose the token with highest probability
    - Non-greedy sampling

The Masked Language Modeling objective

There is a variation on the Language Modeling objective called the *Masked Language Modeling* objective.

- Language Modeling objective: given $s[1:t-1]$, predict $s[t]$
- Masked Language Modeling objective
    - Given $s[1:t]$
    - Randomly chose an index $1 \leq j \leq t$
    - "Mask" token j by replacing it with <MASK> so that the input becomes
    $$s_{(0)}, \ldots s_{(j-1)}, <\text{MASK}>, s_{(j+1)}, \ldots s_{(t)}$$
    - Predict the value behing the mask, e.g., $s_{(j)}$
- The Language Modeling objective is the special case where $j = t$

Unsupervised Pre-Traininng + Supervised Fine-Tuning (Transfer Learning)

How do we adapt a Language Model to solve other Target tasks ?

The obvious answer is via Transfer Learning

- The Language Model has learned a lot about the nature of language
    - perhaps the language-knowledge can be transfered to a new task
- Replace the "head" that predicts the next token
- With a new task-specific head
- Train the new model on labeled examples from the Target task
    - the task-specific head **must** be trained
    - the language-model weights **can** (but don't have to) be adapted

This paradigm is called *Unsupervised Pre-Traininng + Supervised Fine-Tuning*.

Example: Fine-Tuning a Pre-trained Language Model to sentiment

This is a straight-forward application of Transfer Learning

- Replace the Classification Head used for Language Modeling
    - e.g., a head that generated a probability distribution over wo vocabulary
- By an un-trained Binary Classification head (Positive/Negative sentim
- Train on examples. Pairs of
    - sentence
    - label: Positive/Negative

Input Transformations

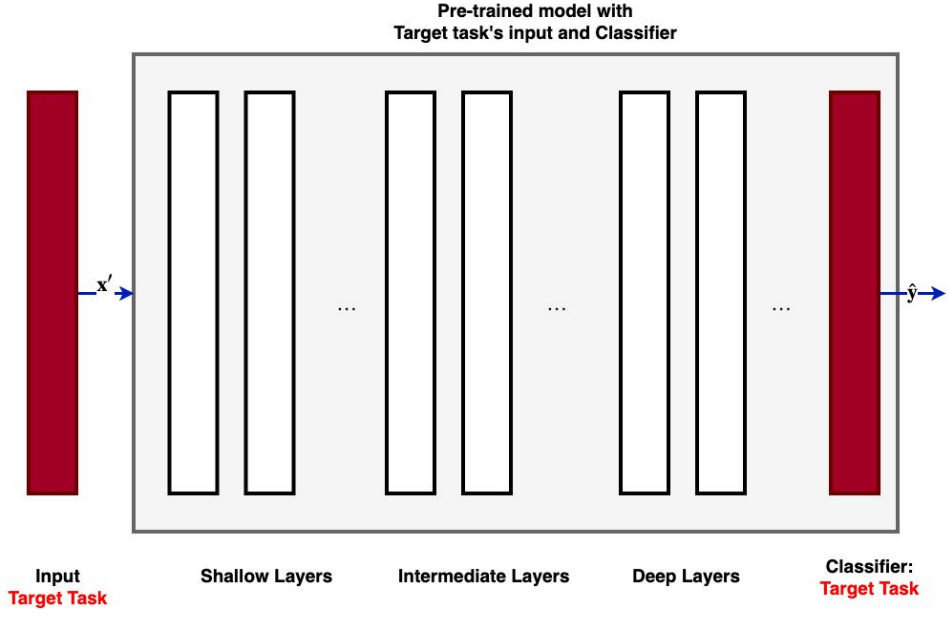
The way to transfer the language-knowledge of a Language Model to a nev
may not always be obvious (as it was for Sentiment Classification).

You may need to

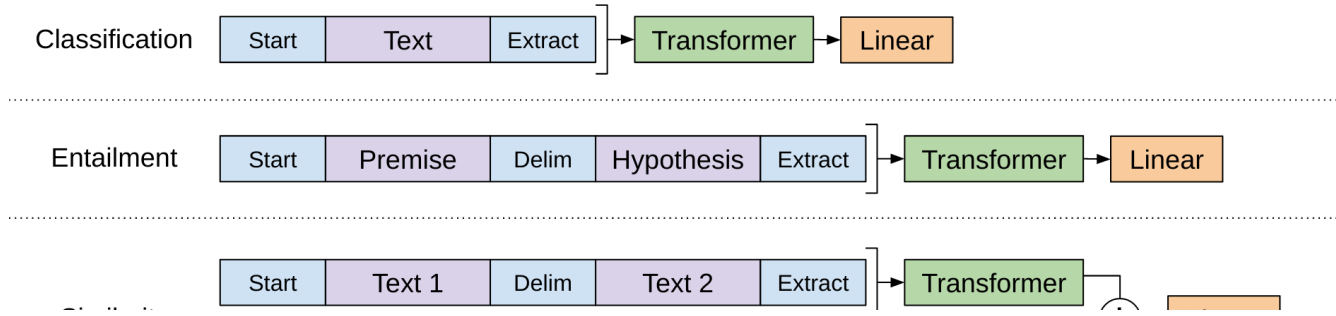
- *transform* your task input from the domain of the Target task
    - into a text string: the domain of the Language Modeling ta
- manipulate the output of the Language Model (text) to transform it
  range of the Target task

See [this paper (https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf#page=4)](https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf#page=4) for some trans

Transfer Learning: replace the head of the pre-trained model



GPT: Task encoding



The top example shows how to use a Language Model to implement Classification

- Apply the head-less Language model (the "Transformer" box) to the sentence to be classified
- Create a new Target task-specific head (the "Linear" box)

We demonstrated this above.

To adapt a Language Model for the task of Multiple Choice Question answering (bottom box)

- train a binary classifier to classify whether each possible answer is correct or not
- run the classifiers in parallel for each possible answer
- choose (as output of the new Target task) the answer with highest binary classification probability

Other uses of a Language Model: Feature based

Rather than using Fine-Tuning to adapt a Language Model to a new task, one can also

- directly re-use the representations created by the Language Model

Recall the behavior of an RNN

- the latent states are "summaries" of prefixes of the sequence

When these representations are produced by an Encoder Transformer trained on a Language Modeling task these representations

- embody great semantic meaning: *Context Sensitive Represenation*
- are much shorter than One Hot Encoding of tokens
- are more meaningful than simple Word Embeddings
    - embody great semantical meaning

See the [ELMo paper (https://arxiv.org/abs/1802.05365)]

An interesting special case

- the representation of the first or final token in the sequence
    - usually a special token <START>, <END>
- is a summary of the *entire sequence* (using a bi-directional RNN or non-masking Encoder Transforer

This representation is a very short encoding of a long sequence

- like hashing

One can use this to implement *Semantic Search*

- "hash" each page on the Web
- "hash" a search query
- The Search returns those web pages whose hash is similar to the hash of the query

Multi-task learning

One area of recent interesting is *multi-task learning*

- Training a model to implement multiple tasks

A model that implements a single task computes

$$p(\text{output} \mid \text{input})$$

A model that implements several tasks computes

Using the Universal API, a Language Model may be adapted to solve *multiple tasks* simultaneously.

This requires you to construct a training set

- with examples from each task
- where each example has an additional "feature" that identifies the task to which it applies

For example

$$\begin{array}{llll} (\textsf{Translate to French}, & \text{English text}, & & \text{French Text}) \\ (\textsf{Answer the question}, & \text{document}, & \text{question}, & \text{answer}) \end{array}$$

The first feature above is the task identifier of the example.

In [2]:
```
print("Done")
```

```
Done
```