Training, Fine Tuning, Inference and Applications of Language Models

Mrinmaya Sachan, Wangchunshu Zhou, Peng Cui, Shehzaad Dhuliawala,
Yifan Hou, Andreas Opedal, Kumar Shridhar,
Alessandro Stolfo, Vilém Zouhar

May 9, 2023

Contents

1 Introduction **5**
1.1 Introduction . . . 5

2 Transfer Learning **7**
2.1 Transfer Learning . . . 7
2.2 ELMo . . . 9
2.3 BERT . . . 11
2.3.1 BERT: Pre-training Objectives . . . 12
2.3.2 BERT Architecture . . . 13
2.3.3 Fine-tuning BERT . . . 14
2.4 Other Transformer Language Models . . . 16
2.4.1 BERT Variants . . . 16
2.4.2 GPTs . . . 19
2.4.3 Seq2seq TLMs . . . 22

3 Parameter efficient finetuning **25**
3.1 Partially Finetuning . . . 25
3.2 Adapter Tuning . . . 27
3.2.1 [Optional Reading] Example: Cross-Lingual Transfer . . . 27
3.2.2 [Optional Reading] LoRA . . . 30
3.2.3 Prefix Tuning . . . 31

4 Prompting and Zero-shot inference **33**
4.1 What is prompting . . . 33
4.1.1 How to prompt? . . . 33
4.2 Prompt Engineering . . . 34
4.2.1 Manual Prompts . . . 35
4.2.2 Automated Prompts . . . 35
4.3 Zero- and Few-Shot Inference . . . 37

4.3.1 In-Context Learning 38
4.3.2 Chain of Thought Prompting 39

5 Multimodality **41**
5.1 Vision Language Models 41
5.1.1 Vision-and-Language Tasks 42
5.1.2 Vision-Language Models: Architectures 44
5.1.3 Vision-Language Models: Pre-training Objectives . . . 47
5.2 Knowledge-Enhancement 51
5.2.1 kNN LM 52
5.2.2 Dynamic Gating kNN LM 53
5.2.3 KnowBERT 54
5.2.4 [Optional] ERNIE 57

6 Additional Topics **61**
6.1 Instruction-Based Training Procedures 61
6.1.1 Instruction Tuning 61
6.1.2 Reinforcement Learning from Human Feedback 63
6.2 Scaling laws and Emergent Behavior 65
6.2.1 Summary of Scaling Laws 66

Chapter 1

Introduction

1.1 Introduction

Welcome to the class notes on "Training, Fine Tuning, Inference and Applications of Language Models" for the Large Language Models class (263-5354-00L). This part of the class focuses on the practical aspects of implementing large language models, their functionalities and applications. Many universities are offering similar courses at the moment, e.g., CS324 at Stanford University (https://stanford-cs324.github.io/winter2022/) and CS 600.471 (https://self-supervised.cs.jhu.edu/sp2023/) at Johns Hopkins University. Their syllabi may serve as useful references.

Chapter 2

Transfer Learning

2.1 Transfer Learning

Transfer learning is the idea of using knowledge gained from training on one task in order to solve other tasks. It takes inspiration from the concept of "transfer of learning" in psychology, which is the phenomenon of a person being able to apply some skill or piece of knowledge they have already acquired to a new situation or context. For instance, a person hoping to learn Swiss German will probably be more efficient in their learning trajectory if they have already acquired knowledge of other related languages, like say Dutch or English. Transfer is believed to be an integral part of the human learning process,[1] thereby begging the question, can machine learning models benefit from similar transfer-related effects?

The idea of transfer learning in neural networks actually dates all the way back to the 70s (Bozinovski, 2020). Bozinovski and Fulgosi (1976) first posed the question of transfer learning as follows: Consider a neural network $f_{\boldsymbol{\theta}}$ that has been trained on a first supervised learning task. Next, it is provided a second supervised learning task with an accompanying train and test split of data. The parameters (or memory) $\boldsymbol{\theta}$ of the network are then updated through some learning algorithm on the train set. Now, is it possible that learning on the first task allows learning on the second task with a smaller (positive transfer learning) or larger (negative transfer learning) set of training data, compared to the case of no previous learning? Bozinovski and

[1] Transfer of learning benefits is in fact one of the main reasons for the strong focus on mathematics in a typical school curricula – it strengthens the student in their ability to apply problem solving, reasoning and several other skills more broadly (Hohensee and Lobato, 2021).

Fulgosi (1976) provided a geometrical model, along with empirical evidence, showing that such was indeed the case.

Years later, when neural networks had again started to gain traction within the AI community, a seminal work by Pratt (1992) introduced the discriminability-based transfer (DBT) algorithm. DBT went beyond simply using the very same network that had been trained on the first task as an initialized network on the second task, by additionally scaling the parameters of the network according to how well they fit the training data of the second task. It was able to achieve the same asymptotic performance as that of randomly initialized networks with fewer training epochs, across several different tasks.

The core idea remains more or less the same today. As we will see however, it is not even necessary to update the parameters of the network in order to achieve positive effects of transfer learning with large language models. Indeed, a recent and very effective trend in transfer learning with large language models is that of demonstrating a new task with a set of examples in the context window of an already trained language model, which nudges the language model into generating text according to the specifications of that new task. This is typically referred to as **prompting** in the context of NLP (also, modulo context and some nuances, called "in-context learning" or "few-shot learning"), and we will discuss it in depth in Chapter 4. Going beyond Bozinovski and Fulgosi's (1976) original definition, we want to encapsulate such forms of transfer learning as well. However, in this course we consider only the case in which the first learning task is that of language modeling. We arrive at the following operational definition:

Definition 2.1.1 (Transfer learning for language models)**.** *Consider a language model* $p_{LM}(\boldsymbol{y}; \boldsymbol{\theta})$ *over* Σ^* *trained on corpus* $\mathcal{D} = \{\boldsymbol{y}^{(n)} \mid \boldsymbol{y}^{(n)} \in \Sigma\}_{n=1}^{N}$*. Next, consider a target task* $\mathcal{T}$*, posed as learning a function* $f : L \mapsto \mathcal{Y}$*, for some input space* $L \subseteq \Sigma^*$ *and output space* $\mathcal{Y}$*. We say that transfer learning occurs if parameterizing* f *as* $f_{\hat{\boldsymbol{\theta}}}$*, with* $\hat{\boldsymbol{\theta}} \subseteq \boldsymbol{\theta}$*, allows for more efficient learning of* $\mathcal{T}$ *compared to initializing* f *as* $f_{\boldsymbol{\theta}'}$*, with* $\boldsymbol{\theta}'$ *being some set of parameters that are sampled randomly from some distribution.*

Note that $\mathcal{T}$ can be any task, as long as it takes a language L as input. The above definition is intentionally left open in what we mean by "efficient learning", and it does not necessarily involve training in the traditional sense of updating the parameters to minimize some loss function. We typically measure efficiency in terms of number of training samples and/or number of training iterations, *ceteris paribus*.

Now that we have formalized what we mean by transfer learning, we can introduce some related terminology that will be used in this and later chapters: The network trained on the source task (in our case language modeling) is called a **pretrained model**, and we refer to the learning process of that model as **pretraining**. The process of updating the weights of a pretrained model for a new target task is called **fine-tuning**.[2] A related concept to transfer learning is **multi-task learning**, which is the idea of sharing learned information across multiple tasks. In contrast to transfer learning, the tasks are learned jointly rather than sequentially.[3]

Language modeling is a particularly suitable task for transfer learning since it is very easy to scale up language modeling data. We only require some corpus of text that approximately captures the domain we want to model. Since the input space of language modeling is the same as the output space it does not require any expensive labeling — it is a **self-supervised** task. In the next sections, we will study a few particular instances and variations of language modeling-based transfer learning, starting with ELMo.

2.2 ELMo

As a first case study of transfer learning based on language models, we consider ELMo (Peters et al., 2018). ELMo was one of the first successful transfer learning models based on language modeling. It leverages the language modeling task to learn word representations, that is, vectors meant to represent the meaning of words. Whereas most previous approaches, such as word2vec (Mikolov et al., 2013) and GloVe (Pennington et al., 2014), trained static word representations, the word representations in ELMo are context-dependent.[4] That means that depending on the sentence fed to ELMo, the representation of a particular word will differ in order to reflect the meaning of that particular sentence. ELMo considers two separate language models, a (standard) forward language model $p_{\text{LM}}(y_t \mid \boldsymbol{y}_{<t})$ as well as a backward language model $p_{\text{LMB}}(y_t \mid \boldsymbol{y}_{>t}) \stackrel{\text{def}}{=} \prod_{t=1}^{T} p_{\text{LMB}}(y_t \mid y_{t+1}, \ldots, y_T)$. Both are implemented by stacking L layers of LSTMs, the parameters of which we refer to as $\overrightarrow{\boldsymbol{\theta}}$ and $\overleftarrow{\boldsymbol{\theta}}$ respectively. For a given input token y_t,

[2] Note that fine-tuning does not encapsulate all forms of transfer learning, as our definition does not necessitate updating the parameters of the language model.

[3] It has become common in recent years to evaluate models on multiple tasks in order to test their multi-task learning abilities. See for instance DecaNLP (McCann et al., 2018).

[4] The model of McCann et al. (2017) is another early contextualized word vector model. However, it was pre-trained on machine translation rather than language modeling.

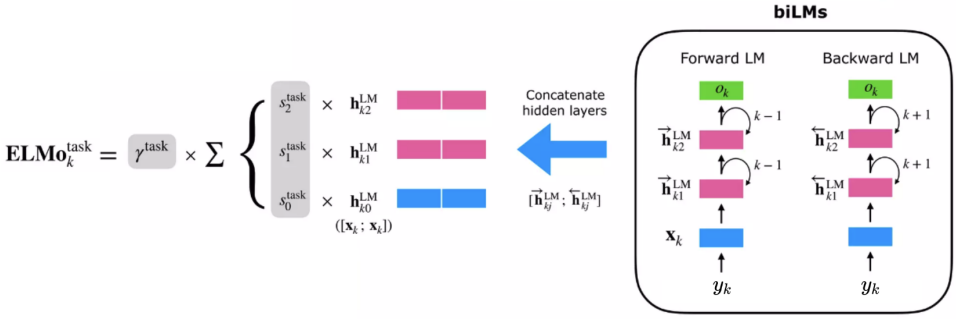


Figure 2.1: Illustration of the ELMo model architecture. Hidden representations from forward and backward language models are concatenated in order to yield representations that consider both left and right context. We can then fine-tune weights for these layer-specific representations for various downstream tasks.

the forward LSTM layers output context-dependent representations $\overrightarrow{\mathbf{h}}_{tl}^{\text{LM}}$, with $l \in [0, L]$ (and analogously, $\overleftarrow{\mathbf{h}}_{tl}^{\text{LM}}$ for the LSTM layers of the backward language model). The deepest representations $\overrightarrow{\mathbf{h}}_{tL}^{\text{LM}}$ and $\overleftarrow{\mathbf{h}}_{tL}^{\text{LM}}$ are fed to a softmax layer to predict the forward and backward probabilities of y_t. In addition to $\overrightarrow{\boldsymbol{\theta}}$ and $\overleftarrow{\boldsymbol{\theta}}$, the parameters for token representations and the softmax layer (denoted together as $\boldsymbol{\theta'}$) are tied between the two networks. All parameters are optimized jointly by maximizing the log likelihoods of the forward and backward models:

$$\mathcal{L}_{\text{ELMo}}(\boldsymbol{\theta}) = \sum_{n=1}^{N} \sum_{t=1}^{T} \log p_{\text{LM}}(y_t{}^{(n)} \mid \boldsymbol{y}_{<t}^{(n)}; \overrightarrow{\boldsymbol{\theta}}, \boldsymbol{\theta'}) + \log p_{\text{LMB}}(y_t{}^{(n)} \mid \boldsymbol{y}_{>t}^{(n)}; \overleftarrow{\boldsymbol{\theta}}, \boldsymbol{\theta'}) \tag{2.1}$$

Now in order to fine-tune for a specific task, we can use the context-specific representations $\mathbf{h}_{tl}^{\text{LM}} = [\overleftarrow{\mathbf{h}}_{tl}^{\text{LM}}; \overrightarrow{\mathbf{h}}_{tl}^{\text{LM}}]$. One could simply take the last layer representations $\mathbf{h}_{tL}^{\text{LM}}$ and use those as input to a separate model that is fine-tuned on another task. The original paper additionally experimented with learning a task-specific representation as a scaled convex combination over the hidden representations, as follows:

$$\mathbf{ELMo}_t^{\text{task}} = \boldsymbol{\gamma}^{\text{task}} \sum_{l=0}^{L} s_l{}^{\text{task}} \mathbf{h}_{tl}^{\text{LM}}, \tag{2.2}$$

where $\sum_{l=0}^{L} s_l{}^{\text{task}} = 1$ are the outputs of a softmax function over the hidden

Task	Previous SOTA		Our baseline	ELMo + baseline	Increase (absolute/ relative)
SQuAD	Liu et al. (2017)	84.4	81.1	85.8	4.7 / 24.9%
SNLI	Chen et al. (2017)	88.6	88.0	88.7 ± 0.17	0.7 / 5.8%
SRL	He et al. (2017)	81.7	81.4	84.6	3.2 / 17.2%
Coref	Lee et al. (2017)	67.2	67.2	70.4	3.2 / 9.8%
NER	Peters et al. (2017)	91.93 ± 0.19	90.15	92.22 ± 0.10	2.06 / 21%
SST-5	McCann et al. (2017)	53.7	51.4	54.7 ± 0.5	3.3 / 6.8%

Figure 2.2: Results of ELMo on six benchmark tasks, compared to previous state-of-the-art results. The performance metrics vary across the tasks. SNLI and SST-5 are measured by accuracy. SQuAD, SRL and NER are measured by F1. Coref is measuerad by average F1.

representations, and $\boldsymbol{\gamma}^{\text{task}} \in \mathbb{R}$ is meant to scale the representations.

Now, consider a neural network-based model $f_{\hat{\boldsymbol{\theta}}}$ that is fine-tuned on some downstream task. As input it takes a sequence of representations $(\mathrm{x}_1, \ldots, \mathrm{x}_T)$. These could be some pre-trained static representations or simple one-hot encodings. To improve the model using contextualized information we can concatenate these input representations with the ELMo representations, yielding input on the form $[\mathrm{x}_t; \mathbf{ELMo}_t^{\text{task}}]$. This only changes the input dimension of $f_{\hat{\boldsymbol{\theta}}}$, the rest of the network can remain unchanged, and be further fine-tuned.

ELMo was indeed somewhat of a breakthrough in NLP transfer learning research: By leveraging ELMo representations for other downstream tasks as just described, the original paper was able to beat what was at the time state-of-the-art performance on six different benchmark datasets. These benchmarks included the tasks of question answering, natural language inference, semantic role labeling, coreference resolution, named entity recognition and sentiment analysis. See Fig. 2.2 for exact numbers.

2.3 BERT

After the success of ELMO, Devlin et al. (2019) pre-trained BERT (Bidirectional Encoder Representations from Transformers), a Transformer-based bidirectional masked language model. BERT is pre-trained with the masked language modeling objective on large scale text corpus. After pre-training, BERT can be fine-tuned to perform different NLP task without the need of design task-specific architectures. BERT advanced the state-of-the-art of many NLP benchmarks at the time it was proposed.

In this section, we first introduce the pre-training objective of BERT. We then describe BERT model architecture and the *pre-training and fine-tuning* paradigm.

2.3.1 BERT: Pre-training Objectives

BERT is pre-trained with two self-supervised tasks: **masked language modeling** and **next sentence prediction**.

Masked Language Modeling Masked language modeling (MLM) is first proposed by Taylor (1953) in the literature, who referred to this as a Cloze task. Devlin et al. (2019) adapted this task as a novel pre-training task to overcome the drawback of the standard unidirectional LM.In the masked language modeling setup, the goal is to predict the omitted token from a piece of text that constitutes a logical and coherent completion. For example, in the piece of text "The students [MASK] to learn about language models", we predict *want* or *like* with high probability for the [MASK] position. The goal of masked language modeling is to approximate the probability distribution over tokens in our vocabulary as the original token at a given masked position. Similarly to the standard language modeling objective, we can choose model parameters by optimizing for the log-likelihood of a dataset $\mathcal{D}$. Albeit in this case, the words at a percentage of randomly-chosen positions in $\mathcal{D}$ are replaced with [MASK] and the model is given **both** sides of context around the masked token in order to make its prediction:

$$\mathcal{L}_{\texttt{MLM}}(\boldsymbol{\theta}) = \sum_{n=1}^{N} \sum_{t=1}^{T} \log p_{\texttt{MLM}}({y_t}^{(n)} \mid \boldsymbol{y}_{<t}^{(n)}, \boldsymbol{y}_{>t}^{(n)}; \boldsymbol{\theta}) \mathbb{1}\{{y_t}^{(n)} = \texttt{[MASK]}\} \tag{2.3}$$

However, this pre-training method will create a mismatch between the pre-training phase and the fine-tuning phase because the mask token does not appear during the fine-tuning phase. Empirically, to deal with this issue, Devlin et al. (2019) used a special [MASK] token 80% of the time, a random token 10% of the time and the original token 10% of the time to perform masking.

Next Sentence Prediction The next sentence prediction objective is included to enable the model to capture the relationships between two consecutive sentences. This is important because many NLP tasks require understanding the relationship between different text inputs (e.g., question

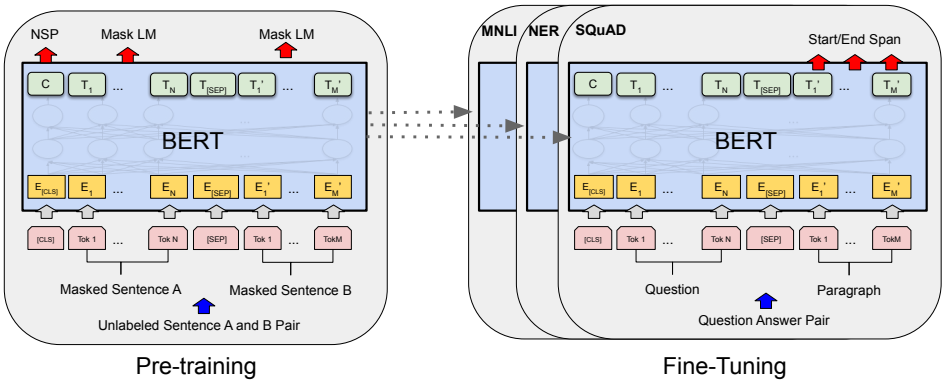


Figure 2.3: Illustration of BERT model architecture.

and context for the question answering task). By pre-training the model to predict whether a given sentence follows another sentence, we can help the model learn the dependencies and relationships between sentences.

The next sentence prediction objective is implemented as follows. Given a pair of sentences, denoted as A and B, the model is trained to predict whether B is the next sentence that follows A. This is done by feeding the two sentences as input to the model, along with a special [CLS] token that serves as the input representation of the entire sequence. The model then generates a binary output that indicates whether B is the next sentence.

BERT is pre-trained with the combination of the masked language modeling objective and the next sentence prediction objective on English Wikipedia data and the BookCorpus (Zhu et al., 2015) dataset, which is a collection of over 11,000 books from various genres. The resulting corpus contains over 20 GB of text with over 3.3 billion words and approximately 800 million sentences. BERT is pre-trained with a batch size of 256 sequences for 1M steps.

2.3.2 BERT Architecture

As illustrated in Figure 2.3, BERT's model architecture is a multi-layer bidirectional Transformer encoder. BERT is a Transformer encoder model. It takes a sequence of text tokens as input and produces their contextualized representations and (optionally) predictions. We denote the number of layers (i.e., Transformer blocks) as L, the hidden size as H, and the number of self-attention heads as A.[5] Devlin et al. (2019) pre-trained two model

[5]In all cases the feed-forward/filter size is set to be $4H$, i.e., 3072 for the $H = 768$ and 4096 for the $H = 1024$.

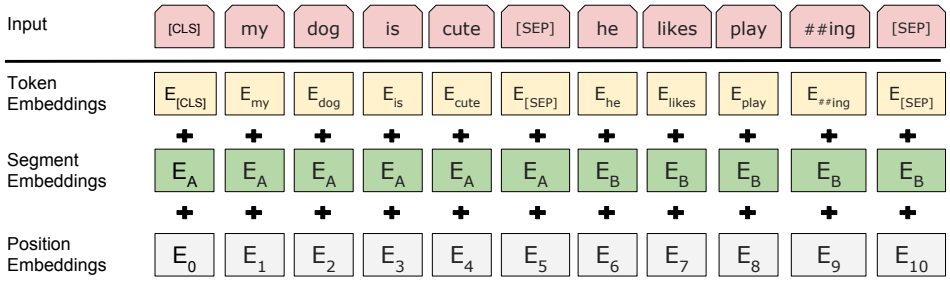


Figure 2.4: BERT input representation. The input embedding are the sum of the token embeddings, the segmentation embeddings and the position embeddings.

sizes: $\mathbf{BERT_{BASE}}$ (L=12, H=768, A=12, Total Parameters=110M) and $\mathbf{BERT_{LARGE}}$ (L=24, H=1024, A=16, Total Parameters=340M).

Input/Output Representations To make BERT handle a variety of down-stream tasks, BERT input representation is able to unambiguously represent both a single sentence and a pair of sentences (e.g., ⟨ Question, Answer ⟩) in one token sequence. BERT use WordPiece embeddings Wu et al. (2016) with a 30,000 token vocabulary. The first token of every sequence is always a special classification token (`[CLS]`). The final hidden state corresponding to this token is used as the aggregate sequence representation for classification tasks. Sentence pairs are packed together into a single sequence. We differentiate the sentences in two ways. First, we separate them with a special token (`[SEP]`). Second, we add a learned embedding to every token indicating whether it belongs to sentence `A` or sentence `B`. For a given token, its input representation is constructed by summing the corresponding token, segment, and position embeddings. A visualization of this construction can be seen in Figure 2.4.

2.3.3 Fine-tuning BERT

To apply pre-trained BERT models on different downstream tasks, we can simply plug in the task-specific inputs and outputs into BERT and fine-tune all the parameters end-to-end. This *pre-training and fine-tuning* paradigm alleviates need of designing or including any task-specific architectures as EMLO does.

As illustrated in Figure 2.5, at the input, sentence `A` and sentence `B` from pre-training are analogous to (1) sentence pairs in paraphrasing, (2)

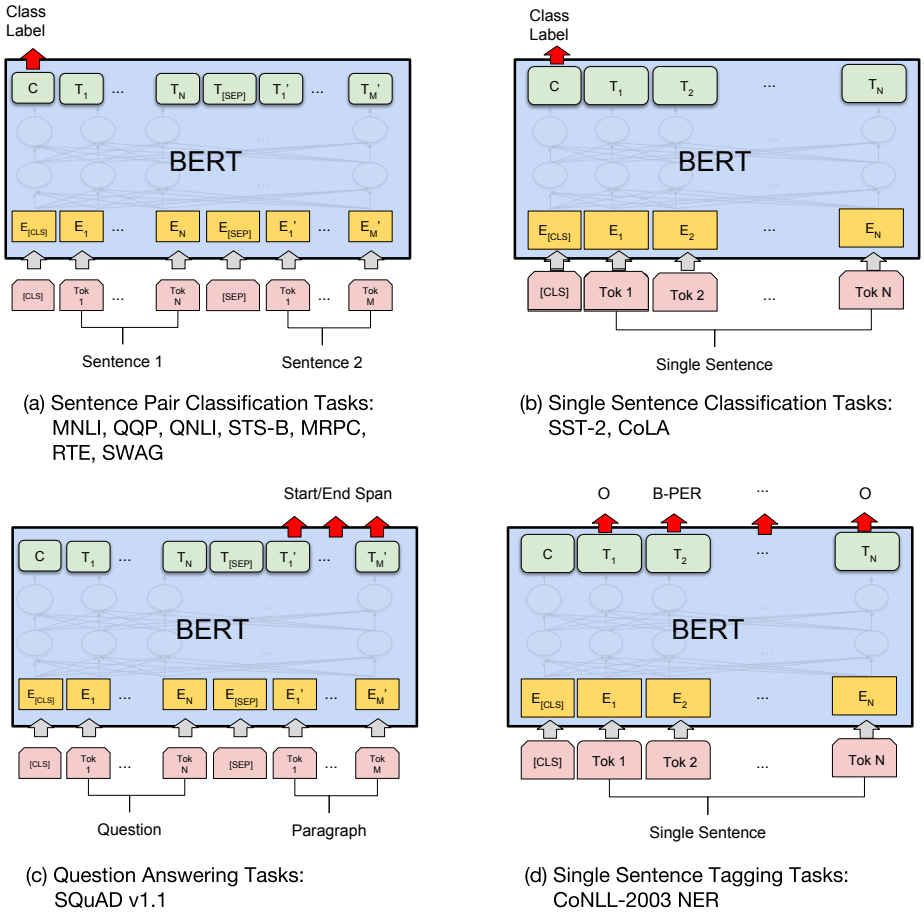


Figure 2.5: Illustration of BERT fine-tuning procedure for different NLP tasks.

hypothesis-premise pairs in entailment, (3) question-passage pairs in question answering, and (4) a degenerate text-$\varnothing$ pair in text classification or sequence tagging. At the output, the token representations are fed into an output layer for token-level tasks, such as sequence tagging or question answering, and the `[CLS]` representation is fed into an output layer for classification, such as entailment or sentiment analysis. As shown in Figure 2.6, BERT substantially outperforms both ELMo and OpenAI GPT, as well as all previous state-of-the-art on the GLUE benchmark. This confirms the effectiveness of pre-training a bi-directional Transformer encoder using the masked language modeling objective.

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	**Average** -
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 2.6: Experimental results of fine-tuning BERT on the GLUE benchmark.

2.4 Other Transformer Language Models

Insipred by the success of BERT, a number of Transformer Language Models (TLMs) have been pre-trained. We categorize these TLMs to three types according to their architecture designs: (i) BERT variants, which are Transformer encoder models, (ii) GPTs, which are Transformer decoder models, and (iii) Seq2Seq TLMs, which are Transformer encoder-decoder models.

2.4.1 BERT Variants

After the success of BERT, the community made a lot effort to improve the performance of BERT-like Transformer encoders. We describe a few representative work as follows:

RoBERTa RoBERTa (Robustly Optimized BERT Approach) is an optimized version of BERT developed by Liu et al. (2019). RoBERTa uses the same model architecture of BERT but with some improvements to the pre-training process. RoBERTa uses a larger pre-training corpus, larger batch size, longer training time, longer input sequence lengths, larger batch size, and a dynamic masking strategy during pre-training.

Specifically, RoBERTa adds CC-NEWs and OpenWebText to the original BERT pre-training data. The resulting corpus contains over 160 GB of text and approximately 2.5 billion word pieces. RoBERTa is pre-trained with a batch size of 8,000 sequences of 512 tokens for 500k steps, resulting in much more computation compared to BERT. The dynamic masking strategy, which randomizes the masking pattern in each epoch and helps the model to learn more robust representations of the input text. These modifications allows it to surpass BERT and achieve state-of-the-art performance on several NLP tasks at the time.

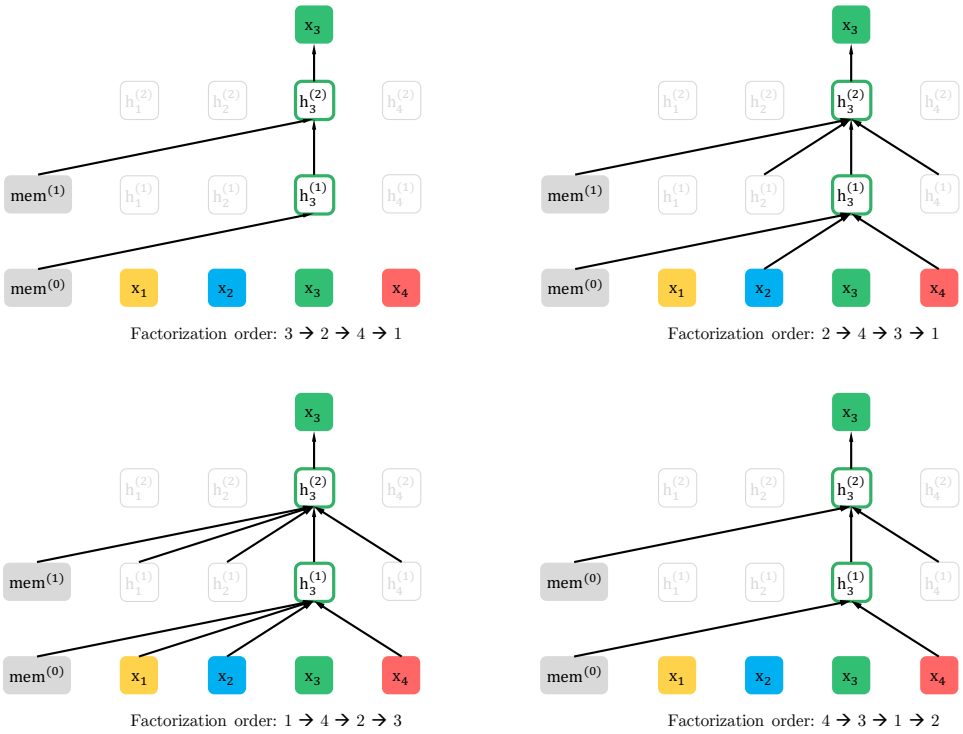


Figure 2.7: Illustration of the factorization scheme in the permutation language modeling objective of XLNet.

[Optional]: XLNet While BERT achieves strong empirical performance on various NLP tasks, it still suffers from a discrepancy between pre-training and fine-tuning. That is because BERT uses the `[MASK]` token during pre-training, but these artificial symbols are absent in the real data at the fine-tuning stage. To alleviate this discrepancy, Yang et al. (2019) proposed XLNet, a pre-trained Transformer encoder model based on the same model architecture as BERT but using a permutation language modeling objective instead of the masked language modeling objective. Permutation language modeling objective maximizes the expected log likelihood of a sequence w.r.t. all possible permutations of the factorization order.

Specifically, as illustrated in Figure 2.7, for a text sequence $\mathbf{x}$, we sample a factorization order $\mathbf{z}$ at a time and decompose the likelihood $p_\theta(\mathbf{x})$ according to factorization order. Note that the original sequence order is kept and permutation in factorization order is achieved by changing the attention mask matrix in the Transformer. Since the same model parameter θ is shared across all factorization orders during training, in expectation, x_t has

seen every possible element $x_i \neq x_t$ in the sequence, hence being able to capture the bidirectional context. Moreover, as this objective fits into the autoregressive framework, it naturally avoids the independence assumption and the pretrain-finetune discrepancy.

$$\mathcal{L} = \sum_{i=1}^{T} \sum_{j=0}^{i-1} \log p_\theta(x_{\pi_{i,j}} | \boldsymbol{x} < \pi_{i,j}) \tag{2.4}$$

where $x_{\pi_{i,j}}$ is the j-th token in the permutation of the first i tokens in the input sequence and $\boldsymbol{x} < \pi_{i,j}$ is the set of tokens that appear before the j-th token in the permutation of the first i tokens.

In addition to the permutation language modeling objective, XLNet also increases the size of pre-training data and pre-training computation budget. By combining these improvements, XLNet achieved state-of-the-art performance on several NLP tasks at the time.

[Optional] ALBERT ALBERT (A Lite BERT) is another BERT variant pre-trained by Lan et al. (2019) focusing on the *parameter efficiency* of BERT-like models. ALBERT reduces the parameter count by (i) factorized embedding parameterization which decomposes the large embedding matrix into two smaller matrices which first project the word to a lower dimensional embedding space and then project it to the hidden space, and (ii) cross-layer parameter sharing, which shares the parameter of each Transformer layer in the model architecture. ALBERT also includes a new sentence order prediction (SOP) task as alternative for the NSP task. The SOP task uses as positive examples the same technique as BERT (two consecutive segments from the same document), and as negative examples the same two consecutive segments but with their order swapped. Compared to the NSP task which focuses on topic prediction, the SOP task focuses primarily on coherence and forces the model to learn finer-grained distinctions about discourse-level coherence properties.

With these techniques, ALBERT reduces the parameter count of $\text{BERT}_{\text{BASE}}$ from 108M to 12M with moderate performance drop. Lan et al. (2019) also pre-trained larger models with larger pre-training corpus and computation budget, resulting in new state-of-the-art performance on several NLP tasks at the time. However, it is worth noting that while ALBERT significantly reduces the parameter count of BERT-like models, the computation cost remains the same and could still be a bottleneck for real-world applications.

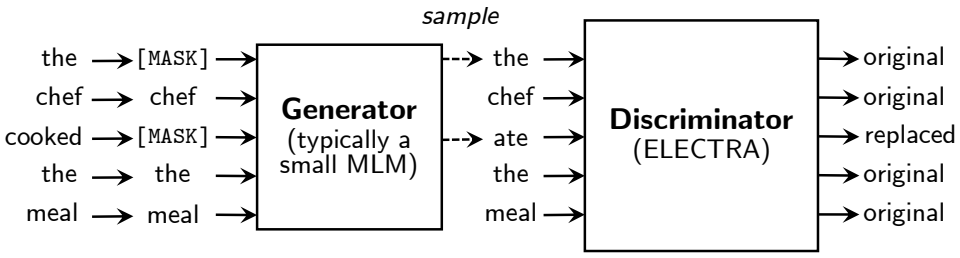


Figure 2.8: Illustration of ELECTRA model and the replaced token detection framework.

[Optional] ELECTRA ELECTRA (Efficiently Learning an Encoder that Classifies Token Replacements Accurately) is another BERT variant pre-trained by Clark et al. (2020). ELECTRA uses the same model architecture with the aforementioned BERT-like models but is pre-trained with a *discriminative* self-supervised objective called the Replaced Token Detection (RTD) task instead of generative ones such as masked language modeling or permutation language modeling.

Specifically, as illustrated in Figure 2.8, the ELECTRA pre-training approach involves training a generator and a discriminator model simultaneously. The generator model is used to create corrupted versions of the input data by replacing some of the tokens with random tokens and is trained with the masked language modeling objective. The discriminator model is a larger and more complex model compared to the generator model and is trained to maximize the probability of correctly distinguishing between the original input data and the corrupted data at token level. After pre-training, the generator model is discarded and only the discriminator model is fine-tuned on downstream tasks. This training framework is more computationally efficient because it provides meaningful training signal on each input token, instead of only 15% of the tokens that are masked. Experiments show that ELECTRA achieves competitive performance compared to the RoBERTa model with fewer computational budget.

2.4.2 GPTs

We then describe another important type of pre-trained language models, i.e., Transformer decoder models. These kind of models are *true* language models according to the definition and can be used for generative tasks. The most representative models in this type are the GPT family pre-trained by OpenAI. We will introduce them as follows.

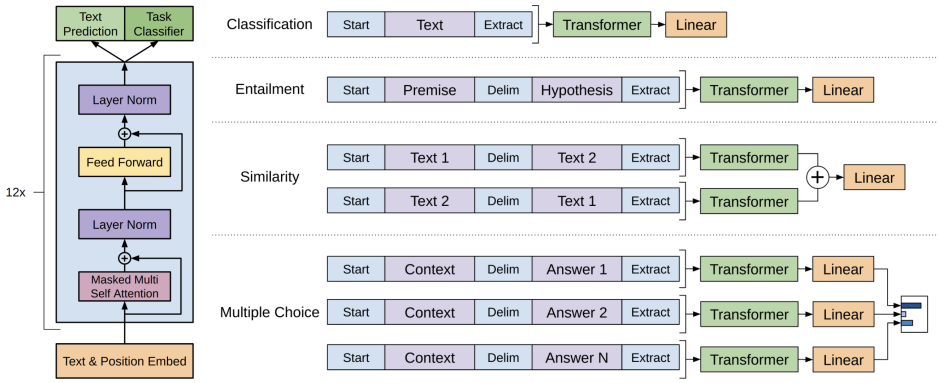


Figure 2.9: Illustration of how to fine-tune GPT on natural language understanding tasks.

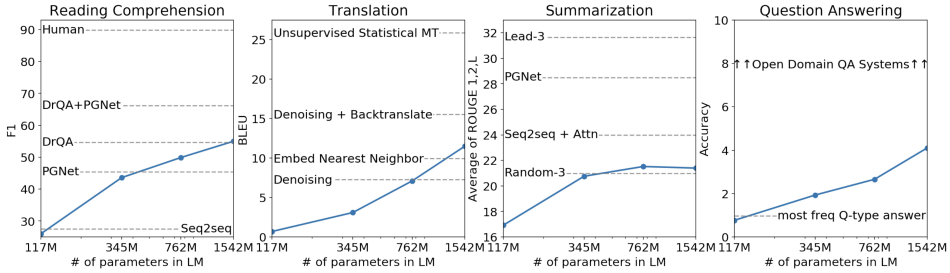


Figure 2.10: Illustration of GPT-2's zero-shot performance on various NLP tasks with respect to model sizes.

GPT A few months before BERT is released, Radford et al. (2018) pre-trained the first version of the GPT (Generative Pre-training Transformers) family. GPT is a decoder-only Transformer language model pre-trained with the standard language modeling objective on large scale web text. After pre-training, GPT can be fine-tuned on various natural language understanding tasks by transforming the inputs into a single text sequence. The sequence is fed into the GPT model and the final hidden state is used as the sequence representation, which is then fed into a fully-connected layer to produce predictions. The GPT fine-tuning procedure is illustrated in Figure 2.9.

GPT-2 After the success of GPT, OpenAI further scales generative pre-training with GPT-2 (Radford et al., 2019) with larger models and more training data. In addition to better performance with fine-tuning, they also find that generative pre-training on large scale text data makes the

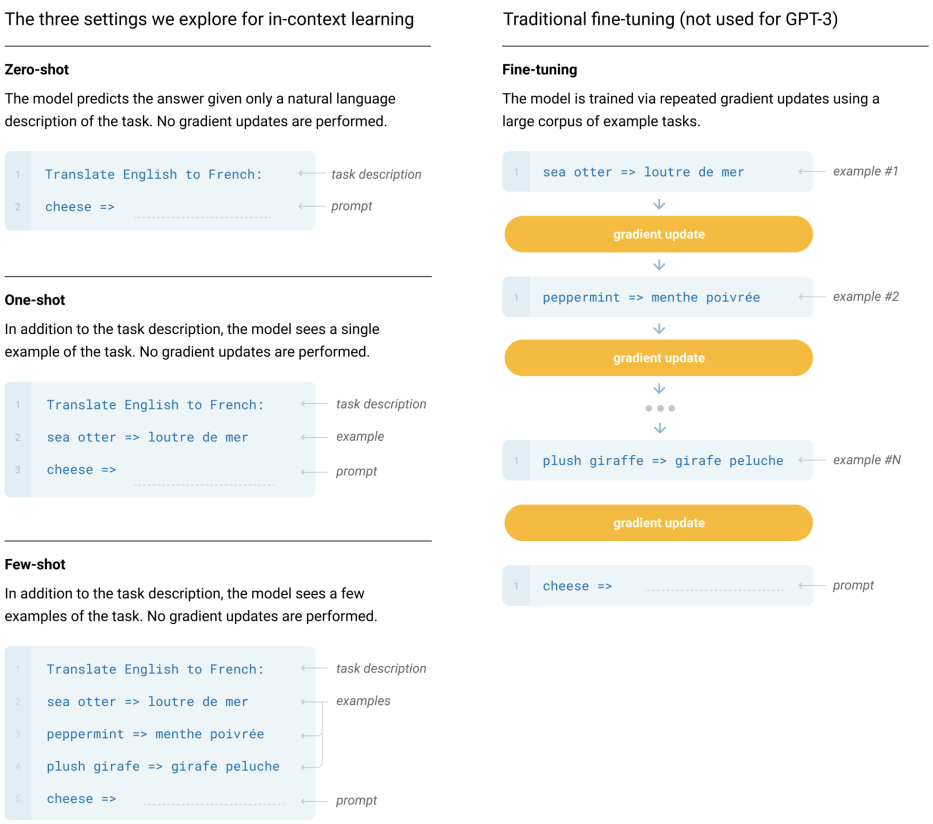


Figure 2.11: Comparison of few-shot in-context learning with zero-shot learning and fine-tuning.

model becomes unsupervised multi-task learners that performs reasonably on various NLP tasks in the zero-shot setting. As shown in Figure 2.10, the zero-shot performance of GPT-2 significantly improves when scaling the model size, and the largest GPT-2 model with 1.5 Billion parameters achieves non-trivial performance on some NLP tasks. This for the first time demonstrate the potential of scaling LLM pre-training for zero-shot inference.

GPT-3 Motivated by the success of scaling model size and pre-training corpus in GPT-2, OpenAI further scales the size of decoder-only Transformer language models to 175 Billion, which is over 100 times larger compared to the largest GPT-2 model. They find that in addition to improved zero-shot performance, GPT-3 also exhibits strong few-shot "in-context learning" ability. As illustrated in Figure 2.11, in-context learning append both the

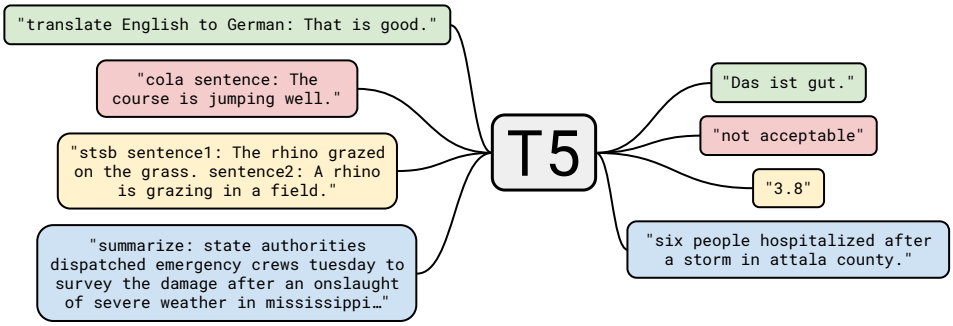


Figure 2.12: Illustration of the text-to-text framework of T5.

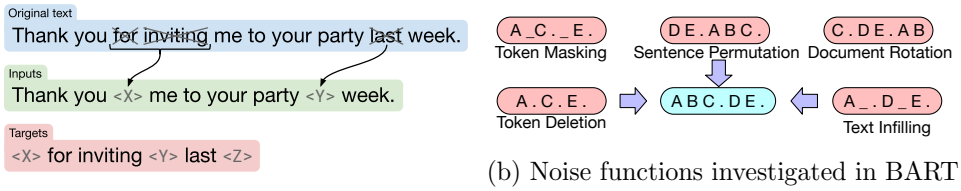
task description and a few demonstrations of the task to the original input as the input for the language model, which produces the prediction by auto-regressively predicting next tokens. Different

2.4.3 Seq2seq TLMs

Apart from encoder-only and decoder-only Transformer language models, researchers also explored pre-training encoder-decoder Transformer language models. Compared to BERT-like encoder-only models and GPT-like decoder-only models, encoder-decoder models suits sequence-to-sequence generation tasks such as machine translation and text summarization better. This is because the encoder help the model better understand the input text with bidirectional attention flow, and the decoder help the model generate more fluent outputs. We will introduce T5 (Raffel et al., 2020) and BART (Lewis et al., 2020), two representative pre-traiend encoder-decoder Transformer language models, in this section.

T5 After the success of BERT and GPTs on natural language understanding and unconditional text generation, it is natural to consider pre-training an encoder-decoder model to improve seq2seq tasks. To this end, Raffel et al. (2020) pre-trained T5 ("Text-to-Text Transfer Transformer"), a powerful pre-trained encoder-decoder language model.

As illustrated in Figure 2.12, T5 is a general-purpose seq2seq model that can handle both natural language understanding and generation tasks with a text-to-text framework: the model take task description and task input as the encoder input and output the answer with the decoder. This paradigm is very influential to future NLP research because it demonstrates the possibility of building a single general-purpose model that can handle all



(a) Text infilling objective.

(b) Noise functions investigated in BART pre-training.

Figure 2.13: Illustrations of the text infilling objective in T5 and other noise functions investigated in BART.

NLP tasks.

T5 is pre-trained by mixing self-supervised data with multi-task supervised data. T5 uses a "text infilling" objective, which is shown to outperform other possible variants, as the source of self-supervision. As illustrated in Figure 2.13a, it randomly masks a portion of contiguous tokens and train the model to predict the masked text spans. Raffel et al. (2020) collected the C4 ("Colossal Clean Crawled Corpus") dataset and construct massive text infilling data with it. For supervised data, T5 convert training data for GLUE, SuperGLUE, machine translation, text summarization, question answering, etc., into text-to-text format by prepending task-specific prefixes. During pre-training, T5 randomly samples text-to-text training data according to the size of different tasks.

After pre-training, T5 can achieve competitive performance on supervised pre-training tasks without fine-tuning by prepending the corresponding prefix to the input. T5 also achieves state-of-the-art performance on a wide range of NLP tasks with task-specific fine-tuning.

BART Concurrently with T5, Lewis et al. (2020) pre-trained BART, another powerful seq2seq pre-trained model. As illustrated in Figure 2.13b, they investigated a number of noise functions as sources for self-supervision, and found the combination of text infilling and sentence permutation to be the best. Then it pre-trained BART on the same pre-training data as RoBERTa and obtained state-of-the-art performance on some natural language generation tasks.

Chapter 3

Parameter efficient finetuning

Pretrained language models are used in a wide range of NLP tasks. When the model size becomes larger and larger, it is more and more difficult to tune the model with limited size of annotated data. Overfitting easily happens, and the cost of model tuning is quite expensive. Thus, to avoid above issues, various parameter-efficient tuning methods are proposed. In the following sections, we will first introduce **partially finetuning** techniques. They are simple but effective, which is widely used in Transformer model tuning. Besides selecting part of parameters for tuning, another line of work explores how to keep the model frozen, and add extra tuned parameters. These newly added and tuned modules are called **adapters**.

3.1 Partially Finetuning

This type of tuning methods (which is also called *specification-based tuning* (Ding et al., 2022)) fine-tune a few inherent parameters while leaving the majority of parameters unchanged in model adaptation. This approach does not seek to change the internal structure of a model but to optimize a small number of internal parameters to solve particular tasks. Generally, such specifications could be implemented based on heuristics or training supervision.

Heuristic Specification. Specification-based methods do not introduce any new parameters in the model, but directly specify part of the parameters to be optimized. The idea is simple but surprisingly effective, Lee et al. (2019) **only fine-tune one-fourth of the final layers** of BERT and RoBERTa and could produce 90% of the performance of full parameter fine-tuning.

BitFit (Zaken et al., 2022) empirically proves that by **only optimizing the bias terms** inside the model and freezing other parameters, the model could still reproduce over 95% performance on several benchmarks. The biased term exists in both attention mechanism and MLP feedforward layer. BitFit can fine-tune only two bias components (the "query" and "middle-of-MLP" bias terms), amounting to half of the bias parameters in the model, and only 0.04% of all model parameters. Corresponding bias terms are colored red. Other bias terms are colored in blue.

$$\texttt{Query in Attention}: \quad \mathbf{Q}(\mathbf{x}) = \mathbf{W}_q\mathbf{x} + \mathbf{b}_q \tag{3.1}$$

$$\mathbf{K}(\mathbf{x}) = \mathbf{W}_k\mathbf{x} + \mathbf{b}_k \tag{3.2}$$

$$\mathbf{V}(\mathbf{x}) = \mathbf{W}_v\mathbf{x} + \mathbf{b}_v \tag{3.3}$$

$$\texttt{MLP in Feedfoward}: \quad \mathbf{h}_2 = \text{Dropout}(\mathbf{W}_1 \cdot \mathbf{h}_1 + \mathbf{b}_1) \tag{3.4}$$

$$\mathbf{h}_3 = \mathbf{g}_{LN_1} \odot \frac{(\mathbf{h}_2 + \mathbf{x}) - \mu}{\sigma} + \mathbf{b}_{LN_1} \tag{3.5}$$

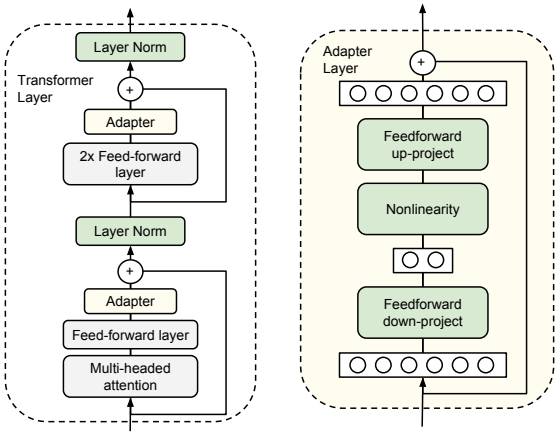
$$\mathbf{h}_4 = \text{GELU}(\mathbf{W}_2 \cdot \mathbf{h}_3 + \mathbf{b}_2) \tag{3.6}$$

$$\mathbf{h}_5 = \text{Dropout}(\mathbf{W}_3 \cdot \mathbf{h}_4 + \mathbf{b}_3) \tag{3.7}$$

$$\text{out} = \mathbf{g}_{LN_2} \odot \frac{(\mathbf{h}_5 + \mathbf{h}_3) - \mu}{\sigma} + \mathbf{b}_{LN_2} \tag{3.8}$$

Empirical results in BitFit also show that even if we use a small random set of parameters for tuning (which obviously will degrade the performance), the model could still yield passable results on the GLUE benchmark. Unfortunately, the work only applies this trick to small-scale models, and there is no guarantee that randomly choosing some parameters to be tuned would remain competitive for larger models. Another valuable observation is that different bias terms may have different functionalities during model adaptation.

Learn the Specification. Rather than manually or heuristically specify which parameters to be updated, one alternative is to "learn" such specifications. *Diff pruning* (Guo et al., 2021) reparameterizes the fine-tuned model parameters as the summation of the pre-trained parameters and the difference vector as $\theta_{FT} = \theta_{LM} + \delta_{Diff}$. Hence, the key issue is to encourage the difference vector δ_{Diff} to be as sparse as possible. This work regularizes the vector by a differentiable approximation to the L_0-norm penalty as $||\delta_{Diff}||_0$ to achieve the goal of sparsity. Practically, because new parameters to be optimized are introduced in the learning phase, *Diff pruning* takes up more GPU memory than full parameter fine-tuning, which may establish barriers in the application on large language models.



(a) Adapters are inserted after the multi-head attention sublayer and the feed-forward sublayer.

(b) The adapter consists of a down-projection, an up-projection, and a skip-connection. The intermediate hidden state typically have smaller size, which is thereby called a "bottleneck".

Figure 3.1: Adapter Houlsby et al. (2019). During adapter tuning, blocks in green are trained, including adapters, layer normalizations, and the final classification layer (not shown in the figure).

3.2 Adapter Tuning

Adapter tuning (Rebuffi et al., 2017) inserts small modules called **adapters** to a model. Adapters can be any network architectures and can be placed anywhere in the model, but many follow Houlsby et al. (2019)'s practice to place a two-layer feed-foward neural network with a bottleneck after each sublayer (including both the multi-head attention sublayer and the feed-forward sublayer) within the transformer (Fig. 3.1):

$$\mathbf{h} \leftarrow \mathbf{h} + f(\mathbf{h}\mathbf{W}_{\text{down}})\mathbf{W}_{\text{up}} \tag{3.9}$$

where f is a nonlinear activation function.

It has been shown that adapter tuning can attain performance comparable to full fine-tuning by training only a small fraction of parameters (Fig. 3.2).

3.2.1 [Optional Reading] Example: Cross-Lingual Transfer

In this section, we introduce the application of adapters to cross-lingual transfer. LLMs pretrained on multiple languages such as multilingual BERT (Devlin et al., 2019) and XLM-R (CONNEAU and Lample, 2019) enables

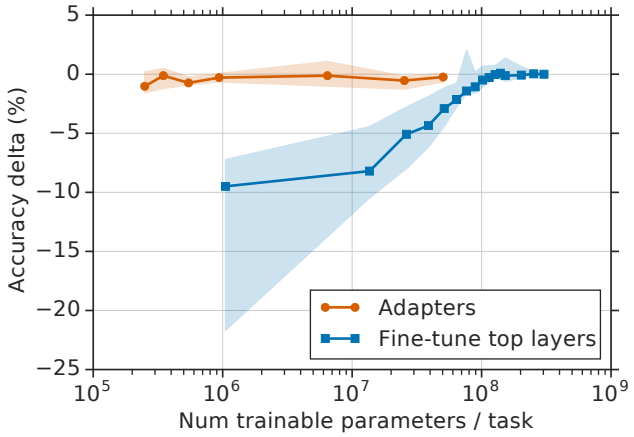


Figure 3.2: Number of trained task-specific parameters for adapter tuning and fine-tuning.

few-shot or even zero-shot cross-lingual transfer to low-resource languages (Pires et al., 2019). However, the capacity of a single model is limited: it cannot cover an unlimited number of languages and better performance on low-resource languages often comes at the cost of worse performance on high-resource languages (compared to monolingual models) (Conneau et al., 2020). To alleviate the issue, we can learn an adapter for each language, which allows the model to incorporate more languages without having the languages interfering with each other.

Pfeiffer et al. (2020) proposed a Multiple ADapters for Cross-lingual transfer (MAD-X) framework (Fig. 3.3) that comprises three types of adapters: language, task, and invertible adapters.

Language Adapter A language adapter is introduced for each distinct language. The language adapters are trained using masked language modeling.

Task Adapter A task adapter is stacked on top of the language adapter to capture task-specific knowledge. During fine-tuning on downstream tasks, language adapters are fixed and only task adapters are trained.

Invertible Adapter There often exists a mismatch between the pre-trained multilingual LLM's vocabulary and the target language's vocabulary.

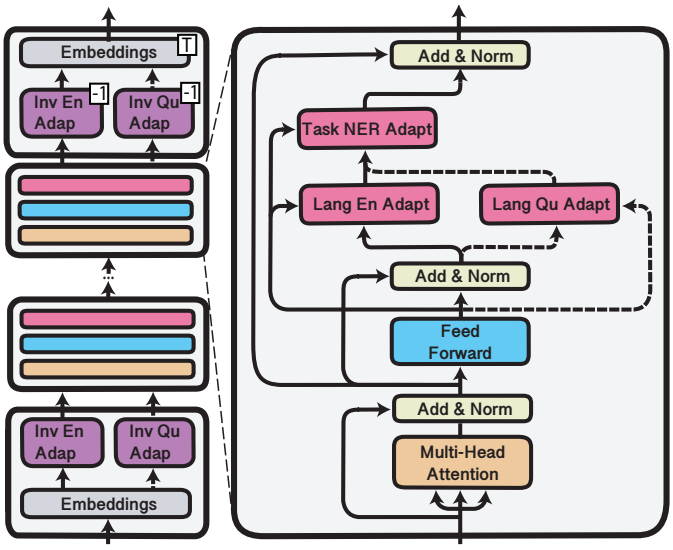


Figure 3.3: MAD-X framework

To mitigate it, invertible adapters are stacked on top of the embedding layer to better accommodate for the target language. Furthermore, since input and out embeddings are tied, the inverses of the adapters are placed before the output embedding layer to revert it for inference. Invertible adapters split an input embedding vector $\mathbf{e}$ into two vectors of equal dimensionality: $\mathbf{e}_1, \mathbf{e}_2$ and applies the following transformations:

$$\begin{aligned} \mathbf{o}_1 &= F(\mathbf{e}_2) + \mathbf{e}_1 \\ \mathbf{o}_2 &= G(\mathbf{o}_1) + \mathbf{e}_2 \\ \mathbf{o} &= [\mathbf{o}_1, \mathbf{o}_2] \end{aligned} \tag{3.10}$$

Its inverse can be computed as follows:

$$\begin{aligned} \mathbf{e}_2 &= \mathbf{o}_2 - G(\mathbf{o}_1) \\ \mathbf{e}_1 &= \mathbf{o}_1 - F(\mathbf{e}_2) \\ \mathbf{o} &= [\mathbf{e}_1, \mathbf{e}_2] \end{aligned} \tag{3.11}$$

F and G can be arbitrary non-linear functions. In Pfeiffer et al. (2020), a function similar to language and task adapters are used (minus the residual connection):

$$\begin{aligned} F(\mathbf{x}) &= \text{ReLU}(\mathbf{x}\mathbf{W}_{\text{down},F})\mathbf{W}_{\text{up},F} \\ G(\mathbf{x}) &= \text{ReLU}(\mathbf{x}\mathbf{W}_{\text{down},G})\mathbf{W}_{\text{up},G} \end{aligned} \tag{3.12}$$

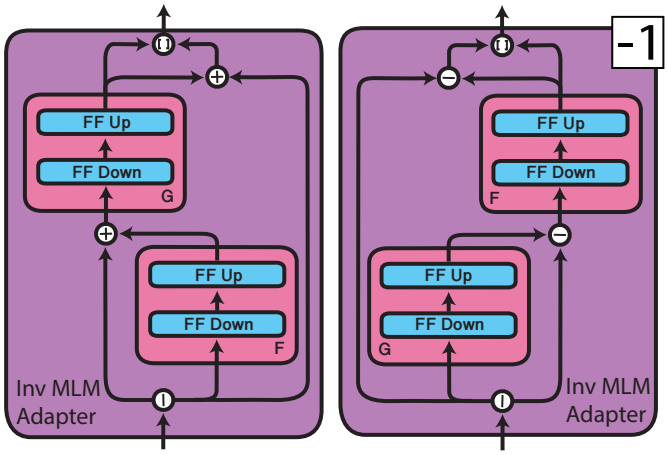


Figure 3.4: Invertible Adapter

Invertible adapters are trained together with language adapters using MLM and are also fixed during task-specific fine-tuning. The architecture of an invertible adapter and its inverse is shown in Fig. 3.4.

3.2.2 [Optional Reading] LoRA

Bottleneck adapters introduce extra compute in adapter layers, which can not be bypassed with model parallelism since they have to be processed sequentially. Even though adapter layers are designed to have very few parameters, it could still lead to a non-negligible latency, especially during inference. To address this issue, Hu et al. (2022) propose LoRA (Fig. 3.5), which injects trainable low-rank matrices to approximate the weight updates. For a pre-trained weight matrix $\mathbf{W} \in \mathbb{R}^{d\times k}$ (e.g. query and value projection matrices in the multi-head attentions), the update is computed with a low-rank decomposition:

$$\Delta\mathbf{W} = \mathbf{BA} \tag{3.13}$$

where $\mathbf{B} \in \mathbb{R}^{d\times r}$, $\mathbf{A} \in \mathbb{R}^{r\times k}$. During training, $\mathbf{W}$ is frozen and only $\mathbf{A}$ and $\mathbf{B}$ are trained. $\mathbf{A}$ is initialized with a random Gaussian distribution, and $\mathbf{B}$ is initialized to be zero. The update is then scaled by a constant $\frac{\alpha}{r}$, which is roughly equivalent to tuning the learning rate. In summary, given a specific input $\mathbf{x}$ to the layer, we have

$$\mathbf{h} = \mathbf{W}\mathbf{x} + \frac{\alpha}{r} \cdot \Delta\mathbf{W}\mathbf{x} = \mathbf{W}\mathbf{x} + \frac{\alpha}{r} \cdot \mathbf{B}\mathbf{A}\mathbf{x}. \tag{3.14}$$

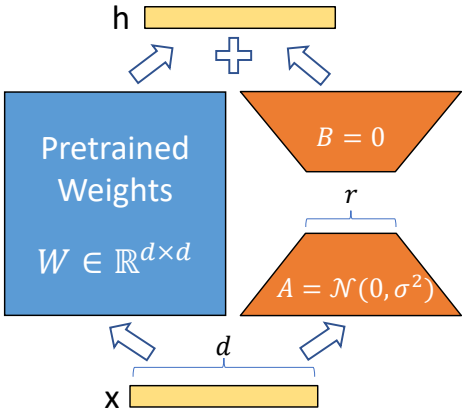


Figure 3.5: LoRA.

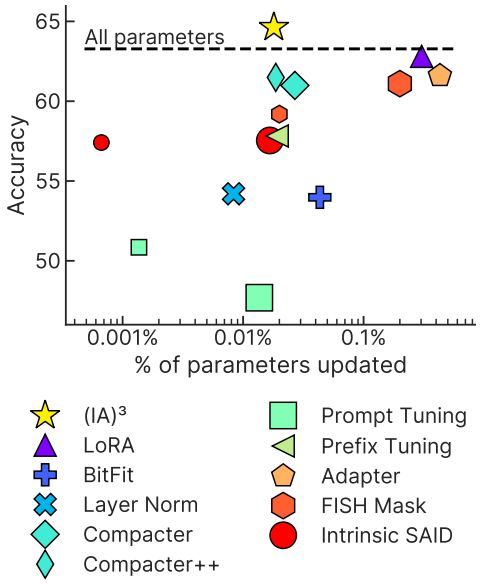


Figure 3.6: Accuracy of PEFT methods applied to T0-3B (Sanh et al., 2022).

3.2.3 Prefix Tuning

Prefix tuning is another PEFT method. But different from others, it has its roots in prompting. Therefore, we will discuss it in details in Chapter 4. In Assignment 2, you will be asked to show how prefix tuning as well as LoRA connects to adapter tuning.

The performance of various PEFT methods are summarized in Fig. 3.6 (Liu et al., 2022). We could not cover all the methods in this course. Feel free to read their papers if you are interested.

Chapter 4

Prompting and Zero-shot inference

4.1 What is prompting

In a traditional supervised learning setting, an output $\boldsymbol{y}$ is generated given the input $\boldsymbol{x}$ and the model parameters θ using the modeling objective $P(y|x;\theta)$. However, for many tasks, the supervised data is unavailable making the training process difficult/impossible. Prompting enables models to circumvent the training issue by learning an LM that models the probability $P(\boldsymbol{x};\theta)$ of text $\boldsymbol{x}$ itself, and using this probability to predict $\boldsymbol{y}$, reducing or obviating the need for large supervised datasets. In other words, prompting is non-invasive: it does not introduce large amounts of additional parameters or require direct inspection of a model's representations. It can be thought of as a lower bound on what the model "knows" and can be used to extract information from LM.

4.1.1 How to prompt?

To prompt an LM, it is important to map the input $\boldsymbol{x}$ to a prompt $\boldsymbol{x}'$. A prompting function $f_{\text{prompt}}(\cdot)$ is applied to modify the input text $\boldsymbol{x}$ into a prompt $\boldsymbol{x}' = f_{\text{prompt}}(\boldsymbol{x})$ (example provided in Table 4.1). Next, a template is defined that consists of two slots: an input slot `[X]` for input $\boldsymbol{x}$ and an answer slot `[Z]` for any generated answer $\boldsymbol{z}$ that may or may not be mapped into $\boldsymbol{y}$ depending on the use case. Next, the highest-scoring $\hat{\boldsymbol{z}}$ that maximizes the score of the LM is taken ($\boldsymbol{z}$ can take all possible values from the vocabulary for generation tasks but can also be modified for controlled

Name	Notation	Example	Description
Input	$\boldsymbol{x}$	I love this movie.	One or multiple texts
Output	$\boldsymbol{y}$	++ (very positive)	Output label or text
Prompting Function	$f_{\text{prompt}}(\boldsymbol{x})$	`[X]` Overall, it was a `[Z]` movie.	A function that converts the input into a specific form by inserting the input $\boldsymbol{x}$ and adding a slot `[Z]` where answer $\boldsymbol{z}$ may be filled later.
Prompt	$\boldsymbol{x}'$	I love this movie. Overall, it was a `[Z]` movie.	A text where `[X]` is instantiated by input $\boldsymbol{x}$ but answer slot `[Z]` is not.
Filled Prompt	$f_{\text{fill}}(\boldsymbol{x}', \boldsymbol{z})$	I love this movie. Overall, it was a bad movie.	A prompt where slot `[Z]` is filled with any answer.
Answered Prompt	$f_{\text{fill}}(\boldsymbol{x}', \boldsymbol{z}^*)$	I love this movie. Overall, it was a good movie.	A prompt where slot `[Z]` is filled with a true answer.
Answer	$\boldsymbol{z}$	"good", "fantastic", "boring"	A token, phrase, or sentence that fills `[Z]`

Table 4.1: Terminology and notation of prompting methods. $\boldsymbol{z}^*$ represents answers that correspond to true output $\boldsymbol{y}^*$. The table is taken from Liu et al. (2023).

classification/generation tasks).

A function $f_{\text{fill}}(\boldsymbol{x}', \boldsymbol{z})$ fills in the location `[Z]` in prompt $\boldsymbol{x}'$ with the potential answer $\boldsymbol{z}$ by searching over the set of all potential answers by calculating the probability of their corresponding filled prompts using a pre-trained LM $P(\cdot; \theta)$.

$$\hat{\boldsymbol{z}} = \underset{\boldsymbol{z} \in \mathcal{Z}}{\text{search}} \, P(f_{\text{fill}}(\boldsymbol{x}', \boldsymbol{z}); \theta). \tag{4.1}$$

This search function could be an *argmax* search that searches for the highest-scoring output or various *sampling* techniques that randomly generate outputs following the probability distribution of the LM.

4.2 Prompt Engineering

Given a task, multiple prompts can work. However, finding the most effective prompts is desired in order to unlock the full potential of the LM. *Prompt Engineering* is the process of designing a prompting function $f_{\text{prompt}}(\boldsymbol{x})$ that results in the most effective performance for the given task.

4.2.1 Manual Prompts

The most straightforward yet somewhat effective technique is to design the prompts for a given task manually. Since the number of required prompts is usually very less (1 to 8 depending on the task and the input sequence memory), designing manual prompts gives more control and flexibility to the user.

4.2.2 Automated Prompts

While the strategy of manually creating prompts is intuitive and allows for solving various tasks with some degree of accuracy, there are several problems with this approach: creating and experimenting with these prompts is an art that takes time and experience, especially for more complicated tasks like multi-step reasoning, and even experienced prompt designers may fail to find optimal prompts manually (Jiang et al., 2020). Searching for automated prompts can be further divided into *discrete prompts* or *continuous prompts*.

Discrete prompts

Discrete prompts (also sometimes known as *hard prompts*), as the name suggests, automatically search for prompts in a discrete space, i.e. the prompts are usually text strings corresponding to natural language. Various methods have been proposed in this line of work and we explore a few approaches here.

Jiang et al. (2020) introduced a *mining-based* approach that automatically identifies templates from a given set of training inputs $\boldsymbol{x}$ and outputs $\boldsymbol{y}$. This approach searches for strings in a large text corpus, such as Wikipedia, that contain both $\boldsymbol{x}$ and $\boldsymbol{y}$, and identifies the middle words or dependency paths between them that can be used as templates, e.g. in the form of "`[X]` middle words `[Z]`".

Paraphrasing methods involve taking an original prompt and creating various alternative prompts to use as training data. These methods include translating the prompt into another language and back (Jiang et al., 2020), using a thesaurus to replace words (Yuan et al., 2021), or using a neural prompt rewriter designed to improve the accuracy of systems that use the prompt (Haviv et al., 2021).

Treating prompts as generation tasks is an obvious choice and Gao et al. (2021) used the pre-trained T5 model for the template search process. Ben-David et al. (2021) further extended it and proposed a domain adaptation algorithm that trains T5 to generate unique domain-relevant features.

Other popular approaches involve *gradient-based* search that finds sequences that trigger the LM to generate desired output and use that as prompts (Wallace et al., 2019). AUTO PROMPT (Shin et al., 2020) creates a prompt based on a template that combines the original input $\boldsymbol{x}$ with trigger tokens (trigger tokens use the gradient-based method as shown in (Wallace et al., 2019)) and the LM predictions for the prompt is then converted to class probabilities by marginalizing over a set of associated label tokens.

Continuous prompts

Prompt construction aims to enable an LM to perform a specific task effectively, and it is not imperative for the prompt to be limited to natural language that can be interpreted by humans. Therefore, some approaches focus on *continuous prompts* (also sometimes known as *soft prompts*) that prompt the model directly in its embedding space.

One such approach is *Prefix Tuning* (Li and Liang, 2021) that prepends a sequence of continuous task-specific vectors to the input, while keeping the LM parameters frozen. Mathematically, this consists of optimizing over the following log-likelihood objective given a trainable prefix matrix M_ϕ and a fixed pre-trained LM parameterized by θ.

$$\max_{\phi} \log P(\boldsymbol{y}|\boldsymbol{x};\theta;\phi) = \max_{\phi} \sum_{y_i} \log P(y_i|h_{<i};\theta;\phi) \tag{4.2}$$

In Eq. 4.2, $h_{<i} = [h_{<i}^{(1)};\cdots;h_{<i}^{(n)}]$ is the concatenation of all neural network layers at time step i. It is copied from M_ϕ directly if the corresponding time step is within the prefix (h_i is $M_\phi[i]$), otherwise it is computed using the pre-trained LM.

However, it will be more interesting to combine the discrete prompts with continuous approaches as prefix tuning might be very sensitive to the changes. Zhong et al. (2021) takes the advantage of the hybrid approach by first defining a template using AUTOPROMPT (Shin et al., 2020)'s (a discrete search method), initialize virtual tokens based on this discovered prompt, then fine-tune the embeddings to increase the performance. Similarly, Liu et al. (2021b) propose "P-tuning", where continuous prompts are learned by inserting trainable variables into the embedded input. Han et al. (2021) further propose prompt tuning with rules (PTR) where logic rules create the templates alongside the virtual tokens whose embedding comes from the pre-trained LM.

4.3 Zero- and Few-Shot Inference

In many cases, prompting methods can be used without *any* explicit training of the language model (LM) for the downstream task. Instead, one can take an LM that has been trained to predict the probability of text $P(\boldsymbol{x})$ and apply it as-is to fill the cloze or prefix prompts that define the task. This is traditionally referred to as the *zero-shot* setting, as there is no training data available for the task of interest.

However, in the scenario where a limited number of labeled examples are available or can be easily annotated, it is possible to augment the prompt to include this additional information. This setting is referred to as *few-shot* inference and consists of including a small collection of input-out exemplars. These exemplars are input-output pairs that serve as demonstrations of the behavior that one would like the LM to emulate. By using these pairs as a guide, the large LMs can learn to carry out the desired task. This simple idea was shown to For example, we can augment the standard prompt "France's capital is `[X]` ." by prepending a few examples such as "Great Britain's capital is London . Germany's capital is Berlin . France's capital is `[X]`". Note that few-shot inference does not involve any parameter updates.

The introduction of GPT-3 (Brown et al., 2020) popularized this approach: the model was shown to benefit from the inclusion of exemplars in the prompt on a wide range of tasks and across different model sizes. An example of this behavior is displayed in Figure 4.1.

Figure 4.1: Results of GPT-3 on the TriviaQA dataset, using different numbers of exemplars in the few-shot prompt. From Brown et al. (2020).

Although the idea of including a small number of examples in the prompt is simple, there are some aspects that we need to pay attention to: (1) what examples should be included in the prompt to make the demonstration effective, and (2) how to order the examples in the prompt. Regarding the example selection, researchers have shown that different demonstrations can result in very different performance (Lu et al., 2021). An approach to tackle this issue consists of using sentence embeddings to sample examples that are close to the input in the embedding space (Liu et al., 2021a; Drori et al., 2022). As for the order of the labeled examples, methods were proposed to score different candidate permutations (Lu et al., 2021).

4.3.1 In-Context Learning

In-context learning is an *emergent behavior* displayed by large LMs that can lead to the models performing previously unseen tasks in a few-shot setting, without any parameter optimization.

Initially pointed out by Brown et al. (2020) in the presentation of the GPT-3 model, this behavior has been the subject of several studies and surveys. Notably, the phenomenon occurs suddenly as the model size increases, leading to the definition of emergent behavior. While including a few exemplars in the prompt does not lead to a significant difference in performance in small-scale models, it leads to a significantly improved behavior in large models (e.g., GPT-3, PaLM). Another aspect that makes this behavior surprising is the mismatch between the LM's training objective and the tasks that the model is able to learn in-context. Typically, large LMs are trained with a self-supervised language modeling objective, and the mechanisms that allow these models to learn a previously unseen task (e.g., text classification) without any parameter optimization are currently an active area of research.

Recent work has hypothesized that in-context learning by few-shot prompting works as a process of meta-optimization on the few examples included in the prompt (Dai et al., 2022). The authors show experimental results that support the idea that, when performing in-context learning, the models produces meta-gradients according to the demonstration exemplars through froward computation. Then these meta gradients are applied to the original LM through attention. This way, in-context learning can be seen as a kind of implicit fine-tuning on the few-shot exemplars.

A popular prompting strategy aimed at eliciting in-context learning is *chain-of-tought* prompting.

4.3.2 Chain of Thought Prompting

Multiple studies investigating the reasoning capabilities of large LMs, highlighted how eliciting the model to produce a step-by-step solution of a problem can lead to a more accurate final answer (Nye et al., 2021). Combining this idea with the intuition of including a set of demonstrations in the prompt, Wei et al. (2022c) introduced the concept of chain of thought (CoT) prompting. Given a question, a *chain of thought* is a coherent sequence of reasoning steps that leads to a final answer. Figure 4.2 shows an example of a math world problem that the model is able to solve through CoT prompting, and that would otherwise be gotten incorrect. The step-by-step reasoning displayed by LM through this prompting technique was subsequently show to be elicited in a zero-shot setting (Kojima et al., 2022). This was achieved simply by adding "Let's think step-by-step" to the prompt before the model's answer.

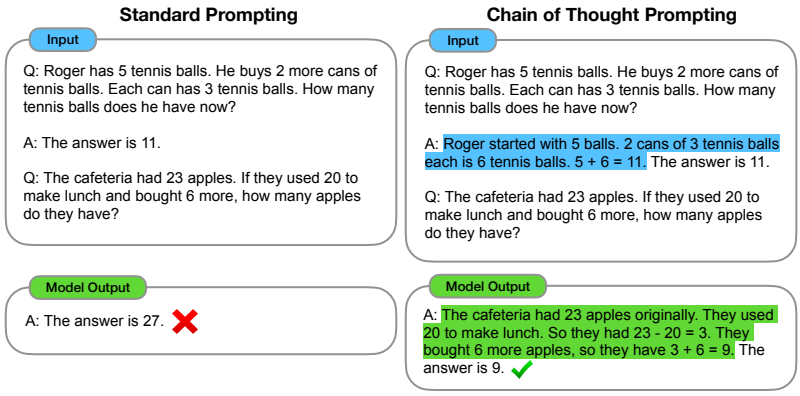


Figure 4.2: Example of chain of thought prompting, from Wei et al. (2022c).

Chain-of-thought prompting has been shown to significantly improve the ability of large language models to perform complex reasoning. In particular, through experiments on arithmetic and commonsense reasoning, it has been found that the ability of generating chains of thought is an *emergent* property of model scale. This means that larger models are better at generating step-by-step reasoning chains than smaller models. This phenomenon is noticeable in the scores reported in Figure 4.3. The results display a significant discrepancy between the solve rate obtained with chain of thought and with standard prompting on the GSM8k dataset.

However, this difference in performance does not seems to be as large for SVAMP and MAWPS. This is because GSM8k consists of complex problems that require multiple reasoning steps, while the other two datasets are made up of simpler problems that, for the most part, only require the application of a single mathematical operator to obtain the final result. This highlights how CoT prompting is most effective in eliciting in-context learning on multi-step reasoning problems.

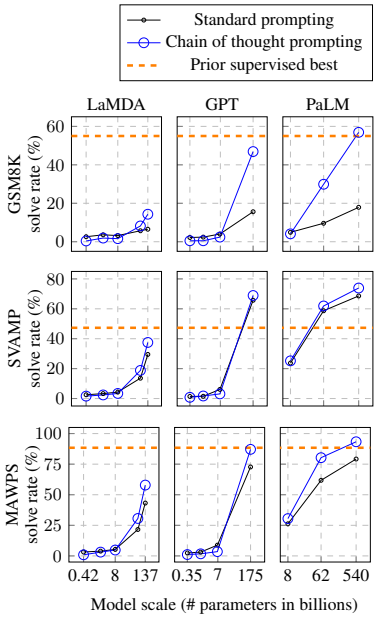


Figure 4.3: Performance of three large LMs on three different math world problem datasets, from Wei et al. (2022c).

Chapter 5

Multimodality

In this chapter we describe how language models deal with multimodality. By multimodality we mean information which is outside of the main input text. Prototypically, the extra modality is visual. For this, consider the case of an image captioning model. It is a language model which also has a module that is able to process an image and generate text based on it. However, there are also other types of multimodalities, which we cover in the subsequent section. These include, for example, knowledge-graphs or knowledge-bases in general and we discuss how they can be used to enhance the model with extra knowledge.

5.1 Vision Language Models

While text-only pre-trained models achieves great success on various NLP tasks, we live in a multimodal world and human brains naturally learn to process multi-sense signals received from the environment to help us make sense of the world around us (Gan et al., 2022). Among multiple modalities, vision and language are two main channels with which humans perceive and communicate, respectively. Therefore, Vision-and-Language (VL) has been a popular research area that sits at the nexus of Computer Vision and Natural Language Processing (NLP), aiming to develop algorithms that endow computers with an ability to effectively learn from multimodal data. Inspired by the great success of LLM pre-training in NLP, which is described in previous chapters, Vision-Language Pre-training (VLP) has attracted rapidly growing attention from both CV and NLP communities and become the main paradigm for modern VL research.

In this section, we first introduce important vision and language tasks in

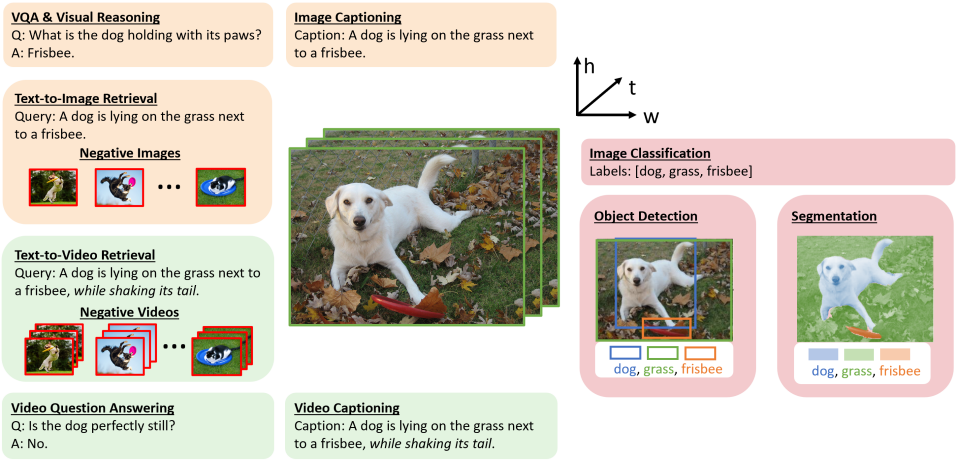


Figure 5.1: Illustration of representative tasks from three categories of VL problems covered in this paper: image-text tasks, vision tasks as VL problems, and video-text tasks.

section 5.1.1 and then present popular model architectures and pre-training objectives for vision-language models (VLMs) in section **??** and section **??**, respectively.

5.1.1 Vision-and-Language Tasks

Vision-and-language tasks, by definition, should include both vision and language modalities in their inputs and outputs. According to Gan et al. (2022), VL tasks can be grouped into three categories: **Image-Text Tasks**, **CV Tasks as VL Tasks**, and **Video-Text Tasks**.

- **Image-Text Tasks.** Image-text tasks are tasks that include images and texts in their inputs and outputs. Image-text tasks are arguably the most important and well studied tasks in VL research. We introduce the most representative image-text tasks below:
  - **VQA and visual reasoning.** Visual Question Answering (VQA) (Antol et al., 2015) is a task where an AI system is given an image and a natural language question about the image, and it must provide an answer to the question in natural language. The answers required in these these tasks can be open-ended free-form texts, or selected from multiple choices. The goal of VQA is to build a system that can understand both

visual and textual information and combine them to answer questions about the content of images. As illustrated in Figure 5.1, given an image of a dog playing frisbee, a VQA system might be asked "What is the dog holding with its paws?" and the answer should be "frisbee."

As extensions to visual question answering, researchers also developed datasets for visual reasoning (Hudson and Manning, 2019; Suhr et al., 2019), visual commonsense reasoning (Zellers et al., 2019), visual dialog (Das et al., 2017), knowledge-based VQA (Marino et al., 2019), scene-text-based VQA (Singh et al., 2019), *etc.*

- **Image captioning.** Image captioning (Vinyals et al., 2015) involves generating a descriptive and semantically meaningful caption for a given image. The goal of image captioning is to understand the visual content of an image and translate it into natural language text. This task involves several sub-tasks, including object recognition, scene understanding, and natural language generation. For example, as illustrated in Figure 5.1, an image captioning system will produce a caption of "A dog is lying on the grass next to a frisbee" given the image.

  In addition to the traditional setting where short single-sentence generation is required (Lin et al., 2014), researchers have also developed datasets for image paragraph captioning (Krause et al., 2017), scene-text-based image captioning (Sidorov et al., 2020), visual storytelling (Huang et al., 2016), and so on.

- **Image-text retrieval.** Image-text retrieval models are required to retrieve the most relevant texts (or images) from a large corpus, given an image (or text) query. Popular image-text retrieval datasets are based on image captioning datasets (Chen et al., 2015; Plummer et al., 2015).

- **Visual grounding.** In the visual grounding (Yu et al., 2016; Plummer et al., 2015) task, the model receives an image, a referring expression, and bounding boxes of objects and concepts in the image. The model then need to predict the bounding box corresponding to the input text query. This task requires the model to understand the relationship between language and vision, and accurately locate and identify the objects and concepts in an image that are referred to in a given language input.

- **Text-to-image generation.** It can be considered as the dual task of image captioning, where the system is required to create a high-fidelity image based on the text input. Most text-to-image generation models

are trained on image captioning datasets (Chen et al., 2015; Plummer et al., 2015).

- **Computer Vision Tasks as VL Problems.** Traditionally, core visual recognition tasks such as image classification, object detection, and segmentation (highlighted with pink in Figure 5.1) are considered as pure vision problems. With the advent of CLIP (Radford et al., 2021) and ALIGN (Jia et al., 2021), researchers have realized that language supervision can play an important role in computer vision tasks. First, the use of noisy image-text data crawled from web allows large-scale pre-training of vision encoders from scratch. Second, instead of treating the supervision signals (*e.g.*, class labels) as one-hot vectors, we take the semantic meaning behind the labels into consideration and cast these computer vision tasks as VL problems. This perspective generalizes the traditional close-set classification or detection models to recognizing unseen concepts in real-world applications, such as open-vocabulary object detection.

- **Video-Text Tasks.** Besides static images, videos are another important type of visual modality. Naturally, all aforementioned image-text tasks have their video-text counterparts, such as video captioning, retrieval, and question answering (highlighted with green in Figure 5.1). The uniqueness of video inputs, in comparison to images, requires an AI system to not only capture spatial information within a single video frame, but also capture the inherent temporal dependencies among video frames.

5.1.2 Vision-Language Models: Architectures

To effective solve the aforementioned VL problems, researchers in the VL community follow the success of LLM pre-training in NLP and pre-trained a number of VLMs that can be effectively fine-tuned on different VL tasks. In this section we describe

Overview

A vision-language model typically consist of a *text encoder*, a *vision encoder*, a fusion module, and (optionally) a decoder.

Given an image-text pair, a VL model first extracts text features $w = \{w_1, \cdots, w_N\}$ and visual features $v = \{v_1, \cdots, v_M\}$ via a text encoder and a vision encoder, respectively. Here, N is the number of tokens in a sentence, and M is the number of visual features for an image, which can be the number of image regions/grids/patches, depending on the specific vision encoder being used.

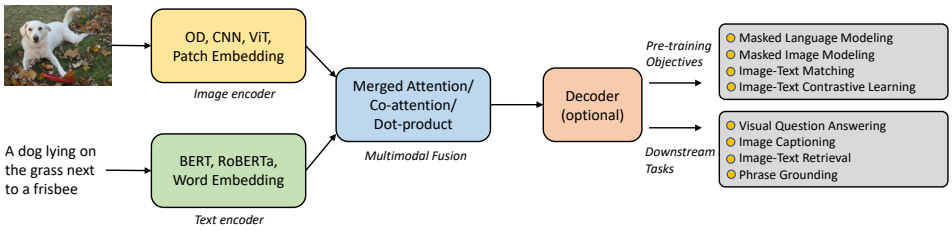


Figure 5.2: Illustration of a general framework for Transformer-based vision-language models.

The text and visual features are then fed into a *multimodal fusion module* to produce cross-modal representations, which are then optionally fed into a *decoder* before generating the final outputs. An illustration of this general framework is shown in Figure 5.2.

In many cases, there are no clear boundaries among image/text backbones, multimodal fusion module, and the decoder. In this paper, we refer to the part of the model that only takes image/text features as input as the corresponding *vision/text encoder*, and the part of the model that takes both image and text features as input as the *multimodal fusion module*. Besides this, if there are additional modules that take the multimodal features as input to generate the output, we call it *decoder*. We next describe different kinds of vision encoder, text encoder, and fusion module in detail.

Vision Encoder. There are three types of vision encoders: (*i*) an object detector (OD), (*ii*) a plain CNN, and (*iii*) a vision Transformer.

- **OD.** The most widely used object detector for VL research is the Faster R-CNN (Ren et al., 2015) pre-trained on the Visual Genome (VG) dataset (Krishna et al., 2017) as in BUTD (Anderson et al., 2018). In VinVL (Zhang et al., 2021), a stronger OD model based on the ResNeXt-152 C4 architecture is pre-trained on multiple public OD datasets (including COCO (Chen et al., 2015), OpenImages (Kuznetsova et al., 2020), Objects365 (Shao et al., 2019) and VG), and significant performance boost is observed across a wide range of VL tasks by using this stronger OD model. Additional care is taken to encode the location information of image regions, which is typically represented as a 7-dimensional vector.[1] Both visual and location features are then fed through a fully-connected layer, to be projected

[1] $[x_1, y_1, x_2, y_2, w, h, w*h]$ (normalized top/left/bottom/right coordinates, width, height, and area)

into the same embedding space. The final embedding for each region is obtained by summing up the two FC outputs and then passing through a layer normalization layer.

- **CNN.** In PixelBERT (Huang et al., 2020) and SOHO (Huang et al., 2021), ResNet-50, ResNet-101 and ResNeXt-152 pre-trained from ImageNet classification are adopted. In CLIP-ViL (Shen et al., 2022), ResNet-50, ResNet-101, and ResNet-50x4 pre-trained from CLIP (Radford et al., 2021) are used. In SimVLM (Wang et al., 2022b), they use the first three blocks (excluding the Conv stem) of ResNet-101 and ResNet-152 for their base and large models, respectively, and a larger variant of ResNet-152 with more channels for the huge model. Typically, it is observed that a stronger CNN backbone results in stronger downstream performance.

- **ViT.** Some recent pre-trained VLMs such as ALBEF (Li et al., 2021) and ViLT (Kim et al., 2021) use Transformer-based vision encoders. Following Dosovitskiy et al. (2021), an image is first split into image patches, which are then flattened into vectors and linearly projected to obtain patch embeddings. A learnable special token `[CLS]` embedding is also prepended to the sequence. These patch embeddings, when summed up together with learnable 1D position embeddings and a potential image-type embedding, are sent into a multi-layer Transformer block to obtain the final output image features. Different ViT variants have been studied for VLP, such as plain ViT (Dosovitskiy et al., 2021), DeiT (Touvron et al., 2021), BEiT (Bao et al., 2022a), Swin Transformer (Liu et al., 2021c), and CLIP-ViT (Radford et al., 2021), to name a few.

In a nutshell, no matter what vision encoder is used, the input image is represented as a set of feature vectors $\mathbf{v} = \{\mathbf{v}_1, \cdots, \mathbf{v}_M\}$. VLMs can be categorized into non end-to-end models, which use an OD model to get vision features for the model, and end-to-end models that directly take raw images as input.

Text Encoder. Following BERT (Devlin et al., 2019) and RoBERTa (Liu et al., 2019), VLMs first segment the input sentence into a sequence of subwords and then insert two special tokens at the beginning and the end of the sentence to generate the input text sequence. After we obtain the text embeddings, existing works either feed them directly to the multimodal fusion module (Li et al., 2019; Chen et al., 2020), or to several text-specific layers (Tan and Bansal, 2019; Lu et al., 2019) before the fusion. For the former, the fusion module is typically initialized with BERT, and the role of

text encoding and multimodal fusion is therefore entangled and absorbed in a single BERT model, and in this case, we consider text encoder as the word embedding layer.

In a nutshell, no matter what text encoder is used, the input text is represented as a set of feature vectors $w = \{w_1, \cdots, w_N\}$.

Multimodal Fusion. For *dual encoders* like CLIP (Radford et al., 2021) and ALIGN (Jia et al., 2021), fusion is essentially computing the similarity between representation in the two modalities, which is typically performed via a dot-product between two global image and text feature vectors. For *fusion encoder*, it takes both $v = \{v_1, \cdots, v_M\}$ and $w = \{w_1, \cdots, w_N\}$ as input, and learns contextualized multimodal representations denoted as $\tilde{v} = \{\tilde{v}_1, \cdots, \tilde{v}_M\}$ and $\tilde{w} = \{\tilde{w}_1, \cdots, \tilde{w}_N\}$. There are mainly two types of fusion modules, namely, *merged attention* and *co-attention*, shown in Figure 5.3. Specifically,

- In a **merged attention** module, the text and visual features are simply concatenated together, and then fed into a single Transformer block. This design has been used in many previous works, such as VisualBERT (Li et al., 2019), Unicoder-VL (Li et al., 2020a), VL-BERT (Su et al., 2019), UNITER (Chen et al., 2020), OSCAR (Li et al., 2020b), VinVL (Zhang et al., 2021), ViLT (Kim et al., 2021), GIT (Wang et al., 2022a), *etc.*
- In a **co-attention** module, on the other hand, the text and visual features are fed into different Transformer blocks independently, and techniques such as cross-attention are used to enable cross-modal interaction. This design has been used in LXMERT (Tan and Bansal, 2019), ViLBERT (Lu et al., 2019), ERNIE-ViL (Yu et al., 2021), METER (Dou et al., 2022), *etc.* Also, in many models, only image-to-text cross-attention modules are used, such as ALBEF (Li et al., 2021), BLIP (Li et al., 2022), CoCa (Yu et al., 2022), and Flamingo (Alayrac et al., 2022). Most ViT-based models adopts co-attention module since the image sequence can be very long and doing merged attention can be very computationally inefficient.

5.1.3 Vision-Language Models: Pre-training Objectives

We then introduce the pre-training tasks used in VLP. We will first introduce masked language modeling and image-text matching, which have been used extensively in almost every VLP model. Then, we will also describe image-text contrastive learning and masked image modeling tasks which are widely used in recent VLMs.

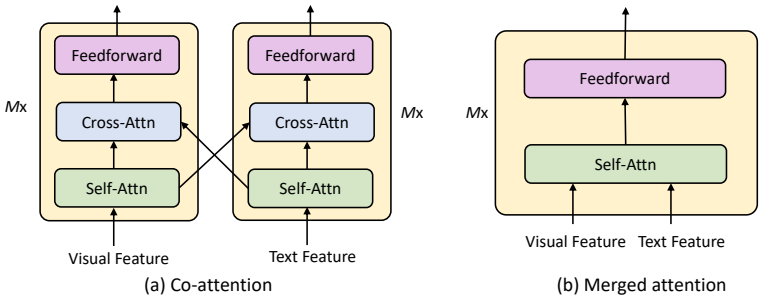


Figure 5.3: Co-attention and merged attention design for multimodal fusion.

Masked Language Modeling (MLM). The MLM objective is first introduced in language model pre-training (*e.g.,* Devlin et al., 2019; Liu et al., 2019). In VLP, MLM with image-text pairs has also proven to be useful. In MLM, given an image-text pair, we randomly mask out the input words with probability of 15%, and replace the masked ones $\tilde{\mathbf{w}}_{\mathbf{m}}$ with special token `[MASK]`.[2] The goal is to predict these masked tokens based on their surrounding words $\tilde{\mathbf{w}}_{\backslash \mathbf{m}}$ and the paired image $\tilde{\mathbf{v}}$, by minimizing the negative log-likelihood:

$$\mathcal{L}_{\text{MLM}}(\theta) = -\mathbb{E}_{(\tilde{\mathbf{w}},\tilde{\mathbf{v}})\sim D} \log P_\theta(\tilde{\mathbf{w}}_{\mathbf{m}}|\tilde{\mathbf{w}}_{\backslash \mathbf{m}}, \tilde{\mathbf{v}}) \,, \tag{5.1}$$

where θ denotes the trainable parameters. Each pair $(\tilde{\mathbf{w}}, \tilde{\mathbf{v}})$ is sampled from the whole training set D. There are several MLM variants used in VLP. Specifically,

- **Seq-MLM**: In order to adapt the pre-trained model for image captioning, it is observed (Zhou et al., 2020; Wang et al., 2021a) that adding a seq2seq *causal mask* during pre-training is beneficial. That is, in Seq-MLM, the model can only use its preceding context to predict the masked token, which is consistent to the way the model performs image captioning during inference.

- **LM**: Direct language modeling is used in BLIP (Li et al., 2022) and CoCa (Yu et al., 2022) for VLP. The model predicts the caption given an image token-by-token autoregressively.

- **Prefix-LM**: Using the encoder-decoder framework as in SimVLM (Wang et al., 2022b) and DaVinCi (Diao et al., 2023), a PrefixLM pre-training

[2]Following BERT, this 15% is typically decomposed into 10% random words, 10% unchanged, and 80% `[MASK]`.

objective is proposed, where a sentence is first split into two parts, and the bi-directional attention is enabled on the prefix sequence and the input image, while a causal attention mask is adopted on the remaining tokens.

Image-Text Matching (ITM). In ITM, given a batch of matched or mismatched image-caption pairs, the model needs to identify which images and captions correspond to each other. Most VLP models treat image-text matching as a binary classification problem. Specifically, a special token (*i.e.*, `[CLS]`) is appended at the beginning of the input sentence to learn a global cross-modal representation. We then feed the model with either a matched or mismatched image-caption pair $\langle \tilde{\mathbf{v}}, \tilde{\mathbf{w}} \rangle$ with equal probability, and a classifier is added on top of the `[CLS]` token to predict a binary label y, indicating whether the sampled image-caption pair is matched. Specifically, denote the output score as $s_\theta(\tilde{\mathbf{w}}, \tilde{\mathbf{v}})$, We apply the binary cross-entropy loss for optimization:

$$\mathcal{L}_{\text{ITM}}(\theta) = -\mathbb{E}_{(\tilde{\mathbf{w}},\tilde{\mathbf{v}})\sim D}[y \log s_\theta(\tilde{\mathbf{w}}, \tilde{\mathbf{v}}) + (1-y)\log(1 - s_\theta(\tilde{\mathbf{w}}, \tilde{\mathbf{v}}))]) \,. \quad (5.2)$$

Besides randomly sampling a negative image-text pair, harder negative pairs can also be mined from an image-text contrastive loss introduced below, which has been shown to be effective in improving the downstream performance, as reported in ALBEF (Li et al., 2021), VLMo (Wang et al., 2021b), and X-VLM (Zeng et al., 2022).

Image-Text Contrastive Learning (ITC). Early VLP models, such as UNITER (Chen et al., 2020) and VinVL (Zhang et al., 2021), do not use ITC for pre-training. Though the ITC loss is widely studied before VLP, in the context of end-to-end VLP, it is mostly popularized by CLIP (Radford et al., 2021) and ALIGN (Jia et al., 2021) to pre-train a dual encoder. Later on, it is also used to pre-train a fusion encoder as in ALBEF (Li et al., 2021). Note that this ITC loss is used on top of the outputs of image and text encoders, before multimodal fusion (*i.e.*, the use of w and v, instead of $\tilde{w}$ and $\tilde{v}$). Specifically, given a batch of N image-text pairs, ITC aims to predict the N matched pairs from all the N^2 possible image-text pairs. With a little bit abuse of notation, let $\{v_i\}_{i=1}^N$ and $\{w_i\}_{i=1}^N$ denote respectively the normalized image vectors and text vectors in a training batch. To compute

image-to-text and text-to-image similarities, we have:

$$s_{i,j}^{i2t} = v_i^\top w_j, \; s_{i,j}^{t2i} = w_i^\top v_j \,, \tag{5.3}$$

$$\mathcal{L}_{\text{ITC}}^{i2t}(\theta) = -\frac{1}{N}\sum_{i=1}^{N} \log \frac{\exp(s_{i,i}^{i2t}/\sigma)}{\sum_{j=1}^{N} \exp(s_{i,j}^{i2t}/\sigma)} \,, \tag{5.4}$$

$$\mathcal{L}_{\text{ITC}}^{t2i}(\theta) = -\frac{1}{N}\sum_{i=1}^{N} \log \frac{\exp(s_{i,i}^{t2i}/\sigma)}{\sum_{j=1}^{N} \exp(s_{i,j}^{t2i}/\sigma)} \,, \tag{5.5}$$

where σ is a learned temperature hyper-parameter, $\mathcal{L}_{\text{ITC}}^{i2t}$ and $\mathcal{L}_{\text{ITC}}^{t2i}$ are image-to-text and text-to-image contrastive loss, respectively.

Masked Image Modeling (MIM). Similar to the MLM objective, researchers have studied various masked image modeling (MIM) tasks for pre-training. Specifically, the model is trained to reconstruct the masked patches or regions $\tilde{\mathbf{v}}_{\mathbf{m}}$ given the remaining visible patches or regions $\tilde{\mathbf{v}}_{\backslash \mathbf{m}}$ and all the words $\tilde{\mathbf{w}}$ as

$$\mathcal{L}_{\text{MIM}}(\theta) = \mathbb{E}_{(\tilde{\mathbf{w}},\tilde{\mathbf{v}})\sim D} P_\theta(\tilde{\mathbf{v}}_{\mathbf{m}} | \tilde{\mathbf{v}}_{\backslash \mathbf{m}}, \tilde{\mathbf{w}}) \,. \tag{5.6}$$

The designs of MIM can be divided into two categories.

- **For OD-based VLP models**, *e.g.*, LXMERT (Tan and Bansal, 2019) and UNITER (Chen et al., 2020), some of the input regions are randomly masked (*i.e.*, the visual features of the masked regions are replaced by zeros), and the model is trained to regress the original region features via minimizing the mean squared error loss. Researchers (Tan and Bansal, 2019; Lu et al., 2019; Chen et al., 2020) have also tried to first generate object labels for each region using a pre-trained object detector, which can contain high-level semantic information, and the model is trained to predict the object labels for the masked regions instead of the original region features.

- **For end-to-end VLP models**, *e.g.*, ViLT (Kim et al., 2021) and METER (Dou et al., 2022), researchers have investigated the use of masked patch regression/classification for masked image modeling. Specifically,

  – For **MIM with discrete VQ tokens**, inspired by BEiT (Bao et al., 2022a), discrete VQ tokens are first extracted for the input patches, and the model is then trained to reconstruct the discrete tokens. Specifically, the VQ-VAE (van den Oord et al., 2017) model in DALL-E (Ramesh

et al., 2021) is first used to tokenize each image into a sequence of discrete tokens. Each image is resized so that the number of patches is equal to the number of tokens, and thus each patch corresponds to a discrete token. Then, we randomly mask 15% of the patches and feed the masked image patches to the model as before, but now the model is trained to predict the discrete tokens instead of the masked patches.

– For **MIM with in-batch negatives**, by imitating MLM which uses a text vocabulary, the model is trained to reconstruct input patches by using a dynamical vocabulary constructed with in-batch negatives. Concretely, at each training step, we sample a batch of image-caption pairs $\{\langle v^k, w^k \rangle\}_{k=1}^{B}$, where B is the batch size. We treat all the patches in $\{\mathbf{v}^k\}_{k=1}^{B}$ as candidate patches. For each masked patch, we mask 15% of the input patches. The model needs to select the original patch within this candidate set. The model is trained to maximize its probability similar to noise contrastive estimation (Gutmann and Hyvärinen, 2010).

Notably, recent state-of-the-art VLP models (*e.g.*, VinVL (Zhang et al., 2021), ALBEF (Li et al., 2021), VLMo (Wang et al., 2021b)) do not apply MIM during pre-training, and in ViLT (Kim et al., 2021) and METER (Dou et al., 2022), the authors also demonstrate that MIM is not helpful for downstream performance. However, there are also recent works that adopt masked vision-language modeling (as in MaskVLM (Kwon et al., 2022) and VL-BEiT (Bao et al., 2022b)), which try to randomly mask patches/tokens while keeping the other modality intact.

5.2 Knowledge-Enhancement

Language models are known to contain knowledge on their own (Petronic et al., 2019; Petroni et al., 2020) and we cover the extraction of this knowledge in a later chapter. Consider a question answering system based on a large language model. When asked about the current president, it is able to provide the correct answer. But on the day of inauguration, the true answer will be different while the model will keep the old one. Because the information about the current president is stored in the model's parameters, the only reliable way of updating this information in the model is to fine-tune it or re-train it from scratch. However, changing a small piece of information should not require these complicated and expensive operations. For this, we turn to knowledge-enhanced language models.

Models which contain information indirectly in their parameters are called **parametric** and their counterparts, models which rely on external

information, are called **non-parametric**. They do have parameters but most of the information is not stored there but in some more human-accessible form and location. The general principle is that during the inference, the model generates a key by which an external, interpretable and editable system is queried. Next, the result of this search is fused into the model and the inference continues. Continuing with the question answering example, the model may construct e.g. an SQL query for the current president in a given country. Upon receiving the answer, it is attended to in the inference and copied to the output. When the true answer changes e.g. on the inauguration day, it can easily be edited by a human or another automated system in the database and the large language model, which serves as question answerer, does not need to be changed at all. To summarize, the advantage of having an external source of information is ease of editability but also interpretability because we can later examine which pieces of information were retrieved and used in the inference process.

We now describe several models that utilize the knowledge retrieval approach.

5.2.1 kNN LM

The language model proposed by Khandelwal et al. (2019) utilizes memorization of the training data during inference in order to predict otherwise sparsely occurring words. In this approach, the authors first compute representations of all sentence prefixes (last self-attention layer) and store them as a mapping to the following word:

$$\texttt{Encoder}: \qquad k = \mathrm{LM}^{\mathrm{rep.}}(X_{<i}) \tag{5.7}$$

$$\texttt{Datastore}: \qquad \mathcal{B} = \{(\texttt{Encoder}(X_{<i}), X_i) | X \in D_{\mathrm{train}}, i < |X|\} \tag{5.8}$$

In autoregressive inference, the prefix is again computed for the current sentence and then 1024 neighbours in the vector space are retrieved:

$$\texttt{Retriever}: \qquad S = \{(r, v) | (r, v) \in \mathcal{N}^{L^2}_{1024}(k)\} \tag{5.9}$$

Then, their corresponding mapped words are combined together into a single distribution over the vocabulary:

$$\texttt{Aggregator}: \qquad p_\xi(\hat{X}_i) \propto \sum_{(r,v)\in S} \mathbb{1}_v \exp(-||r - k||_2 \cdot v) \tag{5.10}$$

Formally, this alone would create its own kNN-based language model p_ξ. However, its output is weighted to the main model's own prediction using

manually set hyperparameter $\lambda \in [0, 1]$. The higher it is, more weight is put to the retrieved prediction as opposed to the current model's prediction:

$$\texttt{Output:} \qquad \lambda \cdot p_\xi + (1 - \lambda) \cdot \mathrm{LM}(X_{<i}) \quad (5.11)$$

The symbolic working of the model is shown in the following set of equations and Fig. 5.4 which is adapted from the original paper. Even with preserving the same amount of trainable parameters, the perplexity on WikiText-103 of a baseline model (Baevski and Auli, 2018) was reduced from 18.7 to 15.8.

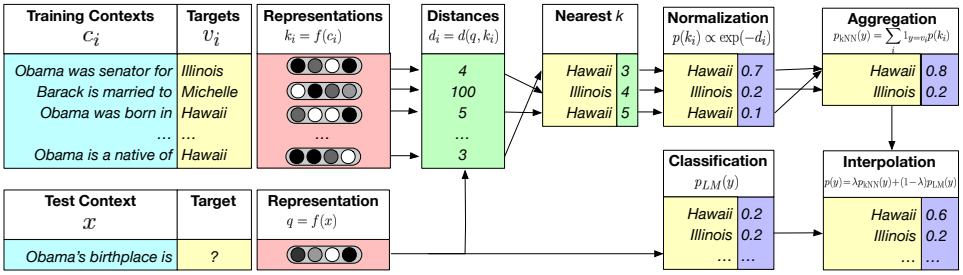


Figure 5.4: An illustration of kNN-LM. A datastore is constructed with an entry for each training set token, and an encoding of its leftward context. For inference, a test context is encoded, and the k most similar training contexts are retrieved from the datastore, along with the corresponding targets. A distribution over targets is computed based on the distance of the corresponding context from the test context. This distribution is then interpolated with the original model's output distribution. (Khandelwal et al., 2019)

5.2.2 Dynamic Gating kNN LM

The previously described language models use the same weighting mechanism across all words, i.e. both common words such as *the, a, one* and very low frequency words, where the retrieval from training data may be particularly useful (e.g. f(Barack's wife is called) $\rightarrow$ Michelle). Yogatama et al. (2021) propose an approach in which the model itself determines this parameter, now a vector, dynamically based on the current sentence prefix (e.g. lower weight to retrieval component for easy words). The datastore is constructed in the same way from the training data as in kNN LM. Their approach modifies the final Eq. (5.11) with a dynamic weight based on the current

hidden state:

$$\texttt{Dynamic weight:} \qquad g = \sigma(w^T \cdot \mathrm{LM}^{\mathrm{rep.}}(X_{<i})) \quad (5.12)$$

$$\texttt{Output:} \qquad (1-g) \odot p_\xi + g \odot \mathrm{LM}(X_{<i}) \quad (5.13)$$

The experimented model (Transformer-XL) was half the size of the original one, but again, on Wikipedia-103, it reduced the perplexity from 19.1 to 17.6

5.2.3 KnowBERT

KnowBERT (Peters et al., 2019) is one typical knowledge-enhanced language model that injecting factual knowledge into *hidden representations* of tokens. The idea is quite intuitive: if a language model can align mentions in the input text with entities in the knowledge base, we can say that factual knowledge is contained in this model. Note that in previous sections, we mentioned that in Transformers, text is first tokenized and vectorized as embeddings. Then, they are processed by Transformer modules. In KnowBERT, mentions are not aligned with entities before input into the model. Instead, they are both first vectorized, and then they are aligned in the format of hidden representations. Next, we illustrate its idea with more details.

Preparation (knowledge base and base language model). Knowledge bases contain rich factual knowledge. The knowledge in them is usually formatted as triples, i.e., (`subj`, `rel`, `obj`). `subj` and `obj` here are entities, which are usually associated with entity labels and descriptions. `rel` indicates the relationship between entities, with labels associated as well. In most cases, the knowledge is vectorized before usage. In KnowBERT, they compute entity embeddings from WordNet as the entity embeddings, where they claimed that relations, and synset definitions are encoded.

As most knowledge-enhanced language models, KnowBERT is not trained from scratch. The **backbone pretrainded language model** is BERT, where we can regard the knowledge enhancement process as secondary pretraining or special finetuning of BERT. The architecture is roughly the same to BERT, with specially designed context mechanism. Next, we introduce the *Knowledge Attention and Recontextualization* (as shown in Figure 5.5) component with details.

(1) Mention-span representations is calculated with below equation:

$$\mathbf{H}_i^{\mathtt{proj}} = \mathbf{H}_i \mathbf{W}_1^{\mathtt{proj}} + \mathbf{b}_1^{\mathtt{proj}}. \quad (5.14)$$

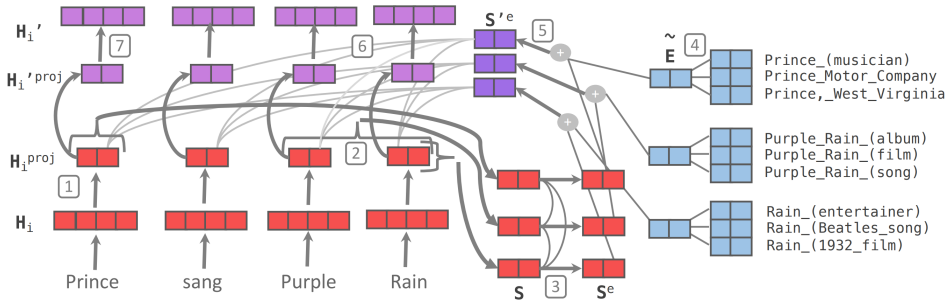


Figure 5.5: The knowledge integration component of KnowBERT (i.e., Knowledge Attention and Recontextualization). BERT word piece representations $\mathbf{H}_i$ are first projected to $\mathbf{H}_i^{\text{proj}}$ (1), then pooled over candidate mentions spans (2) to compute $\mathbf{S}$, and contextualized into $\mathbf{S}^e$ using mention-span self-attention (3). An integrated entity linker computes weighted average entity embeddings $\widetilde{\mathbf{E}}$ (4), which are used to enhance the span representations with knowledge from the knowledge base (5), computing $\mathbf{S}'^e$. Finally, the BERT word piece representations are recontextualized with word-to-entity-span attention (6) and projected back to the BERT dimension (7) resulting in $\mathbf{H}_i'$.

The KAR starts with the knowledge base entity candidate selector that provides a list of candidate mentions which it uses to compute C mention-span representations $\mathbf{s}_m \in \mathbb{R}^E$. $\mathbf{H}_i$ is first projected to the entity dimension with a linear projection, then, the KAR computes mention-span representations, one for each candidate mention, by pooling over all word pieces in a mention-span using the self-attentive span pooling. The mention-spans are stacked into a matrix $\mathbf{S} \in \mathbb{R}^{C \times E}$.

(2) Entity linker functions as its name. Here, it is responsible for performing entity disambiguation for each potential mention from available candidates. It first runs mention-span self-attention to compute embedding as

$$\mathbf{S}^e = \text{TransformerBlock}(\mathbf{S}). \tag{5.15}$$

The span self-attention is identical to the typical transformer layer, exception that the attention mechanism here is **cross-attention**, which is between mention-span vectors instead of word piece vectors. This allows KnowBERT to incorporate global information into each linking decision, so that it can take advantage of entity-entity cooccurrence and resolve which of several overlapping candidate mentions should be linked.

The objective function is based on the supervision signal of entity linking, which can be used to optimize parameters of the entity linker and other weight matrices. Specifically, $\mathbf{S}^e$ is used to score each of the candidate entities while incorporating the candidate entity prior from the knowledge base. Each candidate span m has an associated mention-span vector $\mathbf{s}_m^e \in \mathbf{S}^e$, M_m candidate entities with embeddings $\mathbf{e}_{mk}$ (from knowledge base), and prior probabilities p_{mk}. KnowBERT computes M_m scores using the prior and dot product between the entity span vectors and entity embeddings as

$$\psi_{mk} = \text{MLP}(p_{mk}, \mathbf{s}_m^e \cdot \mathbf{e}_{mk}). \tag{5.16}$$

And thus correspondingly, the loss function of entity linker can be written as

$$\mathcal{L}_{\text{EntityLinker}} = -\sum_m \log \left(\frac{\exp \psi_{mg}}{\sum_k \exp \psi_{mg}} \right) \tag{5.17}$$

(3-5) Knowledge enhanced entity-span representations. KnowBERT injects the knowledge base entity information into the mention-span representations computed from BERT vectors ($\mathbf{s}_m^e$) to form entity span representations. The weighted entity embedding can be then written as

$$\widetilde{\mathbf{e}}_m = \sum_k \psi_{mg} \mathbf{e}_m. \tag{5.18}$$

Then, the entity-span representations are updated with the weighted entity embeddings as

$$\mathbf{s}_m^{'e} = \mathbf{s}_m^e + \widetilde{\mathbf{e}}_m, \tag{5.19}$$

which can be packed into a matrix as $\mathbf{S}^{'e} \in \mathbb{R}^{C \times E}$.

(6-7) Recontextualization happens after updating the entity span representations with the weighted entity vectors. KnowBERT uses them to recontextualize the word piece representations. This is accomplished using a modified transformer layer that substitutes the multi-headed self-attention with a multi-headed attention between the projected word piece representations and knowledge enhanced entity span vectors. As introduced before, we have

$$\mathbf{H}_i^{'\text{proj}} = \text{MLP}(\text{MultiHeadAttn}(\mathbf{H}_i^{\text{proj}}, \mathbf{S}^{'e}, \mathbf{S}^{'e})). \tag{5.20}$$

Then, $\mathbf{H}_i^{\text{proj}}$ is projected back to the BERT dimension with a linear transformation and a residual connection added as:

$$\mathbf{H}_i^{'} = \mathbf{H}_i^{'\text{proj}} \mathbf{W}_2^{\text{proj}} + \mathbf{b}_2^{\text{proj}} + \mathbf{H}_i. \tag{5.21}$$

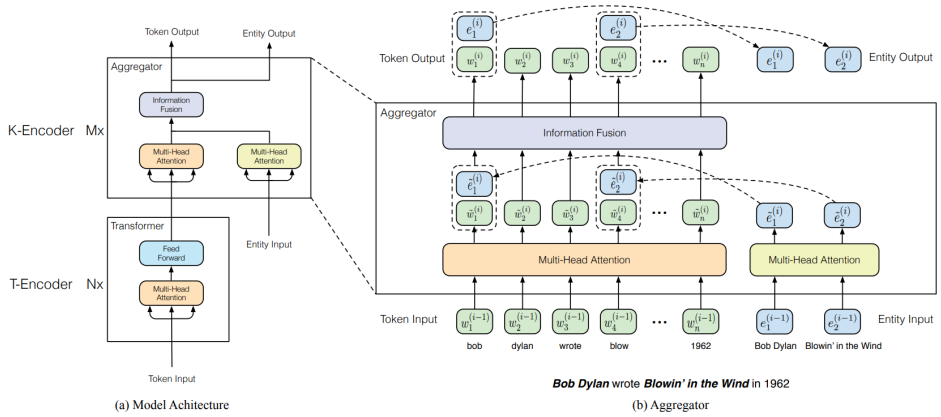


Figure 5.6: The left part is the architecture of ERNIE. The right part is the aggregator for the mutual integration of the input of tokens and entities. Information fusion layer takes two kinds of input: one is the token embedding, and the other one is the concatenation of the token embedding and entity embedding. After information fusion, it outputs new token embeddings and entity embeddings for the next layer

Training process. To avoid catastrophic forgetting, KnowBERT cannot be trained only using the entity linking supervision signal, the pretraining objectives of BERT should also be considered. Thus, simply speaking, the knowledge enhancement objective is defined as:

$$\mathcal{L}_{\text{KnowBERT}} = \mathcal{L}_{\text{BERT}} + \mathcal{L}_{\text{EntityLinker}}. \tag{5.22}$$

Note that there are many engineering tricks of the KnowBERT design. These tricks are commonly used in many knowledge-enhanced language models. In our following sections, we will leave out these tricks for simplicity.

5.2.4 [Optional] ERNIE

ERNIE (Zhang et al., 2019) was proposed prior to KnowBERT, which utilized part of the design mentioned above in the KnowBERT section. Therefore, to avoid repeating content, we briefly introduce the general method design of ERNIE, and mainly focus on the difference between ERNIE and KnowBERT.

Architecture. The knowledge enhancement design of ERNIE can be found in Figure 5.6. We can find that different from KnowBERT, the

whole model architecture of ERNIE consists of two stacked modules: (1) the underlying textual encoder (T-Encoder) responsible to capture basic lexical and syntactic information from the input tokens, and (2) the upper knowledgeable encoder (K-Encoder) responsible to integrate extra token-oriented knowledge information into textual information from the underlying layer, so that we can represent heterogeneous information of tokens and entities into a united feature space.

Objective. In order to inject knowledge into language representation by informative entities, ERNIE proposes a new pre-training task, which randomly masks some token-entity alignments and then requires the system to predict all corresponding entities based on aligned tokens. As the task is similar to training a denoising auto-encoder, it refers to this procedure as a denoising entity auto-encoder (dEA). Considering that the size of entities is quite large for the softmax layer, ERNIE thus only requires the system to predict entities based on the given entity sequence instead of all entities in knowledge base. Given the token sequence $\{w_1, ..., w_n\}$ and its corresponding entity sequence $\{e_1, ..., e_m\}$, the aligned entity distribution for the token w_i is defined as follows,

$$p(e_j|w_i) = \text{softmax}(\text{linear}(w_i^o) \cdot e_j). \tag{5.23}$$

The equation above will be used to compute the cross-entropy loss function for dEA.

Performance. After we know how KnowBERT and ERNIE inject knowledge, we show that they can perform good on knowledge-intensive tasks. Note that the knowledge enhancement can be regarded as the secondary pretraining. Thus, to compare the performance of BERT, ERNIE, and KnowBERT, similar to finetuning, we need first tune them on training data of the downstream task and then test them. As aforementioned, the knowledge can be regarded as triples, which contains entities and relations. Thus, the most intuitive way to evaluate these knowledge-enhanced language models is to see if they perform good on predicting knowledge.

We consider two typical knowledge-intensive downstream tasks for evaluation: relation extraction (Choi et al., 2018) and entity typing (Zhang et al., 2017). Results can be found in Table 5.1.

- Relation extraction: Models are given a sentence with marked a subject and object, and asked to predict which of several different relations (or no relation) holds.

Table 5.1: Performance of BERT, ERNIE, and KnowBERT on two knowledge-intensive tasks. P here is precision, R is recall.

Model	Relation extraction (Choi et al., 2018)			Entity typing (Zhang et al., 2017)		
	P	R	F1-Micro	P	R	F1-Micro
BERT	67.2	64.8	66.0	76.4	71.0	73.6
ERNIE	70.0	66.1	68.0	78.4	72.9	75.6
KnowBERT	71.6	71.4	71.5	78.6	73.7	76.1

- Entity typing: Given an entity mention and its context, entity typing requires systems to label the entity mention with its respective semantic types.

Chapter 6

Additional Topics

This chapter includes...

6.1 Instruction-Based Training Procedures

Shortly after the advent of transformer-based language models, the pretraining-finetuning paradigm was shown to be an effective strategy for the great majority of NLP tasks. The ability of the pretraining stage to boost the performance of these models on downstream tasks led to the hypothesis that language models are unsupervised multitask learners (Radford et al., 2019). This idea suggests that models pretrained with a language modeling objective on large corpora are able to perform better because, during pretraining, these models implicitly learn a latent structure of the language that is useful to subsequently learn more effectively a task involving text. While classic pretraining objectives leverage this ability of language models to implicitly learn information useful for multiple tasks, recent work proposed an explicit multi-task training paradigm that leverages the use of natural language instructions. We can identify two main approaches that follow this paradigm: instruction tuning and reinforcement learning from human feedback.

6.1.1 Instruction Tuning

Instruction tuning consists of finetuning language models on a collection of datasets described via instructions. The way these instructions are incorporated in the training procedure is simply by prepending to the model's input a short description of the task that needs to be carried out (e.g., *Is the sentiment of this movie review positive or negative?* or *Translate the*

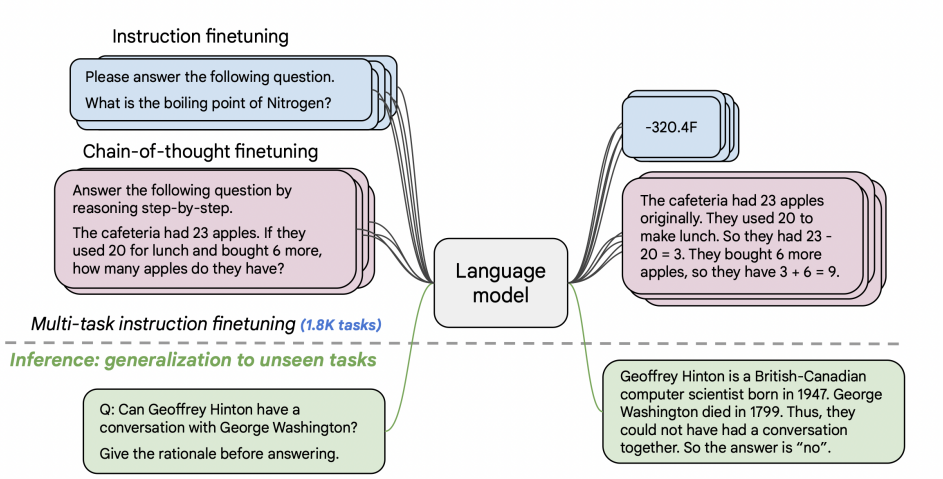


Figure 6.1: Example of instruction tuning, from Chung et al. (2022).

following sentence into Chinese). This approach combines aspects of both pretrain-finetune and prompting paradigms to improve language models' responses to inference-time text interactions. By finetuning a pretrained language model on a mixture of NLP datasets expressed via natural language instructions, the models were shown to perform better on previously unseen tasks (process illustrated in Figure 6.1).

An example of models trained using this procedure is the FLAN (Fine-tuned Language Net) family (Wei et al., 2022a). To evaluate the FLAN models' ability to perform a specific task T (e.g., natural language inference), the model is instruction-tuned on a range of other NLP tasks such as commonsense reasoning, translation, and sentiment analysis. As this setup ensures that the FLAN model has not seen task T in instruction tuning, the model is the evaluated on its ability to perform T in a zero-shot setting. FLAN models were shown to outperform larger LMs prompted with few shots (Figure 6.2). This approach was subsequently scaled to larger models (540B PaLM; Chowdhery et al. 2022) and a larger number of tasks (1836) (Chung et al., 2022). This resulted in models with better generalization capabilities, improved reasoning abilities, and better behavior in open-ended zero-shot generation than non-instruction-tuned models.

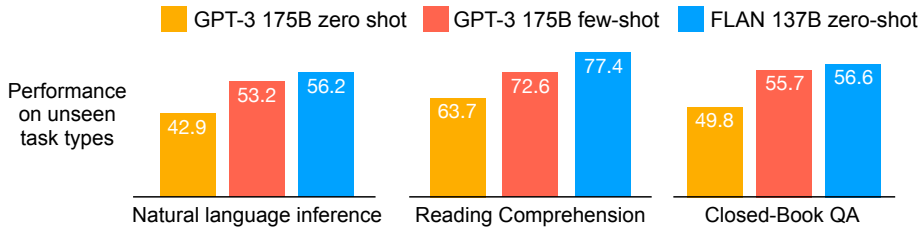


Figure 6.2: Performance of zero-shot FLAN, compared with zero-shot and few-shot GPT-3, from Wei et al. (2022a).

6.1.2 Reinforcement Learning from Human Feedback

A different training approach that leverages human instructions is reinforcement learning from human feedback (Christiano et al., 2017). Reinforcement learning (RL) is a branch of machine learning that focuses on how agents can learn to make decisions through interactions with an environment. In RL, an agent receives rewards or penalties based on its actions in the environment, and its objective is to maximize its cumulative reward over time. The reward function is a critical component of RL, as it specifies the goals and incentives for the agent. The agent's behavior is captured by a policy, which maps observations to actions. The goal of RL is to optimize the policy to maximize the expected cumulative reward. The intuition behind RL from human feedback (RLHF) is to fit a reward function to the human's preferences while simultaneously training a policy to optimize the current predicted reward function.

In this setting, the agent that we are considering is a pretrained language model that needs to be aligned with the user's intention. To this end, human preferences are used as a reward signal to train the model (process illustrated in Fig. 6.3). This procedure incorporates both explicit intentions such as following instructions and implicit intentions such as staying truthful, and not being biased, toxic, or otherwise harmful. RLHF consists of the following steps:

1. Defining a distribution of prompts on which we want the model to produce aligned outputs.

2. Constructing a dataset of human-written demonstrations and using it to train supervised learning baseline model. These demonstrations represent the desired behavior on the input prompt distribution.

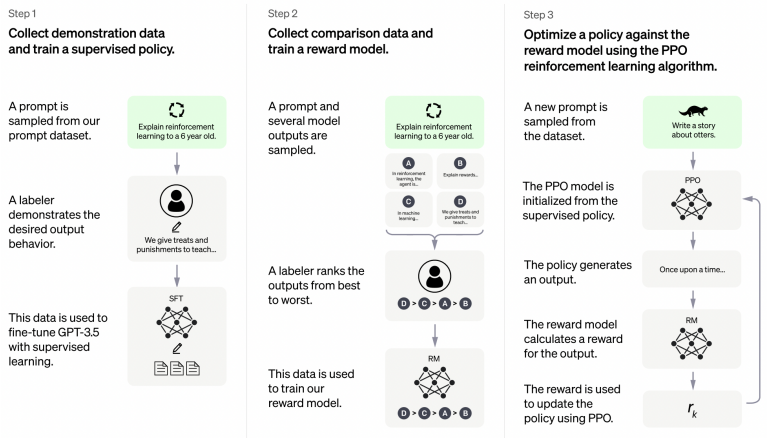


Figure 6.3: RLHF training procedure. Source: https://openai.com/blog/chatgpt.

3. Collecting a dataset D of comparisons between outputs produced by the baseline model, where labelers indicate which output they prefer for a given input. A reward model is then trained to predict the human-preferred output. The loss function for the reward model θ_{RM} is:

$$\mathcal{L}(\theta_{RM}) = -\frac{1}{\binom{K}{2}} E_{(x,y_w,y_l)\sim D}\big[\log(\sigma(r_{\theta_{RM}}(x,y_w) - r_{\theta_{RM}}(x,y_l)))\big],$$

where $r_\theta(x,y)$ is the scalar output of the reward model for prompt x and completion y with parameters θ, y_w is the preferred completion out of the pair of y_w and y_l, and K is the number of ranked responses to prompt x present in D.

4. Using the output of the reward model as a scalar reward, fine-tune the supervised policy to optimize this reward using the proximal policy optimization (PPO) algorithm (Schulman et al., 2017). The objective function during this phase is:

$$\begin{aligned}\text{obj}(\phi) =& E_{(x,y)\sim D_{\pi_\phi^{RL}}}\big[r_{\theta_{RM}}(x,y) - \beta\log(\pi_\phi^{RL}(y|x)/\pi^{SFT}(y|x))\big] + \\ & \gamma E_{x\sim D_{\text{pretrain}}}\big[\log(\pi_\phi^{RL}(x))\big],\end{aligned}$$

where π_ϕ^{RL} is the learned RL policy, π^{SFT} is the supervised trained model, and D_{pretrain} is the pretraining distribution. This objective

combines a per-token Kullback-Leibler penalty from the supervised finetuning (SFT) model to mitigate over-optimization of the reward model, with a pretraining loss term. This terms are weighted by the parameters β and θ, respectively.

This procedure aligns the behavior of the language model to the stated preferences of a specific group of people.

An example of models trained via RLHF is the InstructGPT series (Ouyang et al., 2022). For these models, the 175B-parameter GPT-3 is used a SFT baseline model, while a smaller 6B model is used to model the reward. A human evaluation of the outputs produced by InstructGPT showed the efficacy of the RLHF technique: a set of labelers rated InstructGPT outputs favorably, compared to the predictions of the non-instruction-trained baseline, along multiple axes (e.g., compliance to the constraints in the prompt, lack of hallucinations, use of appropriate language). An aggregation of this result is displayed in Figure 6.4. The same procedure was used to train the popular ChatGPT model, though with slight differences in the data collection setup.

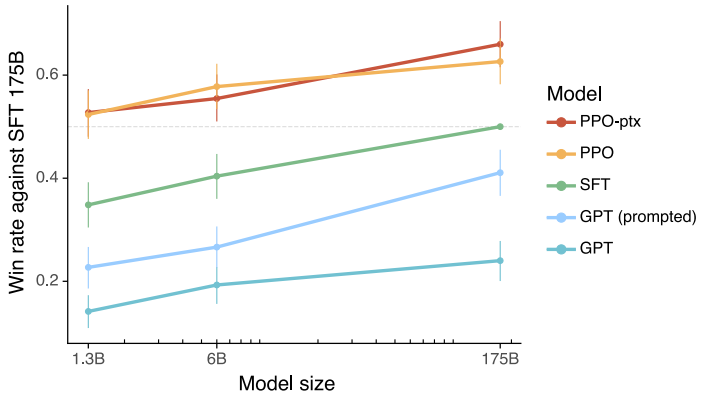


Figure 6.4: Multiple models evaluated by how often their outputs were preferred to those for the 175B SFT model, according to human evaluators. InstructGPT (PPO-ptx), as well as its variant without pretraining mix (PPO), significantly outperform GPT-3.

6.2 Scaling laws and Emergent Behavior

With the improved accuracy of pretrained language models, researchers noticed improvements in performance with increasing model size.

Kaplan et al. (2020) demonstrate that the test-loss of an autoregressive transformer exhibits a power-law relationship when the model's performance is constrained by only one of three factors: the number of non-embedding parameters (N), the size of the dataset (D), or the compute budget allocated optimally (C_{min}). They analyze the relationship between language modeling loss and these factors, with a focus on the Transformer architecture. The broad range of performance levels in language tasks enables us to examine trends across more than seven orders of magnitude in scale.

The authors identify precise power-law scaling patterns for performance in relation to training time, context length, dataset size, model size, and computational resources.

6.2.1 Summary of Scaling Laws

The test loss of a Transformer trained to autoregressively model language can be predicted using a power-law when performance is limited by only either the number of non-embedding parameters N, the dataset size D, or the optimally allocated compute budget C_{min} (see Figure **??**):

1. For models with a limited number of parameters, trained to convergence on sufficiently large datasets: $L(N) = (N_{\text{c}}/N)^{\alpha_N}$; $\alpha_N \sim 0.076$, $N_{\text{c}} \sim 8.8 \times 10^{13}$ (non-embedding parameters)

2. For large models trained with a limited dataset with early stopping: $L(D) = (D_{\text{c}}/D)^{\alpha_D}$; $\alpha_D \sim 0.095$, $D_{\text{c}} \sim 5.4 \times 10^{13}$ (tokens)

3. When training with a limited amount of compute, a sufficiently large dataset, an optimally-sized model, and a sufficiently small batch size: $L(C_{\text{min}}) = \left(C_{\text{c}}^{\text{min}}/C_{\text{min}}\right)^{\alpha_C^{\text{min}}}$; $\alpha_C^{\text{min}} \sim 0.050$, $C_{\text{c}}^{\text{min}} \sim 3.1 \times 10^8$ (PF-days)

These relations hold across eight orders of magnitude in C_{min}, six orders of magnitude in N, and over two orders of magnitude in D. They depend very weakly on model shape and other Transformer hyperparameters (depth, width, number of self-attention heads). The power laws $\alpha_{\text{N}}, \alpha_{\text{D}}, \alpha_C^{\text{min}}$ specify the degree of performance improvement expected as we scale up N, D, or C_{min}; for example, doubling the number of parameters yields a loss that is smaller by a factor $2^{-\alpha_N} = 0.95$. The precise numerical values of $N_{\text{c}}, C_{\text{c}}^{\text{min}}$, and D_{c} depend on the vocabulary size and tokenization and hence do not have a fundamental meaning.

Scaling a model can give rise to some emergent abilities. Wei et al. (2022b) describe an emergent ability as

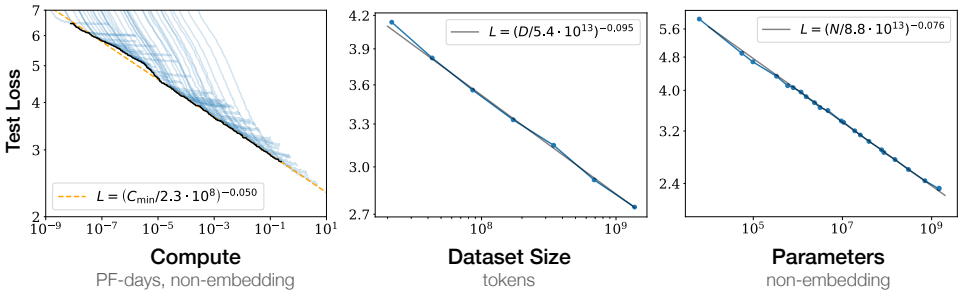


Figure 6.5: Language modeling performance improves smoothly as we increase the model size, datasetset size, and amount of compute used for training. For optimal performance all three factors must be scaled up in tandem.

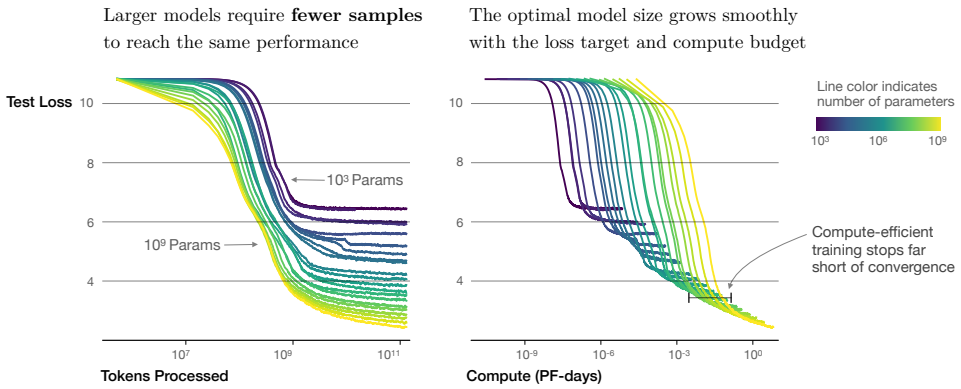


Figure 6.6: Language model training runs, with models ranging in size from 10^3 to 10^9 parameters (excluding embeddings).

> *An ability is emergent if it is not present in smaller models but is present in larger models.*

Wei et al. (2022b) compare how models of varying sizes perform on various tasks and show that abilities to perform some tasks emerge in language models after a certain scale. Figure 6.7 and 6.8 show how language models that are trained above a certain number of FLOPS exhibit increased performance on certain tasks.

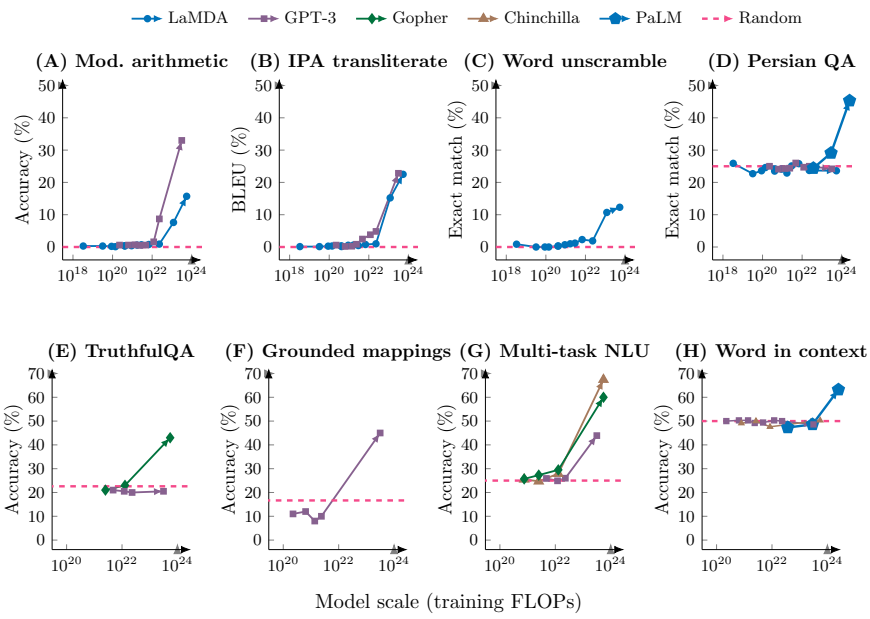


Figure 6.7: Eight examples of emergence in the few-shot prompting setting. Each point is a separate model. The ability to perform a task via few-shot prompting is emergent when a language model achieves random performance until a certain scale, after which performance significantly increases to well above random.

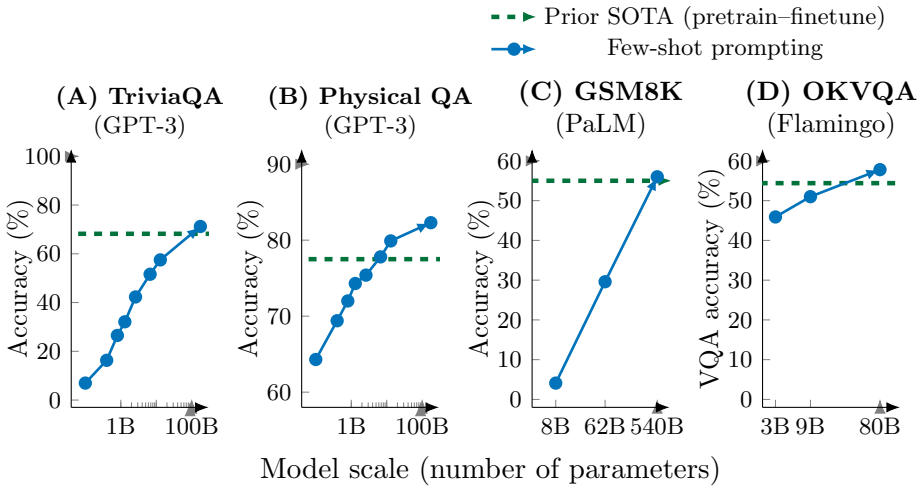


Figure 6.8: On some benchmarks, task-general models (not explicitly trained to perform a task) surpass prior state-of-the-art performance held by a task-specific model.

Index

A

adapter 27

F

fine-tuning 9
finetuning 25

M

masked language modelling 12

N

next sentence prediction 12
non-parametric 51

P

parametric 51
pretrained model 9
pretraining 9
prompting 8

S

self-supervision 9

Bibliography

Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katie Millican, Malcolm Reynolds, et al. 2022. Flamingo: A visual language model for few-shot learning. *arXiv preprint arXiv:2204.14198*.

Peter Anderson, Xiaodong He, Chris Buehler, Damien Teney, Mark Johnson, Stephen Gould, and Lei Zhang. 2018. Bottom-up and top-down attention for image captioning and visual question answering. In *CVPR*.

Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C Lawrence Zitnick, and Devi Parikh. 2015. VQA: Visual question answering. In *ICCV*.

Alexei Baevski and Michael Auli. 2018. Adaptive input representations for neural language modeling. *arXiv preprint arXiv:1809.10853*.

Hangbo Bao, Li Dong, and Furu Wei. 2022a. BEiT: BERT pre-training of image transformers. In *ICLR*.

Hangbo Bao, Wenhui Wang, Li Dong, and Furu Wei. 2022b. VL-BEiT: Generative vision-language pretraining. *arXiv preprint arXiv:2206.01127*.

Eyal Ben-David, Nadav Oved, and Roi Reichart. 2021. PADA: A prompt-based autoregressive approach for adaptation to unseen domains.

Stevo Bozinovski. 2020. Reminder of the first paper on transfer learning in neural networks, 1976. *Informatica*, 44(3).

Stevo Bozinovski and Ante Fulgosi. 1976. The influence of pattern similarity and transfer learning upon training of a base perceptron b2. In *Proceedings of Symposium Informatica*, volume 3, pages 121–126.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Xinlei Chen, Hao Fang, Tsung-Yi Lin, Ramakrishna Vedantam, Saurabh Gupta, Piotr Dollár, and C. Lawrence Zitnick. 2015. Microsoft COCO captions: Data collection and evaluation server. *arXiv preprint arXiv:1504.00325*.

Yen-Chun Chen, Linjie Li, Licheng Yu, Ahmed El Kholy, Faisal Ahmed, Zhe Gan, Yu Cheng, and Jingjing Liu. 2020. UNITER: Universal image-text representation learning. In *ECCV*.

Eunsol Choi, Omer Levy, Yejin Choi, and Luke Zettlemoyer. 2018. Ultra-fine entity typing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 87–96, Melbourne, Australia. Association for Computational Linguistics.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2022. Palm: Scaling language modeling with pathways.

Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. 2017. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. 2022. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416*.

Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. 2020. Electra: Pre-training text encoders as discriminators rather than generators. *arXiv preprint arXiv:2003.10555*.

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. 2020. Unsupervised cross-lingual representation learning at scale. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8440–8451, Online. Association for Computational Linguistics.

Alexis CONNEAU and Guillaume Lample. 2019. Cross-lingual language model pretraining. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Damai Dai, Yutao Sun, Li Dong, Yaru Hao, Zhifang Sui, and Furu Wei. 2022. Why can gpt learn in-context? language models secretly perform gradient descent as meta optimizers. *arXiv preprint arXiv:2212.10559*.

Abhishek Das, Satwik Kottur, Khushi Gupta, Avi Singh, Deshraj Yadav, José MF Moura, Devi Parikh, and Dhruv Batra. 2017. Visual dialog. In *CVPR*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Shizhe Diao, Wangchunshu Zhou, Xinsong Zhang, and Jiawei Wang. 2023. Write and paint: Generative vision-language models are unified modal learners. In *The Eleventh International Conference on Learning Representations*.

Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, Jing Yi, Weilin Zhao, Xiaozhi Wang, Zhiyuan Liu, Hai-Tao Zheng, Jianfei Chen, Yang Liu, Jie Tang, Juanzi Li, and Maosong Sun. 2022. Delta tuning: A comprehensive study of parameter efficient methods for pre-trained language models. *CoRR*, abs/2203.06904.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. 2021. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*.

Zi-Yi Dou, Yichong Xu, Zhe Gan, Jianfeng Wang, Shuohang Wang, Lijuan Wang, Chenguang Zhu, Zicheng Liu, Michael Zeng, et al. 2022. An empirical study of training end-to-end vision-and-language transformers. In *CVPR*.

Iddo Drori, Sarah Zhang, Reece Shuttleworth, Leonard Tang, Albert Lu, Elizabeth Ke, Kevin Liu, Linda Chen, Sunny Tran, Newman Cheng, et al.

2022. A neural network solves, explains, and generates university math problems by program synthesis and few-shot learning at human level. *Proceedings of the National Academy of Sciences*, 119(32):e2123433119.

Zhe Gan, Linjie Li, Chunyuan Li, Lijuan Wang, Zicheng Liu, Jianfeng Gao, et al. 2022. Vision-language pre-training: Basics, recent advances, and future trends. *Foundations and Trends® in Computer Graphics and Vision*, 14(3–4):163–352.

Tianyu Gao, Adam Fisch, and Danqi Chen. 2021. Making pre-trained language models better few-shot learners. In *Association for Computational Linguistics (ACL)*.

Demi Guo, Alexander M. Rush, and Yoon Kim. 2021. Parameter-efficient transfer learning with diff pruning. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, pages 4884–4896. Association for Computational Linguistics.

Michael Gutmann and Aapo Hyvärinen. 2010. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *AISTATS*.

Xu Han, Weilin Zhao, Ning Ding, Zhiyuan Liu, and Maosong Sun. 2021. PTR: Prompt tuning with rules for text classification.

Adi Haviv, Jonathan Berant, and Amir Globerson. 2021. BERTese: Learning to speak to BERT. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 3618–3623, Online. Association for Computational Linguistics.

C. Hohensee and J. Lobato. 2021. *Transfer of Learning: Progressive Perspectives for Mathematics Education and Related Fields*. Research in Mathematics Education. Springer International Publishing.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for NLP. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2790–2799. PMLR.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Ting-Hao Huang, Francis Ferraro, Nasrin Mostafazadeh, Ishan Misra, Aishwarya Agrawal, Jacob Devlin, Ross Girshick, Xiaodong He, Pushmeet Kohli, Dhruv Batra, et al. 2016. Visual storytelling. In *NAACL*.

Zhicheng Huang, Zhaoyang Zeng, Yupan Huang, Bei Liu, Dongmei Fu, and Jianlong Fu. 2021. Seeing Out of tHe bOx: End-to-End pre-training for vision-language representation learning. In *CVPR*.

Zhicheng Huang, Zhaoyang Zeng, Bei Liu, Dongmei Fu, and Jianlong Fu. 2020. Pixel-BERT: Aligning image pixels with text by deep multi-modal transformers. *arXiv preprint arXiv:2004.00849*.

Drew A Hudson and Christopher D Manning. 2019. GQA: A new dataset for real-world visual reasoning and compositional question answering. In *CVPR*.

Chao Jia, Yinfei Yang, Ye Xia, Yi-Ting Chen, Zarana Parekh, Hieu Pham, Quoc V Le, Yunhsuan Sung, Zhen Li, and Tom Duerig. 2021. Scaling up visual and vision-language representation learning with noisy text supervision. In *ICML*.

Zhengbao Jiang, Frank F. Xu, Jun Araki, and Graham Neubig. 2020. How can we know what language models know? *Transactions of the Association for Computational Linguistics*, 8:423–438.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. 2019. Generalization through memorization: Nearest neighbor language models. *arXiv preprint arXiv:1911.00172*.

Wonjae Kim, Bokyung Son, and Ildoo Kim. 2021. ViLT: Vision-and-language transformer without convolution or region supervision. In *ICML*.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *arXiv preprint arXiv:2205.11916*.

Jonathan Krause, Justin Johnson, Ranjay Krishna, and Li Fei-Fei. 2017. A hierarchical approach for generating descriptive image paragraphs. In *CVPR*.

Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen, Yannis Kalantidis, Li-Jia Li, David A Shamma, et al. 2017. Visual genome: Connecting language and vision using crowdsourced dense image annotations. *IJCV*.

Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Malloci, Alexander Kolesnikov, et al. 2020. The open images dataset v4. *IJCV*.

Gukyeong Kwon, Zhaowei Cai, Avinash Ravichandran, Erhan Bas, Rahul Bhotika, and Stefano Soatto. 2022. Masked vision and language modeling for multi-modal representation learning. *arXiv preprint arXiv:2208.02131*.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*.

Jaejun Lee, Raphael Tang, and Jimmy Lin. 2019. What would elsa do? freezing layers during transformer fine-tuning. *CoRR*, abs/1911.03090.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *ACL*, pages 7871–7880. Association for Computational Linguistics.

Gen Li, Nan Duan, Yuejian Fang, Ming Gong, and Daxin Jiang. 2020a. Unicoder-VL: A universal encoder for vision and language by cross-modal pre-training. In *AAAI*.

Junnan Li, Dongxu Li, Caiming Xiong, and Steven Hoi. 2022. Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation. In *ICML*.

Junnan Li, Ramprasaath R Selvaraju, Akhilesh Deepak Gotmare, Shafiq Joty, Caiming Xiong, and Steven Hoi. 2021. Align before fuse: Vision and language representation learning with momentum distillation. In *NeurIPS*.

Liunian Harold Li, Mark Yatskar, Da Yin, Cho-Jui Hsieh, and Kai-Wei Chang. 2019. VisualBERT: A simple and performant baseline for vision and language. *arXiv preprint arXiv:1908.03557*.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*.

Xiujun Li, Xi Yin, Chunyuan Li, Pengchuan Zhang, Xiaowei Hu, Lei Zhang, Lijuan Wang, Houdong Hu, Li Dong, Furu Wei, Yejin Choi, and Jianfeng Gao. 2020b. Oscar: Object-semantics aligned pre-training for vision-language tasks. In *ECCV*.

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. 2014. Microsoft COCO: Common objects in context. In *ECCV*.

Haokun Liu, Derek Tam, Muqeeth Mohammed, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin Raffel. 2022. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. In *Advances in Neural Information Processing Systems*.

Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. 2021a. What makes good in-context examples for gpt-3? *arXiv preprint arXiv:2101.06804*.

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys*, 55(9):1–35.

Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. 2021b. GPT understands, too. *CoRR*, abs/2103.10385.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. 2021c. Swin transformer: Hierarchical vision transformer using shifted windows. In *ICCV*.

Jiasen Lu, Dhruv Batra, Devi Parikh, and Stefan Lee. 2019. Vilbert: Pretraining task-agnostic visiolinguistic representations for vision-and-language tasks. In *NeurIPS*.

Yao Lu, Max Bartolo, Alastair Moore, Sebastian Riedel, and Pontus Stenetorp. 2021. Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity. *arXiv preprint arXiv:2104.08786*.

Kenneth Marino, Mohammad Rastegari, Ali Farhadi, and Roozbeh Mottaghi. 2019. OK-VQA: A visual question answering benchmark requiring external knowledge. In *CVPR*.

Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. 2017. Learned in translation: Contextualized word vectors. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 6294–6305.

Bryan McCann, Nitish Shirish Keskar, Caiming Xiong, and Richard Socher. 2018. The natural language decathlon: Multitask learning as question answering. *arXiv preprint arXiv:1806.08730*.

Tomas Mikolov, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. In *International Conference on Learning Representations*.

Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, et al. 2021. Show your work: Scratchpads for intermediate computation with language models. *arXiv preprint arXiv:2112.00114*.

Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. 2017. Neural discrete representation learning. In *NeurIPS*.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Matthew E. Peters, Mark Neumann, Robert L. Logan IV, Roy Schwartz, Vidur Joshi, Sameer Singh, and Noah A. Smith. 2019. Knowledge enhanced contextual word representations. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 43–54. Association for Computational Linguistics.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana. Association for Computational Linguistics.

Fabio Petroni, Patrick Lewis, Aleksandra Piktus, Tim Rocktäschel, Yuxiang Wu, Alexander H. Miller, and Sebastian Riedel. 2020. How context affects language models' factual predictions. In *Automated Knowledge Base Construction*.

Fabio Petronic, T. Rocktäschel, A. H. Miller, P. Lewis, A. Bakhtin, Y. Wu, and S. Riedel. 2019. Language models as knowledge bases? In *In: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2019*.

Jonas Pfeiffer, Ivan Vulić, Iryna Gurevych, and Sebastian Ruder. 2020. MAD-X: An Adapter-Based Framework for Multi-Task Cross-Lingual Transfer. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7654–7673, Online. Association for Computational Linguistics.

Telmo Pires, Eva Schlinger, and Dan Garrette. 2019. How multilingual is multilingual BERT? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4996–5001, Florence, Italy. Association for Computational Linguistics.

Bryan A Plummer, Liwei Wang, Chris M Cervantes, Juan C Caicedo, Julia Hockenmaier, and Svetlana Lazebnik. 2015. Flickr30k entities: Collecting

region-to-phrase correspondences for richer image-to-sentence models. In *ICCV*.

L. Y. Pratt. 1992. Discriminability-based transfer between neural networks. In *Advances in Neural Information Processing Systems*, volume 5. Morgan-Kaufmann.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. 2021. Learning transferable visual models from natural language supervision. In *ICML*.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. Improving language understanding by generative pre-training.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21:140:1–140:67.

Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. 2021. Zero-shot text-to-image generation. In *ICML*.

Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. 2017. Learning multiple visual domains with residual adapters. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. In *NeurIPS*.

Victor Sanh, Albert Webson, Colin Raffel, Stephen Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Arun Raja, Manan Dey, M Saiful Bari, Canwen Xu, Urmish Thakker, Shanya Sharma Sharma, Eliza Szczechla, Taewoon Kim, Gunjan Chhablani, Nihal Nayak, Debajyoti Datta, Jonathan Chang, Mike Tian-Jian Jiang, Han Wang, Matteo Manica, Sheng Shen, Zheng Xin Yong, Harshit Pandey, Rachel Bawden, Thomas Wang, Trishala Neeraj, Jos Rozen, Abheesht Sharma, Andrea Santilli,

Thibault Fevry, Jason Alan Fries, Ryan Teehan, Teven Le Scao, Stella Biderman, Leo Gao, Thomas Wolf, and Alexander M Rush. 2022. Multitask prompted training enables zero-shot task generalization. In *International Conference on Learning Representations*.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Shuai Shao, Zeming Li, Tianyuan Zhang, Chao Peng, Gang Yu, Xiangyu Zhang, Jing Li, and Jian Sun. 2019. Objects365: A large-scale, high-quality dataset for object detection. In *ICCV*.

Sheng Shen, Liunian Harold Li, Hao Tan, Mohit Bansal, Anna Rohrbach, Kai-Wei Chang, Zhewei Yao, and Kurt Keutzer. 2022. How much can CLIP benefit vision-and-language tasks? In *ICLR*.

Taylor Shin, Yasaman Razeghi, Robert L. Logan IV, Eric Wallace, and Sameer Singh. 2020. AutoPrompt: Eliciting knowledge from language models with automatically generated prompts. In *Empirical Methods in Natural Language Processing (EMNLP)*.

Oleksii Sidorov, Ronghang Hu, Marcus Rohrbach, and Amanpreet Singh. 2020. Textcaps: A dataset for image captioning with reading comprehension. In *ECCV*.

Amanpreet Singh, Vivek Natarajan, Meet Shah, Yu Jiang, Xinlei Chen, Dhruv Batra, Devi Parikh, and Marcus Rohrbach. 2019. Towards vqa models that can read. In *CVPR*.

Weijie Su, Xizhou Zhu, Yue Cao, Bin Li, Lewei Lu, Furu Wei, and Jifeng Dai. 2019. VL-BERT: Pre-training of generic visual-linguistic representations. In *ICLR*.

Alane Suhr, Stephanie Zhou, Ally Zhang, Iris Zhang, Huajun Bai, and Yoav Artzi. 2019. A corpus for reasoning about natural language grounded in photographs. In *ACL*.

Hao Tan and Mohit Bansal. 2019. LXMERT: Learning cross-modality encoder representations from transformers. In *EMNLP*.

Wilson L. Taylor. 1953. "cloze procedure": A new tool for measuring readability. *Journalism Quarterly*, 30(4):415–433.

Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. 2021. Training data-efficient image transformers & distillation through attention. In *ICML*.

Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. 2015. Show and tell: A neural image caption generator. In *CVPR*.

Eric Wallace, Shi Feng, Nikhil Kandpal, Matt Gardner, and Sameer Singh. 2019. Universal adversarial triggers for attacking and analyzing NLP. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 2153–2162. Association for Computational Linguistics.

Jianfeng Wang, Xiaowei Hu, Zhe Gan, Zhengyuan Yang, Xiyang Dai, Zicheng Liu, Yumao Lu, and Lijuan Wang. 2021a. UFO: A unified transformer for vision-language representation learning. *arXiv preprint arXiv:2111.10023*.

Jianfeng Wang, Zhengyuan Yang, Xiaowei Hu, Linjie Li, Kevin Lin, Zhe Gan, Zicheng Liu, Ce Liu, and Lijuan Wang. 2022a. GIT: A generative image-to-text transformer for vision and language. *arXiv preprint arXiv:2205.14100*.

Wenhui Wang, Hangbo Bao, Li Dong, and Furu Wei. 2021b. VLMo: Unified vision-language pre-training with mixture-of-modality-experts. *arXiv preprint arXiv:2111.02358*.

Zirui Wang, Jiahui Yu, Adams Wei Yu, Zihang Dai, Yulia Tsvetkov, and Yuan Cao. 2022b. SimVLM: Simple visual language model pretraining with weak supervision. In *ICLR*.

Jason Wei, Maarten Bosma, Vincent Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V Le. 2022a. Finetuned language models are zero-shot learners. In *International Conference on Learning Representations*.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. 2022b. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. 2022c. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Dani Yogatama, Cyprien de Masson d'Autume, and Lingpeng Kong. 2021. Adaptive semiparametric language models. *Transactions of the Association for Computational Linguistics*, 9:362–373.

Fei Yu, Jiji Tang, Weichong Yin, Yu Sun, Hao Tian, Hua Wu, and Haifeng Wang. 2021. ERNIE-VIL: Knowledge enhanced vision-language representations through scene graphs. In *AAAI*.

Jiahui Yu, Zirui Wang, Vijay Vasudevan, Legg Yeung, Mojtaba Seyedhosseini, and Yonghui Wu. 2022. Coca: Contrastive captioners are image-text foundation models. *TMLR*.

Licheng Yu, Patrick Poirson, Shan Yang, Alexander C Berg, and Tamara L Berg. 2016. Modeling context in referring expressions. In *ECCV*.

Weizhe Yuan, Graham Neubig, and Pengfei Liu. 2021. BARTScore: Evaluating generated text as text generation.

Elad Ben Zaken, Yoav Goldberg, and Shauli Ravfogel. 2022. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 1–9. Association for Computational Linguistics.

Rowan Zellers, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. From recognition to cognition: Visual commonsense reasoning. In *CVPR*.

Yan Zeng, Xinsong Zhang, Hang Li, Jiawei Wang, Jipeng Zhang, and Wangchunshu Zhou. 2022. X^2-VLM: All-In-One pre-trained model for vision-language tasks. *arXiv preprint arXiv:2211.12402*.

Pengchuan Zhang, Xiujun Li, Xiaowei Hu, Jianwei Yang, Lei Zhang, Lijuan Wang, Yejin Choi, and Jianfeng Gao. 2021. VinVL: Revisiting visual representations in vision-language models. In *CVPR*.

Yuhao Zhang, Victor Zhong, Danqi Chen, Gabor Angeli, and Christopher D. Manning. 2017. Position-aware attention and supervised data improve slot filling. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 35–45, Copenhagen, Denmark. Association for Computational Linguistics.

Zhengyan Zhang, Xu Han, Zhiyuan Liu, Xin Jiang, Maosong Sun, and Qun Liu. 2019. ERNIE: Enhanced language representation with informative entities. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1441–1451, Florence, Italy. Association for Computational Linguistics.

Zexuan Zhong, Dan Friedman, and Danqi Chen. 2021. Factual probing is [MASK]: Learning vs. learning to recall. *CoRR*, abs/2104.05240.

Luowei Zhou, Hamid Palangi, Lei Zhang, Houdong Hu, Jason Corso, and Jianfeng Gao. 2020. Unified vision-language pre-training for image captioning and vqa. In *AAAI*.

Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE international conference on computer vision*, pages 19–27.