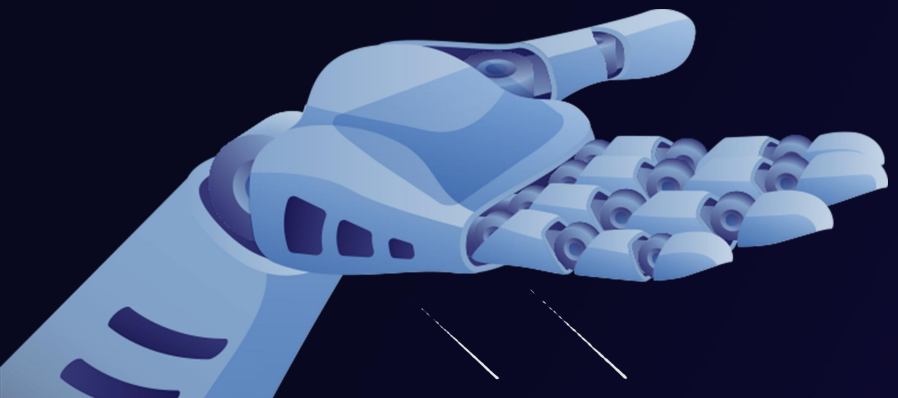
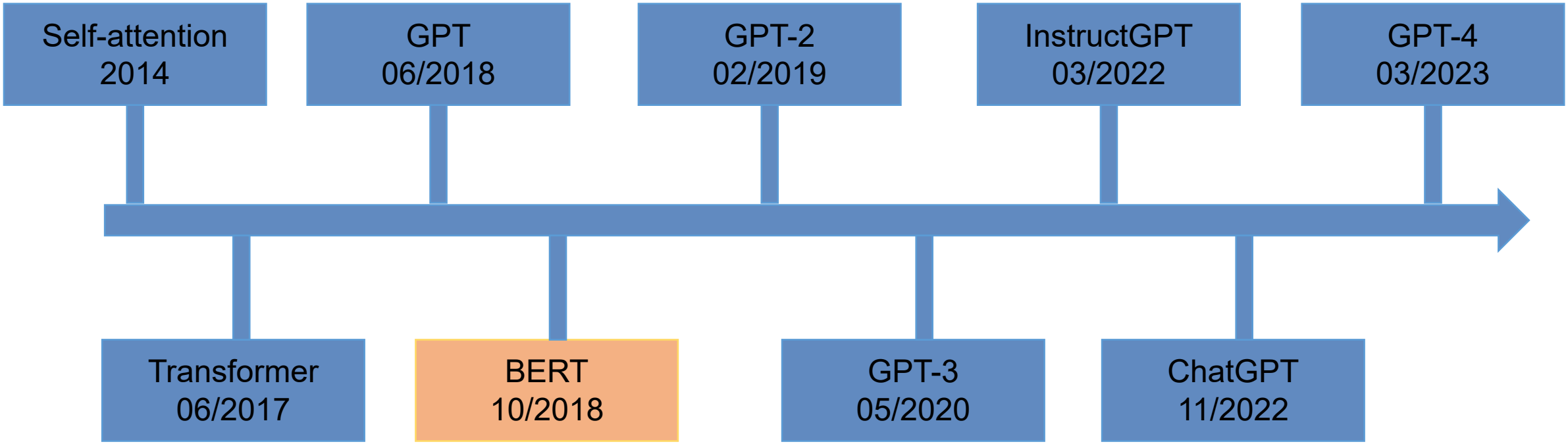
第五章：BERT算法原理及实践案例



CONTENTS

一

5.1 BERT简介



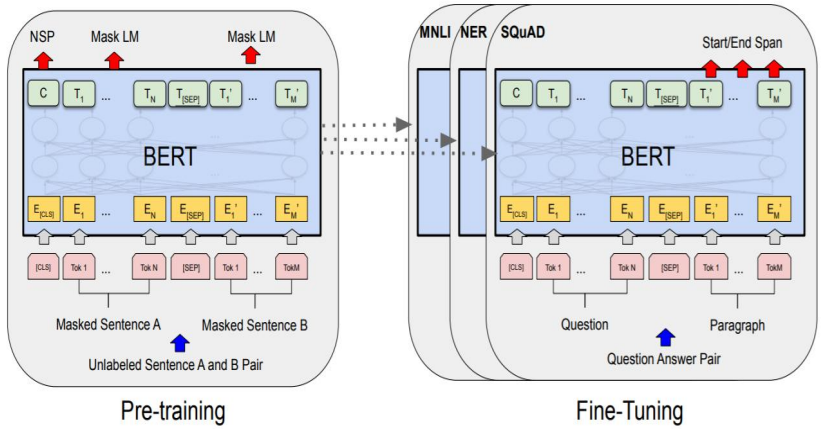
如今，BERT在计算资源有限情况下，在需要上下文理解情况下的任务上仍然表现出色。

5.1 BERT简介

1、BERT是什么

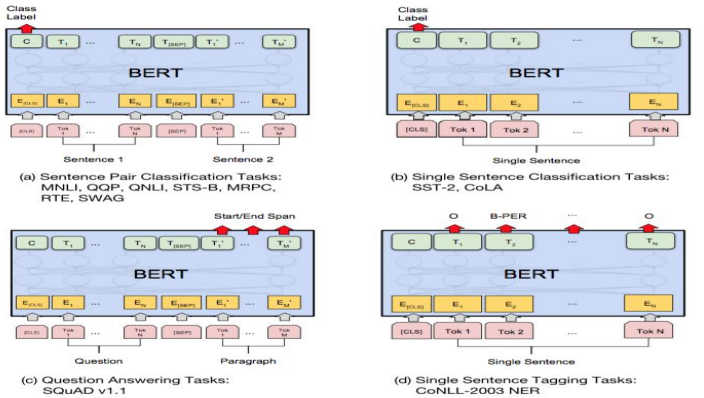
谷歌公司AI团队发布的BERT模型，全称为Bidirectional Encoder Representation from Transformers，是一个预训练的语言表征模型。它采用新的掩码语言模型和下一个句子预测任务，能生成深度的双向语言表征。预训练后，只需要添加一个额外的输出层进行fine-tune，就可以在各种各样的下游任务中取得state-of-the-art的表现。在这过程中并不需要对BERT进行任务特定的结构修改。

2、网络结构



BERT

3、应用场景



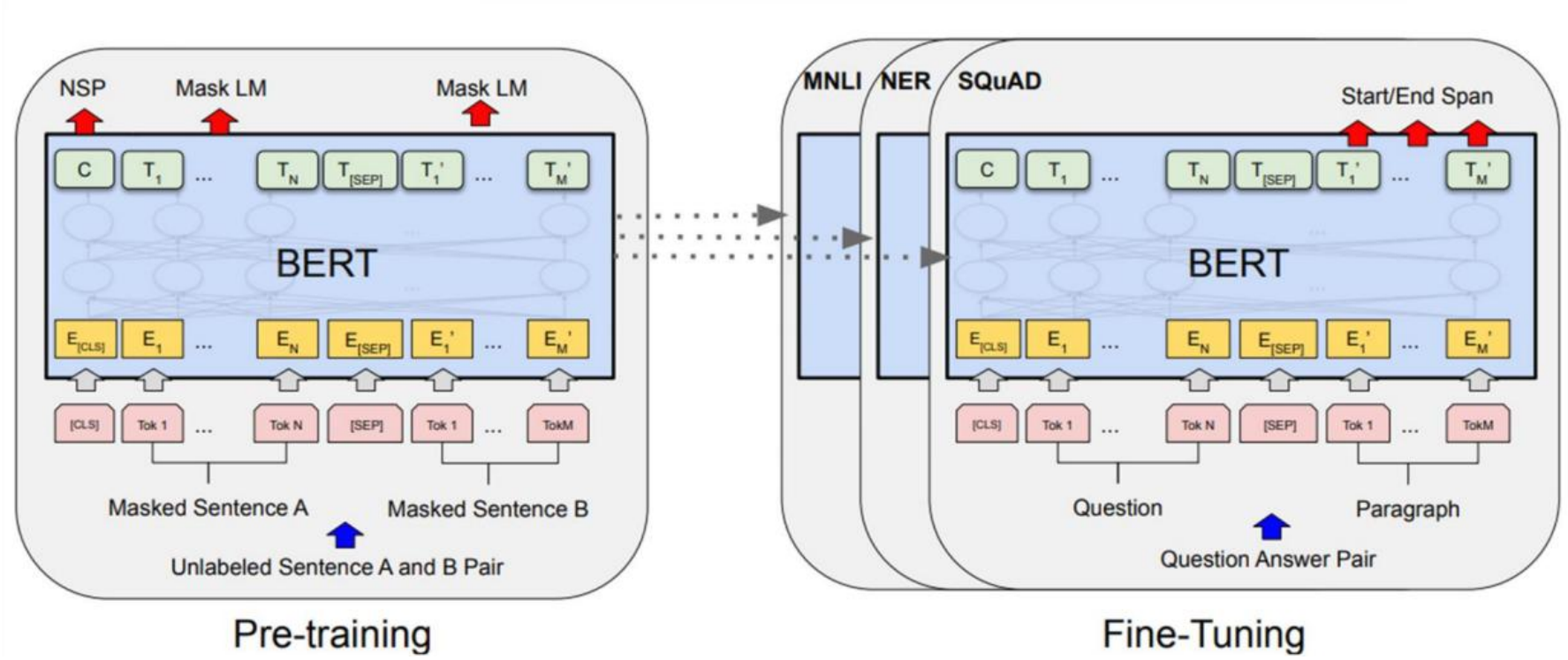
4、算法性能

BERT is conceptually simple and empirically powerful. It obtains new state-of-the-art results on eleven natural language processing tasks, including pushing the GLUE score to 80.5% (7.7% point absolute improvement), MultiNLI accuracy to 86.7% (4.6% absolute improvement), SQuAD v1.1 question answering Test F1 to 93.2 (1.5 point absolute improvement) and SQuAD v2.0 Test F1 to 83.1 (5.1 point absolute improvement).

5.1 BERT简介

1）BERT是使用Transformer的Encoder进行的，双向的，预训练模型。

2）作为预训练模型配合一些与任务相关的网络结构，可以进行各种各样的下游任务微调。



5.2 BERT预训练：整体过程

预训练大致的过程如右图所示。输入经Bert Encoder层编码后，进行MLM与NSP的任务，产生一个联合训练的损失函数从而迭代更新整个模型中的参数。

1）样本：文本、掩码位标签、是否下一个句子标签。

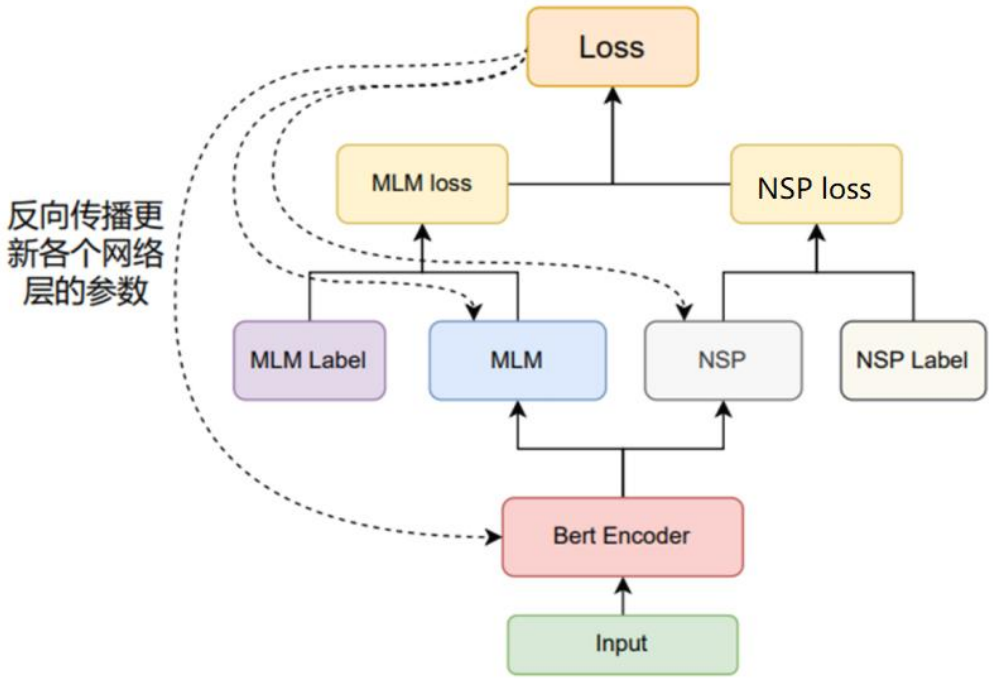
2）输入：有三个部分组成，分别为Token Embeddings、Segment Embeddings和Position Embeddings。

3）BERT Encoder: 由12层的Transformer Encoder组成。

4）Bert Encoder模块后的向量进入MLM和NSP任务的网络，并计算MLM loss和NSP loss。

5）将MLM loss和NSP loss相加得到总的损失。

6）在总损失的基础上进行反向传播更新网络参数。



MLM(Masked Language Modeling)：遮盖句子中若干个词通过周围词去预测被遮盖的词。

NSP(Next Sentence Prediction)：判断句子B在文章是否属于句子A的下一个句子。

5.2 BERT预训练：样本

语料

1）BooksCorpus (800M words)

2）英文维基百科 (2,500M words)

预处理

原始

my dog is cute he likes playing

分词&添加标志符号

[cls] my dog is cute [sep] he likes play #ing [sep]

加工掩码标签

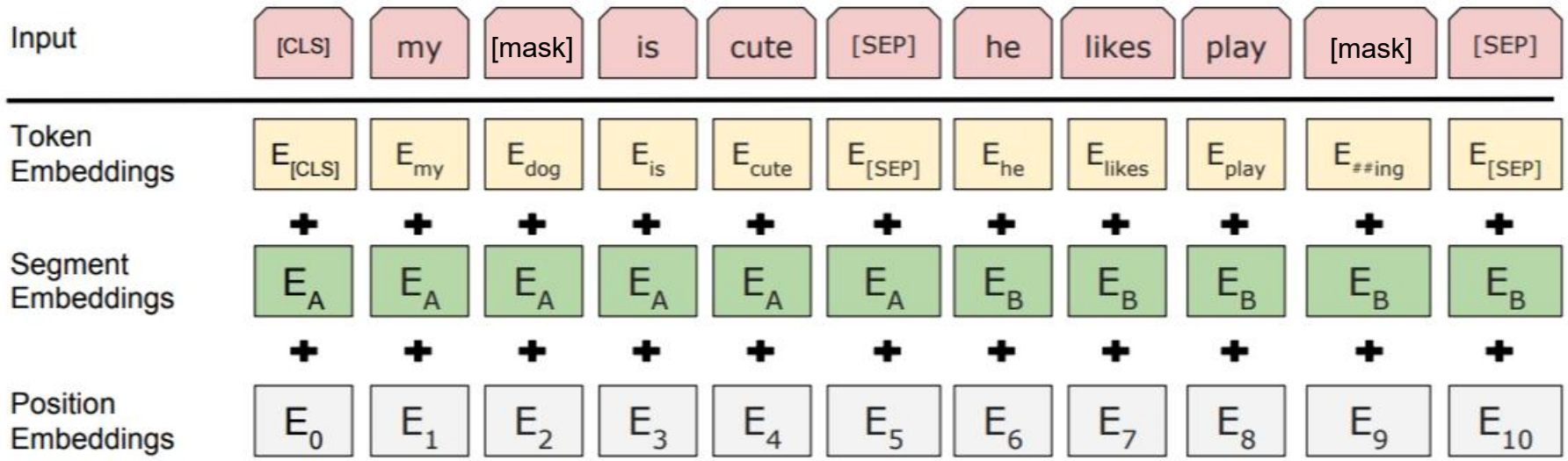
[cls] my [mask] is cute [sep] he likes play [mask] [sep]

加工下一个句子标签

input_x=[cls] my [mask] is cute [sep] he likes play [mask] [sep]
input_y=yes

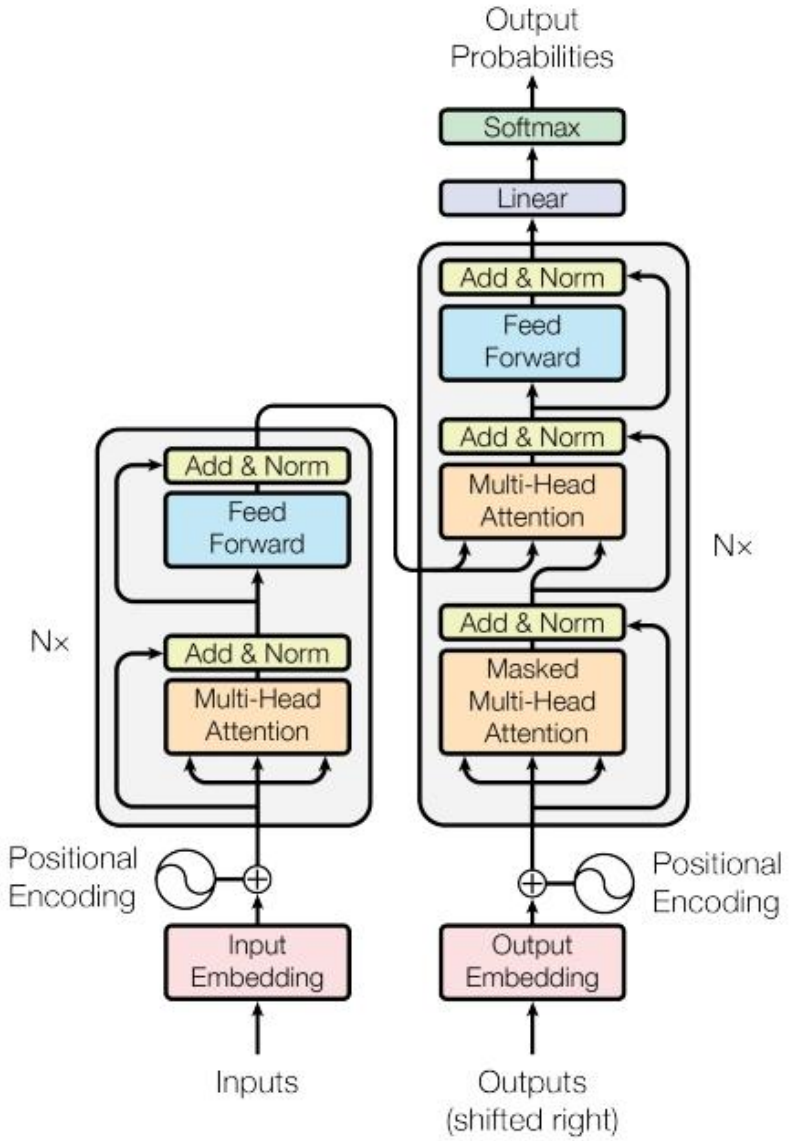
5.2 BERT预训练：输入

将预处理后的输入组合成三种Embedding的相加，分别是Token级别Embedding、句子级别Embedding、位置级别Embedding。



5.2 BERT预训练：编码器

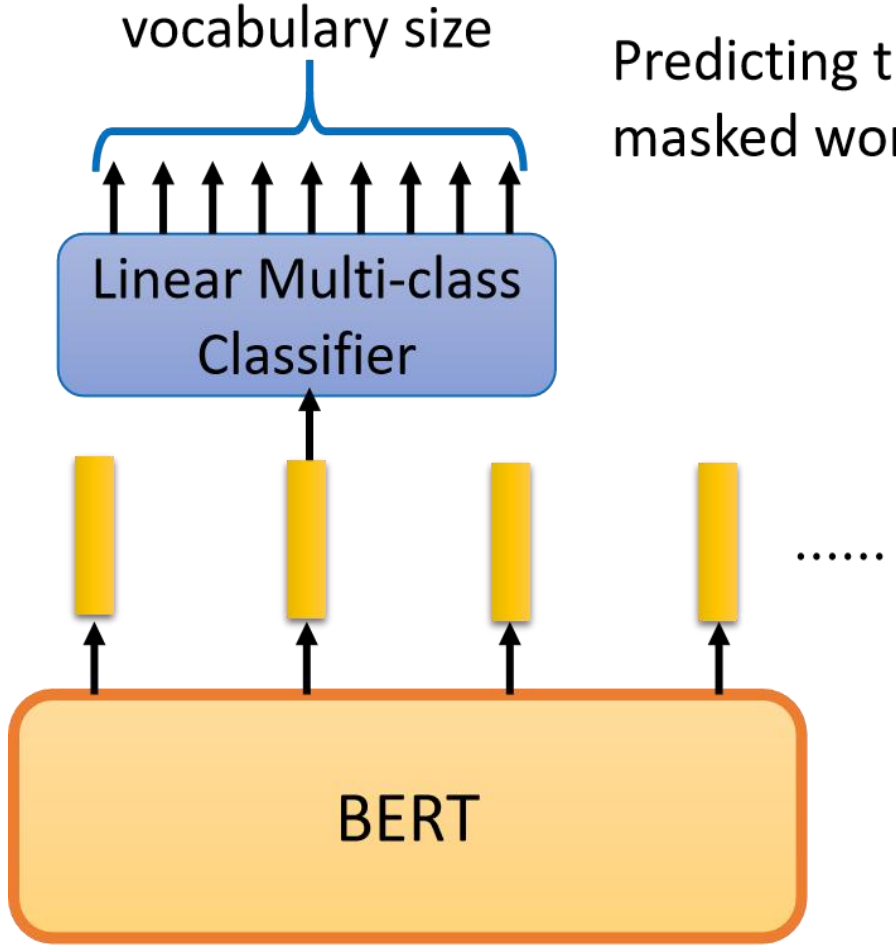
BERT的Encoder由12层的Transfomer Encorder组成。



5.2 BERT预训练：掩码语言模型（MLM）

• 输入：Bert Encoder层的输出，需要预测的词元位置。
• 中间：一个MLP的结构。默认的形式如右图所示。
• 输出：序列类别分布张量。

掩码采样方式：
➢ 每一对句子中随机选择15%位置的词语进行遮盖动作。
➢ 遮盖时80%替换为”<mask>”，10%替换为随机词元，10%不变。过程中<CLS> 与 <SEP>不会被替换。



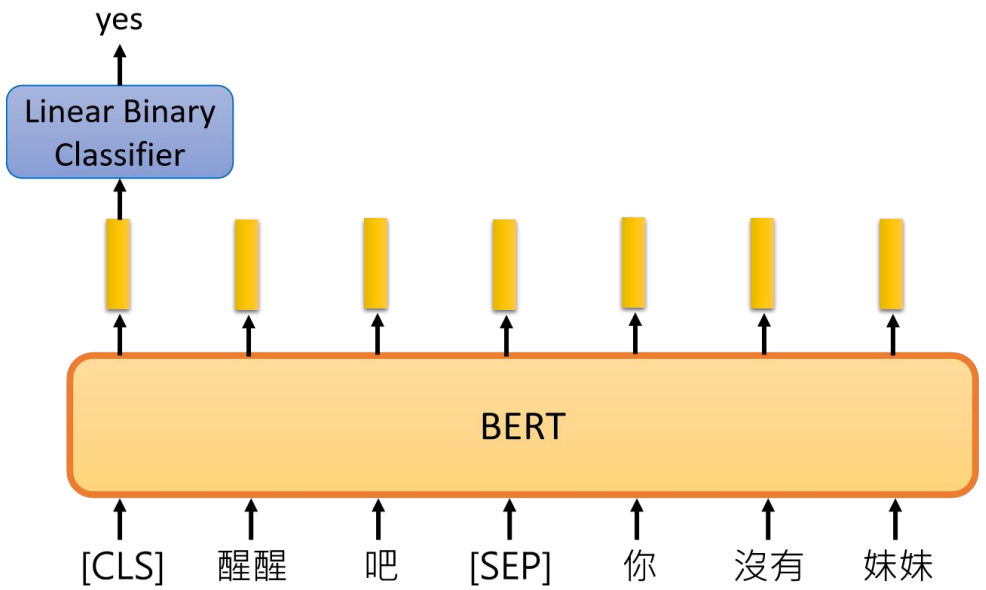
5.2 BERT预训练：下一句预测任务（NSP）

- 输入：Bert Encoder层输出<CLS>位置的张量(也就是序列中的首位)。

- 中间：一个MLP的结构。默认是一个输出维度为2的，激活函数为Softmax的全连接层。

- 输出：类别分布张量。

- 样本采样：采样时，50%的概率将第二句句子随机替换为段落中的任意一句句子。



5.2 BERT预训练：训练参数

batch_size：256 <=>per batch：256sequences * 512 tokens = 128,000 tokens
epochs：40
优化器：Adam (learning_rate = 1e-4, $\beta 1 = 0.9$, $\beta 2 = 0.999$)
L2正则：0.01
warmup steps：10000
dropout：0.1
激活函数：gelu
资源：4 TPUs for bert base, 16 TPUs for bert large, 4 days

5.3 BERT的微调：四大类任务

微调主要是指通过预训练得到的BERT Encoder网络接上各种各样的下游网络进行不同的任务。

4大类任务

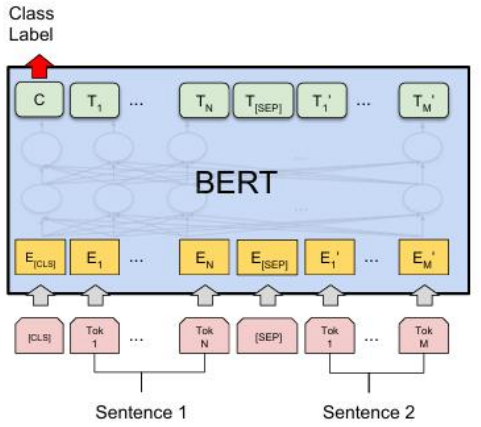
a) 句子对分类，将经过Encoder层编码后的<CLS>对应位置的向量输入进一个多分类的MLP网络中即可。

b) 单句分类，同上。

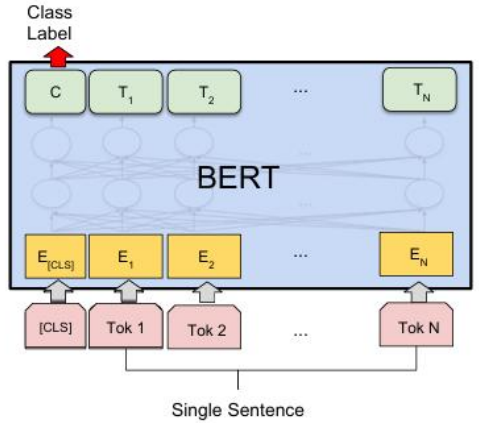
c) 根据问题得到答案，输入是一个问题与一段描述组成的句子对。将经过Encoder层编码后的每个词元对应位置的向量输入进3分类的MLP网络，而类别分别是Start(答案的首位)，End(答案的末尾)，Span(其他位置)。

d) 命名实体识别，将经过Encoder层编码后每个词元对应位置的向量输入进一个多分类的MLP网络中即可。

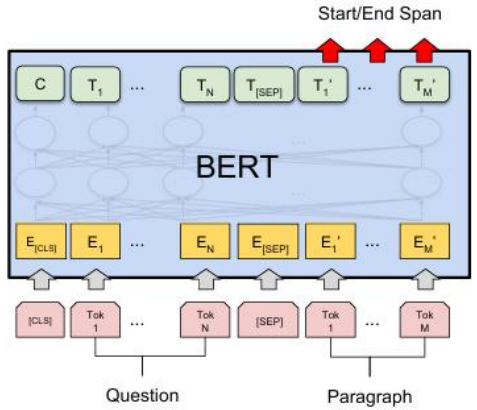
也可以结合实际场景设计更多的微调任务。



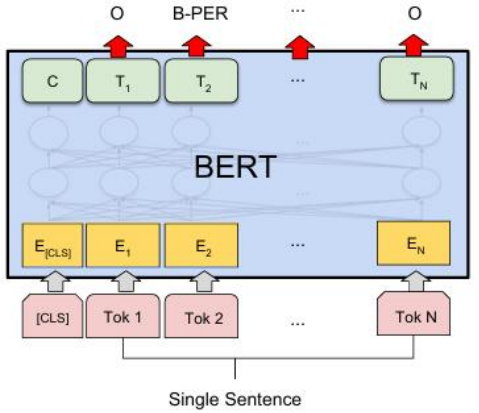
(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG



(b) Single Sentence Classification Tasks: SST-2, CoLA



(c) Question Answering Tasks: SQuAD v1.1



(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

5.3 BERT的微调：消融实验

预训练目标调整

1）基于BERTbase模型架构进行消融实验。

2）NO NSP：是指不使用下一个句子任务。

3）LTR & NSP：是指从左至右的模型，而且不使用NSP任务，这像OpenAI的GPT一样。

4）+BiLSTM：是指在微调时在“LTR+No NSP”的顶部增加了一个随机初始化的BiLSTM。

Tasks	Dev Set MNLI-m (Acc)	QNLI (Acc)	MRPC (Acc)	SST-2 (Acc)	SQuAD (F1)
$BERT_{BASE}$	84.4	88.4	86.7	92.7	88.5
No NSP	83.9	84.9	86.5	92.6	87.9
LTR & No NSP	82.1	84.3	77.5	92.1	77.8
+ BiLSTM	82.1	84.1	75.7	91.6	84.9

模型尺寸调整

1）右图为GLUE任务中6次微调的验证集平均准确率。

2）BERT为第一篇文章来充分证明，即使在小规模的任务上，模型尺寸增长仍带来性能的快速提升，当然这个前提是模型做了充分的预训练。

Hyperparams			Dev Set Accuracy			
#L	#H	#A	LM (ppl)	MNLI-m	MRPC	SST-2
3	768	12	5.84	77.9	79.8	88.4
6	768	3	5.24	80.6	82.2	90.7
6	768	12	4.68	81.9	84.8	91.3
12	768	12	3.99	84.4	86.7	92.9
12	1024	16	3.54	85.7	86.9	93.3
24	1024	16	3.23	86.6	87.8	93.7

5.3 BERT的微调：训练参数

预训练

batch_size：256 <=>per batch：256sequences * 512 tokens = 128,000 tokens
epochs：40
优化器：Adam (learning_rate = 1e-4, $\beta 1 = 0.9$, $\beta 2 = 0.999$)
L2正则：0.01
warmup steps：10000
dropout：0.1
激活函数：gelu
资源：4 TPUs for bert base, 16 TPUs for bert large, 4 days

微调

红色的为与预训练的差异

batch_size：16， 32
epochs：2, 3, 4
优化器：Adam (learning_rate = [5e-5, 3e-5, 2e-5], $\beta 1 = 09$, $\beta 2 = 0999$)
L2正则：0.01
warmup steps：10000
dropout：0.1
激活函数：gelu

5.4 典型微调任务实践

1）句子分类：https://github.com/huggingface/notebooks/blob/main/examples/text_classification.ipynb

2）实体识别：https://github.com/huggingface/notebooks/blob/main/examples/token_classification.ipynb

3）问答：https://github.com/huggingface/notebooks/blob/main/examples/token_classification.ipynb

5.4 典型微调任务实践

1）原始数据加载：可以从接口提供的数据加载，也可以从自定义数据加载。
2）数据处理：AutoTokenizer提供分词和词编码映射功能，同时生成句子标识和掩码标识。
3）构建模型：AutoModelForSequenceClassification构建分类模型网络架构。
4）模型推理：pipeline提供抽象级的推理接口。

第一步：安装和导入必要包

```
[ ]  !pip install datasets
     !pip install evaluate
```

```
[ ]  from datasets import load_dataset
     from transformers import AutoTokenizer, AutoModelForSequenceClassification, TrainingArguments, Trainer
     import numpy as np
     import evaluate
```

第二步：原始数据加载

```
[ ]  dataset = load_dataset("yelp_review_full")
```

第三步：数据处理

```
[ ]  tokenizer = AutoTokenizer.from_pretrained("google-bert/bert-base-cased")

     def tokenize_function(examples):
         return tokenizer(examples["text"], padding="max_length", truncation=True)

     tokenized_datasets = dataset.map(tokenize_function, batched=True)
     small_train_dataset = tokenized_datasets["train"].shuffle(seed=42).select(range(1000))
     small_eval_dataset = tokenized_datasets["test"].shuffle(seed=42).select(range(1000))
```

第四步：构建模型

```
model = AutoModelForSequenceClassification.from_pretrained("google-bert/bert-base-cased", num_labels=5)
metric = evaluate.load("accuracy")


def compute_metrics(eval_pred):
    logits, labels = eval_pred
    predictions = np.argmax(logits, axis=-1)
    return metric.compute(predictions=predictions, references=labels)


training_args = TrainingArguments(output_dir="test_trainer",
                                  eval_strategy="epoch",
                                  per_device_train_batch_size=8,
                                  per_device_eval_batch_size=8,
                                  logging_steps=10,
                                  )
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=small_train_dataset,
    eval_dataset=small_train_dataset,
    compute_metrics=compute_metrics,
)

trainer.train()
```

第五步：模型保存与加载

```
[ ]  trainer.save_model('./my_model/')
```

```
[ ]  model = AutoModelForSequenceClassification.from_pretrained("./my_model/")
```

第六步：模型推理

```
from transformers import pipeline

# 加载文本分类pipeline
classifier = pipeline("text-classification", model="./my_model/", tokenizer="google-bert/bert-base-cased", device=0)

# 输入文本进行推理
texts = ["这是第一个示例文本。", "这是第二个示例文本。"]
results = classifier(texts)

# 输出结果
for result in results:
    print(f"Label: {result['label']}, Score: {result['score']:.4f}")
```

5.4 典型微调任务实践

对于BERT来说，文本相似性任务预分类任务在实现上几乎一样，区别在于，单句分类输入的是一个句子，文本相似性输入的是两个句子。

第一步：安装和导入必要包

```
[ ] !pip install datasets
    !pip install evaluate
```

```
[ ] from datasets import load_dataset
    from transformers import AutoTokenizer, DataCollatorWithPadding, AutoModelForSequenceClassification, import TrainingArguments, Trainer
    import numpy as np
    import evaluate
```

第二步：原始数据加载

```
[ ] dataset = load_dataset("glue", "mrpc")
```

第三步：数据处理

```
[ ] tokenizer = AutoTokenizer.from_pretrained("google-bert/bert-base-cased")


    def tokenize_function(examples):
        return tokenizer(examples["sentence1"], examples["sentence2"], padding="max_length", truncation=True, max_length=128)


    tokenized_datasets = dataset.map(tokenize_function, batched=True)
    data_collator = DataCollatorWithPadding(tokenizer)
```

第四步：构建模型

```
[ ] model = AutoModelForSequenceClassification.from_pretrained("google-bert/bert-base-cased", num_labels=2)
    metric = evaluate.load("accuracy")


    def compute_metrics(eval_pred):
        logits, labels = eval_pred
        predictions = np.argmax(logits, axis=-1)
        return metric.compute(predictions=predictions, references=labels)


    training_args = TrainingArguments(output_dir="test_trainer",
                                      eval_strategy="epoch",
                                      per_device_train_batch_size=8,
                                      per_device_eval_batch_size=8,
                                      num_train_epochs=5,
                                      logging_steps=10,
                                      )
    trainer = Trainer(
        model=model,
        args=training_args,
        train_dataset=tokenized_datasets['train'],
        eval_dataset=tokenized_datasets['validation'],
        data_collator=data_collator,
        tokenizer=tokenizer,
        compute_metrics=compute_metrics,
    )

    trainer.train()
```

第五步：模型保存与加载

```
[ ] trainer.save_model('./my_model/')
```

```
[ ] model = AutoModelForSequenceClassification.from_pretrained("./my_model/")
```

第六步：模型推理

```
[ ] from transformers import pipeline

    # 加载文本分类pipeline
    classifier = pipeline("text-classification", model="./my_model/", tokenizer="google-bert/bert-base-cased", device=0)

    # 输入文本进行推理
    texts = {"text": "this is a good thing for me", "text_pair": "it is a good news"}
    results = classifier(texts)

    # 输出结果
    print(results)
    # for result in results:
    #     print(f"Label: {result['label']}, Score: {result['score']:.4f}")
```

感谢您的观看