RoBERTa：一种稳健优化的 BERT 预训练方法

Abstract

语言模型预训练带来了显着的性能提升，但不同方法之间的仔细比较具有挑战性。训练的计算成本很高，通常在不同大小的私有数据集上进行，并且正如我们将展示的，超参数的选择对最终结果有重大影响。我们提出了 BERT 预训练的复制研究（Devlin 等人，2019），该研究仔细测量了许多关键超参数和训练数据大小的影响。我们发现 BERT 的训练明显不足，但可以匹配或超过其之后发布的每个模型的性能。我们的最佳模型在 GLUE、RACE 和 SQuAD 上取得了最先进的结果。这些结果凸显了以前被忽视的设计选择的重要性，并对最近报告的改进的来源提出了疑问。我们发布我们的模型和代码。 [1]

1. 简介

自训练方法，例如 ELMo (Peters et al., 2018)、GPT (Radford et al., 2018)、BERT (Devlin et al., 2019)、XLM (Lample and Conneau, 2019) 和 XLNet (Yang et al., 2019) al., 2019）带来了显着的性能提升，但确定这些方法的哪些方面贡献最大可能具有挑战性。训练的计算成本很高，限制了可以完成的调整量，并且通常使用不同大小的私人训练数据来完成，限制了我们衡量建模进步效果的能力。

* 平等贡献。

[1] 我们的模型和代码可在以下网址获取：https://github.com/pytorch/fairseq

我们提出了 BERT 预训练的复制研究（Devlin 等人，2019），其中包括对效果的仔细评估超参数调整和训练集大小。我们发现 BERT 训练明显不足，并提出了一种改进的 BERT 模型训练方法，我们称之为 RoBERTa，它可以匹配或超过所有后 BERT 方法的性能。我们的修改很简单，包括：（1）用更大的batches、更多的数据训练模型更长时间；（2)去除下一句预测目标；（3）较长序列的训练；（4）动态改变应用于训练数据的 masking 模式。我们还收集了一个与其他私人使用的数据集大小相当的大型新数据集（CC-NEWS），以更好地控制训练集大小的影响。

在控制训练数据时，我们改进的训练程序改进了 GLUE 和 SQuAD 上已发布的 BERT 结果。当使用额外数据进行更长时间的训练时，我们的模型在公共 GLUE 排行榜上获得了 88.5 分，与 Yang 等人报告的 88.4 分相匹配（2019）。我们的模型在 4/9 的 GLUE 任务上建立了新的最先进技术：MNLI、QNLI、RTE 和 STS-B。我们还在 SQuAD 和 RACE 上匹配最先进的成绩。总体而言，我们重新确定 BERT 的 masked 语言模型训练目标与最近提出的其他训练目标（例如扰动自回归语言模型）具有竞争力（Yang 等人，2019）。 [2]

总之，本文的贡献是：（1）我们提出了一组重要的 BERT 设计选择和训练策略，并介绍了

[2] 通过更多的调整，这些其他方法也可能得到改进。我们将这种探索留给未来的工作。

带来更好的下游任务绩效的替代方案；（2)我们使用了一个新颖的数据集CCNEWS，并确认使用更多的数据进行预训练可以进一步提高下游任务的性能；（3）我们的训练改进表明，masked 语言模型预训

练在正确的设计选择下，与所有其他最近发布的方法相比具有竞争力。我们发布了在 PyTorch 中实现的模型、预训练和 fine-tuning 代码（Paszke et al., 2017）。

2. 背景

在本节中，我们将简要概述 BERT（Devlin 等人，2019）预训练方法以及我们将在下一节中通过实验检查的一些训练选择。

2.1. 设置

BERT 将两个段（标记序列）的串联作为输入： $x_1, \ldots, x_N$ 和 $y_1, \ldots, y_M$。句段通常由多个自然句子组成。这两个片段作为 BERT 的单个输入序列呈现，并用特殊标记分隔它们：
$[CLS], x_1, \ldots, x_N, [SEP], y_1, \ldots, y_M$ ，$[E \cdot O \cdot S]$。 M 和 N 受到约束，使得 $M + N < T$，其中 T 是控制训练期间最大序列长度的参数。

该模型首先在大型未标记文本语料库上进行预训练，然后使用最终任务标记数据进行微调。

2.2. 架构

BERT 使用现在无处不在的 transformer 架构（Vaswani 等人，2017），我们不会详细回顾。我们使用具有 L 层的 transformer 架构。每个块都使用 A 自注意力头和隐藏维度 H。

2.3. 训练目标

在预训练过程中，BERT 使用两个目标：masked 语言建模和下一句预测。

Masked 语言模型 (MLM) 选择输入序列中 tokens 的随机样本，并将其替换为特殊的 token $[MASK]$。 MLM 的目标是预测 masked tokens 时的交叉熵损失。 BERT统一选择输入tokens的 15%进行可能的替换。在所选择的tokens中， 80%被替换为保持不变的 $[MASK]$, 10%，并且 10%被随机选择的词汇token替换。

在最初的实现中，随机 mask 和替换在开始时执行一次并在训练期间保存，尽管在实践中，数据是重复的，因此每个训练句子的 mask 并不总是相同的（参见第 1 节） 4.1）。

下一句预测（NSP） NSP 是一种二元分类损失，用于预测原始文本中两个片段是否相互跟随。正面例子是通过从文本语料库中取出连续的句子来创建的。负样本是通过将不同文档中的片段配对来创建的。正例和负例以相等的概率进行采样。

NSP 目标旨在提高下游任务的性能，例如自然语言推理（Bowman 等人，2015），它需要推理句子对之间的关系。

2.4. 优化

BERT 使用 Adam（Kingma 和 Ba，2015）使用以下参数进行优化： $\beta_1 = 0.9$、
$\beta_2 = 0.999, \epsilon = 1\mathrm{e}-6$ 和 L_2 权重衰减为 0.01。学习率在前 10,000 步中预热至峰值 1e-4，然后线性衰减。 BERT 在所有层和注意力权重上使用 0.1 的 dropout 以及 GELU 激活函数进行训练（Hendrycks 和 Gimpel，2016）。模型针对 $S = 1,000,000$ 更新进行了预训练，其中mini batches 包含最大长度 $B = 256$ tokens 的 $T = 512$ 序列。

2.5. 数据

BERT 在 BOOKCORPUS（Zhu et al., 2015）和英语 WIKIPEDIA 的组合上进行训练，总共 16 GB 个未压缩文本。[3]

3. 实验设置

在本节中，我们将描述 BERT 复制研究的实验设置。

3.1. 实现

我们在 FAIRSEQ 中重新实现了 BERT（Ott 等人，2019）。我们主要遵循原始的 BERT

[3] 杨等人(2019) 使用相同的数据集，但报告在数据清理后仅包含 13 GB 文本。这很可能是由于维基百科数据清理方面的细微差异造成的。第 2 节中给出的优化超参数，但峰值学习率和 warmup 步骤数除外，这些参数针对每个设置分别进行调整。我们还发现训练对 Adam epsilon 项非常敏感，在某些情况下，我们在调整它后获得了更好的性能或提高了稳定性。同样，我们发现设置 $\beta_2 = 0.98$ 可以提高使用大 batch 尺寸进行训练时的稳定性。

我们使用最多 $T = 512$ tokens 的序列进行预训练。与德夫林等人不同（2019），我们不会随机注入短序列，并且我们不会在第一个 90% 更新时使用减少的序列长度进行训练。我们仅使用全长序列进行训练。

我们在 DGX-1 机器上使用混合精度浮点算法进行训练，每台机器都配有通过 Infiniband 互连的 8× 32GB Nvidia V100 GPU（Micikevicius 等人，2018）。

3.2. 数据

BERT 式的预训练很大程度上依赖于大量文本。巴耶夫斯基等人(2019) 证明增加数据大小可以提高最终任务性能。一些工作已经在比原始 BERT 更大、更多样化的数据集上进行了训练（Radford 等人，2019；Yang 等人，2019；Zellers 等人，2019）。不幸的是，并非所有附加数据集都可以公开发布。在我们的研究中，我们专注于收集尽可能多的数据进行实验，使我们能够在每次比较中适当匹配数据的整体质量和数量。

我们考虑了五个不同大小和领域的英语语料库，总计超过 160GB 的未压缩文本。我们使用以下文本语料库：

- BookCorpus（Zhu et al., 2015）加上英文维基百科。这是用于训练 BERT 的原始数据（16 GB）。
- CC-News，我们从 CommonCrawl News 数据集的英文部分收集（Nagel，2016）。该数据包含2016年9月至2019年2月期间抓取的6300万篇英文新闻文章。（过滤后76GB）。[4]
- OpenWebText（Gokaslan 和 Cohen，2019），WebText 相关的公开的Radford 等人描述的语料（2019）。该文本是从 Reddit 上共享的 URL 中提取的 Web 内容，并且至少有 3 票赞成（38GB）。[5]
- Stories，Trinh 和 Le (2018) 中引入的数据集，包含 CommonCrawl 数据的子集，经过过滤以匹配 Winograd 模式的故事风格（31GB）。

3.3. 评估

继之前的工作之后，我们使用以下三个基准评估下游任务的预训练模型。

GLUE 通用语言理解评估 (GLUE) 基准（Wang 等人，2019b）是用于评估自然语言理解系统的 9 个数据集的集合。[6] 任务被定义为单句分类或句子对分类任务。 GLUE 组织者提供训练和开发数据拆分以及提交服务器和排行榜，允许参与者根据私人保留的测试数据评估和比较他们的系统。

对于第 4 节中的复制研究，我们在相应的单任务训练数据（即没有多任务训练或集成）上微调预训练模型后报告开发集的结果。我们的微调程序遵循原始 BERT 论文（Devlin 等人，2019）。

在第 5 节中，我们还报告了从公共排行榜获得的测试集结果。这些结果取决于一些特定于任务的修改，我们在第 5.1 节中对此进行了描述。

SQuAD 斯坦福问答数据集 (SQuAD) 提供一段上下文和一个问题。任务是通过从上下文中提取相关跨度来回答问题。我们评估了 SQuAD 的两个版本：V1.1 和 V2.0（Rajpurkar 等人，2016、2018）。在 V1.1 中，上下文总是包含答案，而对于 V2.0 有些问题没有在所提供的上下文中得到回答，使得任务更具挑战性。

[5] 作者及其附属机构与 OpenWebText 数据集的创建没有任何关系。

[6] 数据集为：CoLA (Warstadt et al., 2018)、Stanford Sentiment Treebank (SST) (Socher et al., 2013)、Microsoft Research Paragraph Corpus (MRPC) (Dolan and Brockett, 2005)、语义文本相似度Benchmark (STS) (Agirre et al., 2007)、Quora Question Pairs (QQP) (Iyer et al., 2016)、MultiGenre NLI (MNLI) (Williams et al., 2018)、Question NLI (QNLI) (Rajpurkar et al., 2018) al., 2016）、识别文本蕴含（RTE） (Dagan et al., 2006；Bar-Haim et al., 2006；Giampiccolo et al., 2007；Bentivogli et al., 2009）和 Winograd NLI (WNLI)（莱维斯克等人，2011）。

对于 SQuAD V1.1，我们采用与 BERT 相同的跨度预测方法（Devlin et al., 2019）。对于 SQuAD V2.0，我们添加了一个额外的二元分类器来预测问题是否可回答，我们通过对分类和跨度损失项求和来联合训练该分类器。在评估过程中，我们仅预测被分类为可回答的对的跨度索引。

RACE ReAding Compressive from Examinations (RACE) (Lai et al., 2017) 任务是一个大规模的阅读理解数据集，包含超过 28,000 个段落和近 100,000 个问题。该数据集是从中国针对中学生和高中生的英语考试中收集的。在 RACE 中，每个段落都与多个问题相关。对于每一个问题，任务是从四个选项中选择一个正确答案。 RACE 的上下文比其他流行的阅读理解数据集要长得多，并且需要推理的问题比例非常大。

4. 训练程序分析

本节探讨并量化了哪些选择对于成功预训练 BERT 模型很重要。我们保持模型架构固定。[7] 具体来说，我们首先使用与 $\text{BERT}_{\text{BASE}}$ 相同的配置（ $L = 12, H = 768, A = 12, 110\text{M}$ 参数）训练 BERT 模型。

4.1. 静态 vs. 动态 Masking

正如第 2 节中所讨论的，BERT 依赖于随机 mask 和预测 tokens。最初的 BERT 实现在数据预处理期间执行一次 mask，从而产生单个静态 mask。为了避免在每个 mask 中的每个训练实例使用相同的 epoch，训练数据被复制 10 次，以便每个序列在 40 个 masked 训练中以 10 种不同的方式成为 epochs。因此，在训练过程中，每个训练序列都会出现四次相同的 mask。

我们将此策略与动态mask进行比较，动态mask是在每次向模型提供序列时生成mask模式。当预训练更多步骤或使用更大的数据集时，这一点变得至关重要。

[7] 研究架构变化，包括更大的架构，是未来工作的一个重要领域。

Masking	SQuAD 2.0	MNLI-m	SST-2
参考	76.3	84.3	92.8
我们的重新实现：			
静态	78.3	84.3	92.5
动态	78.7	84.0	92.9

表 1： $\text{BERT}_{\text{BASE}}$ 的静态和动态 masking 之间的比较。我们报告 F1 的 SQuAD 以及 MNLI-m 和 SST-2 的准确性。报告的结果是 5 次随机初始化（种子）的中位数。参考结果来自Yang等人（2019）。

结果 表 1 比较了 Devlin 等人发布的 BERT $_{BASE}$ 结果（2019）我们用静态或动态 masking 重新实现。我们发现，我们用静态 masking 重新实现的性能与原始 BERT 模型类似，而动态 masking 与静态 masking 相当或稍好。

鉴于这些结果以及动态 masking 的额外效率优势，我们在其余实验中使用动态 masking。

4.2. 模型输入格式和下一句预测

在原始的 BERT 预训练过程中，模型观察两个 concat 生成的文档片段，它们要么从同一文档（带有 $p = 0.5$）连续采样，要么从不同的文档采样。除了 masked 语言建模目标之外，该模型还经过训练，通过辅助下一句预测 (NSP) 损失来预测观察到的文档片段是否来自相同或不同的文档。

NSP 损失被假设是训练原始 BERT 模型的一个重要因素。德夫林等人(2019) 观察到删除 NSP 会损害性能，QNLI、MNLI 和 SQuAD 1.1 的性能会显着下降。然而，最近的一些工作对 NSP 损失的必要性提出了质疑（Lample 和 Conneau，2019；Yang 等人，2019；Joshi 等人，2019）。

为了更好地理解这种差异，我们比较了几种替代训练形式：

- SEgMENT-PAIR+NSP：这遵循 BERT 中使用的原始输入格式（Devlin 等人，2019），但有 NSP 损失。每个输入都有一对段，每个段可以包含多个自然句子，但总组合长度必须小于 512 tokens。

模型	SQuAD 1.1/2.0	MNLI-m	SST-2	RACE
我们的重新实现（带有 NSP 损失）：				
段对	**90.4/78.7**	84.0	92.9	64.2
句子对	**88.7/76.2**	82.9	92.1	63.0
我们的重新实现（没有 NSP 损失）：				
完整句子	90.4/79.1	84.7	92.5	64.8
文档语句	**90.6/79.7**	84.7	92.7	65.6
$\text{BERT}_{\text{BASE}}$	**88.5/76.3**	84.3	92.8	64.3
$\text{XLNet}_{\text{BASE}}\,(\text{K}=7)$	-181.3	85.8	92.7	66.1
$\text{XLNet}_{\text{BASE}}\,(\text{K}=6)$	−/**81.0**	85.6	93.4	66.7

表 2：通过 BoOKCORPUS 和 WIKIPEDIA 预训练的基础模型的开发集结果。所有模型均针对 1M 步骤进行训练，batch 大小为 256 个序列。我们报告 SQuAD 的 F1 以及 MNLI-m、SST-2 和 RACE 的准确性。报告的结果是五次随机初始化（种子）的中值。 $\text{BERT}_{\text{BASE}}$ 和 $\text{XLNet}_{\text{BASE}}$ 的结果来自 Yang 等人（2019）。

- SENTENCE-PAIR+NSP：每个输入包含一对自然句子，要么从一个文档的连续部分采样，要么从单独的文档采样。由于这些输入明显短于 512 个 tokens，因此我们增加 batch 大小，以便 tokens 的总数保持与 SEGMENT-PAIR+NSP 相似。我们保留 NSP 损失。
- 完整句子：每个输入都包含从一个或多个文档连续采样的完整句子，因此总长度最多为 512 tokens。输入可能跨越文档边界。当我们到达一个文档的末尾时，我们开始从下一个文档中采样句子，并在文档之间添加一个额外的分隔符 token。我们消除了 NSP 损失。
- DOC-SENTENCES：输入的构造与 FULL-SENTENCES 类似，但它们不能跨越文档边界。在文档末尾附近采样的输入可能短于 512 tokens，因此我们在这些情况下动态增加 batch 大小，以实现与 FULLSENTENCES 类似的总 tokens 数量。我们消除了 NSP 损失。

结果 表 2 显示了四种不同设置的结果。我们首先比较 Devlin 等人的原始 SEGMENT-PAIR 输入格式 (2019) 到句子对格式；两种格式都保留了 NSP 损失，但后者使用单个句子。我们发现**使用单个句子会损害下游任务的性能**，我们假设这是因为模型无法学习远程依赖关系。接下来，我们比较没有 NSP 损失的训练和使用单个文档（DOC-SENTENCES）中的文本块的训练。我们发现此设置优于最初发布的 $\text{BERT}_{\text{BASE}}$ 结果，并且与 Devlin （2019）等人相比，**删除 NSP loss 匹配或略微提高了下游任务性能**。最初的 BERT 实现可能只是删除了损失项，同时仍然保留了 SEGMENT-PAIR 输入格式。

最后，我们发现限制来自单个文档的序列（DOC-SENTENCES）比打包来自多个文档的序列（FULL-SENTENCES）稍好。然而，由于 DOC-SENTENCES 格式会导致 batch 大小变化，因此我们在其余实验中使用 FULLSENTENCES，以便于与相关工作进行比较。

4.3. 使用大 batches 进行训练

神经机器翻译领域过去的工作表明，当学习率适当提高时，使用非常大的 mini-batches 进行训练既可以提高优化速度，也可以提高最终任务性能（Ott et al., 2018）。最近的工作表明，BERT 也适合大型 batch 训练（You et al., 2019）。

德夫林等人(2019) 最初训练 $\text{BERT}_{\text{BASE}}$ 的 1M 步骤，batch 大小为 256 个序列。这在计算成本上相当于通过梯度累积来训练具有 batch 大小的 125 K 序列的 2 K 步骤，或者训练具有 8 K batch 大小的 31 K 的步骤。

bsz	步骤	Ir	ppl	MNLI-m	SST-2
256	1M	$1e-4$	3.99	84.7	92.7
2 K	125 K	$7e-4$	**3.68**	**85.2**	**92.9**
8 K	31 K	$1e-3$	3.77	84.6	92.8

表 3：在不同 batch 大小 (*ppl*) 的 BOOKCORPUS 和 WIKIPEDIA 上训练的基础模型对保留训练数据 (*bsz*) 的困惑度和开发集准确性。我们调整每个设置的学习率（lr）。模型对数据进行相同次数的传递 (epochs)，并且具有相同的计算成本。

在表 3 中，我们比较了 $\text{BERT}_{\text{BASE}}$ 的 perplexity 和下游任务的性能，当我们增加 batch 大小并控制训练数据的传递次数时。我们观察到，使用大的 batches 进行训练可以提高 perplexity 语言建模目标的 masked，以及最终任务的准确性。大的 batches 也更容易通过分布式数据并行训练 [8] 进行并行化，在后面的实验中，我们使用 8 K 序列的 batches 进行训练。

值得注意的是你等人(2019) 用更大的 batche 大小训练 BERT，最多可达 32 K 序列。我们将进一步探索大型 batch 训练的局限性留给未来的工作。

4.4. 文本编码

字节对编码 (BPE)（Sennrich 等人，2016）是字符级和单词级表示形式的混合体，可以处理自然语言语料库中常见的大型词汇表。 BPE 依赖于子词单元，而不是完整的单词，这些子词单元是通过对训练语料库进行统计分析来提取的。

BPE 词汇表大小的范围通常为 10 K − 100 K 个子字单元。然而，在对大型且多样化的语料库（例如本工作中考虑的语料库）进行建模时，unicode 字符可以占此词汇表的很大一部分。雷德福等人 (2019) 介绍了 BPE 的巧妙实现，它使用字节而不是 unicode 字符作为基本子字单元。使用字节可以学习适度大小（50K 单位）的子词词汇表，该词汇表仍然可以对任何输入文本进行编码，而不会引入任何“未知”tokens。

[8] 大型 batch 训练即使没有大规模并行硬件，也可以通过梯度累积来提高训练效率，其中来自多个 mini-batch 的梯度在每个优化步骤之前在本地累积。 FAIRSEQ 原生支持此功能（Ott 等人，2019）。

最初的 BERT 实现（Devlin 等人，2019）使用大小为 30 K 的字符级 BPE 词汇表，该词汇表是在使用启发式 token 化规则对输入进行预处理后学习的。继雷德福等人之后(2019)，我们考虑使用包含 50 K 子字单元的更大字节级 BPE 词汇来训练 BERT，而不需要对输入进行任何额外的预处理或 token 化。这分别为 $\text{BERT}_{\text{BASE}}$和 $\text{BERT}_{\text{LARGE}}$添加了大约 15M和 20M附加参数。

早期的实验显示这些编码之间只有细微的差异，Radford 等人(2019) BPE 在某些任务上的最终任务性能稍差。尽管如此，我们相信通用编码方案的优点超过了性能的轻微下降，并在我们的其余实验中使用这种编码。这些编码的更详细比较留待将来的工作。

5. RoBERTa

在上一节中，我们建议对 BERT 预训练过程进行修改，以提高最终任务性能。我们现在汇总这些改进并评估它们的综合影响。我们将此配置称为 RoBERTa，即稳健优化的 BERT 方法。具体来说，RoBERTa 使用动态 masking（第 4.1 节）、无 NSP 损失的完整句子（第 4.2 节）、大 mini-batches（第 4.3 节）和更大的字节级 BPE（第 4.4 节）进行训练。

此外，我们还研究了之前工作中未充分强调的另外两个重要因素：（1）用于预训练的数据，以及（2）数据的训练次数。例如，最近提出的 XLNet 架构（Yang 等人，2019）使用比原始 BERT（Devlin 等人，2019）多近 10 倍的数据进行预训练。它还使用大八倍的 batch 大小进行训练，优化步骤数量减半，因此与 BERT 相比，预训练中的序列数量是 BERT 的四倍。

为了帮助区分这些因素与其他建模选择（例如预训练目标）的重要性，我们首先按照 $\text{BERT}_{\text{LARGE}}$ 架构 (L=24、H=1024, A=16,355M参数）训练 RoBERTa。我们在可比较的 BooKCORPUS 加 WIKIPEDIA 数据集上预训练 100 K 步骤，如德夫林等人（2019）。我们使用 1024 个 V100 GPU 对模型进行大约一天的预训练。

Model	data	bsz	steps	SQuAD (v1.1/2.0)	MNLI-m	SST-2
RoBERTa						
with BOOKS + WIKI	16GB	8K	100K	93.6/87.3	89.0	95.3
+ additional data (§3.2)	160GB	8K	100K	94.0/87.7	89.3	95.6
+ pretrain longer	160GB	8K	300K	94.4/88.7	90.0	96.1
+ pretrain even longer	160GB	8K	500K	**94.6/89.4**	**90.2**	**96.4**
$\text{BERT}_{\text{LARGE}}$						
with BOOKS + WIKI	13GB	256	1M	90.9/81.8	86.6	93.7
$\text{XLNet}_{\text{LARGE}}$						
with BOOKS + WIKI	13GB	256	1M	94.0/87.8	88.4	94.4
+ additional data	126GB	2K	500K	94.5/88.8	89.8	95.6

表 4：当我们预训练更多数据（16GB → 160 GB 文本）和预训练更长的 (100 K → 300 K → 500 K 步骤时，RoBERTa 的开发集结果。每行都累积了上面各行的改进。 RoBERTa 符合 BERT LARgE 的架构和训练目标。 $\text{BERT}_{\text{LARGE}}$ 和 $\text{XLNet}_{\text{LARGE}}$ 的结果来自 Devlin 等人（2019）和杨等人（2019），分别。所有 GLUE 任务的完整结果可在附录中找到。

结果 我们在表 4 中展示了我们的结果。在控制训练数据时，我们观察到 RoBERTa 比最初报告的 $\text{BERT}_{\text{LARGE}}$ 结果有了很大的改进，重申了我们在第 4 节中探索的设计选择的重要性。

接下来，我们将这些数据与第 3.2 节中描述的三个附加数据集结合起来。我们使用与之前 (100 K) 相同数量的训练步骤对组合数据进行训练。总的来说，我们对 160 GB 文本进行了预训练。我们观察到所有下游任务的性能进一步提高，验证了数据大小和多样性在预训练中的重要性。 [9]

最后，我们对 RoBERTa 进行预训练的时间明显更长，将预训练步骤的数量从 100 K 增加到 300 K，然后进一步增加到 500 K。我们再次观察到下游任务性能的显着提升，并且 300K 和 500K 步模型在大多数任务中都优于 XLNetLARGE。
请注意，即使我们训练时间最长的模型似乎也不会过度拟合我们的数据，并且可能会从额外的训练中受益。

在本文的其余部分，我们在三个不同的基准上评估我们最好的 RoBERTa 模型：GLUE、SQuaD 和 RACE。具体来说，我们认为 RoBERTa 在 3.2 节中介绍的所有五个数据集上训练了 500 K 步骤。

[9] 我们的实验将数据大小和多样性的增加结合在一起。我们把对这两个维度的更仔细的分析留到未来的工作中。

5.1. GLUE结果

对于 GLUE，我们考虑两种微调设置。在第一个设置（单任务，开发）中，我们仅使用相应任务的训练数据，分别对每个 GLUE 任务进行微调 RoBERTa。我们考虑对每个任务进行有限的超参数扫描，具有 batch 大小 $\in \{16, 32\}$ 和学习率 $\in \{1e-5, 2e-5, 3e-5\}$，第一个 6% 步骤采用线性 warmup，然后线性衰减到 0。我们微调 10 epochs，并根据开发集上每个任务的评估指标执行早期停止。其余的超参数与预训练期间保持相同。在此设置中，我们报告五次随机初始化中每个任务的中值开发集结果，无需模型集成。

在第二个设置（ensembles，测试）中，我们通过 GLUE 排行榜将 RoBERTa 与测试集上的其他方法进行比较。虽然许多向 GLUE 排行榜的提交依赖于多任务微调，**但我们的提交仅依赖于单任务微调**。对于 RTE、STS 和 MRPC，我们发现从 MNLI 单任务模型开始进行微调很有帮助，而不是从 baseline预训练 RoBERTa 开始。我们探索了一个稍宽的超参数空间，如附录中所述，并且ensemble 每个任务有 5 到 7 个模型。

	MNLI	QNLI	QQP	RTE	SST	MRPC	CoLA	STS
开发上的单任务单模型								
BERT LARGE	86.6/−	92.3	91.3	70.4	93.2	88.0	60.6	90.0
XLNet LARGE	89.8/−	93.9	91.8	83.8	95.6	89.2	63.6	91.8
RoBERTa	**90.2/90.2**	**94.7**	**92.2**	**86.6**	**96.4**	**90.9**	**68.0**	**92.4**
Ensembles on test (from leaderboard as of July 25, 2019)								
ALICE	88.2/87.9	95.7	**90.7**	83.5	95.2	92.6	**68.6**	91.1
MT-DNN	87.9/87.4	96.0	89.9	86.3	96.5	92.7	68.4	91.1
XLNet	90.2/89.8	98.6	90.3	86.3	**96.8**	**93.0**	67.8	91.6
RoBERTa	**90.8/90.2**	**98.9**	90.2	**88.2**	96.7	92.3	67.8	**92.2**

表 5：GLUE 的结果。所有结果均基于 24 层架构。 BERT $_{\text{LARGE}}$ 和 XLNet LARGE $^{\text{results}}$ 来自 Devlin 等人（2019）和杨等人（2019），分别。 RoBERTa 在开发集上的结果是五次运行的中值。RoBERTa 在测试集上的结果是单任务模型的 ensemble。对于 RTE、STS 和 MRPC，我们从 MNLI 模型而不是baseline预训练模型开始进行微调。平均值是从 GLUE 排行榜获得的。

针对特定任务的修改 两个 GLUE 任务需要针对特定任务的微调方法来实现有竞争力的排行榜结果。

QNLI：最近在 GLUE 排行榜上提交的内容采用了 QNLI 任务的成对排名公式，其中从训练集中挖掘候选答案并相互比较，单个（问题，候选）对被分类为肯定（Liu 等人）等人，2019b，a；Yang 等人，2019）。这种公式显着简化了任务，但不能直接与 BERT 相比较（Devlin 等人，2019）。在最近的工作之后，我们采用排名方法来提交测试，但为了与 BERT 直接比较，我们报告基于纯分类方法的开发集结果。

WNLI：我们发现所提供的 NLI 格式数据很难处理。相反，我们使用来自 SuperGLUE 的重新格式化的 WNLI 数据（Wang 等人，2019a），它指示查询代词和所指对象的范围。我们使用 Kocijan 等人的margin排名损失对 RoBERTa 进行微调（2019）。对于给定的输入句子，我们使用 spaCy（Honnibal 和 Montani，2017）从句子中提取额外的候选名词短语并微调我们的模型，以便它为积极指称短语分配比任何生成的消极候选短语更高的分数。这种表述的一个不幸的后果是我们只能使用正训练样本，这排除了所提供的一半以上的训练样本。 [10]

[10] 虽然我们只使用提供的 WNLI 训练数据，但通过使用额外的代词消歧数据集来增强结果可能会得到改善。

结果 我们的结果如表 5 所示。在第一个设置（单任务，开发）中，RoBERTa 在所有 9 个 GLUE 任务开发集上都取得了最先进的结果。至关重要的是，RoBERTa 使用与 $\text{BERT}_{\text{LARGE}}$ 相同的 masked 语言建模预训练目标和架构，但始终优于 $\text{BERT}_{\text{LARGE}}$ 和 $\text{XLNet}_{\text{LARGE}}$。与我们在这项工作中探索的数据集大小和训练时间等更平凡的细节相比，这引发了关于模型架构和预训练目标的相对重要性的问题。

在第二个设置（ensembles，测试）中，我们将 RoBERTa 提交到 GLUE 排行榜，并在 9 项任务中的 4 项上取得了最先进的结果以及迄今为止的最高平均分数。这尤其令人兴奋，因为与大多数其他顶级提交不同，RoBERTa 不依赖于多任务微调。我们预计未来的工作可以通过结合更复杂的多任务微调程序来进一步改善这些结果。

5.2. SQuAD结果

与过去的工作相比，我们对 SQuAD 采用了更简单的方法。特别是，虽然 BERT（Devlin 等人，2019）和 XLNet（Yang 等人，2019）都使用额外的 QA 数据集来增强其训练数据，**但我们仅使用提供的 SQuAD 训练数据对 RoBERTa 进行微调**。杨等人（2019）还采用了自定义分层学习率 schedule 来微调

Model	SQuAD 1.1		SQuAD 2.0	
	EM	F1	EM	F1
Single models on dev, w/o data augmentation				
$\text{BERT}_{\text{LARGE}}$	84.1	90.9	79.0	81.8
$\text{XLNet}_{\text{LARGE}}$	**89.0**	94.5	86.1	88.8
RoBERTa	88.9	**94.6**	**86.5**	**89.4**
Single models on test (as of July 25, 2019)				
$\text{XLNet}_{\text{LARGE}}$			$86.3^{\dagger}$	$89.1^{\dagger}$
RoBERTa			86.8	89.8
XLNet + SG-Net Verifier			$\mathbf{87.0}^{\dagger}$	$\mathbf{89.9}^{\dagger}$

表 6：SQuAD 的结果。 † 表示结果取决于额外的外部训练数据。 RoBERTa 在开发和测试设置中仅使用提供的 SQuAD 数据。 $\mathbf{BERT}_{\text{LARGE}}$ 和 $\mathbf{XLNet}_{\text{LARGE}}$ 结果来自 Devlin 等人（2019）和杨等人（2019），分别。

XLNet，而我们对所有层使用相同的学习率。

对于 SQuAD v1.1，我们遵循与 Devlin 等人相同的微调程序（2019）。对于 SQuAD v2.0，我们还对给定问题是否可回答进行分类；我们通过对分类和跨度损失项求和来联合训练该分类器与跨度预测器。

结果 我们在表 6 中展示了我们的结果。在 SQuAD v1.1 开发集上，RoBERTa 与 XLNet 的最新开发集相匹配。在 SQuAD v2.0 开发集上，RoBERTa 创下了新的最先进水平，比 XLNet 提高了 0.4 点 (EM) 和 0.6 点 (F1)。

我们还将 RoBERTa 提交到公共 SQuAD 2.0 排行榜，并评估其相对于其他系统的性能。大多数顶级系统都建立在 BERT（Devlin 等人，2019）或 XLNet（Yang 等人，2019）之上，这两者都依赖于额外的外部训练数据。相比之下，我们提交的内容不使用任何额外的数据。

我们的单一 RoBERTa 模型的性能优于除其中一个提交的单一模型之外的所有模型，并且是不依赖数据增强的评分系统中最高的评分系统。

5.3. RACE结果

在 RACE 中，系统提供了一段文本、一个相关问题和四个候选答案。系统需要对四个候选答案中哪一个是正确的进行分类。

我们通过 concate- 修改 RoBERTa 来完成此任务

型号	精度	中	高
测试中的单个模型（截至 2019 年 7 月 25 日）			
$\mathbf{BERT}_{\text{LARGE}}$	72.0	76.6	70.1
XLNet $_{\text{LARGE}}$	81.7	85.4	80.2
RoBERTa	83.2	86.5	81.3

表 7：RACE 测试集的结果。 RoBERTa LARGE 和 XLNetLARGE 结果来自 Yang 等人。（2019）。

将每个候选人的答案与相应的问题和段落一起列出。然后，我们对这四个序列中的每一个进行编码，并将生成的 [CLS] 表示传递通过全连接层，该层用于预测正确答案。我们会截断长度超过 128 tokens 的问答对，并在需要时截断段落，以便总长度最多为 512 tokens。

RACE 测试集的结果如表 7 所示。RoBERTa 在初中和高中环境中均取得了最先进的结果。

6. 相关工作

预训练方法是根据不同的训练目标设计的，包括语言建模（Dai and Le，2015；Peters et al.，2018；Howard and Ruder，2018）、机器翻译（McCann et al.，2017）和masked语言建模（Devlin 等人，2019；Lample 和 Conneau，2019）。最近的许多论文都使用了针对每个最终任务的微调模型的基本方法（Howard 和 Ruder，2018；Radford 等人，2018），并使用 masked 语言模型目标的某些变体进行预训练。然而，较新的方法通过多任务 fine tuning (Dong et al., 2019)、合并实体 embeddings (Sun et al., 2019)、跨度预测 (Joshi et al., 2019) 和多重任务提高了性能自回归预训练的变体（Song et al., 2019；Chan et al., 2019；Yang et al., 2019）。通常还可以通过在更多数据上训练更大的模型来提高性能（Devlin 等人，2019；Baevski 等人，2019；Yang 等人，2019；Radford 等人，2019）。我们的目标是复制、简化并更好地调整 BERT 的训练，作为更好地理解所有这些方法的相对性能的参考点。

7. 结论

在预训练 BERT 模型时，我们仔细评估了许多设计决策。我们发现，通过更长时间地训练模型、在更多数据上使用更大的 batches 可以显着提高性能；删除下一个句子的预测目标；较长序列的训练；并动态改变应用于训练数据的 masking 模式。我们改进的预训练程序（我们称之为 RoBERTa）在 GLUE、RACE 和 SQuAD 上实现了最先进的结果，无需对 GLUE 进行多任务微调或为 SQuAD 提供额外数据。这些结果说明了这些以前被忽视的设计决策的重要性，并表明 BERT 的预训练目标与最近提出的替代方案相比仍然具有竞争力。

我们还使用了一个新颖的数据集 CC-NEWS，并在以下位置发布了用于预训练和微调的模型和代码：https://github.com/pytorch/fairseq。