General Purpose Text Embeddings from Pre-trained Language Models for Scalable Inference

Jingfei Du* **Myle Ott*** **Haoran Li** **Xing Zhou** **Veselin Stoyanov**
Facebook AI
{jingfeidu,myleott,aimeeli,xingz,ves}@fb.com

Abstract

The state of the art on many NLP tasks is currently achieved by large pre-trained language models, which require a considerable amount of computation. We aim to reduce the inference cost in a setting where many different predictions are made on a single piece of text. In that case, computational cost during inference can be amortized over the different predictions (tasks) using a shared text encoder. We compare approaches for training such an encoder and show that encoders pre-trained over multiple tasks generalize well to unseen tasks. We also compare ways of extracting fixed- and limited-size representations from this encoder, including pooling features extracted from multiple layers or positions. Our best approach compares favorably to knowledge distillation, achieving higher accuracy and lower computational cost once the system is handling around 7 tasks. Further, we show that through binary quantization, we can reduce the size of the extracted representations by a factor of 16 to store them for later use. The resulting method offers a compelling solution for using large-scale pre-trained models at a fraction of the computational cost when multiple tasks are performed on the same text.

1 Introduction

Large pre-trained language models achieve state-of-the-art performance on many Natural Language Processing (NLP) tasks (Peters et al., 2018; Radford et al., 2018; Devlin et al., 2019). However, inference for these models requires significant computational resources, which limits their practical use. Recent trends show that scaling models up (Liu et al., 2019b; Lan et al., 2020; Raffel et al., 2019; Li et al., 2020) in terms of computation still improves end task performance, raising questions about whether and how the most accurate models can be applied in real-world settings.

This computational burden is exacerbated by the need to fine-tune end-to-end a separate model for each task. Since each model has a new set of parameters, none of the computation can be shared by models for different tasks during inference. This is particularly inefficient in real-world settings that require multiple predictions about each input. For example, given a news article, we may want to predict its topic (Zhang et al., 2015), sentiment (Pang and Lee, 2004; Maas et al., 2011; Socher et al., 2013; Zhang et al., 2015), overall text quality (Pitler and Nenkova, 2008), whether it is humorous (Yang et al., 2015) or offensive (Schmidt and Wiegand, 2017; Zampieri et al., 2019) and so on.

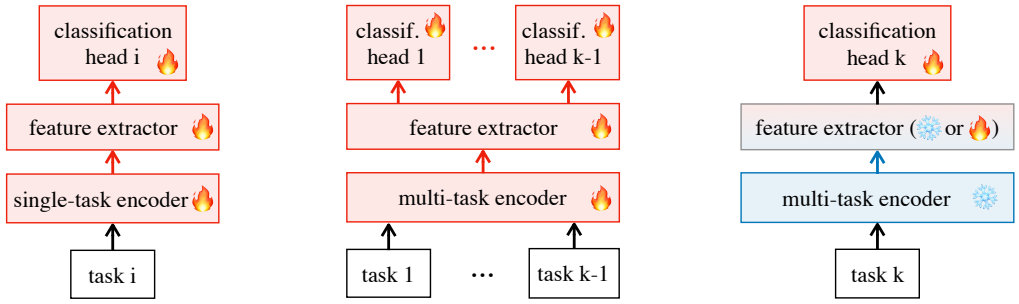
Knowledge Distillation (KD) is a way of reducing the computation required by large pre-trained LMs (Hinton et al., 2015; Sanh et al., 2019). However, there is a sizeable gap in accuracy between the best models using knowledge distillation and the full fine-tuned models. Another way of speeding up computation is through system optimizations such as quantization and operator fusion (Zafrir et al., 2019). These techniques can reduce the amount of computation significantly, but may not be sufficient by themselves and can be combined with the methods we discuss.

In this paper we look at new ways to make inference computationally efficient focusing on the case where different models (models for different tasks) are run over the same piece of text. We propose new methods to run multiple task-specific models in a way that amortizes the computation over the different tasks. The central idea is to compute the activations for the full model once and use smaller task-specific models on top of it. We explore three possible ways for sharing computation.

The first solution is inspired by work on general purpose text encoders (Kiros et al., 2015; Hill

*Equal contribution.

Findings of the Association for Computational Linguistics: EMNLP 2020, pages 3018–3030



(a) Single-task finetuning. (b) Multi-task pre-training. (c) Leave-one-task-out finetuning.

Figure 1: An illustration of the finetuning approaches explored in this work. (a) In *single-task finetuning*, an encoder model is fine-tuned end-to-end for a given task. (b) In *multi-task pre-training*, an encoder model is jointly trained over $k-1$ tasks, each with their own classification head. (c) In *leave-one-task-out finetuning*, a multi-task encoder is frozen and used to extract features for an unseen (k^{th}) task. Following Peters et al. (2019), we use 🔥 and ❄ to denote components that are fine-tuned for each task or frozen, respectively.

et al., 2016; Conneau et al., 2017; Subramanian et al., 2018), which produce fixed-size representations (i.e., sentence embeddings) that can be shared across tasks. We add only small task-specific layers on top of these fixed-size representations. Unfortunately, when evaluated on unseen tasks, we find that models that rely on fixed-size representations often underperform single-task baselines by a large margin, in agreement with past work (Subramanian et al., 2018; Peters et al., 2019; Raffel et al., 2019; Wang et al., 2019a).

The second solution is a *multi-task system* (Caruana, 1997; Collobert and Weston, 2008; Ruder, 2017), where a single model is jointly trained to handle many tasks (see Figure 1b). If most layers are shared, the overall inference cost can be nearly k times less than for k separate single-task models, while providing competitive task accuracy (Liu et al., 2019a; Raffel et al., 2019; Wang et al., 2019a). However, multi-task systems work best when the set of tasks is known in advance. Adding new tasks requires retraining the multi-task model and re-incurring training costs, thus limiting the utility of this approach in real-world systems where new classification tasks may be introduced periodically.

We propose a third solution: a multi-task encoder that is shared across tasks and produces *limited-size* representations that grow with the length of the input, similar to contextualized word representations (Peters et al., 2018). We evaluate our representations on 14 text classification tasks using a *leave-one-task-out* evaluation protocol (see Figure 1c), where a multi-task encoder model is trained on $k-1$ tasks, frozen and used as a static feature extractor for an unseen k^{th} task.[1] We find an important ingredient to performing well on an unseen (k^{th}) task is to extract features from multiple layers and positions of the encoder. Ultimately, our general purpose encoders offer a better tradeoff between task accuracy and inference cost than either fixed-size representations or distilled models, while requiring minimal additional computation to handle new tasks.

We also consider the case in which not all of the predictions can be done at the same time and intermediate representations have to saved. In that context, we study the relationship between representation size and end-task performance. We find that features extracted by our encoders are amenable to heavy quantization enabling a 16x reduction in the size of the extracted features with negligible impact on unseen task performance.

2 Related Work

Self-supervised pre-training, typically through language modeling, has advanced the state of the art for many NLP tasks (Peters et al., 2018; Radford et al., 2018; Devlin et al., 2019). There are two dominant ways of adapting pre-trained models to downstream tasks: (1) finetuning, which often results in the best accuracy (Devlin et al., 2019); and (2) feature extraction, which can be significantly more efficient during inference when there are multiple end tasks. Peters et al. (2019) compare these and find finetuning outperforms feature extraction for BERT; however, they use features immediately after pre-training, whereas we also consider features after multi-task finetuning.

[1] We consider a *task* to be synonymous with a *dataset*.

Multi-task learning (MTL) has a rich history in machine learning (Caruana, 1997; Ruder, 2017) and NLP (Collobert and Weston, 2008; Luong et al., 2016). Multi-task models can potentially leverage similarities across tasks to achieve higher end-task accuracy than single-task models (Clark et al., 2019; Liu et al., 2019a; Phang et al., 2018; Wang et al., 2019a). Compared to single-task models, a multi-task model can also be more efficient during inference by sharing computation across tasks. Most work in multi-task learning assumes that the set of end-tasks is fixed and known in advance and training is performed for all tasks together. This setup can present challenges in the real world where tasks may require different retraining schedules and new tasks may be frequently added or removed.

General purpose text encoders are usually pre-trained with a mix of supervised and self-supervised training objectives and produce fixed-size representations (Kiros et al., 2015; Hill et al., 2016; Conneau et al., 2017; Subramanian et al., 2018). Unlike multi-task learning, general purpose text encoders are typically evaluated on *unseen tasks*, which is more representative of real-world settings in which new tasks may be added periodically. Unfortunately, these approaches often underperform single-task baselines (McCann et al., 2018; Liu et al., 2019a; Wang et al., 2019a).

Another line of work has explored adapting pre-trained models by adding additional task-specific capacity at each layer (Houlsby et al., 2019), however these methods do not improve inference efficiency since there is no task-independent computation that can be shared across tasks.

Knowledge Distillation (Buciluǎ et al., 2006; Hinton et al., 2015) is a technique where a more efficient student model is trained to mimic the behaviour of a larger or ensembled teacher model. A knowledge distilled version of BERT (Sanh et al., 2019) has been proposed to reduce the computation required by large pre-trained language models. DistilRoBERTa reaches 95% of RoBERTa-base's performance on GLUE while being twice faster.

Quantization and other compression techniques have been explored for word embeddings (Shu and Nakayama, 2017; Tissier et al., 2019) and sentence embeddings (Shen et al., 2019). Recent work has also explored quantization for contextualized word representations, generally showing that quantization-aware training is necessary to achieve reasonable end task performance (Zafrir et al., 2019; Fan et al., 2020). Quantization is complementary to the approaches we consider and is explored more in Section 5.

task	type	# train	# dev	# label
MNLI	NLI	393K	20K	3
QNLI	NLI	105K	5.4K	2
QQP	PP	364K	391K	2
RTE	NLI	2.5K	3K	2
SST-2	SA	17K	1.8K	2
MRPC	PP	3.7K	1.7K	2
CoLA	LA	8.5K	1K	2
AG-news	DOC	120K	7.6K	4
Amazon-5	SA	3M	650K	5
Amazon-2	SA	3.6M	400K	2
Yelp-5	SA	650K	50K	5
Yelp-2	SA	560K	38K	2
DBpedia	DOC	560K	70K	14

Table 1: Task statistics.

3 Experimental Setup

Our goal is to develop text encoders that produce representations which achieve high accuracy for multiple task with little task-specific processing. We first introduce our tasks, encoder models and finetuning framework.

3.1 Tasks

We consider 14 text classification tasks, spanning sentiment analysis (SA), natural language inference (NLI), paraphrase identification (PP), document categorization (DOC) and linguistic acceptability (LA). Tasks are chosen for their diversity and usage in recent related work, ensuring that our baselines are representative of the state of the art.

Details about each task are given in Table 1. The SA, DOC and LA tasks consist of making predictions about a single text input, while NLI and PP tasks require classifying a pair of text inputs. For pair tasks we concatenate the text with a special separator token following Liu et al. (2019b). Since many of our tasks are part of evaluation benchmarks such as GLUE (Wang et al., 2019b) and the test sets are not publicly available, we report accuracy on the corresponding development sets.

3.2 Encoder models

Our encoder models are based on RoBERTa (Liu et al., 2019b), an optimized version of BERT (Devlin et al., 2019) that achieves competitive performance on most of the tasks considered in this work. We primarily use the public

RoBERTa$_{\text{LARGE}}$ model consisting of 24 Transformer layers (Vaswani et al., 2017), 1024 dimensional representations and 355M parameters. We refer the reader to Devlin et al. (2019) for more details about the BERT architecture and Liu et al. (2019b) for more details about RoBERTa.

We also consider a Knowledge Distilled (KD) version of RoBERTa called DistilRoBERTa (Sanh et al., 2019), which consists of 6 Transformer layers, 768-dim representations and 82M parameters. The distilled model contains 1/4 as many parameters and requires 1/7 as much computation (FLOPs) as the full model. We present a more detailed comparison of the computational requirements for these encoder models in Section 6.5.

3.3 Finetuning

We consider two methods for finetuning encoder models, illustrated in Figure 1. Finetuning hyperparameters and other methodological details are given in the Appendix.

3.3.1 Single-task finetuning

Single-task finetuning is the most common way of adapting pre-trained language models to a given task (see Figure 1a). When applied to large pre-trained models (e.g., RoBERTa) it often results in the best end-task accuracy, but requires the full model to be run for every task and thus has the highest inference costs for a set of k tasks. Computation can be reduced by using a smaller pre-trained models—including knowledge distilled models (e.g., DistilRoBERTa).

Single-task finetuning serves as an upper bound. Our goal is to achieve similar accuracy as large single-task models with reduced inference costs.

3.3.2 Leave-one-task-out finetuning

We also consider *leave-one-task-out* finetuning, illustrated in Figures 1b and 1c. We pre-train a multi-task encoder on $k-1$ tasks and extract frozen features for a k^{th} task. Freezing the encoder allows us to amortize the inference cost over tasks. The leave-one-task-out setup allows us to evaluate generalization on tasks *unseen* in the training of the encoder. This replicates the real-world setting of adding new tasks to an existing frozen encoder. Leave-one-task-out finetuning has two stages:

1. Multi-task pre-training: We train a single model end-to-end over $k-1$ tasks (Figure 1b). The majority of the encoder weights are shared across tasks, except for a classification head (see Section 3.4) that is unique to each task.

It is important for the multi-task model to properly weight different tasks, so that larger tasks do not dominate smaller ones (Raffel et al., 2019; Wang et al., 2019a). We adopt a loss-reweighting technique inspired by Raffel et al. (2019). At each step, we sample a batch of data for every task and update our model according to a sum of the losses, weighted by: $\alpha_i = D_i^{\left(\frac{1}{T}\right)} / \sum_j D_j^{\left(\frac{1}{T}\right)}$, where D_i is the number of training examples for task i and T is a temperature controlling weight uniformity. When $T = 1$, task weights are proportional to data size, and as $T \to 0$, task weights become uniform. We use a fixed temperature of $T = 0.1$, which performed best in early experiments.

2. Leave-one-task-out finetuning: In the second stage, we freeze the multi-task encoder's weights and use it as a feature extractor for an unseen k^{th} task (see Figure 1c). The extracted features are fed to a new, randomly initialized classification head, which is fine-tuned over the training data for the k^{th} task. We repeat this process k times, with each task held out once, and report the corresponding held-out task performance.

3.4 Classification heads

Each task has a *classification head* that takes features as input and makes a prediction. While related work uses task-specific classification layers (Peters et al., 2018, 2019; Liu et al., 2019a), we adopt a unified architecture for all tasks. We follow the original BERT setup (Devlin et al., 2019) and use a two-layer Multi-Layer Perceptron (MLP) with inner dimension equal to the pooled feature dimension and a `tanh` activation function. The classification head is always fine-tuned for the end task.

4 Feature extraction and pooling

A common way to extract features from BERT-like models is to take the representation in the last Transformer layer corresponding to a special `CLS` token prepended to the input sequence (Devlin et al., 2019). Recent work has also explored extracting features from every position and layer, then linearly combining the layers with task-specific weights (Peters et al., 2019; Tenney et al., 2019).

We propose a more general framework for extracting features, shown in Figure 2. We extract features from several layers of the encoder and

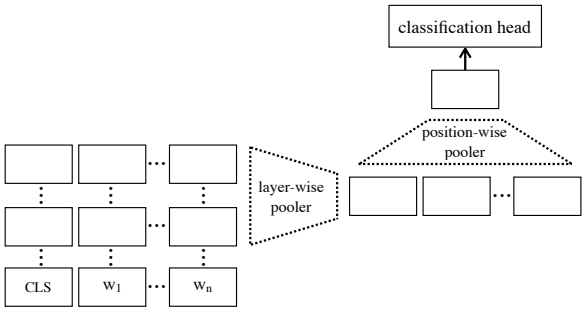


Figure 2: Features are extracted from multiple encoder layers, pooled across layers, then positions, and finally passed to a task-specific classification head. Some feature extraction and pooling approaches have additional task-specific parameters that require finetuning.

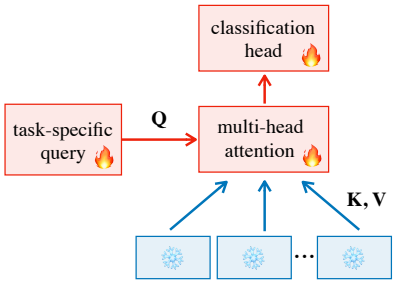


Figure 3: Our proposed multi-head attention pooler. The extracted features are frozen (❄) and used as both the keys (**K**) and values (**V**). Each task has its own query (**Q**), multi-head attention module and classification head, all of which are fine-tuned (🔥).

then *pool* them, first across layers and then across positions, before feeding them to a task-specific classification head. This framework subsumes both the `CLS` token and weighted layer combination approaches. We consider several ways of *layer-wise pooling* and *position-wise pooling*:

Layer-wise pooling approaches:

- LAST-LAYER: only use the last layer. This setting is used by Devlin et al. (2019).
- LAYER-AVG: average the last m layers. We tune m for each setting, but find that $m = 16$ works best in most cases.
- LEARNED-COMB: learn a task-specific weighted combination over all layers. This setting is used by Peters et al. (2019) and Tenney et al. (2019).

Position-wise pooling approaches:

- CLS: extract features from the first position. This setting is used by Devlin et al. (2019).
- POSITION-AVG: average features across positions.

layer-wise pooling	position-wise pooling	quantization *fp16*	*int8*	*bool*
LAST-LAYER or LAYER-AVG	CLS or POSITION-AVG	2K	1K	128
	MHA	100K	50K	6K
LEARNED-COMB	MHA	2.3M	1.2M	150K

Table 2: Estimated storage cost (in bytes) to store features for a 50 token input.

- MHA: pool features with a task-specific Multi-Head Attention (MHA) layer (Devlin et al., 2019). We learn a task-specific query and use features as the keys and values (see Figure 3).

5 Storage Considerations and Quantization

In a real-world settings it may be necessary to store extracted features for later use, such as when new tasks are introduced that require "backfilling" classifications for older content (Shen et al., 2020). Storage costs quickly become impractical when pooling over multiple hidden layers and positions, with some approaches (Section 4) requiring features from every layer and position in the encoder. For RoBERTa$_{\text{LARGE}}$, with 24 layers and 1024 dimension representations, a 50 token input would thus emit `50*24*1024` half-precision floating point numbers and require 2.3MB of storage!

We consider quantization methods, described below, for reducing the storage of extracted features. With quantization, we replace floating point numbers with alternative representation formats that have reduced bit width. We will show in Section 6 that extracted features are surprisingly robust: they show little degredation in end-task accuracy even with binary quantization. Recent work has made similar observations in the context of 8-bit integer quantization for BERT model weights and activations (Zafrir et al., 2019).

We explore both 8-bit (uint8) and 1-bit (boolean) quantization of extracted features (see Table 2). We apply quantization prior to leave-one-task-out finetuning (Section 3.3.2) to simulate a real-world setting in which only quantized features are available. For 8-bit quantization, we use PyTorch (Paszke et al., 2019) to learn scale and zero-point parameters which map floating point numbers to the range 0-255. For 1-bit quantization, we apply the `sign` function to binarize each feature dimension.

6 Results and Discussion

Table 3 presents our main results for the 14 tasks introduced in Section 3.1. Detailed results of all tasks are included in Table 5 in the Appendix.

6.1 Baselines

Table 3 (a) shows results for models fine-tuned end-to-end on a single task. This approach yields the best end-task accuracy but has the highest inference costs (see discussion in Section 3.3.1).

We observe that `DistilRoBERTa` achieves competitive accuracy across many tasks with only 1/4 as many parameters and 1/7 of the computation of the full RoBERTa model. Multi-task pre-training (see Section 3.3.2) prior to single-task fine-tuning improves results with an average gain of +0.2%. This is consistent with recent work (Liu et al., 2019a; Wang et al., 2019a), but somewhat at odds with the findings of Raffel et al. (2019), who report slightly worse performance with multi-task pre-training. It remains an open question under what conditions multi-task pre-training improves end task accuracy for single-task models.

6.2 Feature extraction and pooling

6.2.1 Without multi-task pre-training

Table 3 (b) shows results for single-task models with a frozen encoder and fine-tuned classification head. We observe that freezing the pre-trained RoBERTa model and extracting features from the last layer's `CLS` token performs poorly, with a 15% drop in accuracy compared to the end-to-end fine-tuned version (90.5% $\rightarrow$ 75.5%). This is expected, since the `CLS` token is not heavily used in the RoBERTa pre-training process (Liu et al., 2019b).[2] If we instead average features across positions in the last layer, we see slightly higher accuracy compared to the `CLS` token alone (77.7% vs. 75.5%), while our multi-head attention (MHA) pooling further improves accuracy to 83.3%, confirming the importance of task-specific position-wise pooling.

We next consider different layer-wise pooling strategies, still using the MHA position-wise pooling. Taking a simple average over the top 16 layers improves accuracy by +2.2% compared to using just the last layer (85.5% vs. 83.3%). If we instead learn a task-specific weighted combination of layers, similar to Peters et al. (2019), we gain an additional +0.1% compared to using a simple average. However, using a task-specific combination of layers introduces significant storage costs (see Table 2), thus we focus on the LAYER-AVG pooling approach in the rest of our experiments.

6.2.2 With multi-task pre-training

Table 3 (c) presents results for leave-one-task-out multi-task pre-training (Section 3.3.2), in which the encoder is fine-tuned on $k-1$ tasks, then frozen.

In this setting, the last layer's `CLS` token now encodes general task information, achieving a higher average accuracy than any of the frozen encoders which did not have leave-one-task-out multi-task pre-training (85.9% vs. 85.6%). As before, our multi-head attention (MHA) position-wise pooling strategy performs best, outperforming the `CLS` approach by +0.9% and the POSITION-AVG strategy by +0.8%. Layer-wise pooling across multiple layers provides an additional 1.6-1.7% gain.

6.3 Quantization

Table 3 (c) also shows the effect of feature quantization on task accuracy. We quantize extracted features after leave-one-task-out multi-task pre-training and use LAYER-AVG / MHA pooling, which offers the best balance between storage efficiency and accuracy. In early experiments, we considered whether to quantize before or after layer-wise pooling and found that quantization before layer-wise pooling was slightly better for 1-bit quantization and had no impact on 8-bit quantization.

We observe no performance loss with 8-bit quantization. Surprisingly, 1-bit quantization only reduces accuracy by 0.4%, still outperforming distillation-based methods (88.0% vs. 87.1%), while reducing storage costs by a factor of 16 (to 1024 bits per token; see Table 2).

To understand why quantization works so well, we use a word-sense disambiguation (WSD) task to probe if semantic information encoded in the original and quantized features is preserved. Following Peters et al. (2018), we apply a nearest neighbor classifier over word sense centroids, obtained by averaging features for each word sense over training data. We use the data and splits from Reif et al. (2019). We extract features from the 16th layer of the multi-task RoBERTa encoder (Table 3 (e)), which performed best in pilot experiments, and compare them before and after 1-bit quantization.

The F1 scores shown in Table 4 show that RoBERTa features achieve similar results before

[2] RoBERTa does not pre-train with a Next Sentence Prediction (NSP) objective, thus the `CLS` token is mostly unused.

model	G FLOPs	SA	NLI	PP	DOC	LA	AVERAGE
(a) *Single-task finetuning (end-to-end):*							
BERT	-	86.8	83.1	89.7	97.1	83.1*	87.6
XLNet	-	87.6	89.2	90.5	**97.4**	84.5*	89.5
DistilRoBERTa	61	86.6	80.7	89.6	97.1	84.3	87.1
RoBERTa	430	88.2	91.3	91.8	**97.4**	86.3	90.5
+ leave-one-task-out multi-task pre-training	430	**88.2**	**91.6**	**92.1**	**97.4**	**87.2**	**90.7**
(b) *Single-task finetuning (frozen encoder):*							
RoBERTa							
+ LAST-LAYER / CLS	31	80.8	58.8	68.2	94.9	69.1	75.5
+ LAST-LAYER / POSITION-AVG	31	80.3	63.0	75.6	95.0	75.0	77.7
+ LAST-LAYER / MHA	34	86.1	72.7	79.0	96.7	80.2	83.3
+ LAYER-AVG / MHA	34	86.9	77.7	**83.0**	**96.9**	**82.7**	85.5
+ LEARNED-COMB / MHA	34	**87.0**	**78.0**	82.8	**96.9**	82.5	**85.6**
(c) *Leave-one-task-out finetuning (frozen multi-task encoder):*							
RoBERTa							
+ LAST-LAYER / CLS	31	87.4	82.8	81.8	94.9	76.4	85.9
+ LAST-LAYER / POSITION-AVG	31	87.4	83.0	81.9	95.1	77.1	86.0
+ LAST-LAYER / MHA	34	87.5	84.6	83.5	96.2	77.2	86.8
+ LAYER-AVG / MHA	34	**87.9**	87.8	**85.7**	96.8	82.4	88.4
+ LEARNED-COMB / MHA	34	**87.9**	**87.9**	**85.7**	**96.9**	82.3	**88.5**
RoBERTa (8-bit quantization)							
+ LAYER-AVG / MHA	34	**87.9**	87.7	**85.7**	96.8	**82.6**	88.4
RoBERTa (1-bit quantization)							
+ LAYER-AVG / MHA	34	87.8	87.1	84.6	96.6	81.3	88.0
(d) *Leave-one-task-group-out finetuning (frozen multi-task encoder):*							
RoBERTa							
+ LAYER-AVG / MHA	31	87.0	81.3	85.3	96.7	82.4	86.6
(e) *Multi-task pre-training over all tasks (frozen multi-task encoder; no additional finetuning):*							
RoBERTa							
+ LAST-LAYER / CLS	31	87.7	89.6	89.3	97.2	82.6	89.3

Table 3: Results on 14 tasks, grouped by task type (see Section 3.1). We consider different layer-wise and position-wise pooling strategies introduced in Section 4. We also report the estimated inference cost for 14 tasks (in G FLOPs) for each strategy. Bold results indicate the most accurate method in each section. BERT results are from Yang et al. (2019) and Sun et al. (2019). XLNet results are from Yang et al. (2019). DistilRoBERTa and RoBERTa results are recomputed ourselves. Full results for each task is given in the Appendix. (*) we recomputed accuracy for CoLA, since BERT and XLNet originally reported a different metric.

	original	1-bit quant.
baseline (most freq. sense)	64.8	-
BERT (Reif et al., 2019)	71.1	-
RoBERTa	71.2	71.1

Table 4: WSD results (F1 score) for original and 1-bit quantized features.

and after 1-bit quantization. Both results are comparable to those from Reif et al. (2019), confirming that 1-bit quantization at least preserves word sense information in the extracted features.

6.4 Generalization

We used the *leave-one-task-out* setting to evaluate generalization to unseen tasks. We now consider the case where an entire *task type* (see Section 3.1) is held out during multi-task pre-training. For example, we pre-train an encoder over non-NLI tasks and evaluate the frozen features on NLI tasks. Results presented in the fourth section (d) of Table 3 show that performance drops considerably from the corresponding leave-one-task-out setting (Table 3 (c)). Average accuracy decreases from 88.4% to 86.6%. Accuracy on NLI tasks decreases the most from 87.8% to 81.3%, consistent with past work showing positive transfer between NLI tasks (Phang et al., 2018). Thus, it is important to pre-train the encoder over a variety of task types to maximize generalization to new tasks.

Another alternative to leaving tasks out is to pre-train the encoder over *all* k *tasks* and evaluate it on each task without additional finetuning. This setting is useful when the set of all tasks is known in advance and does not change. Results (in the final section of Table 3 (e)) show that when models are part of the multi-task finetuning they perform 3.4% better on average as opposed to when they are held out (89.3% vs. 85.9%).

6.5 Computational cost during inference

Table 3 reports cumulative inference cost (over 14 tasks) for each method. Single-task finetuning is the most accurate and the least efficient approach. Approaches using knowledge distillation and frozen encoders reduce FLOPs by an order of magnitude.

Figure 4 shows the number of FLOPs required for inference as a function of the number of tasks performed on the same text. While single-task finetuning of the full model is never efficient, distilled models are the most efficient for systems with 7 or fewer tasks. Frozen encoder approaches become the most efficient option when more than 7 tasks are prerformed on the same piece of text.

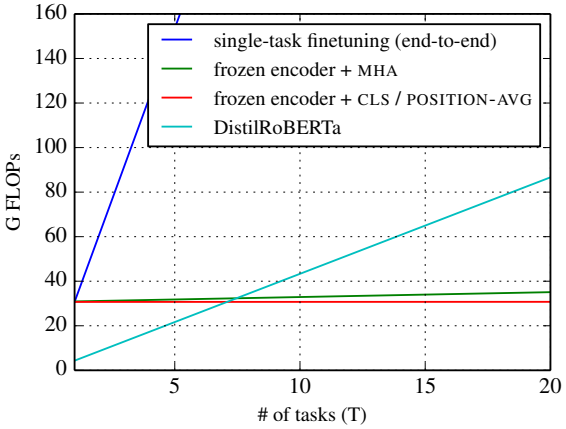


Figure 4: Estimated computational cost (in FLOPs) to run RoBERTa inference for T tasks over a single input. The cost for single-task models grows linearly with the number of tasks, whereas approaches based on a frozen encoder are much more efficient. Distilled models are particularly efficient when the number of tasks is small, but the cost scales linearly and becomes less efficient than a frozen encoder when the number of tasks $T > 7$.

7 Conclusion

We study how to improve the efficiency of large-scale pre-trained models so that can be used in practical settings. We show that when several tasks are performed on a single piece of text, the computation can be effectively amortized reducing the amount of computation per task. Compared to distillation, the shared computation method achieves higher accuracy and reduces computational cost after 7 tasks need to be performed on the same piece of text. We show that the shared features can be quantized with very little loss in accuracy, which means that the intermediate computation can be stored for later use. In total, the techniques that we present provide a compelling solution for running large-scale pre-trained models in applications where multiple predictions are made on the same piece of text.

References

Cristian Buciluǎ, Rich Caruana, and Alexandru Niculescu-Mizil. 2006. Model compression. In *Proceedings of the 12th ACM SIGKDD international*

conference on Knowledge discovery and data mining, pages 535–541. ACM.

Rich Caruana. 1997. Multitask learning. *Machine learning*, 28(1):41–75.

Kevin Clark, Minh-Thang Luong, Urvashi Khandelwal, Christopher D Manning, and Quoc V Le. 2019. Bam! born-again multi-task networks for natural language understanding. *arXiv preprint arXiv:1907.04829*.

Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM.

Alexis Conneau, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes. 2017. Supervised learning of universal sentence representations from natural language inference data. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 670–680, Copenhagen, Denmark. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *North American Association for Computational Linguistics (NAACL)*.

Angela Fan, Pierre Stock, Benjamin Graham, Edouard Grave, Remi Gribonval, Herve Jegou, and Armand Joulin. 2020. Training with quantization noise for extreme model compression.

Felix Hill, Kyunghyun Cho, and Anna Korhonen. 2016. Learning distributed representations of sentences from unlabelled data. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1367–1377, San Diego, California. Association for Computational Linguistics.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

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. *International Conference on Machine Learning (ICML)*.

Diederik Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*.

Ryan Kiros, Yukun Zhu, Ruslan R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Skip-thought vectors. In *Advances in neural information processing systems*, pages 3294–3302.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. Albert: A lite bert for self-supervised learning of language representations. In *International Conference on Learning Representations*.

Zhuohan Li, Eric Wallace, Sheng Shen, Kevin Lin, Kurt Keutzer, Dan Klein, and Joseph E Gonzalez. 2020. Train large, then compress: Rethinking model size for efficient training and inference of transformers. *arXiv preprint arXiv:2002.11794*.

Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. 2019a. Multi-task deep neural networks for natural language understanding. *arXiv preprint arXiv:1901.11504*.

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

Minh-Thang Luong, Quoc V Le, Ilya Sutskever, Oriol Vinyals, and Lukasz Kaiser. 2016. Multi-task sequence to sequence learning. In *International Conference on Learning Representations (ICLR)*.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA. Association for Computational Linguistics.

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*.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. FAIRSEQ: A fast, extensible toolkit for sequence modeling. In *North American Association for Computational Linguistics (NAACL): System Demonstrations*.

Bo Pang and Lillian Lee. 2004. A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In *Proceedings of the 42nd annual meeting on Association for Computational Linguistics*, page 271. Association for Computational Linguistics.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems (NIPS)*, pages 8024–8035.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *North American Association for Computational Linguistics (NAACL)*.

Matthew Peters, Sebastian Ruder, and Noah A Smith. 2019. To tune or not to tune? adapting pretrained representations to diverse tasks. *arXiv preprint arXiv:1903.05987*.

Jason Phang, Thibault Fvry, and Samuel R. Bowman. 2018. Sentence encoders on stilts: Supplementary training on intermediate labeled-data tasks. *arXiv preprint arXiv:1811.01088*.

Emily Pitler and Ani Nenkova. 2008. Revisiting readability: A unified framework for predicting text quality. In *Proceedings of the conference on empirical methods in natural language processing*, pages 186–195. Association for Computational Linguistics.

Alec Radford, Karthik Narasimhan, Time Salimans, and Ilya Sutskever. 2018. Improving language understanding with unsupervised learning. Technical report, OpenAI.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*.

Emily Reif, Ann Yuan, Martin Wattenberg, Fernanda B Viegas, Andy Coenen, Adam Pearce, and Been Kim. 2019. Visualizing and measuring the geometry of BERT. In *Advances in Neural Information Processing Systems*, pages 8592–8600.

Sebastian Ruder. 2017. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098*.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. DistilBERT, a distilled version of bert: smaller, faster, cheaper and lighter. In *NeurIPS EMC^2 Workshop*.

Anna Schmidt and Michael Wiegand. 2017. A survey on hate speech detection using natural language processing. In *Proceedings of the Fifth International Workshop on Natural Language Processing for Social Media*, pages 1–10.

Dinghan Shen, Pengyu Cheng, Dhanasekar Sundararaman, Xinyuan Zhang, Qian Yang, Meng Tang, Asli Celikyilmaz, and Lawrence Carin. 2019. Learning compressed sentence representations for on-device text processing. *arXiv preprint arXiv:1906.08340*.

Yantao Shen, Yuanjun Xiong, Wei Xia, and Stefano Soatto. 2020. Towards backward-compatible representation learning. *arXiv preprint arXiv:2003.11942*.

Raphael Shu and Hideki Nakayama. 2017. Compressing word embeddings via deep compositional code learning. *arXiv preprint arXiv:1711.01068*.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Empirical Methods in Natural Language Processing (EMNLP)*.

Sandeep Subramanian, Adam Trischler, Yoshua Bengio, and Christopher J Pal. 2018. Learning general purpose distributed sentence representations via large scale multi-task learning. In *International Conference on Learning Representations*.

Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. 2019. How to fine-tune bert for text classification? *arXiv preprint arXiv:1905.05583*.

Ian Tenney, Dipanjan Das, and Ellie Pavlick. 2019. Bert rediscovers the classical nlp pipeline. *arXiv preprint arXiv:1905.05950*.

Julien Tissier, Christophe Gravier, and Amaury Habrard. 2019. Near-lossless binarization of word embeddings. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7104–7111.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*.

Alex Wang, Jan Hula, Patrick Xia, Raghavendra Pappagari, R Thomas McCoy, Roma Patel, Najoung Kim, Ian Tenney, Yinghui Huang, Katherin Yu, et al. 2019a. Can you tell me how to get past sesame street? sentence-level pretraining beyond language modeling. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4465–4476.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019b. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *International Conference on Learning Representations (ICLR)*.

Diyi Yang, Alon Lavie, Chris Dyer, and Eduard Hovy. 2015. Humor recognition and humor anchor extraction. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2367–2376.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. *arXiv preprint arXiv:1906.08237*.

Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. 2019. Q8bert: Quantized 8bit bert. In *5th Workshop on Energy Efficient Machine Learning and Cognitive Computing*.

Marcos Zampieri, Shervin Malmasi, Preslav Nakov, Sara Rosenthal, Noura Farra, and Ritesh Kumar. 2019. Semeval-2019 task 6: Identifying and categorizing offensive language in social media (offenseval). In *Proceedings of the 13th International Workshop on Semantic Evaluation*, pages 75–86.

Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *Advances in Neural Information Processing Systems (NIPS)*, pages 649–657.

A Finetuning Methodology

We largely adopt the finetuning procedure and hyperparameters from Liu et al. (2019b). We use the Adam optimizer (Kingma and Ba, 2015) with $\beta_1 = 0.9, \beta_2 = 0.98, \epsilon = 1e-6$. We search over learning rates $\in$ {1,2,3}e-5 and batch sizes $\in$ {16,32} for each task. We finetune for 10 epochs[3] and report the best dev set accuracy for each task, which we measure at epoch boundaries. We linearly warm up the learning rate for the first 6% of finetuning updates and linearly decay the rate to 0 for the remaining updates. We apply dropout with $p = 0.1$ and finetune with weight decay of 0.01.

For finetuning the multi-task encoders, we use a learning rate of 1e-5 and batches consisting of 4 samples from each task. For example, a leave-one-task-out encoder for MNLI would be finetuned on batches containing 4 samples from each of the (13) non-MNLI tasks, for a total batch size of 52. As described in Section 3.3.1, the task losses are weighted with a mixing temperature $\alpha = 0.1$ and summed, following Raffel et al. (2019).

We perform our experiments using the FAIRSEQ toolkit (Ott et al., 2019), PyTorch (Paszke et al., 2019) and Nvidia V100 GPUs. We train with mixed precision, following Liu et al. (2019b).

B Detailed Results

The results for all setups over 14 tasks can be found in Table 5.

[3] Due to the large size of the training sets, we finetune for only a single epoch for the Amazon-{2,5} data, and for only five epochs for the Yelp-{2,5} and DBpedia data.

Model	MNLI	QNLI	QQP	RTE	SST2	MRPC	CoLA	IMDB	AG	Amzn5	Amzn2	Yelp5	Yelp2	DBpd	Avg
(a) *Single-task finetuning (end-to-end):*															
BERT	86.6	92.3	91.3	70.4	93.2	88.0	83.1	95.5	94.8	65.8	97.4	70.7	98.1	99.4	87.6
XLNet	89.8	93.9	91.8	83.8	95.6	89.2	84.5	96.2	95.5	67.7	97.4	70.7	98.1	99.4	89.5
DistilRoBERTa	83.9	91.0	91.2	67.2	93.5	88.0	84.3	94.4	94.9	66.3	97.1	70.5	97.9	99.3	87.1
RoBERTa	90.3	94.6	92.3	88.9	96.7	91.3	86.3	96.4	95.4	67.9	97.6	71.9	98.4	99.3	90.5
+ leave-one-task-out multi-task pre-training	90.3	94.6	92.2	89.9	96.7	92.1	87.2	96.6	95.5	68.0	97.6	72.2	98.4	99.3	90.7
(b) *Single-task finetuning (frozen encoder):*															
RoBERTa															
+ LAST-LAYER / CLS	55.7	67.4	67.4	53.3	84.0	69.0	69.1	89.1	91.0	58.2	94.6	62.7	96.2	98.7	75.5
+ LAST-LAYER / POSITION-AVG	56.7	72.7	80.3	59.7	88.1	70.9	75.0	87.2	91.5	57.3	93.7	60.6	95.0	98.4	77.7
+ LAST-LAYER / MHA	75.7	81.6	86.0	60.7	92.5	71.9	80.3	94.3	94.1	65.1	96.9	69.9	98.0	99.3	83.3
+ LAYER-AVG / MHA	83.1	87.3	88.1	62.8	94.3	77.9	82.7	95.5	94.4	65.9	97.2	70.6	98.2	99.3	85.5
+ LEARNED-COMB / MHA	83.4	87.4	88.1	63.1	94.4	77.4	82.5	95.5	94.5	66.0	97.2	70.8	98.2	99.3	85.6
ALBERT (Lan et al., 2020)															
+ LAST-LAYER / CLS	67.2	72.4	78.3	53.2	86.8	71.6	69.9	90.1	92.2	60.1	93.7	66.6	98.0	97.8	78.4
+ LAST-LAYER / POSITION-AVG	64.6	77.9	79.2	59.4	86.9	75.0	69.9	89.1	93.2	59.2	93.7	66.8	98.0	98.0	79.4
+ LAST-LAYER / MHA	82.8	88.8	86.7	64.2	91.6	75.5	83.3	95.5	94.5	65.5	97.0	70.0	98.2	99.3	85.3
+ LAYER-AVG / MHA	84.2	89.1	88.4	67.8	95.2	80.1	84.7	95.5	94.4	65.9	97.2	70.8	98.2	99.3	86.5
+ LEARNED-COMB / MHA	84.4	90.2	88.9	73.1	95.8	84.1	86.5	95.5	94.4	65.9	97.2	70.8	98.2	99.3	87.5
(c) *Leave-one-task-out finetuning (frozen multi-task encoder):*															
RoBERTa															
+ LAST-LAYER / CLS	76.2	84.3	84.7	87.8	94.0	78.8	76.4	96.7	90.7	66.4	97.6	71.2	98.7	99.1	85.9
+ LAST-LAYER / POSITION-AVG	76.3	84.6	84.7	88.2	93.8	79.1	77.1	96.7	91.1	66.3	97.6	71.0	98.7	99.0	86.0
+ LAST-LAYER / MHA	79.8	87.6	86.2	86.5	94.1	80.8	77.2	96.7	93.2	66.5	97.6	71.5	98.7	99.2	86.8
+ LAYER-AVG / MHA	86.6	91.6	88.7	85.1	96.0	82.7	82.4	96.7	94.4	66.8	97.6	71.7	98.7	99.3	88.4
+ LEARNED-COMB / MHA	87.0	91.7	88.8	85.1	96.1	82.7	82.3	96.7	94.4	66.8	97.6	71.8	98.7	99.3	88.5
RoBERTa (8-bit quantization)															
+ LAYER-AVG / MHA	86.7	91.5	88.6	85.1	96.0	82.7	82.6	96.7	94.4	66.7	97.6	71.6	98.7	99.3	88.4
RoBERTa (1-bit quantization)															
+ LAYER-AVG / MHA	85.5	90.8	88.2	85.1	95.8	81.0	81.3	96.5	93.9	66.6	97.6	71.4	98.7	99.3	88.0
(d) *Leave-one-task-group-out finetuning (frozen multi-task encoder):*															
RoBERTa															
+ LAYER-AVG / MHA	85.2	90.4	88.7	68.3	94.3	82.0	82.4	95.6	94.2	65.8	97.2	70.6	98.2	99.3	86.6
(e) *Multi-task pre-training over all tasks (frozen multi-task encoder; no additional finetuning):*															
RoBERTa															
+ LAST-LAYER / CLS	89.3	93.7	89.6	85.9	94.8	89.0	82.6	96.7	95.1	66.5	97.5	71.9	98.6	99.3	89.3

Table 5: Extended results table for all 14 tasks. See Table 3 for more details.