ReLoRA: Use PEFT for pre-training, not just for fine-tuning

ReLoRA:
High-Rank Training Through Low-Rank updates

Vladislav Lialin,
Sherin Muckatira,
Namrata Shivagunde,
Anna Rumshisky

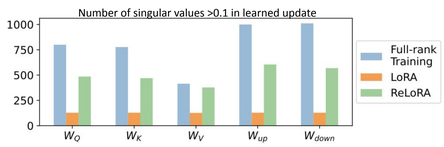
BACKGROUND: Parameter-efficient fine-tuning (PEFT) methods have democratized large model fine-tuning by reducing memory and compute requirements. Can we apply them for pre-training as well and pre-train language models much cheaper while keeping the performance?

METHOD

1. LoRA all linear layers
2. Merge + reset LoRA every ~5K iters
3. Prune 95% optimizer state every reset
4. Re-warmup learning rate every reset

RESULTS

- ReLoRA performs a high-rank update through a sequence of low-rank updates

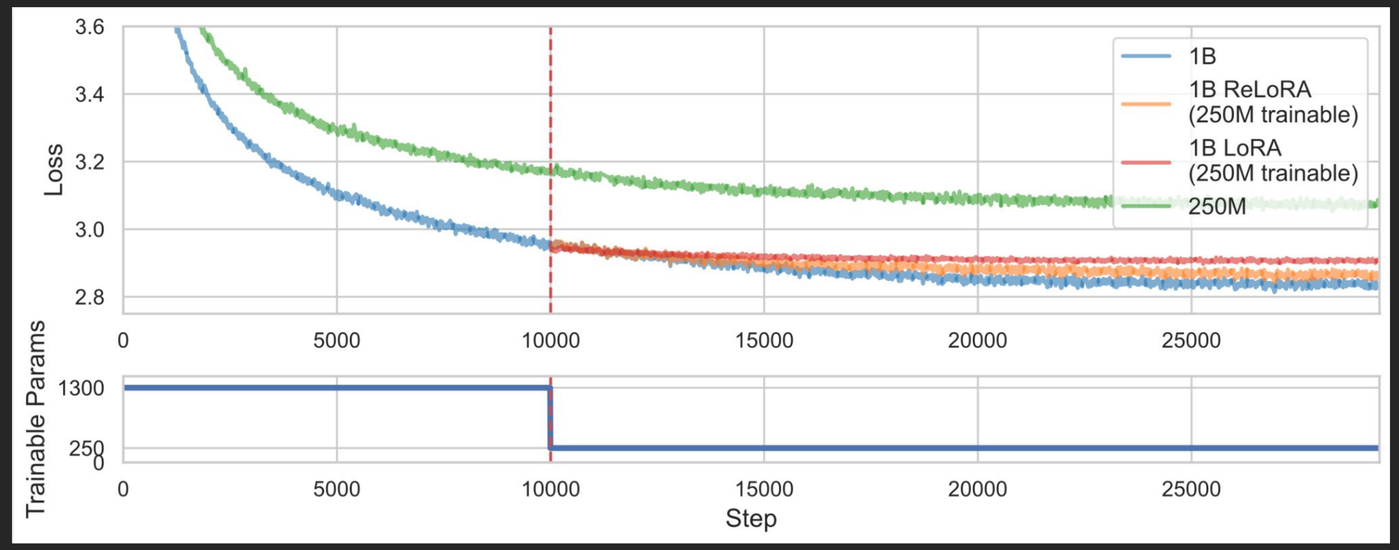


- ReLoRA is the first PEFT method that achieves similar performance to full-rank training **at scale**

	60M	130M	250M	350M	1.3B
Full training	33.81 (60M)	23.65 (130M)	22.39 (250M)	18.66 (350M)	16.83 (250M)
Control	36.52 (43M)	27.30 (72M)	25.43 (99M)	23.65 (130M)	21.73 (250M)
LoRA	47.44 (43M)	34.17 (72M)	36.60 (99M)	57.11 (125M)	-
LoRA + Warm Start	**34.73** (43M)	**25.46** (72M)	**22.86** (99M)	**19.73** (125M)	**18.23** (250M)
ReLoRA	**34.46** (43M)	**25.04** (72M)	**22.48** (99M)	**19.32** (125M)	**17.27** (250M)
Training tokens	1.2B	2.6B	6.8B	6.8B	23.1B

- This allows to use ReLoRA during pre-training and achieve the following speedups @1B

	8xA100	6xA6000 (Ada)	2x3090
Full-rank throughput	137 ex/sec	84 ex/sec	8.8 ex/sec
ReLoRA throughput	157 ex/sec	124 ex/sec	17.8 ex/sec
Immediate speedup	15%	48%	102%
Warm-start adjusted ReLoRA throughput	149 ex/sec	111 ex/sec	14.8 ex/sec
Total speedup	9%	32%	51%




```python
model = ReLoRA(model)
# Train LoRA, LayerNorm and Embeddings
optimizer = Adam(model.trainable_parameters())
scheduler = JaggedScheduler(optimizer, rewarmup_steps=100)

for step, batch in enumerate(dataloader):
    loss = model(**batch)

    loss.backward()
    optimizer.step()
    scheduler.step()

    if step % reset_freq == 0:
        model.merge_and_reinit_lora()
        prune(optimizer)
        scheduler.begin_rewarmup()
```

```python
class ReLoRALayer:
    def __init__(self, in_dim, out_dim, r):
        self.weight = tensor(in_dim, out_dim, requires_grad=False)
        self.bias = tensor(out_dim)

        self.lora_A = tensor(in_dim, r)
        self.lora_B = tensor(r, out_dim)
        self.scale = 1/r

    def forward(self, x):
        h = x @ self.weight + self.bias
        h += self.scale * x @ self.lora_A @ self.lora_B
        return h

    def merge_and_reinit(self):
        self.weight += self.scale * self.lora_A @ self.lora_B
        init.zeros_(self.lora_A)
        init.kaiming_(self.lora_B)
```

Github

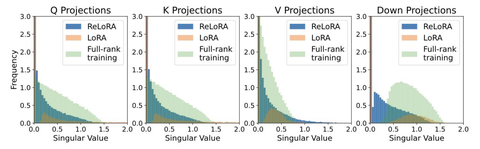
Pre-training Ablations (perplexity)

Restarts	Optimizer Reset	Jagged Schedule	Warm Start	Perplexity (↓)
×	×	×	×	34.17
✓	×	×	×	34.25
✓	✓	×	×	*(diverged)*
✓	×	✓	×	34.29
✓	✓	✓	×	29.77
×	×	×	✓	25.46
✓	✓	✓	✓	25.04
Regular training				23.65

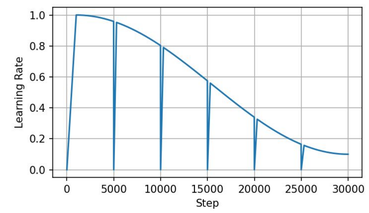
Downstream performance of 350M models pre-trained in this study

Model	SST-2	MNLI	QNLI	QQP	RTE	STS-B	MRPC	CoLA	Avg
Regular	89.0	71.0	83.4	84.5	**64.4**	**83.9**	77.0	**35.4**	**73.5**
ReLoRA-pretrained	89.5	**72.3**	**83.9**	**86.0**	57.4	83.3	78.4	31.1	72.7

Singular values spectra of the weight difference between ReLoRA and LoRA at 5,000 iterations (warm start) and 20,000 iterations



Jagged Scheduler example (re-warmups are exaggerated)



UMASS LOWELL

EleutherAI