😱 Сюжеты, возникающие при обучении больших нейросетевых моделей.

Даня Меркулов

AIRI / Университет Сириус

```
RuntimeError: cuda runtime error (2) : out of memory at /data/users/soumith/miniconda2/cond
```

how can i solve this error?

apaszke commented on Mar 8, 2017 Member

You're running out of memory on the GPU. It's not a bug.

😄 16 🎉 3

$p \approx 160\cdot 10^9$ $\quad \min\limits_{x\in\mathbb{R}^p} \frac{1}{N}\sum_{i=1}^{N} f_i(x) = f(x)$ $\quad N \approx 10^{10}, 10^{11}$

обуч. выборка

$$\nabla f = \frac{1}{N}\sum_{i=1}^{N} \nabla f_i(x)$$

🎁 Large batch training

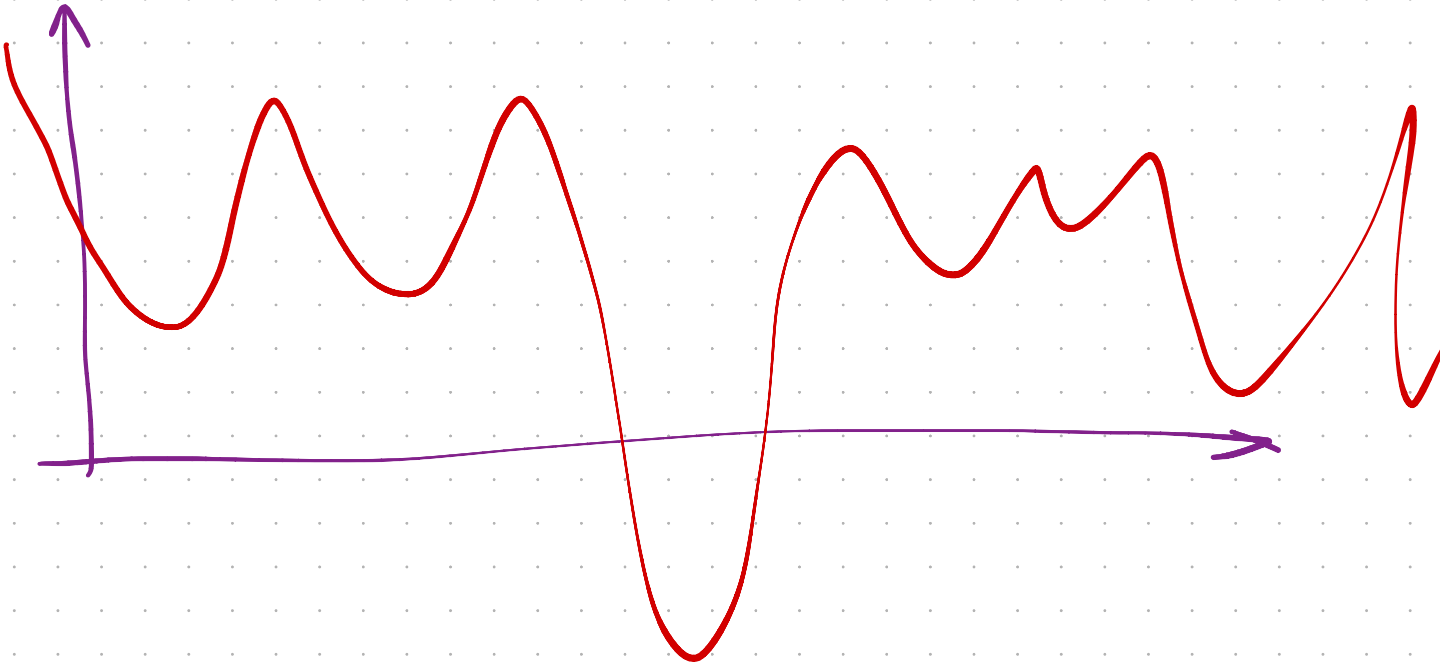
$$g = \frac{1}{b}\sum_{i=1}^{b} \nabla f_i(x)$$

$b \approx 64$

128

256

$g \approx \nabla f$



БОЛЬШЕ b $\Rightarrow$ меньше дисперсия g

ПАРАЛЛЕЛИЗМ.

MEGATRON
DEEPSPEED

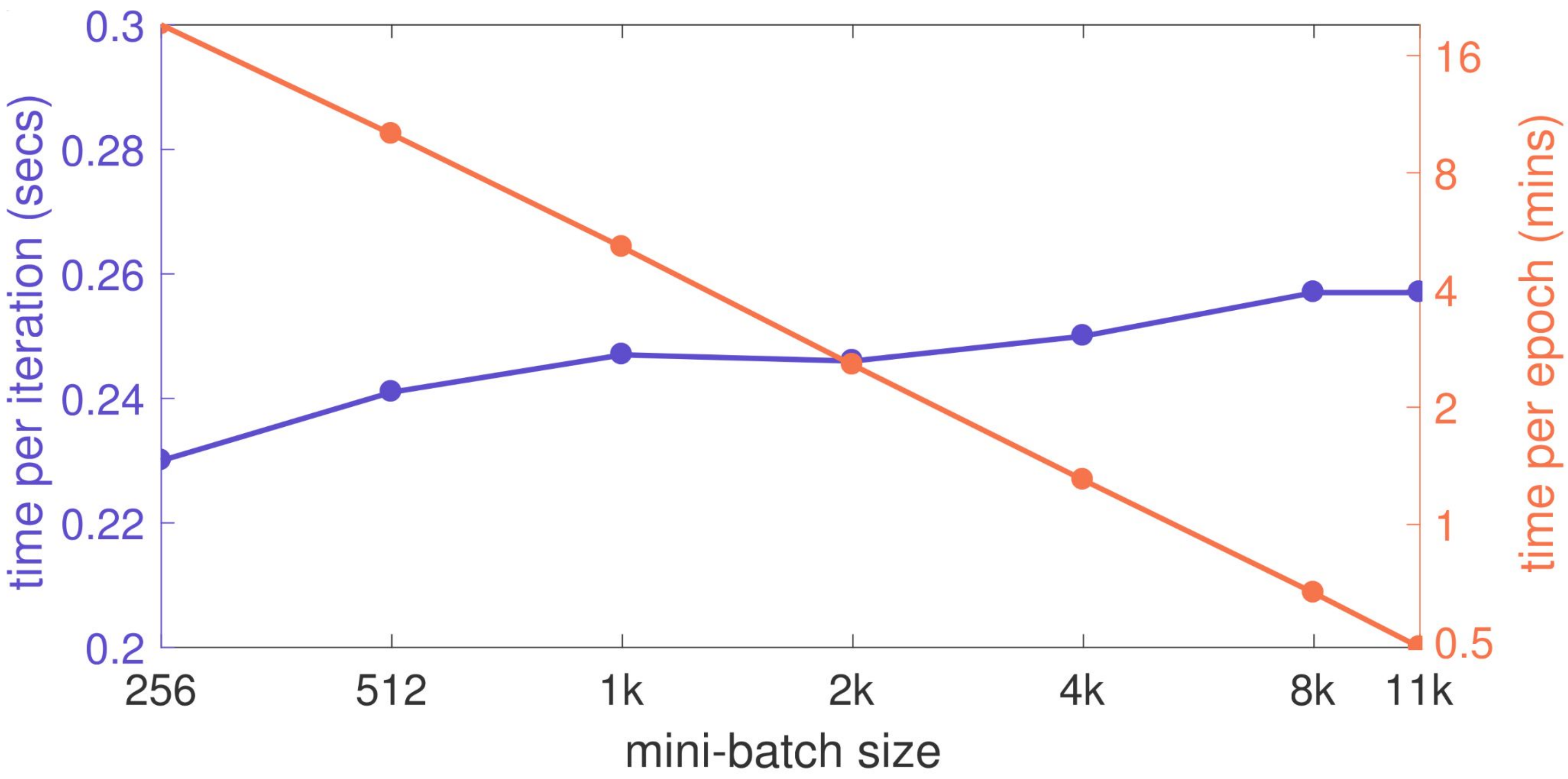
x_k $\quad g_k \quad x_{k+1} = x_k - \eta \cdot g_k$

$$\boxed{x_{k+t} = x_k - \sum_{i=k}^{k+t} \eta \cdot g_i}$$

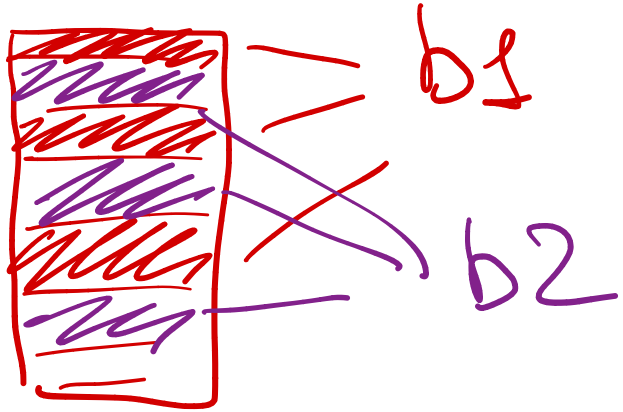
$$\boxed{x_{k+1} = x_k - t \cdot \eta \cdot g_k}$$

$b = 10^2$ 10^8 шагов 10^{10} $b = 10^5$ iter per epoch 10^5

Размер батча и время, затрачиваемое на одну эпоху.



При наличии достаточной памяти GPU, увеличение размера батча позволяет утилизировать ресурсы параллельных вычислений.



📜 **Paper**
Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour.

AIRI / Университет Сириус

Просто так увеличить размер батча не получится.

kn	η	top-1 error (%)
256	0.05	23.92 ±0.10
256	0.10	23.60 ±0.12
256	0.20	23.68 ±0.09
8k	$0.05 \cdot 32$	24.27 ±0.08
8k	$0.10 \cdot 32$	23.74 ±0.09
8k	$0.20 \cdot 32$	24.05 ±0.18
8k	0.10	41.67 ±0.10
8k	$0.10 \cdot \sqrt{32}$	26.22 ±0.03

(a) **Comparison of learning rate scaling rules.** A reference learning rate of $\eta = 0.1$ works best for $kn = 256$ (23.68% error). The linear scaling rule suggests $\eta = 0.1 \cdot 32$ when $kn = 8k$, which again gives best performance (23.74% error). Other ways of scaling η give worse results.

Обучение ResNet-50 на датасете ImageNet с разными вариантами увеличения размера батча.

📜 **Paper**
Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour.

Просто так увеличить размер батча не получится.

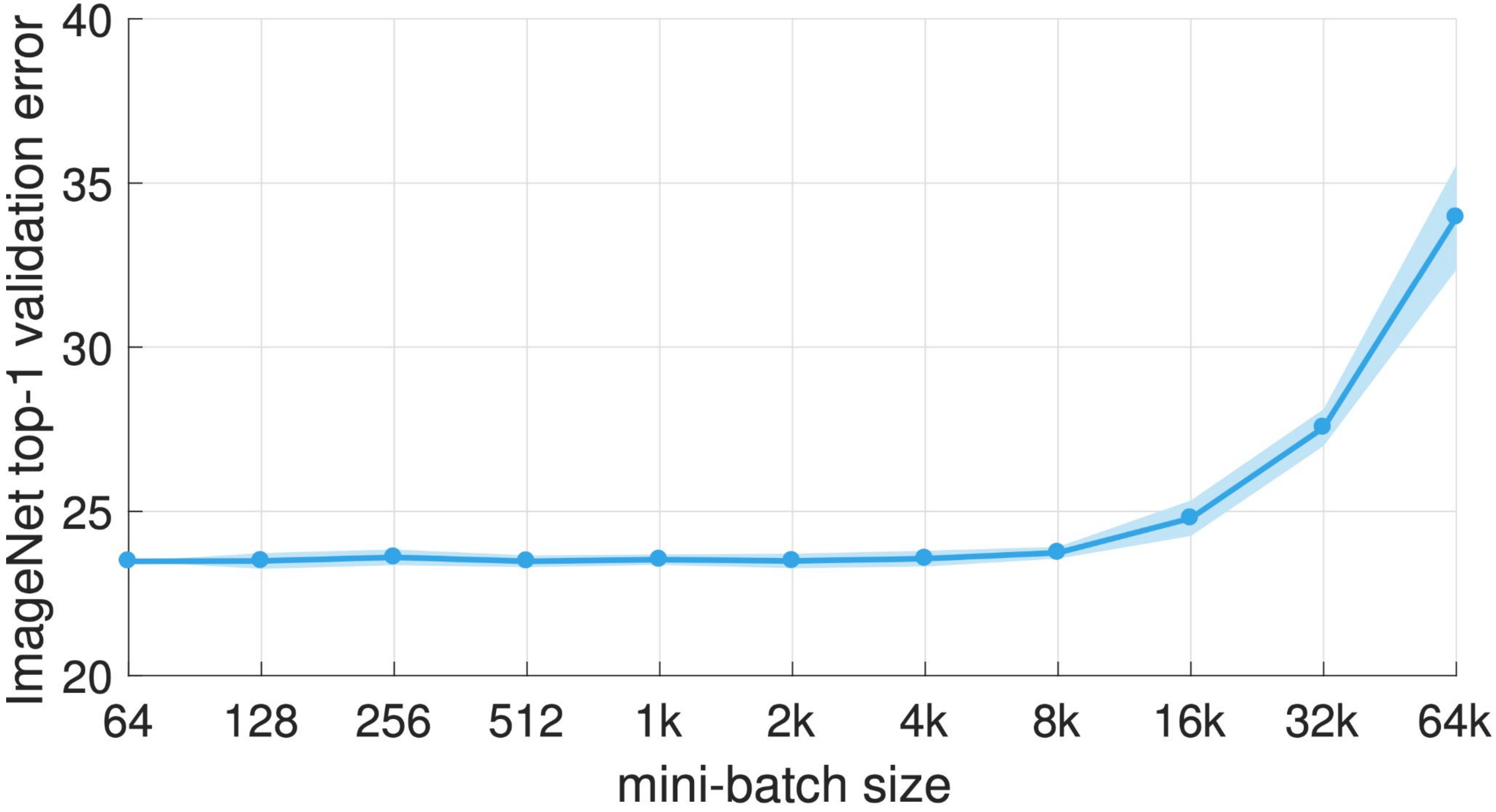
kn	η	top-1 error (%)
256	0.05	23.92 ±0.10
256	0.10	23.60 ±0.12
256	0.20	23.68 ±0.09
8k	$0.05 \cdot 32$	24.27 ±0.08
8k	$0.10 \cdot 32$	23.74 ±0.09
8k	$0.20 \cdot 32$	24.05 ±0.18
8k	0.10	41.67 ±0.10
8k	$0.10 \cdot \sqrt{32}$	26.22 ±0.03

(a) **Comparison of learning rate scaling rules.** A reference learning rate of $\eta = 0.1$ works best for $kn = 256$ (23.68% error). The linear scaling rule suggests $\eta = 0.1 \cdot 32$ when $kn = 8$k, which again gives best performance (23.74% error). Other ways of scaling η give worse results.

Обучение ResNet-50 на датасете ImageNet с разными вариантами увеличения размера батча.

📜 **Paper**
Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour.

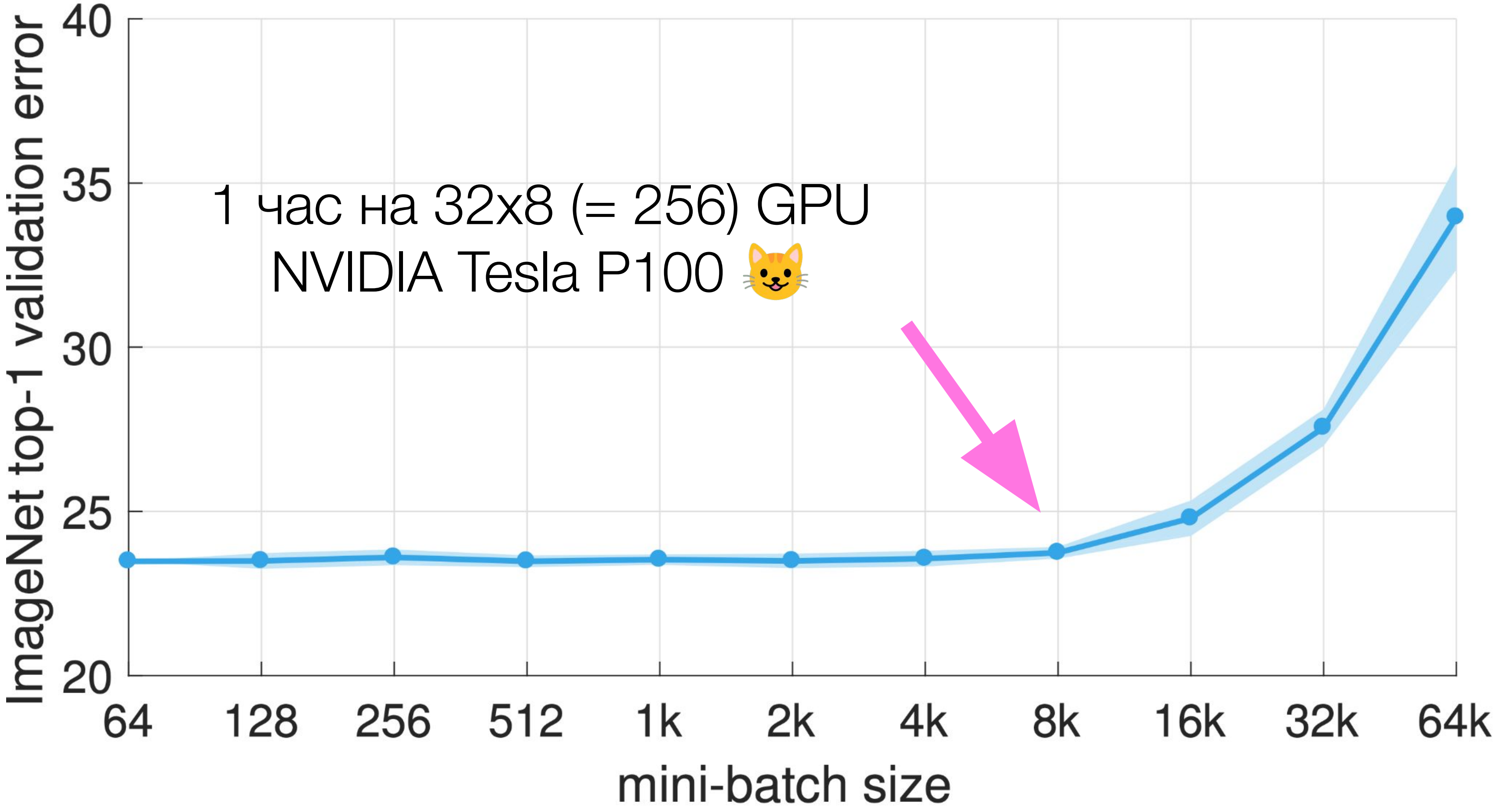
Просто так увеличить размер батча не получится.



При наличии достаточной памяти GPU, увеличение размера батча позволяет утилизировать ресурсы параллельных вычислений.

📜 **Paper**
Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour.

Просто так увеличить размер батча не получится.



При наличии достаточной памяти GPU, увеличение размера батча позволяет утилизировать ресурсы параллельных вычислений.

📜 **Paper**
Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour.

Как увеличивать размер батча?

1. Linear scaling rule.

   При увеличении размера батча в k раз, learning rate увеличивать тоже в k раз:

$$\hat{\alpha} = k\alpha$$

$$\hat{\alpha} = \sqrt{k} \cdot \alpha$$

2. Gradual warmup.

   На первых эпохах (итерациях) learning rate увеличивать постепенно от начального до целевого.

3. Square root scaling rule

лучше для AdamW, RmSprop

📜 **Paper**
Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour.

📜 **Paper**
Learning Rates as a Function of Batch Size: A Random Matrix Theory Approach to Neural Network Training.

Как ещё увеличивать размер батча?

LARS (Layer-wise Adaptive Rate Scaling):

Linear scaling rule + для каждого слоя свой learning rate, который шкалируется на норму весов этого слоя.

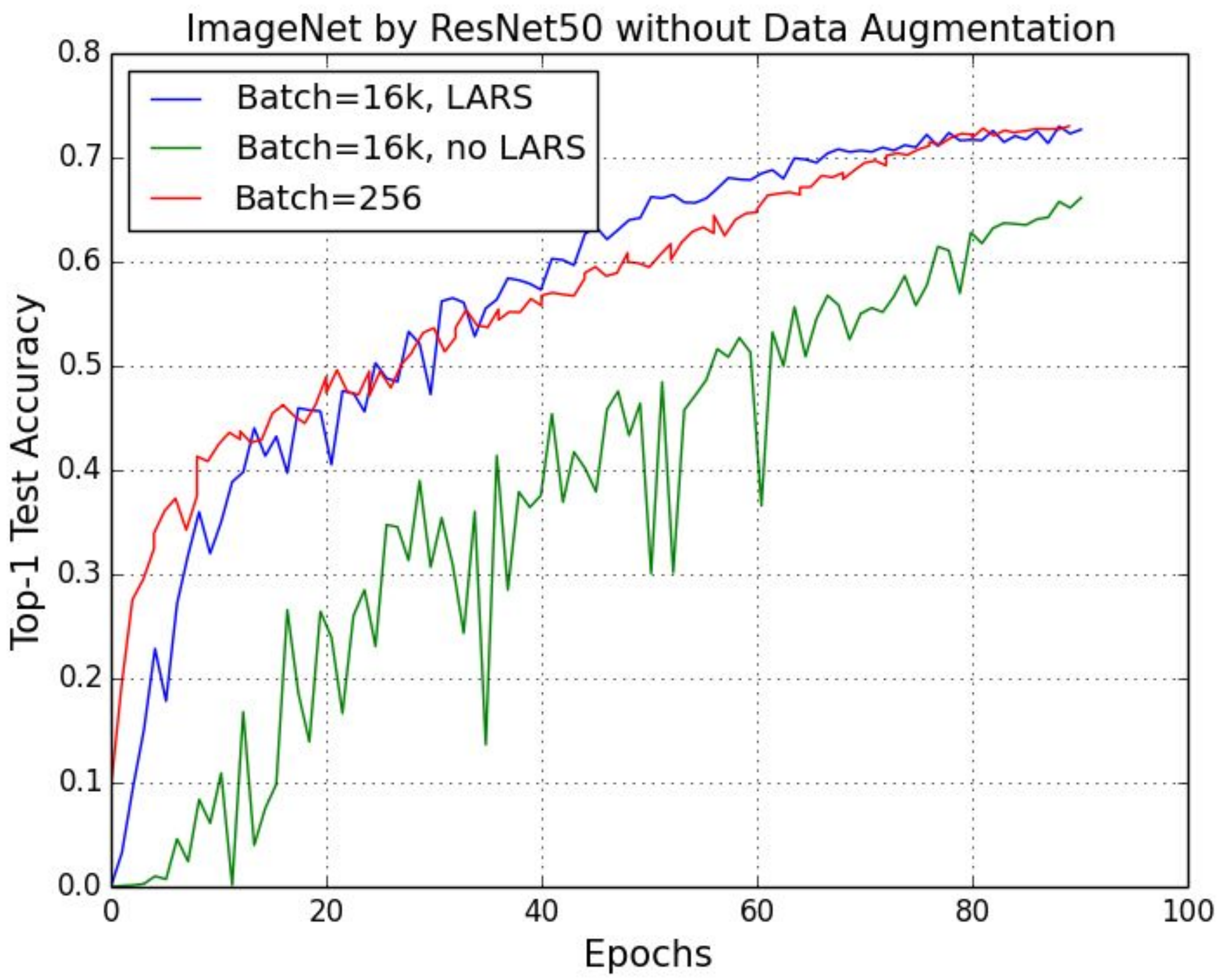
$$\hat{\alpha}_l = k\alpha_l \frac{\|w_l\|_2}{\|\nabla_{w_l} L\|_2},$$

$$l = 1, \dots, N_{layers}$$

📜 **Paper**

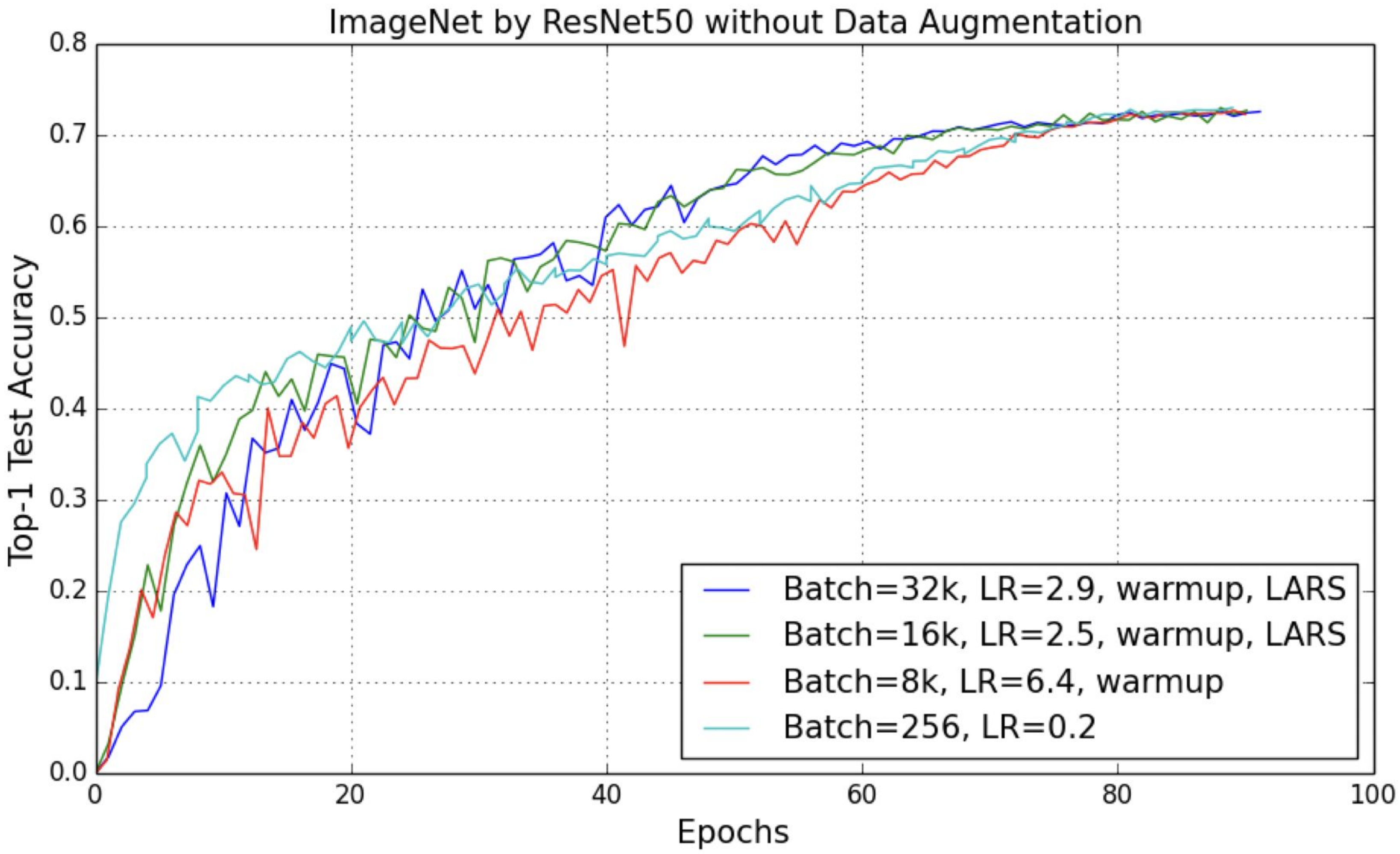
Large batch training of convolutional networks.

Как ещё увеличивать размер батча? LARS.



📜 **Paper**
Large batch training of convolutional networks.

Как ещё увеличивать размер батча? LARS.



📜 **Paper**

Large batch training of convolutional networks.

Как ещё увеличивать размер батча?

LAMB = LARS + Adam 🦄

Algorithm 1 LARS

Input: $x_1 \in \mathbb{R}^d$, learning rate $\{\eta_t\}_{t=1}^T$, parameter $0 < \beta_1 < 1$, scaling function ϕ, $\epsilon > 0$

Set $m_0 = 0$

for $t = 1$ **to** T **do**

  Draw b samples S_t from $\mathbb{P}$

  Compute $g_t = \frac{1}{|\mathcal{S}_t|} \sum_{s_t \in \mathcal{S}_t} \nabla \ell(x_t, s_t)$

  $m_t = \beta_1 m_{t-1} + (1 - \beta_1)(g_t + \lambda x_t)$

  $x_{t+1}^{(i)} = x_t^{(i)} - \eta_t \frac{\phi(\|x_t^{(i)}\|)}{\|m_t^{(i)}\|} m_t^{(i)}$ for all $i \in [h]$

end for

Algorithm 2 LAMB

Input: $x_1 \in \mathbb{R}^d$, learning rate $\{\eta_t\}_{t=1}^T$, parameters $0 < \beta_1, \beta_2 < 1$, scaling function ϕ, $\epsilon > 0$

Set $m_0 = 0$, $v_0 = 0$

for $t = 1$ **to** T **do**

  Draw b samples S_t from $\mathbb{P}$.

  Compute $g_t = \frac{1}{|\mathcal{S}_t|} \sum_{s_t \in \mathcal{S}_t} \nabla \ell(x_t, s_t)$.

  $m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$

  $v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$

  $m_t = m_t / (1 - \beta_1^t)$

  $v_t = v_t / (1 - \beta_2^t)$

  Compute ratio $r_t = \frac{m_t}{\sqrt{v_t} + \epsilon}$

  $x_{t+1}^{(i)} = x_t^{(i)} - \eta_t \frac{\phi(\|x_t^{(i)}\|)}{\|r_t^{(i)} + \lambda x_t^{(i)}\|} (r_t^{(i)} + \lambda x_t^{(i)})$

end for

📜 **Paper**

Large batch optimization for Deep Learning: training BERT in 76 minutes.

Как ещё увеличивать размер батча? LAMB.

Solver	batch size	steps	F1 score on dev set	TPUs	Time
Baseline	512	1000k	90.395	16	81.4h
LAMB	512	1000k	91.752	16	82.8h
LAMB	1k	500k	91.761	32	43.2h
LAMB	2k	250k	91.946	64	21.4h
LAMB	4k	125k	91.137	128	693.6m
LAMB	8k	62500	91.263	256	390.5m
LAMB	16k	31250	91.345	512	200.0m
LAMB	32k	15625	91.475	1024	101.2m
LAMB	64k/32k	8599	90.584	1024	76.19m

📜 **Paper**
Large batch optimization for Deep Learning: training BERT in 76 minutes.

Как ещё увеличивать размер батча? LAMB.

Solver	batch size	steps	F1 score on dev set	TPUs	Time
Baseline	512	1000k	90.395	16	81.4h
LAMB	512	1000k	91.752	16	82.8h
LAMB	1k	500k	91.761	32	43.2h
LAMB	2k	250k	91.946	64	21.4h
LAMB	4k	125k	91.137	128	693.6m
LAMB	8k	62500	91.263	256	390.5m
LAMB	16k	31250	91.345	512	200.0m
LAMB	32k	15625	91.475	1024	101.2m
LAMB	64k/32k	8599	90.584	1024	76.19m

76 минут
на 1024 TPU 😺

📜 **Paper**
Large batch optimization for Deep Learning: training BERT in 76 minutes.

🔋 Gradient accumulation

Идея аккумуляции градиента.

$$g = \frac{1}{b}\sum_{i=1}^{b} \nabla f_i(x)$$

$\tilde{b} = 1$ (влезло)

$$g_1 = \frac{\nabla f_{i_1}(x)}{60}$$

$$g_2 = \frac{\nabla f_{i_2}(x)}{10}$$

$\vdots$

$$g_{10} = \frac{\nabla f_{i_{10}}(x)}{10}$$

$g_1 + \dots g_{10} = \tilde{g}$

$x_{k+1} = x_k - \tilde{g}$

Gradient accumulation

▶️ БЫЛО:

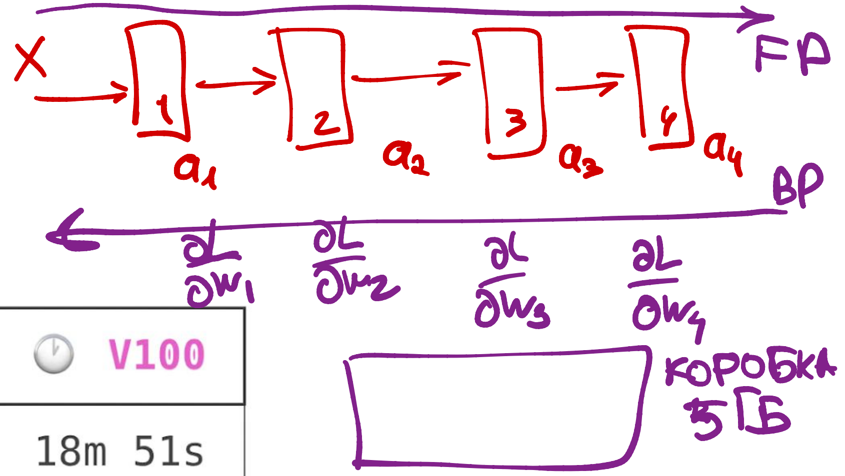
```python
# loop through batches
for (inputs, labels) in data_loader:

    # extract inputs and labels
    inputs = inputs.to(device)
    labels = labels.to(device)

    # passes and weights update
    with torch.set_grad_enabled(True):

        # forward pass
        preds = model(inputs)
        loss  = criterion(preds, labels)

        # backward pass
        loss.backward()

        # weights update
        optimizer.step()
        optimizer.zero_grad()
```

⏩ СТАЛО:

```python
# batch accumulation parameter
accum_iter = 4

# loop through enumaretad batches
for batch_idx, (inputs, labels) in enumerate(data_loader):

    # extract inputs and labels
    inputs = inputs.to(device)
    labels = labels.to(device)

    # passes and weights update
    with torch.set_grad_enabled(True):

        # forward pass
        preds = model(inputs)
        loss  = criterion(preds, labels)

        # normalize loss to account for batch accumulation
        loss = loss / accum_iter

        # backward pass
        loss.backward()

        # weights update
        if ((batch_idx + 1) % accum_iter == 0) or (batch_idx + 1 == len(data_loader)):
            optimizer.step()
            optimizer.zero_grad()
```

🎮 Ссылки

🎮 Ссылки. Gradient checkpointing.



🌏 **Site**
Training larger-than-memory PyTorch models using gradient checkpointing.

checkpointing

	Batch Size	Memory	🕐 T4	🕐 V100
✘	64	9.42 GB	47m 30s	18m 51s
✓	64	3.71 GB	1h 0m 34s	23m 48s

🌏 **Site**
Hugging Face. Efficient training on a single GPU.

```python
training_args = TrainingArguments(
    per_device_train_batch_size=1, gradient_accumulation_steps=4, gradient_checkpointing=True,
)
```

GPT3small_Puskin without checkpointing (batch = 8,sec_len =512)

index	used_mem	delta_mem	delta_time
begin	5804	0	0
forward	13006	7202	0.04
backward	14576	8772	1.2
optim_step	14576	8772	0.02
end	5840	36	0.13
total	15109.75	0	1.396

GPT3small_Puskin with checkpointing (batch = 8,sec_len =512)

index	used_mem	delta_mem	delta_time
begin	4828	0	0
forward	6398	1570	0.45
backward	7968	3140	1.14
optim_step	7968	3140	0.05
end	4828	0	0.01
total	15109.75	0	1.655

🎮 Ссылки. Activation quantization.

👨‍💻 **Repository**
FewBit - Compression schema for gradient of activations in backward pass

	Task	Batch Size	GELU	Linear Layer	Peak Memory, GiB	Saving, %
1	MRPC	128	Vanilla	Vanilla	11.30	0.0
2	MRPC	128	3-bit	Vanilla	9.75	13.8
3	MRPC	128	Vanilla	Randomized	9.20	18.6
4	MRPC	128	3-bit	Randomized	7.60	32.7

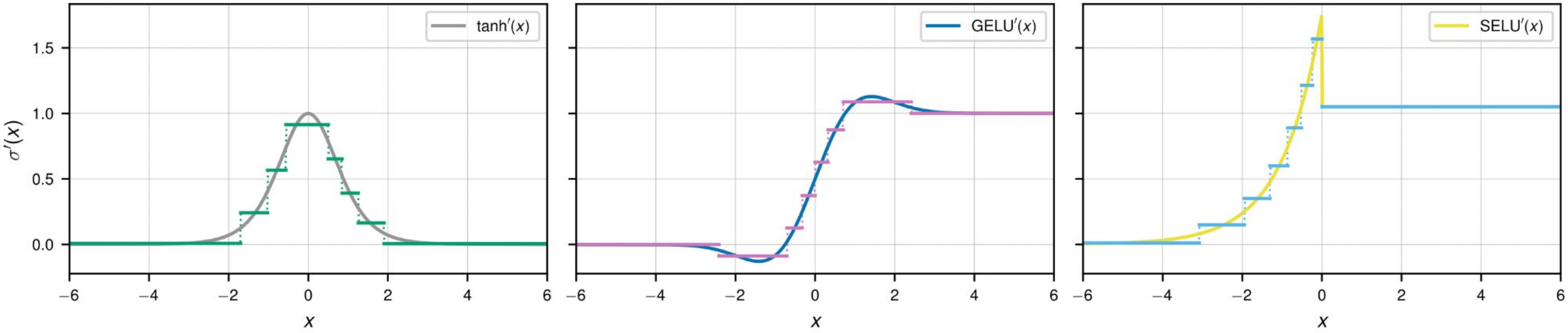


Figure 1. Optimized 3-bit piecewise-constant approximations of the derivatives of activation functions.

🎮 Ссылки. Automatic Mixed Precision training.

👨‍💻 Repository
Automatic Mixed Precision Tutorials using pytorch

Fine-tuning **32-bit** model for 215 iterations.
Time: **01:32**
Peak GPU memory consumption: **8325 MB**
Loss: 2.7370730377906978

Fine-tuning **16-bit** model for 215 iterations.
Time: 01:01
Peak GPU memory consumption: **6691 MB**
Loss: 2.737271473019622

Gradient descent without gradient
Learning to learn

🌏 Site
PyTorch: CUDA Automatic Mixed Precision training examples

- B : Baseline (FP32)
- AMP : Automatic Mixed Precision Training (AMP)

Algorithm	Test Accuracy	GPU Memory	Total Training Time
B - 1080 Ti	94.13	10737MB	64.9m
B - 2080 Ti	94.17	10855MB	54.3m
AMP - 1080 Ti	94.07	6615MB	64.7m
AMP - 2080 Ti	94.23	7799MB	37.3m

Post-local SGD

Практика