Сюжеты, возникающие при обучении больших моделей

Даня Меркулов

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

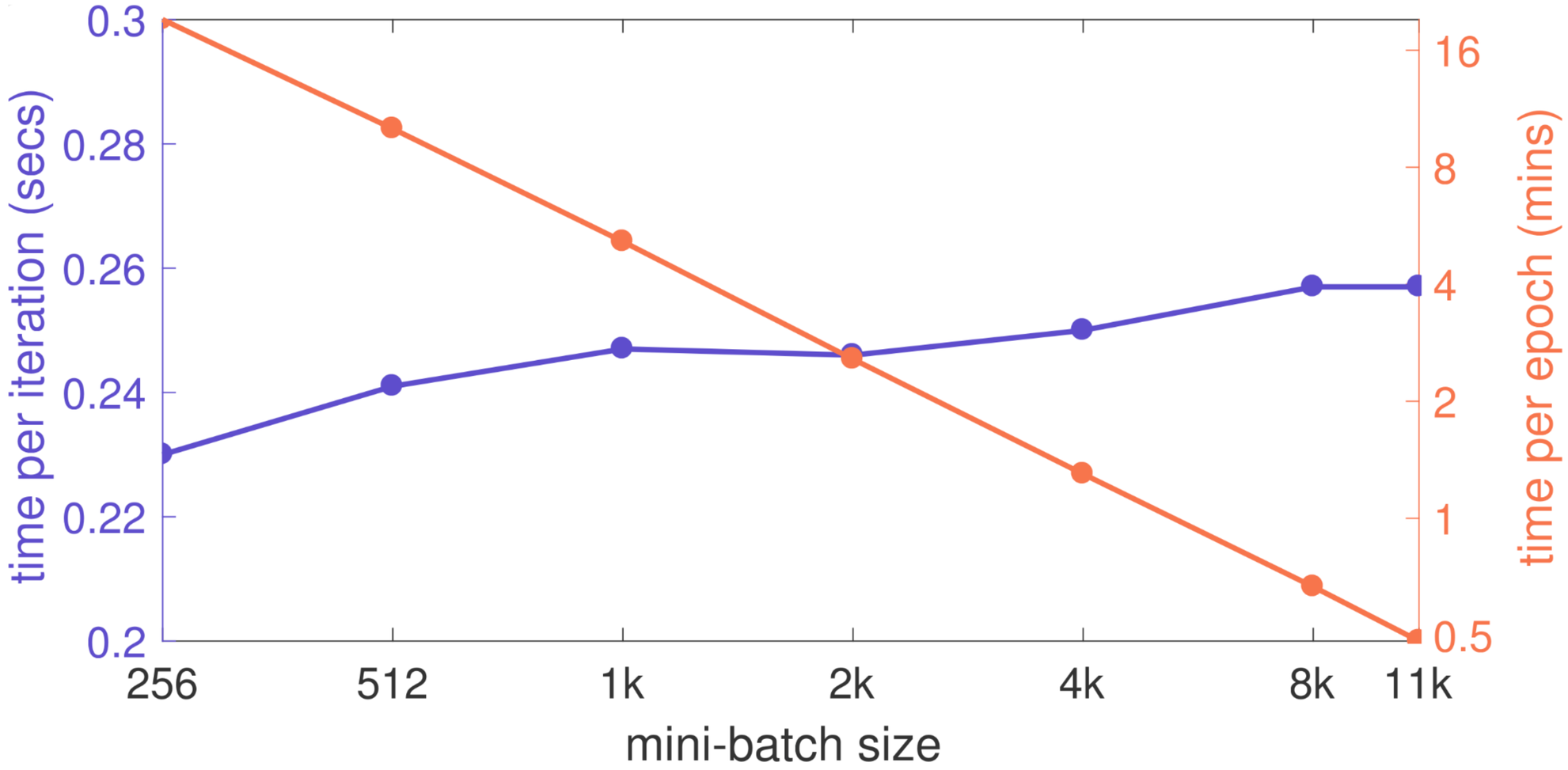
how can i solve this error?

apaszke commented on Mar 8, 2017 — Member

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

16 3

Размер батча и время эпохи



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

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

Просто так увеличить размер батча не выйдет

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

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

ℹ️ Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour.

Просто так увеличить размер батча не выйдет

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

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

ℹ️ Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour.

Просто так увеличить размер батча не выйдет

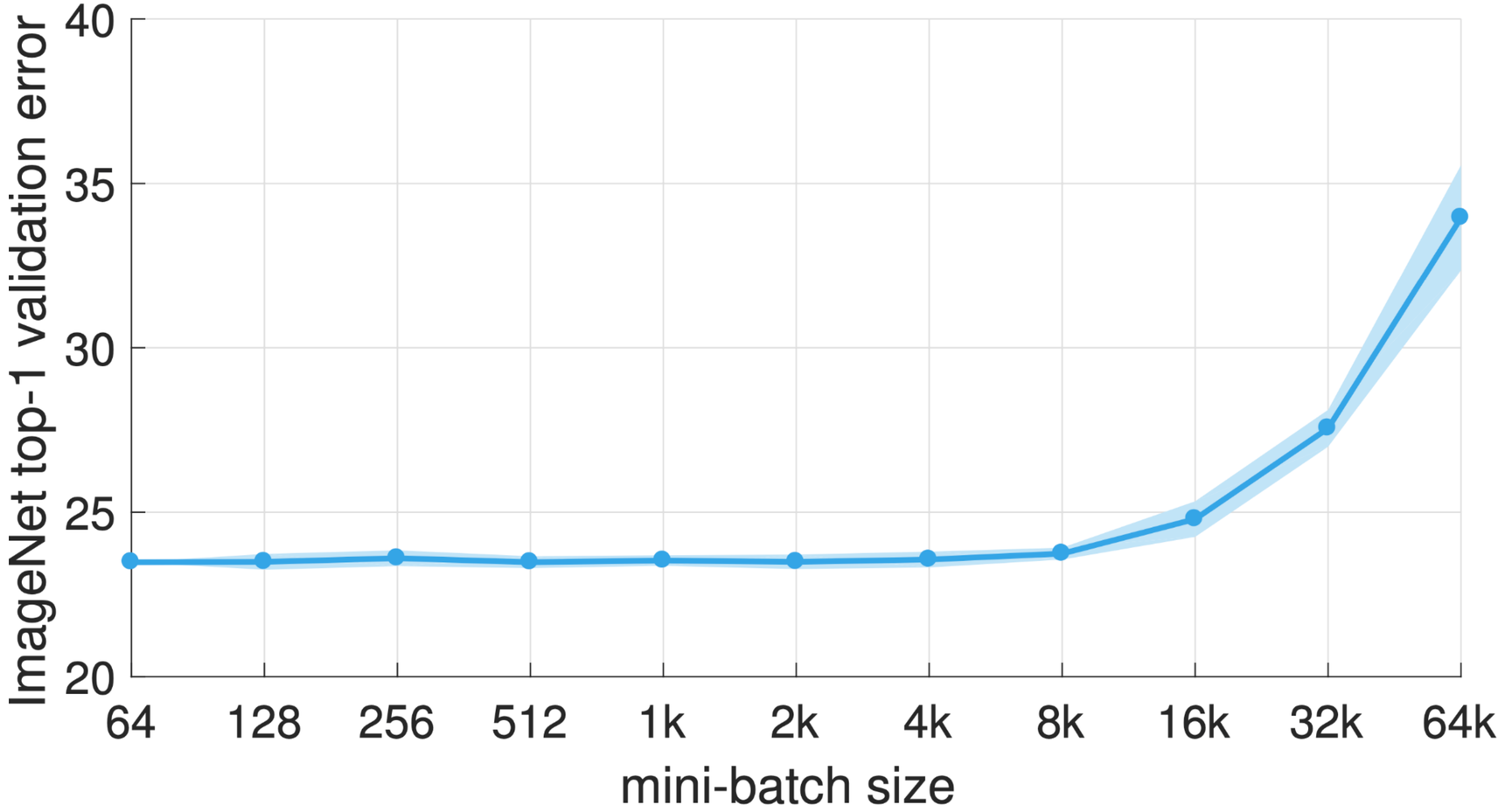
Обучение ResNet-50 на датасете ImageNet с разными вариантами увеличения размера батча.

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.

ℹ️ Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour.

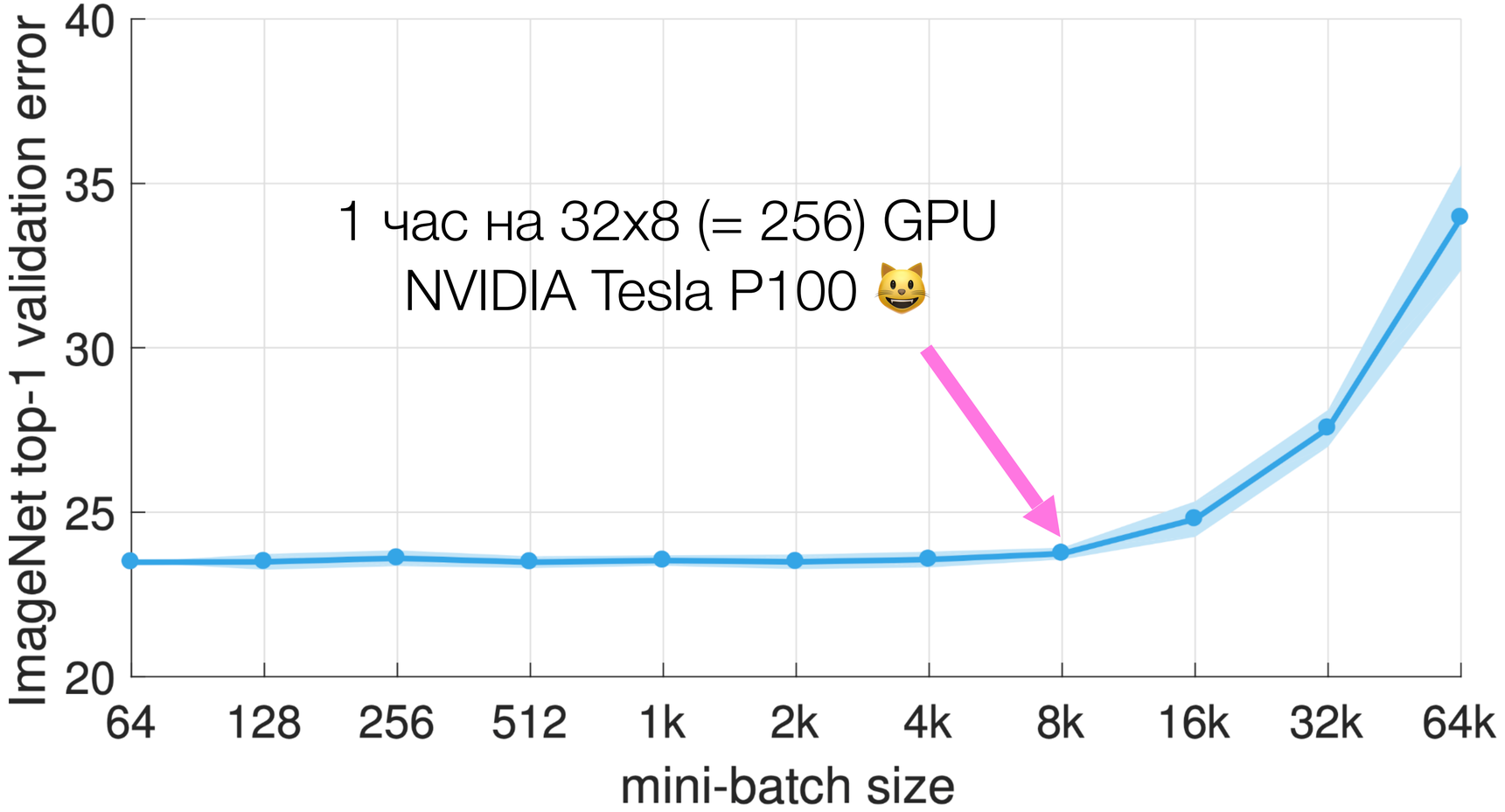
Просто так увеличить размер батча не выйдет



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

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

Просто так увеличить размер батча не выйдет



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

ℹ️ 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:

SGD

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

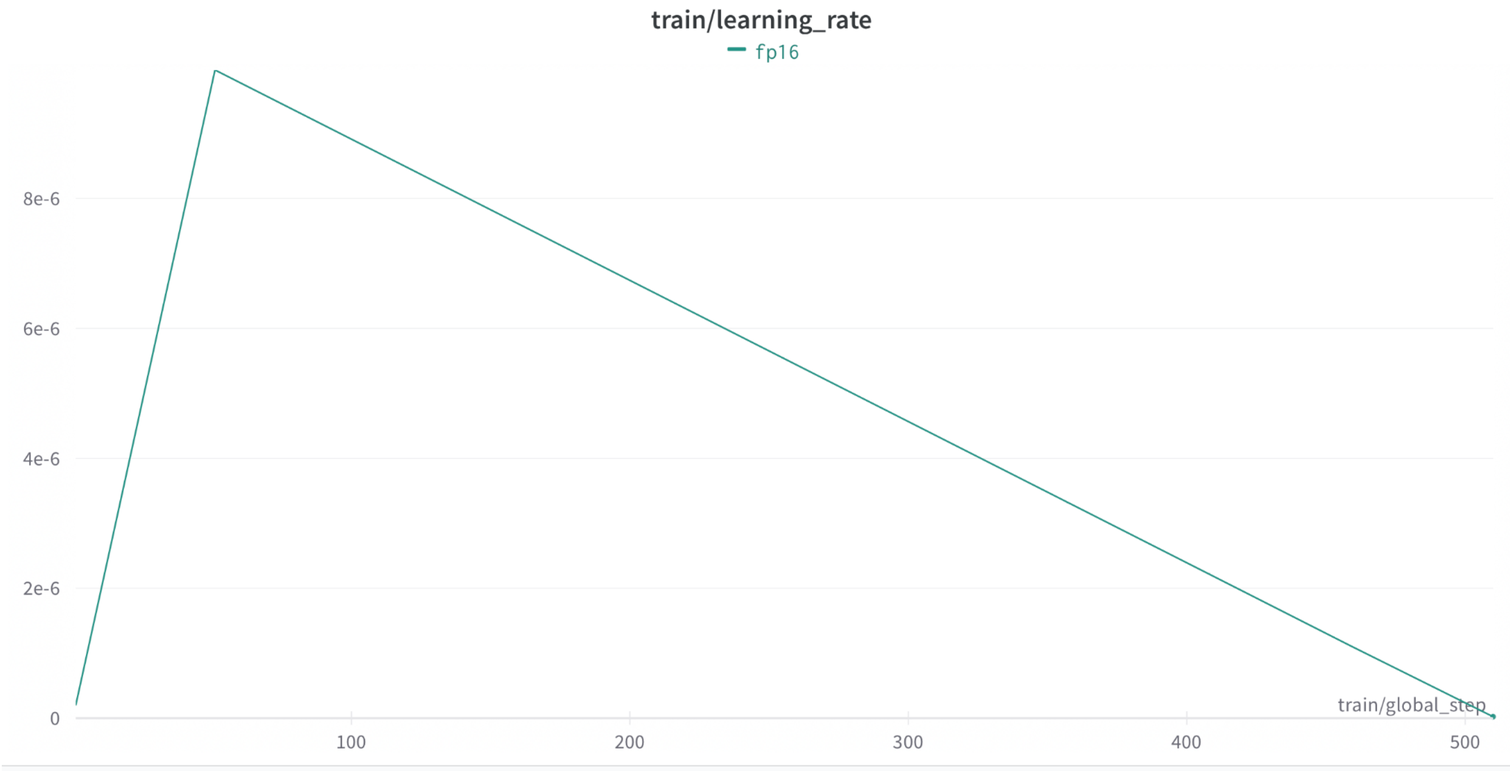
Adaptive algorithms (Adam, RMSProp and etc.)

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

2. Gradual warmup.

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

3. Square root scaling rule



ℹ️ Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. 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.

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

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

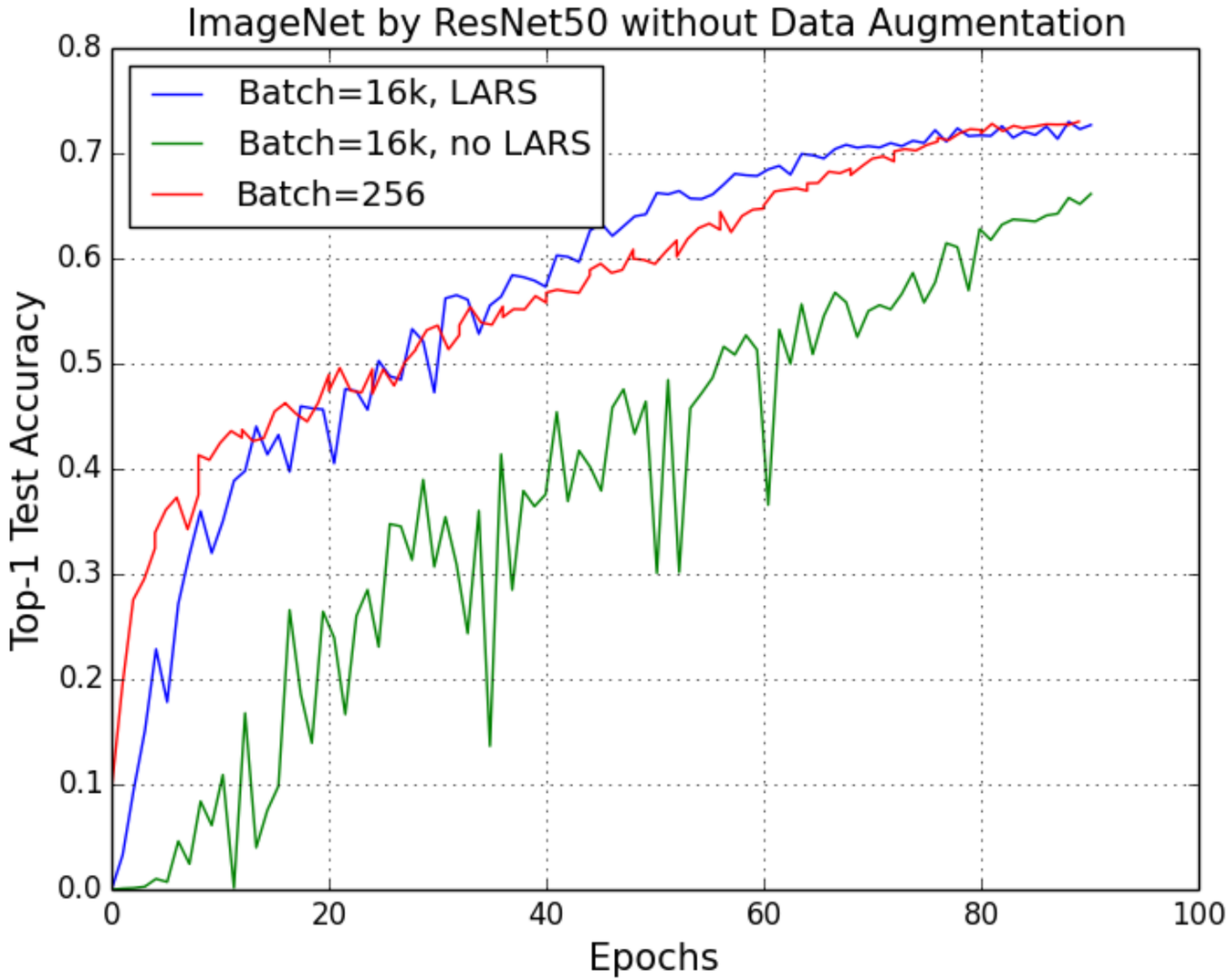
w_l - веса l-ого слоя

$\nabla_{w_l} L$ - градиент по l-ому слою

ℹ️ Large batch training of convolutional networks.

Как ещё увеличивать размер батча?

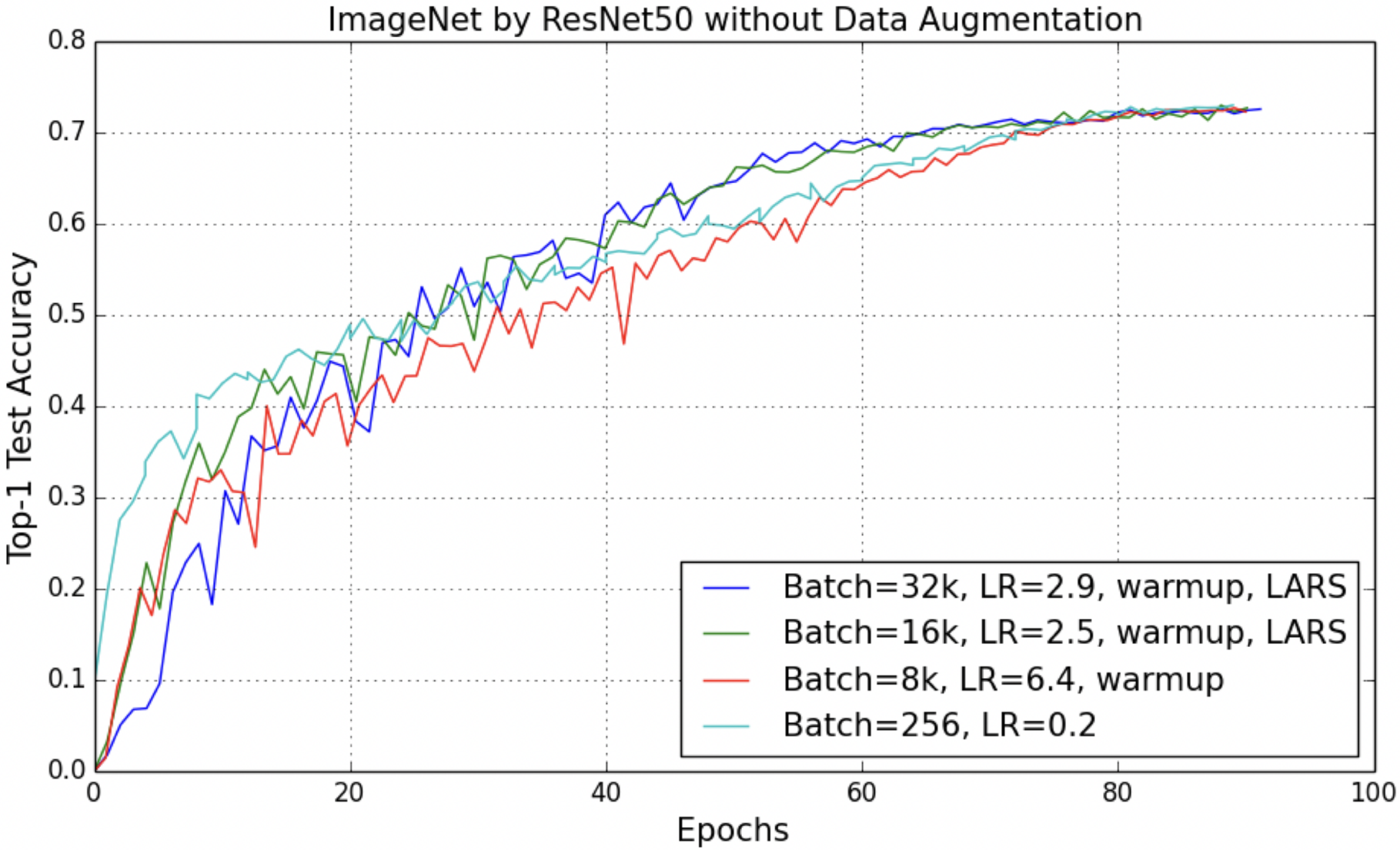
LARS



ℹ️ Large batch training of convolutional networks.

Как ещё увеличивать размер батча?

LARS



ℹ️ 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

ℹ️ 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

ℹ️ 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 😺

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

Аккумуляция градиентов

▶️ БЫЛО:

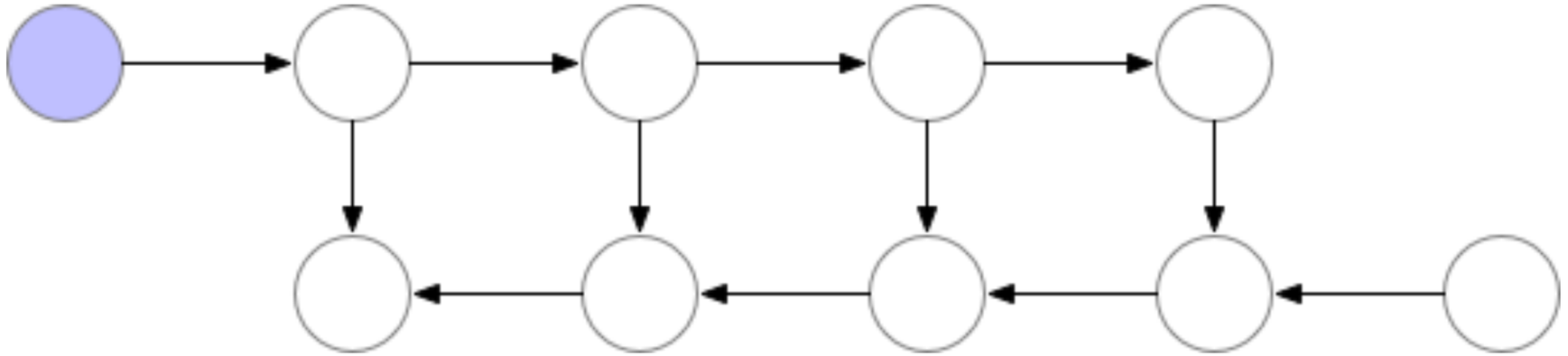
```python
# loop through batches
for (inputs, labels) in data_loader:

    # 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:

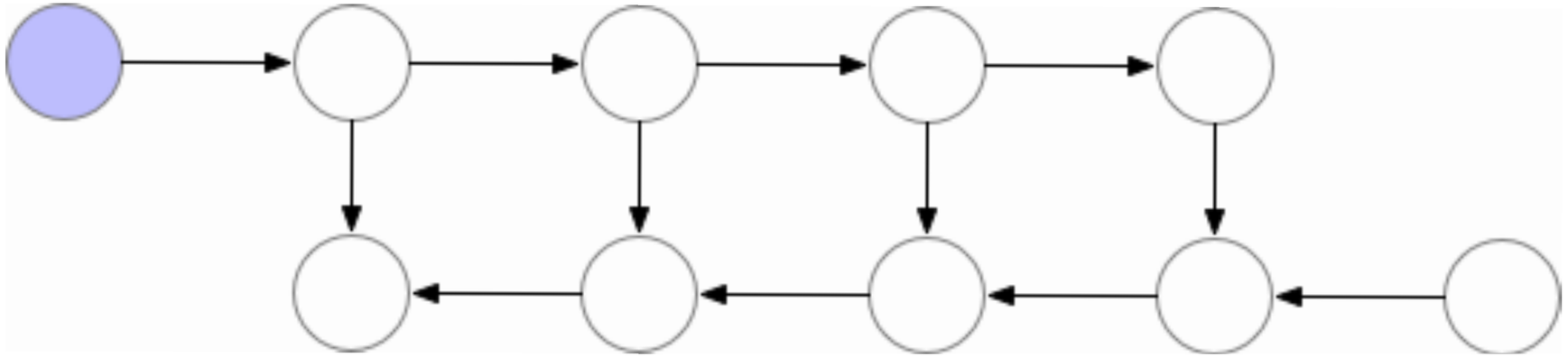
```python
# batch accumulation parameter
accum_iter = 4

# 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()
```

Gradient checkpointing
Идея

Обычные Forward и Backward проходы.

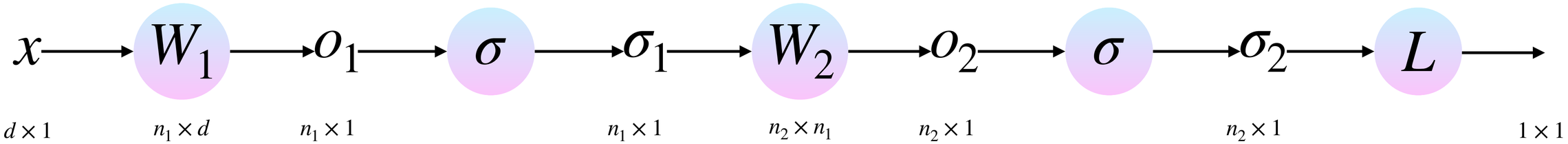


Экономный вариант Forward и Backward проходов. Для вычисления градиентов нужны значения активаций с предыдущих шагов. Вместо их хранения, они пересчитываются заново.



ℹ️ https://github.com/cybertronai/gradient-checkpointing

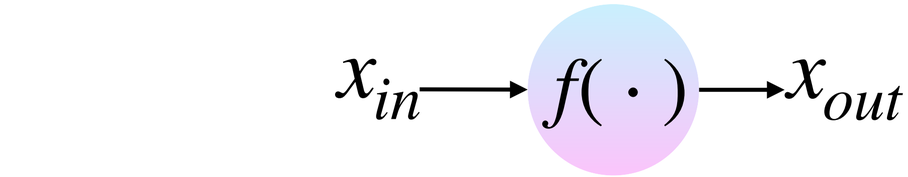
Gradient checkpointing
Идея



Знаем:

производную $\frac{\partial L}{\partial x_{out}}$ и

якобиан преобразования J



вычислить:

производную $\frac{\partial L}{\partial x_{in}}$

$$\frac{\partial L}{\partial x_{in}} = J \frac{\partial L}{\partial x_{out}}$$

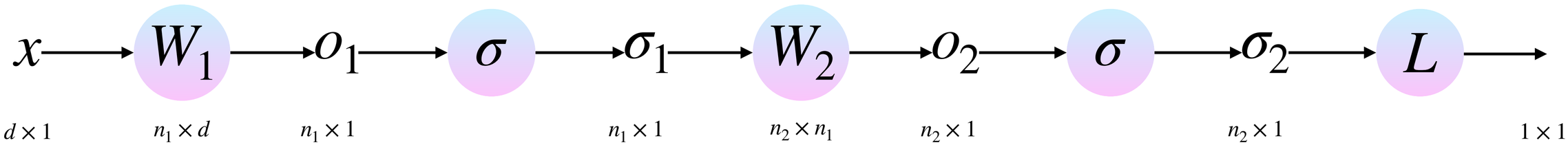
$n_{in} \times 1 \quad n_{in} \times n_{out} \quad n_{out} \times 1$

$$\left(\frac{\partial L}{\partial x_{in}}\right)^\top = \left(\frac{\partial L}{\partial x_{out}}\right)^\top J^\top$$

$1 \times n_{in} \quad 1 \times n_{out} \quad n_{out} \times n_{in}$

Gradient checkpointing

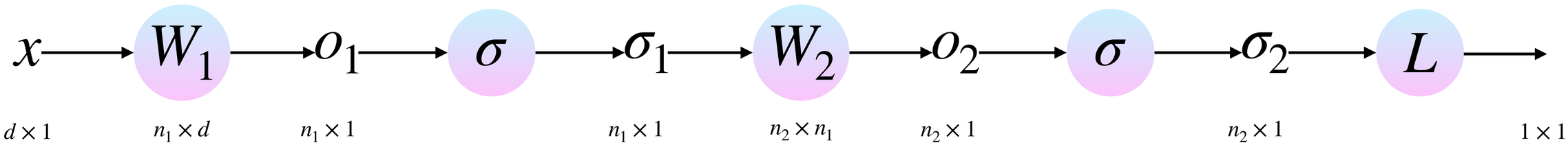
Идея



$$L = \frac{1}{2}\|\sigma_2 - y\|^2$$

Gradient checkpointing

Идея

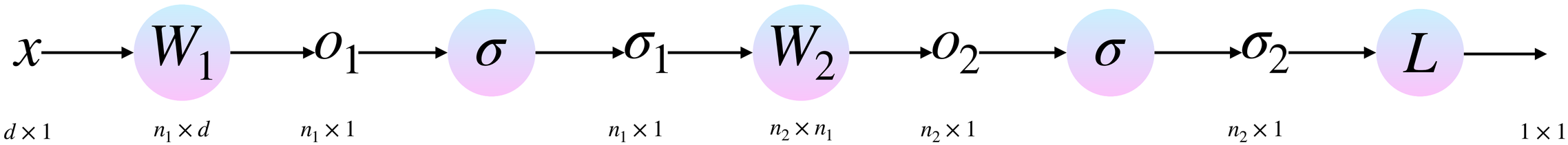


$$L = \frac{1}{2}\|\sigma_2 - y\|^2$$

$$\frac{\partial L}{\partial \sigma_2} = \sigma_2 - y$$

Gradient checkpointing

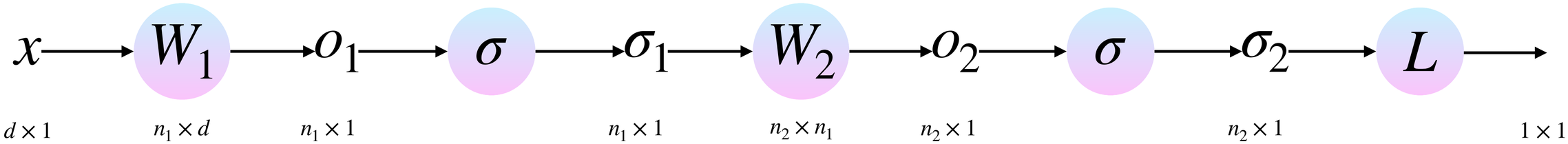
Идея



$$L = \frac{1}{2}\|\sigma_2 - y\|^2$$

$$\frac{\partial L}{\partial \sigma_2} = \sigma_2 - y \qquad \frac{\partial L}{\partial o_2} = J_\sigma \frac{\partial L}{\partial \sigma_2}$$

Gradient checkpointing
Идея

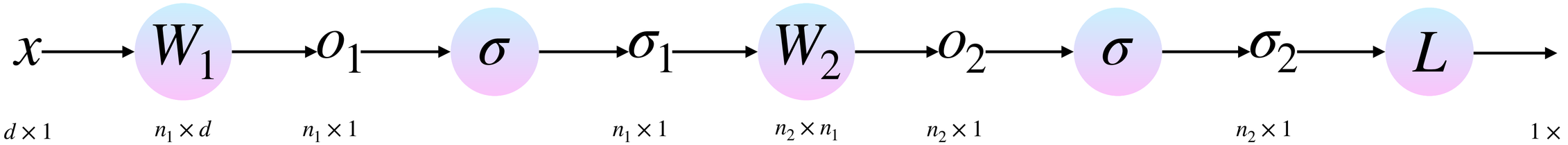


$$L = \frac{1}{2}\|\sigma_2 - y\|^2$$

$$J_\sigma = \text{diag}\left(\sigma(o_2)(1 - \sigma(o_2))\right) = \text{diag}\left(\sigma_2(1 - \sigma_2)\right)$$

$$\frac{\partial L}{\partial \sigma_2} = \sigma_2 - y \qquad \frac{\partial L}{\partial o_2} = J_\sigma \frac{\partial L}{\partial \sigma_2}$$

Gradient checkpointing
Идея



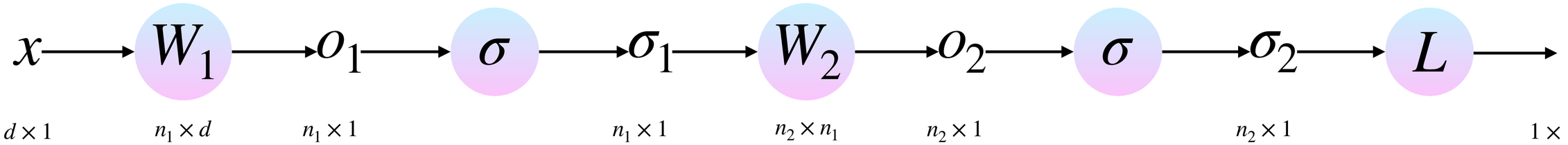
$$L = \frac{1}{2}\|\sigma_2 - y\|^2$$

$$J_\sigma = \text{diag}\left(\sigma(o_2)(1 - \sigma(o_2))\right) = \text{diag}\left(\sigma_2(1 - \sigma_2)\right)$$

$$\frac{\partial L}{\partial \sigma_2} = \sigma_2 - y$$

$$\frac{\partial L}{\partial o_2} = J_\sigma \frac{\partial L}{\partial \sigma_2} = \text{diag}\left(\sigma_2(1 - \sigma_2)\right)(\sigma_2 - y) = \sigma_2 \odot (1 - \sigma_2) \odot (\sigma_2 - y)$$

Gradient checkpointing
Идея



$$L = \frac{1}{2}\|\sigma_2 - y\|^2$$

$$J_\sigma = \text{diag}\left(\sigma(o_2)(1 - \sigma(o_2))\right) = \text{diag}\left(\sigma_2(1 - \sigma_2)\right)$$

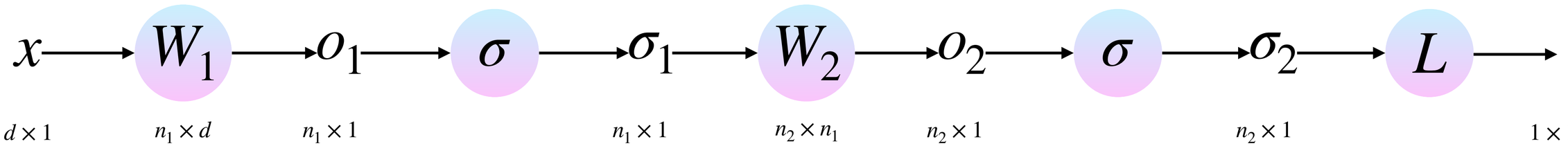
$$\frac{\partial L}{\partial \sigma_2} = \sigma_2 - y \qquad \frac{\partial L}{\partial o_2} = J_\sigma \frac{\partial L}{\partial \sigma_2} = \text{diag}\left(\sigma_2(1 - \sigma_2)\right)(\sigma_2 - y) = \sigma_2 \odot (1 - \sigma_2) \odot (\sigma_2 - y)$$

$$\underset{n_1 n_2 \times 1}{\frac{\partial L}{\partial W_2}} = \underset{n_1 n_2 \times n_2}{J_{W_2}(W_2)} \underset{n_2 \times 1}{\frac{\partial L}{\partial o_2}}$$

$n_2 \times n_1$

$(n_2 \times n_1) \times n_2$

Gradient checkpointing
Идея



$$L = \frac{1}{2}\|\sigma_2 - y\|^2$$

$$J_\sigma = \text{diag}\left(\sigma(o_2)(1 - \sigma(o_2))\right) = \text{diag}\left(\sigma_2(1 - \sigma_2)\right)$$

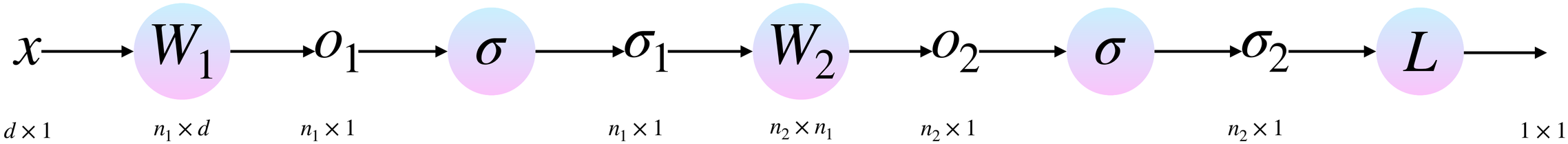
$$\frac{\partial L}{\partial \sigma_2} = \sigma_2 - y \qquad \frac{\partial L}{\partial o_2} = J_\sigma \frac{\partial L}{\partial \sigma_2} = \text{diag}\left(\sigma_2(1 - \sigma_2)\right)(\sigma_2 - y) = \sigma_2 \odot (1 - \sigma_2) \odot (\sigma_2 - y)$$

$$\frac{\partial L}{\partial W_2} = J_{W_2}(W_2)\frac{\partial L}{\partial o_2} = \sigma_2 \odot (1 - \sigma_2) \odot (\sigma_2 - y)\sigma_1^\top$$

$n_2 \times n_1$ $\quad$ $n_2 \times 1$ $\quad$ $1 \times n_1$

Gradient checkpointing

Идея



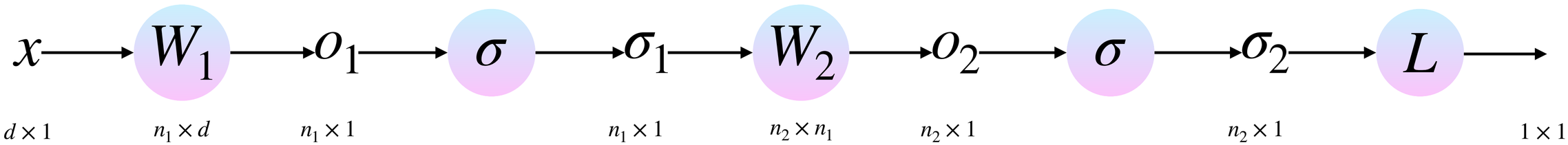
$$\frac{\partial L}{\partial o_2} = \sigma_2 \odot (1 - \sigma_2) \odot (\sigma_2 - y) \qquad J_{W_2}(\sigma_1) = W_2^\top$$

$$\underset{n_1 \times 1}{\frac{\partial L}{\partial \sigma_1}} = \underset{n_1 \times n_2}{J_{W_2}(\sigma_1)} \underset{n_2 \times 1}{\frac{\partial L}{\partial o_2}} \qquad \frac{\partial L}{\partial \sigma_1} = W_2^\top \sigma_2 \odot (1 - \sigma_2) \odot (\sigma_2 - y)$$

$$\frac{\partial L}{\partial o_1} = \sigma_1 \odot (1 - \sigma_1) \odot \frac{\partial L}{\partial \sigma_1} = \sigma_1 \odot (1 - \sigma_1) \odot W_2^\top \sigma_2 \odot (1 - \sigma_2) \odot (\sigma_2 - y)$$

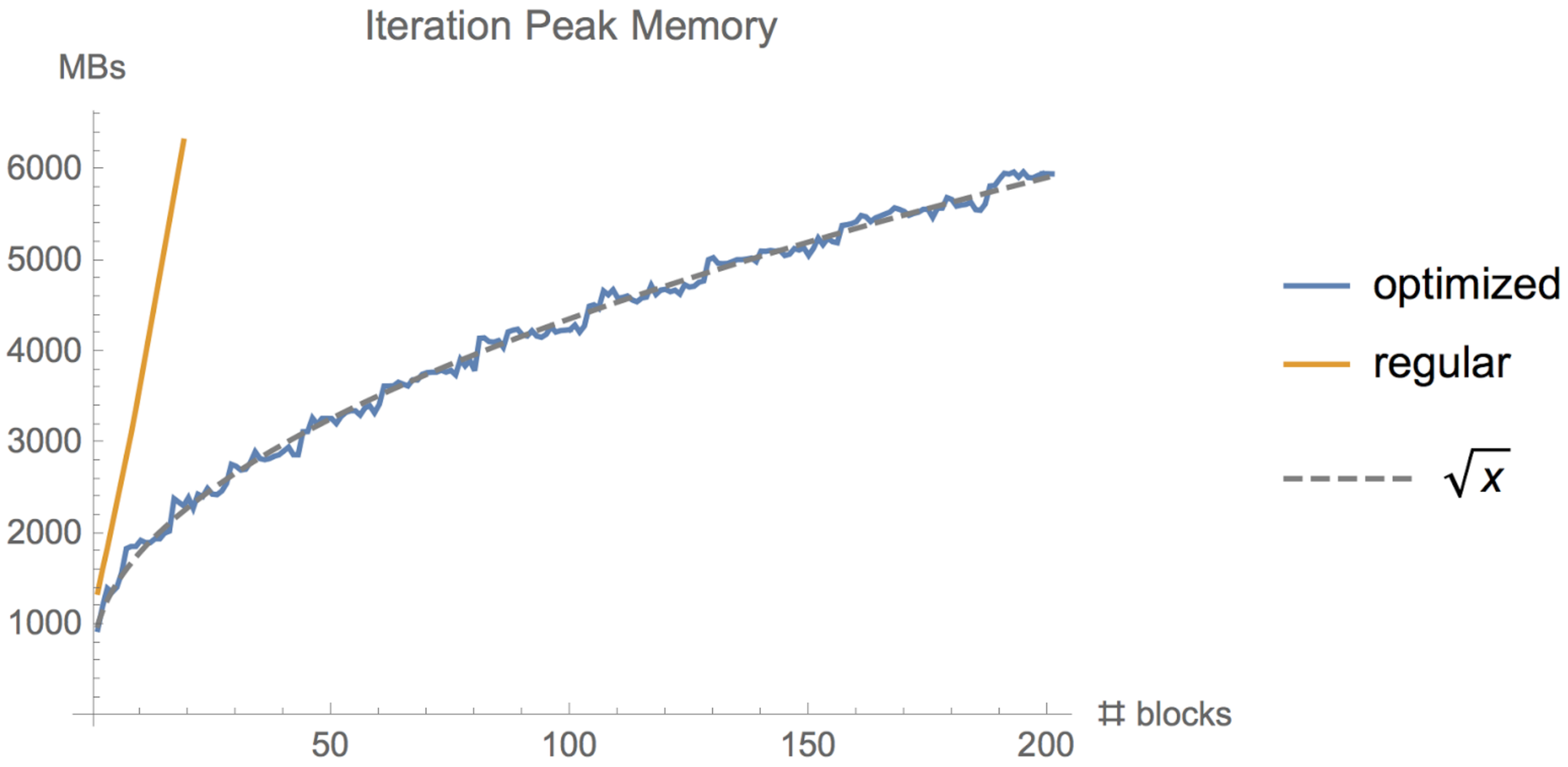
Gradient checkpointing

Идея



$$\frac{\partial L}{\partial W_1} = J_{W_1}(W_1)\frac{\partial L}{\partial o_1} = \sigma_1 \odot (1 - \sigma_1) \odot W_2^\top \sigma_2 \odot (1 - \sigma_2) \odot (\sigma_2 - y)x^\top$$

Gradient checkpointing
Идея



Memory used while training a ResNet model with large batch size

ℹ️ https://github.com/cybertronai/gradient-checkpointing

Gradient checkpointing

Примеры

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

```
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

ℹ️ Training larger-than-memory PyTorch models using gradient checkpointing. Hugging Face. Efficient training on a single GPU.

Activation quantization

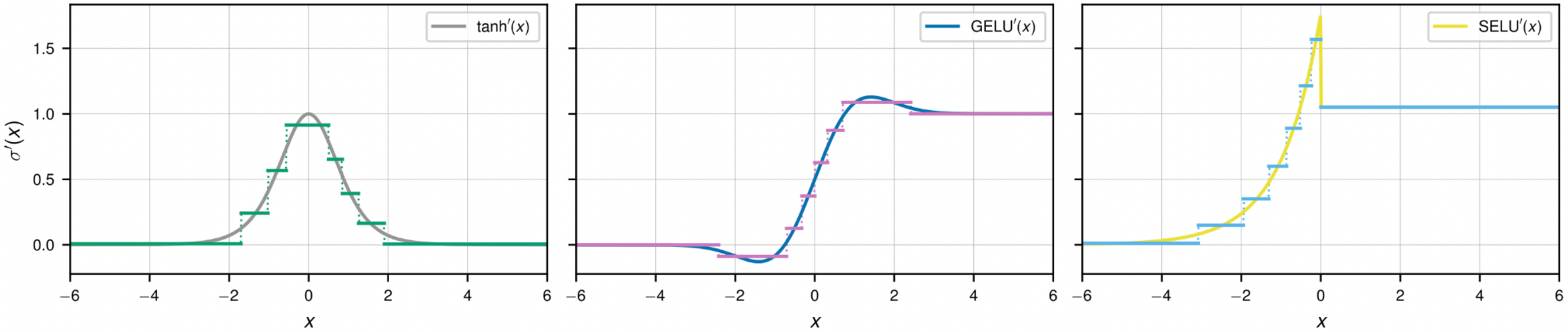


Figure 1. Optimized 3-bit piecewise-constant approximations of the derivatives of activation functions.

	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

ℹ️ FewBit - Compression schema for gradient of activations in backward pass

Automatic Mixed Precision training

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

- 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

ℹ️ Automatic Mixed Precision Tutorials using pytorch PyTorch: CUDA Automatic Mixed Precision training examples

LION optimizer

Symbolic Discovery of Optimization Algorithms

Algorithm 1 AdamW Optimizer

given $\beta_1, \beta_2, \epsilon, \lambda, \eta, f$
initialize $\theta_0, m_0 \leftarrow 0, v_0 \leftarrow 0, t \leftarrow 0$
while θ_t not converged **do**
  $t \leftarrow t + 1$
  $g_t \leftarrow \nabla_\theta f(\theta_{t-1})$
  update EMA of g_t **and** g_t^2
  $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
  $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$
  bias correction
  $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$
  $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$
  update model parameters
  $\theta_t \leftarrow \theta_{t-1} - \eta_t (\hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon) + \lambda \theta_{t-1})$
end while
return θ_t

Algorithm 2 Lion Optimizer (ours)

given $\beta_1, \beta_2, \lambda, \eta, f$
initialize $\theta_0, m_0 \leftarrow 0$
while θ_t not converged **do**
  $g_t \leftarrow \nabla_\theta f(\theta_{t-1})$
  update model parameters
  $c_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
  $\theta_t \leftarrow \theta_{t-1} - \eta_t (\text{sign}(c_t) + \lambda \theta_{t-1})$
  update EMA of g_t
  $m_t \leftarrow \beta_2 m_{t-1} + (1 - \beta_2) g_t$
end while
return θ_t

Symbolic Discovery of Optimization Algorithms